

USING CONCEPTUAL SPACES FOR ARTIFICIAL INTELLIGENCE

Dissertation
zur Erlangung des Doktorgrades
des Fachbereichs Humanwissenschaften
der Universität Osnabrück

vorgelegt
von
Lucas Bechberger
aus
Baden-Baden

Osnabrück, 2022

Dedicated to my wife Lera and my son Kai.

ABSTRACT

Concepts are mental representations of categories and lie at the core of human cognition: They allow us to generalize from individual sensations, which is a requirement for both logical reasoning and effective communication. The cognitive framework of conceptual spaces proposes to represent concepts as convex regions in low-dimensional similarity spaces with interpretable dimensions. It has received attention from a wide variety of disciplines, including psychology, philosophy, linguistics, neuroscience, and artificial intelligence. Conceptual spaces can be interpreted as an intermediate representational layer bridging the traditional dichotomy into symbolic and subsymbolic approaches. As such, the conceptual spaces framework promises to solve the symbol grounding problem by relating symbols to regions in the conceptual layer, whose dimensions are grounded in subsymbolic processes.

This dissertation investigates how conceptual spaces can be applied in artificial agents. Using the distinction of three representational layers (subsymbolic, conceptual, and symbolic), we identify three core problems that need to be solved, namely, the formalization of the conceptual layer and learning processes for attaching it to the subsymbolic and the symbolic layer, respectively.

In order to implement conceptual spaces in artificial agents, one needs a thorough mathematical formalization of the framework which closes many of the remaining degrees of freedom of the general proposal. For instance, instead of only requiring conceptual regions to be convex, we need a parametric description of these regions such that they can be easily represented in a computer program.

The first major contribution of this dissertation consists in providing such a mathematical formalization of the conceptual layer, along with a publicly available implementation. This formalization makes use of star-shapedness instead of convexity in order to encode correlations between domains in a geometric way. It furthermore offers a wide variety of cognitive operations, both for creating new concepts and for quantifying relations between concepts.

The second problem concerns the grounding of the conceptual space in subsymbolic processing: We need to find an automated way of mapping raw perceptual input (such as camera images or other sensor values) into the conceptual layer. Only if such a mapping is available, an artificial agent can successfully conceptualize its surroundings and harness the representational power of the conceptual layer.

In this dissertation, we introduce two important approaches for grounding the dimensions of a conceptual space. On the one hand, we consider machine learning techniques for dimensionality reduction,

with a special focus on representation learning with deep neural networks. The mapping learned by this approach can be applied to novel inputs but has little to no psychological grounding. On the other hand, we consider an approach which is based on dissimilarity ratings obtained from human subjects in psychological experiments. These ratings can be translated into a spatial representation with a technique called multidimensional scaling. Here, a psychological grounding of the similarity space is obvious, but an application to novel inputs is very difficult.

The second major contribution of this dissertation consists in the proposal of a hybrid procedure, which attempts to combine the advantages of these two traditional approaches: The structure of the conceptual space is initialized based on human dissimilarity ratings, and deep neural networks are then subsequently used to learn a mapping from raw inputs to coordinates in this space. We support this proposal by two experimental studies on two different datasets.

Thirdly, we cannot assume that the designer of an autonomous system can foresee all possible circumstances in which this system may operate. Therefore, a fixed set of manually created concepts will not be sufficient for an autonomous system – the artificial agent needs to be equipped with the ability to learn and adapt concepts based on the observations it makes. In other words, we need an automated way of extracting symbols from the conceptual layer.

In this dissertation, we discuss several ways of making current machine learning algorithms more cognitively plausible. This includes the need for incremental processing (observations are typically made one after another) and unsupervised or semi-supervised learning (not all observations come with an explicit label). We furthermore consider the grounding of concepts in language games (where a group of agents needs to negotiate a common conceptualization of their environment) and the incorporation of explicit top-down constraints from the symbolic layer (reflecting the influence of abstract background knowledge).

Overall, the content of this dissertation presents progress with respect to all three core issues. It illustrates how conceptual spaces can be used as a neural-symbolic approach to cognitive AI, which is able to integrate many ideas from different strands of research.

ZUSAMMENFASSUNG

Konzepte sind mentale Repräsentationen von Kategorien und bilden den Kern menschlicher Kognition: Sie erlauben es uns von einzelnen Wahrnehmungen zu generalisieren, was eine Voraussetzung für logisches Schließen und effektive Kommunikation ist. Das kognitive Framework der Conceptual Spaces schlägt vor, Konzepte als konvexe Regionen in niedrigdimensionalen Ähnlichkeitsräumen mit interpretierbaren Dimensionen darzustellen. Es hat große Aufmerksamkeit in verschiedenen Disziplinen erfahren, unter anderem Psychologie, Philosophie, Linguistik, Neurowissenschaften und künstliche Intelligenz. Conceptual Spaces können als eine intermediäre Repräsentationsschicht interpretiert werden, welche die traditionelle Zweiteilung in symbolische und subsymbolische Ansätze überbrückt. Das Conceptual Spaces Framework verspricht das Symbol Grounding Problem zu lösen, indem Symbole mit Regionen in der konzeptuellen Schicht verknüpft werden, deren Dimensionen in subsymbolischen Prozessen verankert sind.

Diese Dissertation untersucht wie Conceptual Spaces in künstlichen Agenten angewendet werden können. Mithilfe der Unterscheidung in drei Repräsentationsschichten (subsymbolisch, konzeptuell, sowie symbolisch) identifizieren wir drei Kernprobleme, die gelöst werden müssen, nämlich die Formalisierung der konzeptuellen Schicht sowie Lernprozesse zur Verknüpfung mit der subsymbolischen und der symbolischen Schicht.

Um Conceptual Spaces in künstlichen Agenten zu implementieren benötigt man eine gründliche mathematische Formalisierung des Frameworks, die viele der im generellen Ansatz verbliebenen Freiheitsgrade schließt. Anstatt nur zu fordern, dass konzeptuelle Regionen konvex sind, benötigt man beispielsweise eine parametrische Beschreibung dieser Regionen, sodass sie in einem Computerprogramm gut repräsentiert werden können.

Der erste Hauptbeitrag dieser Dissertation besteht darin solch eine mathematische Formalisierung der konzeptuellen Schicht bereitzustellen, gemeinsam mit einer öffentlich verfügbaren Implementierung. Diese Formalisierung benutzt Sternförmigkeit statt Konvexität, um Korrelationen zwischen Domänen in einer geometrischen Art und Weise darzustellen. Sie bietet außerdem eine große Bandbreite an kognitiven Operationen, sowohl um neue Konzepte zu erzeugen, als auch um Beziehungen zwischen Konzepten zu quantifizieren.

Das zweite Problem bezieht sich auf die Verankerung des Conceptual Space in subsymbolischen Prozessen: Wir benötigen einen automatisierten Weg um perzeptuellen Input (wie zum Beispiel Kamerabilder oder andere Sensorwerte) in die konzeptuelle Schicht zu projizieren. Nur

wenn eine solche Funktion verfügbar ist, kann ein künstlicher Agent erfolgreich seine Umgebung konzeptualisieren und das Repräsentationsvermögen der konzeptuellen Schicht ausnutzen.

In dieser Dissertation stellen wir zwei wichtige Ansätze für die Verankerung der Dimensionen eines Conceptual Space vor. Einerseits betrachten wir Techniken des maschinellen Lernens zur Dimensionsreduktion mit einem besonderen Augenmerk auf Repräsentationslernen mit tiefen neuronalen Netzen. Die Funktion, die von diesen neuronalen Netzen gelernt wird, kann auf neue Stimuli angewendet werden, hat aber nur wenig bis keine psychologische Plausibilität. Andererseits betrachten wir einen Ansatz, der auf Ähnlichkeitsbewertungen von Teilnehmern psychologischer Experimente basiert. Diese Bewertungen können mithilfe einer Technik namens Multidimensional Scaling in eine räumliche Repräsentation überführt werden. In diesem Fall ist eine psychologische Verankerung des Ähnlichkeitsraumes offensichtlich, allerdings ist eine Anwendung auf neue Stimuli sehr schwierig.

Der zweite Hauptbeitrag dieser Dissertation besteht aus dem Vorschlag einer hybriden Vorgehensweise, welche versucht die Vorteile der beiden traditionellen Ansätze zu verbinden: Die Struktur des Conceptual Space wird basierend auf menschlichen Ähnlichkeitsbewertungen initialisiert und tiefe neuronale Netze werden anschließend verwendet, um eine Funktion von rohen Inputs zu Koordinaten in diesem Raum zu lernen. Wir untermauern diesen Vorschlag mit zwei experimentellen Studien auf zwei unterschiedlichen Datensets.

Drittens können wir nicht annehmen, dass der Entwickler eines autonomen Systems alle möglichen Umstände vorhersehen kann, in welchen dieses System agieren wird. Deshalb wird eine festgelegte Menge an manuell erstellten Konzepten für autonome Systeme nicht ausreichen – der künstliche Agent muss mit der Fähigkeit ausgestattet sein, Konzepte aufgrund der gemachten Beobachtungen zu erzeugen und zu modifizieren. In anderen Worten: Wir benötigen einen automatisierten Weg, um Symbole aus der konzeptuellen Schicht zu extrahieren.

In dieser Dissertation diskutieren wir verschiedene Möglichkeiten, um aktuelle Algorithmen des maschinellen Lernens kognitiv plausibler zu machen. Dies schließt die Notwendigkeit für inkrementelle Verarbeitung (Beobachtungen werden typischerweise sequentiell gemacht) und unüberwachten oder semi-überwachten Lernens (nicht alle Beobachtungen haben ein explizites Label) ein. Wir betrachten außerdem die Verankerung von Konzepten in Language Games (wo eine Gruppe von Agenten eine gemeinsame Konzeptualisierung ihrer Umgebung aushandeln muss) und die Berücksichtigung expliziter top-down Beschränkungen aus der symbolischen Schicht (was den Einfluss abstrakten Hintergrundwissens widerspiegelt).

Insgesamt präsentiert der Inhalt dieser Dissertation Fortschritte bezüglich aller drei Kernprobleme. Er zeigt, wie Conceptual Spaces als ein neurosymbolischer Ansatz kognitiver KI genutzt werden kann, der es ermöglicht, verschiedene Forschungsstränge zu integrieren.

PUBLICATIONS

The research presented in this thesis has appeared previously in the following publications:

- *Formalization and Implementation of Conceptual Spaces*
 - Lucas Bechberger and Kai-Uwe Kühnberger. A Thorough Formalization of Conceptual Spaces. In Gabriele Kern-Isberner, Johannes Fürnkranz, and Matthias Thimm, editors, *KI 2017: Advances in Artificial Intelligence*, volume 10505 of *Lecture Notes in Computer Science*, pages 58–71, Cham, 2017. Springer International Publishing.
 - Lucas Bechberger and Kai-Uwe Kühnberger. A Comprehensive Implementation of Conceptual Spaces. In *5th International Workshop on Artificial Intelligence and Cognition*, 2017.
 - Lucas Bechberger. The Size of a Hyperball in a Conceptual Space. *arXiv preprint, arXiv:1708.05263*, 2017.
 - Lucas Bechberger and Kai-Uwe Kühnberger. Measuring Relations Between Concepts In Conceptual Spaces. In Max Bramer and Miltos Petridis, editors, *37th SGAI International Conference on Artificial Intelligence*, pages 87–100. Springer International Publishing, 2017.
 - Lucas Bechberger and Kai-Uwe Kühnberger. Formal Ways for Measuring Relations between Concepts in Conceptual Spaces. *Expert Systems*, 38(7):e12348, 2018.
 - Lucas Bechberger and Kai-Uwe Kühnberger. Formalized Conceptual Spaces with a Geometric Representation of Correlations. In Mauri Kaipainen, Frankz Zenker, Antti Hautamäki, and Peter Gärdenfors, editors, *Conceptual Spaces: Elaborations and Applications*, volume 405 of *Synthese Library*, pages 29–58. Springer International Publishing, Cham, 2019.
- *Hybrid Approach for Grounding Conceptual Spaces*
 - Lucas Bechberger and Kai-Uwe Kühnberger. Towards Grounding Conceptual Spaces in Neural Representations. In *12th International Workshop on Neural-Symbolic Learning and Reasoning*, 2017.
 - Lucas Bechberger and Elektra Kypridemou. Mapping Images to Psychological Similarity Spaces Using Neural Networks. In *6th International Workshop on Artificial Intelligence and Cognition*, pages 26–39, 2018.

- Lucas Bechberger and Margit Scheibel. Analyzing Psychological Similarity Spaces for Shapes. In Mehwish Alam, Tanya Braun, and Bruno Yun, editors, *Ontologies and Concepts in Mind and Machine*, pages 204–207, Cham, 2020. Springer International Publishing.
- Lucas Bechberger and Kai-Uwe Kühnberger. *Generalizing Psychological Similarity Spaces to Unseen Stimuli – Combining Multidimensional Scaling with Artificial Neural Networks*. In Lucas Bechberger, Kai-Uwe Kühnberger, and Mingya Liu, editors, *Concepts in Action: Representation, Learning, and Application*, volume 9 of *Language, Cognition, and Mind*, pages 11–36. Springer International Publishing, Cham, 2021.
- Lucas Bechberger and Kai-Uwe Kühnberger. Grounding Psychological Shape Space in Convolutional Neural Networks. In *3rd International Workshop on Cognition: Interdisciplinary Foundations, Models, and Applications*, 2021.
- *Learning Conceptual Regions*
 - Lucas Bechberger. Towards Conceptual Logic Tensor Networks. In *5th Workshop on Cognition And Ontologies (CAOS)*, 2021.
- *General Concept Research*
 - Lucas Bechberger and Mingya Liu. *Concepts in Action: Introduction*. In Lucas Bechberger, Kai-Uwe Kühnberger, and Mingya Liu, editors, *Concepts in Action: Representation, Learning, and Application*, volume 9 of *Language, Cognition, and Mind*, pages 1–9. Springer International Publishing, Cham, 2021.

The following manuscript about the proposed hybrid approach is currently in preparation:

- Lucas Bechberger and Margit Scheibel. Modeling the Holistic Perception of Everyday Object Shapes with Conceptual Spaces. In preparation.

The research presented in this dissertation has been presented orally at the following occasions without an associated official publication:

- Lucas Bechberger and Kai-Uwe Kühnberger. Towards Concept Formation in Conceptual Spaces. In *Interdisciplinary College*, 2017.
- Lucas Bechberger. Conceptual Spaces: A Bridge Between Neural and Symbolic Representations? In *Dagstuhl Seminar “Human-Like Neural-Symbolic Computing”*, 2017.
- Lucas Bechberger. Conceptual Spaces for Artificial Intelligence: Formalization, Domain Grounding, and Concept Formation. In *Workshop: Concept Learning and Reasoning in Conceptual Spaces*, 2017.

- Lucas Bechberger. Machine Learning in Conceptual Spaces: Two Learning Processes. In *1st International Workshop "Concepts in Action: Representation, Learning, and Application"*, 2018.
- Lucas Bechberger. Representing Correlations in Conceptual Spaces. In *Research Colloquium of the Düsseldorf Center for Logic and Philosophy of Science*, 2018.
- Lucas Bechberger. Using Conceptual Spaces for Cognitive AI. In *TriCoLore 2.0 Workshop*, 2019.
- Lucas Bechberger and Margit Scheibel. Representing Complex Shapes with Conceptual Spaces. In *2nd International Workshop "Concepts in Action: Representation, Learning, and Application"*, 2020.
- Lucas Bechberger and Kai-Uwe Kühnberger. Using Convolutional Neural Networks to Map Line Drawings Into Psychological Shape Space. In *3rd International Workshop "Concepts in Action: Representation, Learning, and Application"*, 2021.
- Lucas Bechberger. Grounding Psychological Similarity Spaces in Deep Neural Networks. In *Computational Cognition Workshop*, 2021.

Finally, published research by the author, which is not part of this dissertation, includes the following publications:

- Lucas Bechberger, Maria Schmidt, Alex Waibel, and Marcello Federico. Personalized News Event Retrieval for Small Talk in Social Dialog Systems. In *12th ITG Symposium on Speech Communication*, 2016.
- Lucas Bechberger. Review of the Book "Cognitive Design for Artificial Minds" by Antonio Lieto. *ACM Computing Reviews*, 2021.
- Paola Gega, Mingya Liu, and Lucas Bechberger. Numerical Concepts in Context. In Lucas Bechberger, Kai-Uwe Kühnberger, and Mingya Liu, editors, *Concepts in Action: Representation, Learning, and Application*, volume 9 of *Language, Cognition, and Mind*, pages 93–119. Springer International Publishing, Cham, 2021.

All relevant source code for the research presented in this dissertation has been made publicly available on GitHub. The implementation of the conceptual spaces formalization can be found at <https://github.com/lbechberger/ConceptualSpaces> (current version: 1.3.2). Source code for reproducing the results of the two experimental studies on the hybrid domain-grounding approach is available at <https://github.com/lbechberger/LearningPsychologicalSpaces> (current version: 1.6).

PREFACE

This dissertation marks the end of a long PhD journey, which started with first ideas for a PhD project in late 2015: I had the ambitious plan to create an overall system for connecting the symbolic to the subsymbolic layer through a complex concept formation algorithm. Of course, things turned out to be much more complicated than I initially thought, so I got quickly sidetracked into several sub-projects which contained even more hidden complexity.

My aim of implementing conceptual spaces in software required a mathematical formalization of the framework. What I had originally estimated to be an easy one-month task quickly led down a rabbit hole and kept me quite busy for almost two years.

Also the idea of grounding conceptual spaces in deep learning seemed straightforward in the beginning, but required much more experimentation and engineering than I had initially expected. A collaboration with Elektra Kypridemou, which started at the AIC workshop in 2017, sparked the idea for an hybrid approach: Why not combine deep learning with psychological similarity spaces? Combining two approaches of course also doubled the workload, so this sub-project has kept me busy for another two years and was finished just in time for submitting this dissertation.

Along the way, I had somehow managed to collect several other project ideas, on which I was eager to work on, but which unfortunately never made it to the top of my priority list: After having read about neural networks that could discover interpretable features from unlabeled data, I tried to use them for my purposes. However, replicating the results from the original paper was not very successful. Despite the assistance from several students, the project never made it off the ground and finally had to be abandoned.

At a Dagstuhl seminar in 2017, I was introduced to Logic Tensor Networks, a neural framework for incorporating background knowledge in the form of logical formulas into machine learning. I was immediately hooked by the idea and started playing around with this approach in the context of conceptual spaces. As it turned out, my initial setup was not working very well, and while I had some good ideas about how to improve things, I also had to put a (temporary) end to this project in order to finish my dissertation.

Also creating the concept formation algorithm I had originally envisioned turned out to be infeasible given my finite time budget and my plethora of sub-projects. Nevertheless, I am fairly satisfied with the outcome of my PhD studies: I was able to cover a broad range of interesting topics and I feel that I made meaningful progress on most of them. In addition to the sub-projects I had to abandon, there are

still many other open ends and ideas for further research – certainly enough for at least one complete researcher life.

This dissertation presents the research I have conducted throughout my PhD. These scientific contributions have also been made possible through the support from a variety of directions:

I would like to thank my supervisor Kai-Uwe Kühnberger, who was very open to my unsolicited application, who gave me complete freedom to explore different ideas in my research, and who was always approachable and willing to give me valuable feedback on my work. I would also like to thank the members of the AI group and my colleagues at the Institute of Cognitive Science for creating a good working climate and for giving me valuable impulses and feedback at several internal talks and discussions. I want to especially highlight three reading groups on conceptual spaces (Eleni Gregoromichelaki and Nico Potyka), concept research (Danny Arlen de Jesus Gomez Ramirez, Ulf Krumnack, and Mingya Liu) and deep learning (Holger Finger, Ulf Krumnack, Julius Schöning, Leon Sütfeld, and many others), which have helped me to obtain a better overview of the respective fields and to gain a deeper understanding of the respective literature.

I am also grateful to my collaborators Elektra Kypridemou and Margit Scheibel for the respective joint studies. Combining expertise from different backgrounds was crucial for the success of these collaborations and very stimulating for my own research. Moreover, I would like to thank Peter Gärdenfors, Rob Goldstone, Antonio Lieto, Luciano Serafini, and Michael Spranger for acting as informal mentors. They provided me with insightful discussions and guidance, both in person and via email. The CARLA summer school, which I co-organized with Ulf Krumnack and Mingya Liu, and the CARLA workshop, which we were able to establish as a regular event (and which is currently co-organized by Viviana Haase, Nicolás Araneda Hinrichs, Mingya Liu, Stefan Schneider, Corina Strößner, Alessandro Panunzi, and Paola Vernillo) have been crucial in shaping my understanding of concept research as an interdisciplinary endeavor. Moreover, I have immensely benefited from interactions with other researchers at various conferences and workshops, as well as from a number of personal email exchanges (for instance with Jessica Horst and Michael Hout, who were very open about sharing the raw similarity ratings for their NOUN data set).

Last, but certainly not least, I want to thank my family and friends for their support. This includes my brother Thomas, who has helped me to fine-tune the style template I used for formatting this dissertation. My special thanks go above all to my wife Lera, who has been by my side throughout the whole PhD journey, who moved all across Germany for me, and who always helped me to keep my focus on the important things, both in research and in life.

Overview

Abstract	v
Zusammenfassung	vii
Publications	ix
Preface	xiii
1 Introduction	1
I Formalizing Conceptual Spaces	
2 Defining Conceptual Regions	53
3 Operations for Combining Concepts	101
4 Measuring Relations between Concepts	147
II Machine Learning and Optimization Background	
5 General Machine Learning Background	211
6 Representation Learning With Artificial Neural Networks	281
7 Learning Concepts in a Cognitive Way	357
8 Multidimensional Scaling and a Hybrid Proposal	433
III Experimental Studies	
9 Feasibility Study on Novel Objects	513
10 A Dataset on Shape Perception	561
11 A Psychological Similarity Space for Shapes	623
12 Learning a Mapping into Shape Space	665
IV Summary	
13 Conclusions	725
Bibliography	751
List of Figures	791
List of Tables	809
List of Algorithms	819
List of Definitions, Lemmata, Propositions, and Corollaries	821
V Appendix	
A Defining Conceptual Regions	825
B Operations for Combining Concepts	833
C Measuring Relations Between Concepts	843
D Feasibility Study on Novel Objects	867
E A Dataset on Shape Perception	889
F A Psychological Similarity Space for Shapes	895
G Learning a Mapping into Shape Space	901

Declaration

913

Contents

Abstract	v
Zusammenfassung	vii
Publications	ix
Preface	xiii
1 Introduction	1
1.1 Concepts From the View of Psychology	2
1.1.1 Psychological Theories of Concepts	3
1.1.2 Empirical Observations about Concepts	6
1.2 The Conceptual Spaces Framework	14
1.2.1 Overview of the Framework	14
1.2.2 Relation to Psychology	17
1.2.3 The Classical Layers of Representation	22
1.2.4 Using the Conceptual Layer to Solve the Symbol Grounding Problem	25
1.2.5 Relation to Machine Learning	29
1.2.6 Neural Grounding	33
1.3 Contributions of This Dissertation	38
1.3.1 The Envisioned AI System	40
1.3.2 A Comprehensive Formalization of the Framework	43
1.3.3 Obtaining the Dimensions of a Conceptual Space	45
1.3.4 Learning Conceptual Regions	49
I Formalizing Conceptual Spaces	
2 Defining Conceptual Regions	53
2.1 The Original Conceptual Spaces Framework	54
2.1.1 The Structure of a Conceptual Space	54
2.1.2 Conceptual Regions	56
2.2 An Argument Against Convexity	60
2.2.1 Convex Sets Cannot Encode Cross-Domain Cor- relations	60
2.2.2 Potential Solutions	62
2.3 A Parametric Definition of Concepts	65
2.3.1 Formalizing Conceptual Similarity Spaces	66
2.3.2 Crisp Conceptual Regions	67
2.3.3 Fuzzy Conceptual Regions	69
2.3.4 Implementation and Example: Fruit Space	74
2.4 Related Work	79
2.4.1 Related General Ideas	79
2.4.2 Prior Formalizations of Conceptual Spaces	82
2.4.3 Composite Concepts	86

2.5	Detailed Comparison to Other Formalizations	88
2.5.1	Comparison to Adams and Raubal	91
2.5.2	Comparison to Rickard	93
2.5.3	Comparison to Lewis and Lawry	95
2.6	Summary	98
3	Operations for Combining Concepts	101
3.1	Intersection	102
3.1.1	Intersection of Cores	103
3.1.2	Intersection of Concepts	107
3.1.3	Implementation and Example	109
3.2	Union	116
3.2.1	Definition	116
3.2.2	Implementation and Example	118
3.3	Negation	122
3.4	Subspace Projection	124
3.4.1	Definition	125
3.4.2	Implementation and Example	126
3.5	Axis-Parallel Cut	127
3.5.1	Definition	131
3.5.2	Implementation and Example	132
3.6	Supported Applications	135
3.6.1	Concept Formation	135
3.6.2	Concept Combination	136
3.7	Comparison to Other Formalizations	142
3.8	Summary	145
4	Measuring Relations between Concepts	147
4.1	Concept Size	148
4.1.1	The Size of a Fuzzified Cuboid's α -Cut	149
4.1.2	The Size of a Concept	155
4.1.3	Implementation and Example	157
4.2	Subsethood	161
4.2.1	Definition	162
4.2.2	Implementation and Example	166
4.3	Implication	168
4.3.1	Definition	168
4.3.2	Implementation and Example	169
4.4	Concept Similarity	171
4.4.1	Similarity as Inverse Distance	172
4.4.2	Definition	177
4.4.3	Implementation and Example	179
4.5	Betweenness	184
4.5.1	Betweenness in the Literature	185
4.5.2	Definition	187
4.5.3	Implementation and Example	190
4.6	Supported Applications	199
4.6.1	Concept Formation	199
4.6.2	Commonsense Reasoning	200
4.7	Comparison to Other Formalizations	203
4.8	Summary	206

II Machine Learning and Optimization Background	
5	General Machine Learning Background 211
5.1	General Notions in Machine Learning 212
5.1.1	Task and Model 213
5.1.2	Experience and Evaluation 215
5.1.3	Fitting the Model's Parameters 222
5.1.4	Practical Considerations 225
5.2	Machine Learning Algorithms 228
5.2.1	Linear Regression 229
5.2.2	Logistic Regression 233
5.2.3	Support Vector Machines 238
5.2.4	Decision Trees 250
5.2.5	k Nearest Neighbors 255
5.3	Dimensionality Reduction 258
5.3.1	Feature Selection 259
5.3.2	Feature Extraction 265
5.3.3	Metric Learning 271
5.4	Summary 279
6	Representation Learning With Artificial Neural Networks 281
6.1	Representation Learning 282
6.1.1	Desiderata for Good Representations 283
6.1.2	Evaluating Representations 285
6.2	Artificial Neural Networks 291
6.2.1	Fully Connected Feedforward Networks 293
6.2.2	Convolutional Neural Networks 298
6.2.3	Regularization 305
6.2.4	Backpropagation 309
6.2.5	Training Algorithms 311
6.2.6	Other Considerations 315
6.3	Network Architectures for Representation Learning . . 318
6.3.1	Autoencoders 319
6.3.2	Word Embeddings 323
6.3.3	Generative Adversarial Networks 331
6.3.4	Variational Autoencoders 337
6.4	Representation Learning for Conceptual Spaces 346
6.4.1	General Considerations 347
6.4.2	The Rectangle Domain 353
6.5	Summary 356
7	Learning Concepts in a Cognitive Way 357
7.1	Concept Learning in Conceptual Spaces 359
7.1.1	Learning Concepts from Few Examples 359
7.1.2	The Bayesian Perspective 361
7.1.3	The Machine Learning Perspective 367
7.2	Learning Concepts under Knowledge-Based Constraints 370
7.2.1	General Approaches 372
7.2.2	Logic Tensor Networks 377
7.2.3	Towards Conceptual Logic Tensor Networks . . 384
7.3	Learning Concepts from Unlabeled Data 393

7.3.1	Classical Clustering Techniques	393
7.3.2	Hierarchical Concept Formation with COBWEB	400
7.3.3	Adaptive Resonance Theory	406
7.3.4	Incremental Category Learning with SUSTAIN	410
7.4	Learning Concepts through Communication	414
7.4.1	General Considerations	415
7.4.2	Vertical Transmission	420
7.4.3	Horizontal Transmission	423
7.5	Summary	431
8	Multidimensional Scaling and a Hybrid Proposal	433
8.1	Obtaining Dissimilarity Ratings	435
8.1.1	Direct Methods	437
8.1.2	The Spatial Arrangement Method	438
8.1.3	Indirect Methods	441
8.1.4	General Considerations	443
8.1.5	Extracting Dissimilarity Ratings From Datasets	446
8.2	Multidimensional Scaling as an Optimization Problem	451
8.2.1	The Optimization Problem	451
8.2.2	Metric and Nonmetric MDS	454
8.2.3	General Remarks	456
8.2.4	An Illustrative Example	457
8.3	Analytical Solutions with Classical MDS	460
8.3.1	Mathematical Background	460
8.3.2	The Algorithm	463
8.3.3	An Illustrative Example	466
8.4	Kruskal's Gradient Descent Approach	469
8.4.1	Gradient Descent for Stress	469
8.4.2	Monotone Regression	471
8.4.3	Example for Monotone Regression	473
8.4.4	The Algorithm	475
8.4.5	An Illustrative Example	477
8.5	Iterative Stress Majorization Through SMACOF	480
8.5.1	Mathematical Background	480
8.5.2	A Majorizing Function for Stress	482
8.5.3	The Algorithm	486
8.5.4	An Illustrative Example	487
8.6	Evaluating MDS Solutions	490
8.6.1	Stress-based Evaluation	491
8.6.2	Interpretability-based Evaluation	493
8.7	A Hybrid Proposal	498
8.7.1	Proposed Procedure	498
8.7.2	Possible Network Architectures	501
8.7.3	Related Work	505
8.8	Summary	509
III Experimental Studies		
9	Feasibility Study on Novel Objects	513
9.1	Comparing Metric and Nonmetric MDS	514
9.1.1	The NOUN Dataset	514

9.1.2	Methods	516
9.1.3	Results	523
9.1.4	Discussion	532
9.2	Machine Learning Experiments	533
9.2.1	General Methods	534
9.2.2	Analyzing Feature Spaces and Baselines	539
9.2.3	Comparing Feature Spaces and Regressors	542
9.2.4	Comparing MDS Algorithms	548
9.2.5	Generalization to Other Target Spaces	553
9.2.6	Discussion	557
9.3	Summary	559
10	A Dataset on Shape Perception	561
10.1	Related Work	562
10.1.1	Shape Perception in (Neuro-)Psychology	562
10.1.2	The Shape Domain in Conceptual Spaces	571
10.1.3	Shape Features in Computer Vision	577
10.1.4	Shape Sensitivity of CNNs	582
10.2	Our New Dataset	591
10.2.1	Stimuli	591
10.2.2	Elicitation of Dissimilarity Ratings	593
10.2.3	Elicitation of Feature Ratings	595
10.3	Analysis of the Dataset	600
10.3.1	Statistical Tools	600
10.3.2	Comparing Visual and Conceptual Similarity	607
10.3.3	Comparing Pre-Attentive and Attentive Feature Ratings	615
10.4	Summary	620
11	A Psychological Similarity Space for Shapes	623
11.1	Obtaining Similarity Spaces with MDS	624
11.1.1	Comparing Mean Dissimilarities to Median Dissimilarities	624
11.1.2	Extracting the Similarity Spaces	626
11.2	Are Distances and Dissimilarities Correlated?	629
11.2.1	Methods	630
11.2.2	Baselines	631
11.2.3	Similarity Spaces	639
11.3	Are Conceptual Regions Well-Formed?	645
11.3.1	Overlap of Conceptual Regions	646
11.3.2	Size of Conceptual Regions	648
11.4	Are There Interpretable Directions?	650
11.4.1	Methods	651
11.4.2	General Observations	653
11.4.3	Individual Features	657
11.5	Summary	661
12	Learning a Mapping into Shape Space	665
12.1	Sketch Recognition	667
12.1.1	Datasets	667
12.1.2	Approaches	670

12.2 Overall Approach	674
12.2.1 Data	675
12.2.2 Architecture	677
12.2.3 Training, Evaluation, and Hyperparameters	680
12.3 Obtaining Baseline Networks	684
12.3.1 Sketch Classification	684
12.3.2 Sketch Reconstruction	689
12.4 Transfer Learning	694
12.4.1 Photograph-Based Classification	695
12.4.2 Sketch Classification	699
12.4.3 Sketch Reconstruction	704
12.5 Multitask Learning	707
12.5.1 Sketch Classification	707
12.5.2 Sketch Reconstruction	709
12.6 Generalization to Other Target Spaces	712
12.6.1 Classification	712
12.6.2 Reconstruction	714
12.7 Summary	719
IV Summary	
13 Conclusions	725
13.1 Formalizing the Conceptual Layer	726
13.1.1 Lessons Learned	726
13.1.2 Open Ends	728
13.1.3 Outlook	730
13.2 Obtaining the Dimensions	734
13.2.1 Lessons Learned	734
13.2.2 Open Ends	737
13.2.3 Outlook	738
13.3 Learning Conceptual Regions	745
13.3.1 Lessons Learned	745
13.3.2 Outlook	746
13.4 The Big Picture	748
Bibliography	751
List of Figures	791
List of Tables	809
List of Algorithms	819
List of Definitions, Lemmata, Propositions, and Corollaries	821
V Appendix	
A Defining Conceptual Regions	825
A.1 An Argument Against Convexity	825
A.2 Formalizing Conceptual Similarity Spaces	826
A.3 Crisp Conceptual Regions	827
A.4 Fuzzy Conceptual Regions	829
B Operations for Combining Concepts	833
B.1 Intersection	833
B.2 Union	835

B.3	Subspace Projection	837
B.4	Axis-Parallel Cut	840
C	Measuring Relations Between Concepts	843
C.1	Size	843
C.2	Subsethood	855
C.3	Similarity	859
C.4	Betweenness	860
D	Feasibility Study on Novel Objects	867
D.1	Pixel Baseline with Uniform Weights	867
D.2	Pixel Baseline with Optimized Weights	873
D.3	Unregularized Regression for Different MDS Algorithms	878
D.4	Lasso Regression for Different MDS Algorithms	880
D.5	Lasso Regression for Different Dimensionality	883
E	A Dataset on Shape Perception	889
E.1	Correlations between Features	889
E.2	Attentive and Pre-Attentive Ratings	892
F	A Psychological Similarity Space for Shapes	895
F.1	Pixel Baseline for the Mean Matrix	895
F.2	Pixel Baseline for the Median Matrix	897
G	Learning a Mapping into Shape Space	901
G.1	Class Structure of Sketch Datasets	902
G.2	Transfer Learning on Sketch Classification	904
G.3	Transfer Learning on Sketch Reconstruction	906
G.4	Multitask Learning on Sketch Classification	907
G.5	Multitask Learning on Sketch Reconstruction	909
G.6	Classification-Based Generalization Experiments	910
G.7	Reconstruction-Based Generalization Experiments	912
	Declaration	913

1 | INTRODUCTION

1.1	Concepts From the View of Psychology	2
1.1.1	Psychological Theories of Concepts	3
1.1.2	Empirical Observations about Concepts	6
1.2	The Conceptual Spaces Framework	14
1.2.1	Overview of the Framework	14
1.2.2	Relation to Psychology	17
1.2.3	The Classical Layers of Representation	22
1.2.4	Using the Conceptual Layer to Solve the Symbol Grounding Problem	25
1.2.5	Relation to Machine Learning	29
1.2.6	Neural Grounding	33
1.3	Contributions of This Dissertation	38
1.3.1	The Envisioned AI System	40
1.3.2	A Comprehensive Formalization of the Framework	43
1.3.3	Obtaining the Dimensions of a Conceptual Space	45
1.3.4	Learning Conceptual Regions	49

Recent years have seen great successes of artificial intelligence systems based on deep neural networks. These include areas such as computer vision (e.g., superhuman performance in image classification [193]), natural language processing (e.g., language models capable of generating fluent texts in various styles [83]), and reinforcement learning (e.g., superhuman performance in the ancient game of Go [371]). However, in all of these cases, the input-output mapping learned by the neural networks is quite opaque and cannot be easily analyzed or interpreted by human experts. The recent successes of deep neural networks have thus been accompanied with an urge for more human-like, explainable AI [113, 133, 259, 273, 274].

*Recent successes of
deep learning*

On the other hand, classical AI approaches based on formal logics and hand-coded rules have already for decades been successfully applied to various tasks such as planning [165], automated theorem proving [355], and representing information in a structured way for the semantic web [171]. These approaches result in systems which can be easily debugged and which consist of interpretable components. However, they are usually ill-equipped to deal with noise, uncertainty, and inconsistencies. Moreover, it is unclear how to link their abstract symbolic descriptions to perception and action in the real world.

*Symbolic approaches
to artificial
intelligence*

Their individual successes indicate that both the connectionist and the symbolic paradigm have their merits while at the same time facing considerable limitations. The paradigm of neural-symbolic integration [61, 111, 112, 279] proposes to devise hybrid systems encompassing both neural and symbolic aspects in order to combine the strengths of both approaches while eliminating their weaknesses. Our research can

*Neural-symbolic
integration*

be seen in the context of neural-symbolic integration with an emphasis cognitive AI, i.e., intelligent systems inspired by and based on findings from cognitive psychology [177, 259, 273, 274]. By using human cognition as a starting point for artificial systems, cognitive AI attempts to avoid the limitations and pitfalls of purely mathematically derived models. In this dissertation, we will focus on the psychological notion of concepts, which are mental representations of categories in the world allowing the human mind to abstract from individual observations. More specifically, we employ a geometric representation of conceptual knowledge in order to build a bridge between connectionist and symbolic AI systems.

Overview

This introductory chapter sets the stage for the research contributions described in the remainder of this dissertation. In Section 1.1, we present different psychological theories about the mental representation of conceptual knowledge and consider several important empirical observations with respect to the processes involved in learning and using concepts. This serves as a starting point for our research on concept-based cognitive AI. We then give a broad overview of the conceptual spaces framework in Section 1.2, describing its main components and its relation to psychological theories, the symbol grounding problem, machine learning, and neuroscience. Moreover, we summarize four important example applications of this framework in the context of artificial intelligence. Finally, in Section 1.3, we preview the structure of this dissertation and its main research contributions, namely, a novel mathematical formalization of the conceptual spaces framework and a hybrid approach for grounding its dimensions.

The content of this chapter is partially based on material published in [50] (Section 1.1) and [44] (Section 1.3.1).

1.1 CONCEPTS FROM THE VIEW OF PSYCHOLOGY

What are concepts?

It is impossible to talk about human cognition without talking about concepts since they form an abstraction of reality that is central to the functioning of the human mind. A concept can be defined as a "nonlinguistic psychological representation of a class of entities in the world" [298, Chapter 11]. According to Goldstone [170], one of the most important functions of concepts is that they allow us to treat all members of a given category as equivalent for a given task, even though we might be well aware of differences between them. This way, concepts allow us to communicate efficiently, to make predictions (e.g., about unobserved features of a category member), and to generalize beyond individual objects and observations. *But how are concepts represented mentally? How are they acquired? How does the human mind use concepts in cognitive tasks?* Such questions have been a subject of discussion since antiquity and remain highly relevant in multiple fields, including (cognitive) psychology, philosophy, linguistics, and artificial intelligence (see [49, 114, 189, 272, 275, 298]).

The remainder of this section is mainly based on the thorough overview of psychological concept research provided by Murphy [298]. We first introduce different psychological theories about the mental representation of concepts in Section 1.1.1. Afterwards, we summarize empirical observations about concept usage in Section 1.1.2 and discuss to what extent they can be explained by the different theories.

1.1.1 Psychological Theories of Concepts

In the following, we will briefly describe four broad views from the psychological literature about how concepts are mentally represented. We will also highlight connections to other disciplines, especially artificial intelligence and machine learning.

The *classical view* of concepts dates back to Aristotle and has been the predominant approach held implicitly by many psychologists in the beginning of concept research (e.g., [204]). It can be summarized as follows [298, Chapter 2]: Concepts are mentally represented as definitions, which provide a list of necessary and jointly sufficient conditions for membership in the category. If one of these conditions does not apply to an observation, this observation cannot be a member of the category (necessity). Moreover, if an observation fulfills all of the listed conditions, then it must be a member of the category (joint sufficiency). In other words, the definition includes everything that belongs to the category and excludes everything that does not. The membership in the category is assumed to be binary, i.e., any given observation either belongs to the category or it does not belong to the category – there are no borderline cases. Moreover, the classical view does not make any distinction between category members – all observations meeting the definition are equally good representatives of the category. Finally, since concepts are based on definitions, one can use logical connectives like AND, OR, and IF to define more complex concepts based on simpler ones. The resulting set of conditions for such a complex concept can also be interpreted as a set of logical rules determining category membership. The classical view of concepts is thus tightly connected to formal logics, where concepts can be interpreted as well-defined sets and logical connectives can be used to combine them with each other.

The classical view on concepts

The classical view has been challenged on theoretical grounds, for instance by Wittgenstein [437] who noted that it is very difficult to provide a definition of many everyday concepts such as SPORTS based on a set of necessary and jointly sufficient conditions. If concepts were represented mentally by definitions, one would, however, not expect such difficulties in listing the respective conditions. We will see in Section 1.1.2 that the classical view also fails to account for many empirical observations related to concept learning and concept use. It has therefore been essentially abandoned in the field of psychology.

Challenges for the classical view

Relation to artificial intelligence

Nevertheless, logic-based representations are still used in other sub-fields of cognitive science which are concerned with concepts: For instance, theoretical linguists pursue the goal of assigning more precise meaning to natural language expressions by mainly applying logic-based formalisms [17]. Moreover, the conceptual core of the semantic web [59] is based on large ontologies [171] containing hierarchies of concepts formulated in description logics. Finally, formal logics are also prevalent in symbolic approaches to artificial intelligence [341].

The prototype view on concepts

The *prototype theory* of concepts dates back to the pioneering work of Rosch [336]. It assumes that a concept is not represented by a logical definition, but that it is based on a prototypical member. Category membership is then not based on the fulfillment of a list of conditions, but on the similarity of the observation to the category prototype. Murphy [298, Chapter 3] distinguishes two interpretations of this general idea: On the one hand, one may interpret the prototype as the best example of the category. This best example can either be an actual observation or inferred from multiple observations (e.g., by defining an average). However, this interpretation may be of limited practical usefulness, for instance because it cannot encompass information about the variability of the category. A different interpretation sees the prototype as a summary representation of the whole category, describing typical and possible features of its members.

Prototypes as feature lists

The most straightforward implementation of the prototype theory represents each concept by a list of features which are weighted according to their relative importance [298, Chapter 3]. Classification can then be conducted by comparing the features of the observed object to the feature list representing the concept. This can be as rudimentary as checking whether the weighted sum of matching features minus the weighted sum of non-matching features exceeds a given threshold.

Schema-based representations of prototypes

A more advanced model is based on schemata [298, Chapter 3]: A schema is a set of slots with possible fillers. Each slot can also have restrictions on the possible fillers, both with respect to their general type and with respect to their actual values. As each slot can only be filled with a single filler, the different fillers applicable to one slot compete with each other. This prevents concrete examples from containing conflicting information such as having both features `FLIES` and `DOES NOT FLY`. The different slots can furthermore be connected to each other by constraining each others values, thus encoding correlations. If a concept is represented as a schema, then its slots define the relevant features (which can again be weighted based on relevance), and the possible fillers can be weighted based on their frequency.

Relation to artificial intelligence

This idea of schema-based representations is quite similar to Minsky's frames [294], a knowledge representation framework from artificial intelligence that has also been adopted in cognitive science [32]. These frames have a potentially recursive attribute-value structure, which can be augmented by structural invariants and constraints [393]. Also ontologies in the area of the semantic web [171] often make use of similar representational structures when defining features of a concept.

In the field of machine learning, a naive Bayes classifier [295, Chapter 6] can be seen as a probabilistic variant of the "weighted feature list" approach: Each class is represented by its list of possible features and their respective probabilities, as estimated based on frequencies in a given dataset.

In contrast to both the classical and the prototype view, the *exemplar theory* of concepts [287] rejects the idea that concepts are represented by using summary representation of the whole category. Instead, the exemplar theory argues that each concept is represented as the set of all observations of category members that have been made so far. For instance, the DOG concept is represented by the set of all encounters with actual dogs. A clear argument for this exemplar view is that in the beginning of learning a concept, i.e., when the first example is observed, there is not enough information for forming an abstraction, so one has to memorize this example itself. The exemplar theory, however, goes further by postulating that forming an abstraction is not necessary at all, even after many examples have been observed.

Concept membership in the exemplar view is based on the similarity of the observation to the stored exemplars of the concept. It can be calculated as follows [298, Chapter 3]: The similarity of an observation to a given exemplar is based on their similarity with respect to individual features. For each feature, a matching score is defined, where a value of one indicates a perfect match and a value of zero represents the greatest possible mismatch. Based on the importance of a feature, this score can be raised or lowered – for instance, a large difference on an unimportant feature can still result in a similarity score close to one. These scores of individual features are then aggregated through multiplication in order to obtain the overall similarity between the given observation and the given exemplar. By summing over all exemplars of the given concept, one can obtain an overall similarity measure. In this way of computing observation-concept similarity, it is better to have a high overlap with few exemplars than moderate overlap with many exemplars. The overall similarity score can then be compared to a given threshold in order to determine whether the given observation belongs to the given concept.

The exemplar view can be linked to so-called *lazy learning* algorithms in the field of machine learning [295, Chapter 8]: For instance, in a k nearest neighbor classifier (to be introduced in Chapter 5)) each class is represented by a set of examples. A newly observed observation is then classified by finding the k most similar examples seen so far and by choosing the class with the highest frequency among these k examples. Although the exact mechanism of assigning class membership differs from the procedure described above, the underlying idea is quite similar.

The fourth approach to representing concepts we will consider here can be called the *knowledge view*. It emphasizes that concepts do not occur in isolation, but always stand in relations to other concepts and to our general knowledge about the world [299]. It is based on the observation that category learning is not only inductive (i.e., based

*The exemplar view
on concepts*

*Concept membership
with exemplars*

*Relation to artificial
intelligence*

*The knowledge view
on concepts*

Observation	Classical View	Prototype View	Exemplar View	Knowledge View
Typicality Effects	×	✓	✓	–
Exemplar Effects	×	×	✓	–
Correlations	×	–	✓	✓
Hierarchies	×	✓	×	–
Basic Level	–	–	×	–
Induction	×	✓	×	✓
Concept Combination	×	✓	×	✓
Knowledge Effects	–	–	–	✓

Table 1.1: Overview of empirical observations about concept learning and concept usage, and their relation with different psychological theories ("✓" means "predicted", "×" means "conflicting", and "–" means "neutral").

on observations) but often influenced by prior knowledge (e.g., for determining which features are relevant). The knowledge view focuses mainly on knowledge-based learning while largely ignoring empirical learning. Individual concepts are often interpreted as mental "micro-theories" about specific aspects of the world [298, Chapter 3]. These micro-theories are often incomplete and only partially integrated, but they provide explanations and relations to other micro-theories. For example, under the knowledge view the concept `DOG` represents the role dogs play in our broader theories about biology, family life, and hunting. Moreover, the `DOG` concept contains micro-theories about the anatomy of dogs (e.g., using the tongue to regulate body temperature) and their behavior (e.g., circumstances under which dogs bark).

The knowledge view is again related to ontologies [171] from the semantic web area, which provide a formal way of describing networks of concepts. Moreover, in artificial intelligence it can be related to research on commonsense reasoning [113], which tries to overcome the limitation of purely logic-based symbolic systems by incorporating general world knowledge in order to draw plausible conclusions even if they are not justified by logical deduction.

Relations to artificial intelligence

1.1.2 Empirical Observations about Concepts

Overview

In this section, we will summarize several empirically observable effects regarding the learning and use of concepts by humans. For each of these effects, we will also argue to which extent it is compatible with the

different views presented in Section 1.1.1. Table 1.1 shows an overview of this analysis. As one can see, the classical view is not able to explain any of these effects while the three other approaches have different strengths and weaknesses. Let us now take a look at the individual effects in more detail. Again, we base our summary mainly on Murphy's comprehensive overview [298].

One of the strongest and most reliable effects in the categorization literature is the so-called *typicality effect* [298, Chapter 2]. Roughly speaking, the typicality effect is the observation that the members of a category differ with respect to their typicality – there are category members which are more typical than others. For example a ROBIN is a very typical BIRD, but a PENGUIN is a very atypical one. The typicality effect is also closely related but not identical to the observation that there are borderline cases for which no clear membership decision can be made. For instance, it is debatable whether TOMATO is a FRUIT or a VEGETABLE. Such borderline cases often are atypical members of one category which have some features in common with other categories. A TOMATO is technically a FRUIT, albeit a very atypical one which shares many features with members of the VEGETABLE category. In general, a membership judgment for such borderline cases is slower than for typical members or typical non-members of a category. The typicality effect influences also many other behaviors and judgments which are based on relating an observation to a category. For instance, typical category members are produced more frequently and earlier than atypical members when giving examples for category members. They are also learned faster and are more useful for making inferences about other category members.

The classical view cannot explain such typicality effects since all category members are assumed to be equally good examples of the category. The prototype view on the other hand was explicitly developed in order to explain these typicality effects – the more similar an observation is to the category prototype, the more typical it is considered to be. However, also the exemplar view predicts typicality effects: A typical member is expected to have a high similarity to most or all exemplars of the category, while an atypical member is only similar to a very small number of exemplars. Finally, the knowledge view does not explicitly predict typicality effects, but it also does not have any conflicting assumptions.

With respect to both category learning and categorization, one can also observe *exemplar effects* [298, Chapter 4]: While learning a category, the order in which examples are observed can influence the performance when making classifications. For example, if three consecutive observations share a certain subset of features, test items which also have this subset of features are classified quite fast. If the same three observations during learning do, however, not appear right after one another, the same test item is classified considerably slower. Moreover, superficial similarities between examples from the learning phase can

*Typicality effects:
some category
members are more
typical than others*

*Typicality effects and
theories of concepts*

*Exemplar effects:
individual examples
matter*

influence which other commonalities among them are enforced when extracting a general category description. For example, when learning about groups of people, Ross et al. [337] presented participants of their study with two exemplar members: One was described as liking ice cream and having bought nails, while the other one was described as liking Westerns and having bought a swim suit. The third exemplar was always described to have bought both wood and a towel. If it was furthermore described as liking sherbet, participants were reminded of the first exemplar and generalized that all members of this group like carpentry. However, if the third exemplar contained information about liking cowboys, the overall group was interpreted as liking swimming. Ross et al. noted that both generalizations were equally justified by the three exemplars in both conditions and that therefore the third exemplar's value of the otherwise irrelevant feature has determined the resulting generalization.

Exemplar effects in classification

Also when classifying novel observations, one can observe effects based on individual examples from the learning phase. For instance, observations which are superficially similar to a category member seen during training tend to be classified more accurately than observations without such a matching exemplar. Moreover, even when an explicit categorization rule is given, superficial similarity to an exemplar from another category increases the risk of misclassifications. Finally, if a known exemplar is categorized right after an observation that is similar to this exemplar, classification takes place faster than if the same exemplar was preceded by an unrelated observation.

Exemplar effects and theories of concepts

Exemplar models naturally incorporate exemplar effects during classification. However, they have difficulties explaining the exemplar effects observed during learning, since these effects seem to relate to a generalization of the exemplars. Such a generalization beyond exemplars is, however, explicitly refuted by the exemplar view. The prototype theory on the other hand struggles to explain the categorization effects discussed above as it assumes classification to be based solely on the similarity to the category prototype. The exemplar effects with respect to learning do not directly conflict with the prototype view, although prototype models typically do not make any concrete claims about the process of inferring a prototype from examples. Finally, the knowledge view has little to say about exemplar effects, although some of the generalizations discussed above seem to involve at least some degree of background knowledge. The classical view identifies concepts with rule-based definitions and can therefore not explain that exemplar effects are present even if such a classification rule is explicitly given. Finally, one should note that exemplar effects are strongest for artificial categories with a poor internal structure. It is thus unclear how relevant they are for natural categories with a richer structure.

Correlations: feature values don't vary independently from each other

Another important aspect of concepts are *correlations* among features [298, Chapter 5]: Concepts typically contain clusters of correlated features which play an important role both for learning [66] and reasoning [298, Chapter 8]. Murphy [298, Chapter 5] argues that such

correlations are always implicitly encoded if we define concepts based on a set of features: Typically, the features associated with a particular concept are not associated with other concepts. Thus, if we observe some of the features considered typical for the BIRD concept (e.g., HAS FEATHERS and HAS WINGS), we expect that also other features associated with the BIRD concept are present (e.g., FLIES and HAS BEAK). Moreover, certain subsets of features within a have have even stronger correlations with one another [395]. For instance, within the BIRD concept, the features SINGS and SMALL are strongly correlated with each other, even though both may not be very important to the overall concept.

Murphy, however, notes that correlations usually do not influence typicality judgments or classification performance. Moreover, humans tend to consciously notice such correlations during learning only if they are encouraged to do so through a secondary task such as predicting missing features based on a list of given ones. Murphy explains this by the assumption that the main task during category learning is to associate features with categories, not features with other features, which would be more demanding due to the much larger number of possible combinations. The correlations between features are then learned not directly, but indirectly through their common association with the same category. In a series of simulations and experiments, Billman et al. [65, 66, 220] have highlighted that individual, unrelated correlations (e.g., having four features A, B, C, and D, with high correlations between the feature pairs A–B and C–D, but no other correlations) are less helpful in learning than systems of such correlations (e.g., three features A, B, and C having high pairwise correlations in all combinations). In natural categories, one can expect that features come in such clusters of correlations.

Correlations also play an important role in concept combinations: For instance, spoons in general are considered to be small, but wooden spoons are expected to be large [288]. In this case, a correlation between material and size is used to make additional inferences.

The exemplar view predicts correlations among features, since they are implicitly encoded in the set of exemplars: Most BIRD exemplars that have the feature SINGS also have the feature SMALL, hence one can state that small birds tend to sing and vice versa. Please note that in the exemplar view, this conclusion is not drawn for all birds in general. Instead, whenever making an individual observation (e.g., a small bird), only the exemplars relevant to this observation are retrieved (i.e., BIRD exemplars which have the feature SMALL) and other features they have in common (e.g., SINGS) are used to make predictions for this individual observation. The prototype view is able to explain the overall correlation of features with categories (e.g., birds having features HAS WINGS and FLIES), but it does not necessarily predict strongly correlated, but relatively infrequent features within a concept (e.g., small birds tending to sing). However, schema-based prototype models contain mechanisms for explicitly encoding also such intra-conceptual correlations. The knowledge approach is in principle also in line with feature correlations as at least some of these correlations can be explained based

Task and context matter

Correlations and concept combinations

Correlations and theories of concepts

on an underlying theory. For example, COLOR and TASTE of a banana are correlated due to the underlying process of riping. The classical view can in principle account for general feature-category correlations if features are used as conditions in the definition of a concept. However, it struggles to accommodate correlations within a concept, since they cannot be adequately represented by a set of necessary and jointly sufficient conditions.

Conceptual hierarchies: all dogs are mammals

One can furthermore observe that concepts tend to be arranged in a *conceptual hierarchy* which can be represented by a directed "isa" relation (e.g., DOG isa MAMMAL isa ANIMAL) [298, Chapter 7]. This relation is typically assumed to be asymmetric and transitive, and to inherit features (i.e., if all mammals have blood, then also all dogs have blood). When modeling a concept hierarchy in an explicit way (similar to the ontologies mentioned above), then each concept can be represented with a node and each isa relationship by a directed edge between two nodes. The resulting graph is then expected to represent a taxonomy in the form of a tree (i.e., any category can have only one immediate superordinate). Based on the principle of cognitive economy, common features can then be stored at the highest abstraction level possible (e.g., having blood would not be associated with the DOG concept, but with the MAMMAL OR ANIMAL concept).

Violations of transitivity

However, there is experimental evidence indicating that transitivity is often violated: For instance, people agree that a CAR SEAT IS A CHAIR and that a CHAIR IS FURNITURE, but they insist that a CAR SET IS NOT FURNITURE. This indicates that conceptual hierarchies may (at least sometimes) be computed on the fly by comparing the relevant features among concepts. For example, by observing that DOG has all important features of MAMMAL, we can conclude that DOG is a special kind of MAMMAL. Murphy [298, Chapter 10] furthermore notes that while children's word usage is taxonomic (associating DOG with CAT), their sorting behavior is often thematic (associating DOG with BONE). The same can be observed to some degree in adults, highlighting that not only hierarchical relations but also thematic relations are relevant when using concepts in cognitive tasks.

The basic level of categorization: a default level of abstraction

Associated strongly with the conceptual hierarchy is the *basic level of categorization*, which is for instance often used when naming objects (e.g., describing something as a "picture of a dog" instead of a "picture of an animal" or a "picture of a Labrador") [298, Chapter 7]. Moreover, humans are also faster in verifying claims about basic level concepts (e.g., "birds have feathers") than about superordinate or subordinate concepts (e.g., "robins have feathers"). The basic level seems to be a compromise between informativeness and distinctiveness. Concepts at the basic level therefore are both easy to distinguish from each other and informative enough for making practically useful inferences. In general, subordinate concepts refine the basic level concept by adding more information while leaving the general picture unchanged. Also the intra-category similarity only increases slightly when going from the basic level to the subordinate level. Murphy [298, Chapter 7] also notes

that superordinate categories tend to be described by mass nouns (e.g., FURNITURE, FRUIT), whereas basic level concepts tend to be described by count nouns (e.g., CHAIR, APPLE), which may correspond to a distinction into collections and classes [298, Chapter 10].

The classical view is consistent with an explicit encoding of conceptual hierarchies where definitions for subordinate concepts are simply special cases of the definitions for their respective superordinate. However, it is unable to explain violations of the transitivity assumption as observed in the CAR SEAT example from above. Moreover, it does not predict any basic level effects. Also the prototype theory does not predict basic level effects, but it is consistent with them. Moreover, it can explain violations of transitivity in the conceptual hierarchy if this hierarchy is constructed dynamically by comparing prototypes: While the overlap of features between CAR SEAT and CHAIR as well as between CHAIR and FURNITURE may be sufficiently large to relate them hierarchically, the overlap between CAR SEAT and FURNITURE may be too low. The exemplar theory has problems with reflecting concept hierarchies in general: It is unclear how to make statements such as "all dogs are animals" based on exemplars unless every DOG exemplar is also annotated as being a ANIMAL exemplar. This would, however, require that each exemplar is annotated with all possible categories it may belong to, which seems infeasible or at least highly inefficient. Finally, also the knowledge view does not predict conceptual hierarchies or basic level effects, but is in principle able to accommodate them.

Conceptual hierarchies, basic level effects, and theories of concepts

Induction is the reasoning process of inferring information or knowledge about objects or categories [298, Chapter 8]. One important variant is called category-based induction which describes that by knowing to which category an object belongs, one can make predictions about unobserved features based on the general information associated with this category. For instance, by knowing that Wilbur is a DOG, we can infer that Wilbur probably has four legs and a tail, barks, and likes to play fetch. These inferences can be mainly explained based on intra-category similarity: As all members of the category are similar to each other, any category member can be expected to share their common features. Category-based induction can therefore also be related to the general correlations between features and categories as discussed above.

Induction: making educated guesses

Another aspect of induction is the transfer of knowledge between categories (e.g., the presence of a given feature). This transfer is influenced by different factors: Firstly, the typicality of the source category plays a crucial role in making generalizations. A certain disease is more likely to effect all birds if it has been observed for robins than if it has been observed for penguins. Moreover, the similarity of the source and the target category increases the likelihood of the induction: A disease infesting ducks is more likely to also affect a geese than eagles. If multiple source categories are given, their diversity can also give important information. For example, if DUCK, PENGUIN, and EAGLE all have a certain feature in common, we expect that this feature can also be observed for ROBIN. Finally, also the type of feature plays a role in

Knowledge transfer between categories

Induction and theories of concepts

making inductions: While biological features (such as the chemical composition of an animal's blood) are generalized from RABBIT to WHALE, but not from TUNA to WHALE, the opposite effect can be observed for behavioral features (such as feeding strategies).

While the classical view can explain category-based induction (since all features are given in the concept's definition), it cannot account for the effects based on similarity and typicality observed for inductions across categories. Prototype models on the other hand can account for both types of inductions. In fact, prototype-like representations are often implicitly assumed in studies about induction. Also the exemplar view can in principle also account for similarity effects. However, it has problems to account for a generalization across hierarchy levels (e.g., from ROBIN to BIRD), since conceptual hierarchies cannot be well represented with exemplars. Moreover, explicitly adding features to a category based on cross-category induction is difficult in an exemplar-based representation since all individual exemplars would need to be updated with this information. All other views which make use of a summary representation can easily incorporate such pieces of information. The knowledge approach is supported by the effects observed with respect to the feature type, where background knowledge is required in order to decide that physiological features can be generalized based on a common biological background (e.g., from RABBIT to WHALE), while behavioral features require a common habitat (e.g., TUNA and WHALE).

Concept combination: creating new concepts based on existing ones

In addition to transferring information between concepts, one can also directly combine them in order to arrive at novel concepts. This process of *concept combination* has received considerable attention in the literature (see e.g., [187, 189, 288, 379]), also since it can be related to the interpretation of natural language phrases [298, Chapters 11 and 12]. Especially conjunctive concept combinations in the form of so-called *modifier-head* phrases have been investigated. A naive model of conjunctive concept combination could use the intersection of the concept's extensions (i.e., the sets of their respective members). However, this simple intersection-based approach faces various difficulties [298, Chapter 12]: It can neither deal with relative adjectives in examples such as LARGE BABY nor model non-predicating adjectives as in ATOMIC ENGINEER. Also noun-noun combinations such as MOVIE PSYCHIATRIST cannot be modeled by simple set intersections. Moreover, while the set intersection is commutative (i.e., $A \cap B = B \cap A$), this does not hold for concept combinations, where BIRD DOG and DOG BIRD are clearly not identical. Furthermore, one can often observe conjunction effects, where for example a picture of brown apple is judged to be more typical for BROWN APPLE than for both BROWN THING or for APPLE. A related effect are *overextensions*, where for instance CHESS is not considered to be a SPORT, but it is judged to be a SPORT WHICH IS A GAME [187]. Also prior knowledge seems to play an important role in concept combinations: A TIGER SQUIRREL is typically interpreted as a SQUIRREL with stripes, hence taking only a single feature from the modifier. Moreover, a MOUNTAIN JACKET is presumably a JACKET one wears while being in the

mountains, while a MOUNTAIN MAGAZINE is expected to be a MAGAZINE *about* mountains and hiking.

As concept combination under the classical view is based on set intersection, it cannot account for the variety of effects described above. One should, however, note that there have been recent attempts to model typicality effects in concept combination (as observed in the BROWN APPLE example) with special variants of description logics [263, 333], which are implicitly based on the classical view on concepts. Again, most of the models used for explaining concept combination (such as the selective modification model by Smith et al. [379]) implicitly make use of a prototype-based representation where information about the overall concept is given in a schema-like structure and where concept combination operates on the individual slots and fillers. The exemplar theory of concepts has difficulties in modeling concept combination since it assumes that no summary representation is necessary. Modeling the effects mentioned above only based on exemplars is therefore quite challenging. Finally, almost every concept combination example from above involves some sort of background knowledge in order to decide which modifications to apply to the original concepts. This is a very strong support for the knowledge view, which can also explain the emergence of additional features (such as an EMPTY STORE making a deficit) based on causal explanations.

Concept combination and theories of concepts

Let us finally consider *knowledge effects* as a topic cutting across all the above-mentioned empirical observations. As we have seen above, prior knowledge influences the selection of relevant features, learning categories and applying them in categorization, making inductions, and combining concepts. Typically, everyday knowledge can be used to explain a category, but not to predict its existence [298, Chapter 6]. The knowledge employed when it comes to learning and applying conceptual knowledge is also usually quite shallow: One knows that knives are made of metal, because metal is hard. However, the reason implying that metal is hard is typically unknown and irrelevant. Naturally, knowledge effects are the strong suit of the knowledge view, while all other approaches have little to say about them.

Knowledge effects: background knowledge plays a role

As Murphy [298, Chapter 13] argues, the overall picture emerging from Table 1.1 and the discussion above indicates that the prototype theory (especially in the schema-based variant) seems to be a good starting point for future research. The knowledge approach is in some sense complementary to it, since it emphasizes the relations between concepts instead of the way individual concepts are represented. Murphy thus argues for a combination of these two approaches. However, he notes that also the exemplar view should not be neglected since the learning of prototypes needs to start from individual examples. Also Lieto et al. [260] have argued for an approach integrating multiple views in order to arrive at a more complete model of conceptual knowledge.

Taking stock

1.2 THE CONCEPTUAL SPACES FRAMEWORK

Overview

In this section, we present the cognitive framework of conceptual spaces [179, 181] which proposes to represent concepts as convex regions in psychological similarity spaces. Moreover, at different points in this section, we will refer to example applications of conceptual spaces in artificial intelligence. After presenting the main aspects of the framework in Section 1.2.1, we relate it to other psychological theories of concepts in Section 1.2.2. We then introduce the two classical layers of representation along with the symbol grounding problem in Section 1.2.3, before illustrating how conceptual spaces can serve as an intermediate representation format for integrating the two classical layers in Section 1.2.4. Afterwards, we show how conceptual spaces can be related to feature spaces from machine learning in Section 1.2.5. Finally, in Section 1.2.6, we discuss how conceptual spaces can be grounded in neural processing.

1.2.1 Overview of the Framework

In the following, we summarize the main points of the conceptual spaces framework as presented by Gärdenfors [179, 181].

Quality dimensions and semantic similarity

A conceptual space is a similarity space spanned by a number of interpretable *quality dimensions* (e.g., TEMPERATURE, TIME, HUE, PITCH) which are cognitively relevant and typically assumed to be based on perception. One can measure the semantic distance between two observations with respect to each of these dimensions. By aggregating these individual distance measures, one can obtain a global notion of semantic distance. Semantic similarity is assumed to be inversely related to semantic distance. More specifically, Gärdenfors uses an exponentially decaying function of distance in order to define similarity as $Sim(x, y) = e^{-c \cdot d(x, y)}$ with a sensitivity parameter $c > 0$.

Domains contain dimensions which inherently belong together

We furthermore assume that the conceptual space is structured into so-called *domains*. Typically, each perceptual modality (e.g., COLOR, SHAPE, SIZE, TASTE, WEIGHT, and SOUND) is represented by one domain, which consists of the dimensions immediately relevant to this modality. The COLOR domain for instance can be represented by the three dimensions HUE, SATURATION, and LIGHTNESS (see Figure 1.1), while the SOUND domain is spanned by the two dimensions PITCH and LOUDNESS. Based on psychological evidence [19, 211, 367, 368], Gärdenfors defines that distance within a domain is measured with the Euclidean metric, and that the domain-wise distances are then aggregated with the Manhattan metric into a global notion of semantic distance.

Properties are convex regions

Gärdenfors defines *properties* like RED, ROUND, and SWEET as CONVEX regions within a single domain (namely, COLOR, SHAPE, and TASTE, respectively). A property thus corresponds to a set of observations from a single perceptual modality. By requiring that the regions must be

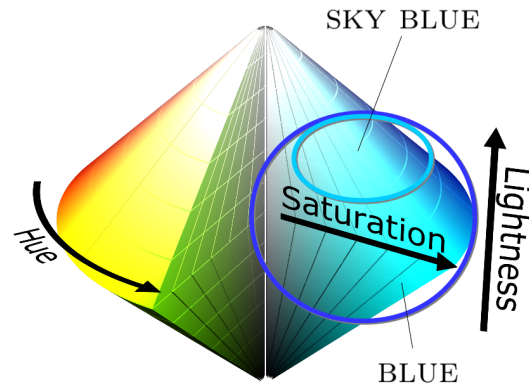


Figure 1.1: A conceptual space for the COLOR domain (based on [364], image license CC BY-SA 3.0) with dimensions HUE, SATURATION, and LIGHTNESS, and with conceptual regions for BLUE and SKY BLUE.

convex, Gärdenfors enforces a certain well-formedness: If both x and z belong to the property RED, then any linear interpolation $y = t \cdot x + (1-t) \cdot z$ (with $t \in [0, 1]$) also belongs to RED. Even though it is not necessary to restrict properties to being convex, Gärdenfors argues that the convexity assumption gives rise to many empirical predictions. Moreover it is supported by the principle of cognitive economy since convex regions are easier to store and process than arbitrarily shaped regions. For instance, the intersection of two convex regions is guaranteed to be again convex. Concept hierarchies as introduced in Section 1.1.2 are an emergent property of this spatial representation: If the SKY BLUE region is a subset of the BLUE region (cf. Figure 1.1), this implicitly encodes that SKY BLUE is a special shade of BLUE.

Based on properties, Gärdenfors now defines full-fledged *concepts* like APPLE or DOG as involving multiple domains. More specifically, a concept is represented by one convex region per domain, a set of salience weights (which represent the relevance of the given domain to the given concept) and information about cross-domain correlations. The APPLE concept may thus be represented by the RED region in the COLOR domain, the SWEET region in the TASTE domain, and the ROUND region in the SHAPE domain. Presumably, the salience weight for the SHAPE domain would be highest, since shape information is crucial in determining membership to the APPLE concept. Please note that these salience weights can also be used to represent the current context. For example, when eating an apple, its TASTE is much more prominent than when throwing it – in the latter case, SIZE and WEIGHT are more relevant.

As Murphy [298, Chapter 11] has argued, word classes can be grounded in different types of concepts: Nouns, verbs, and adjectives, arise from concepts for objects, events, and properties, respectively. This is also captured in the conceptual spaces framework: Generally, adjectives like RED, ROUND, and SWEET are represented by properties in their respective domains, while nouns such as APPLE and DOG are represented by concepts. Events are defined as composite structures consisting of agent, patient, force vector, and result vector [181, Chapters 8, 9, and

Concepts involve multiple domains

Word classes in conceptual spaces

10]. Here, agent and patient are concepts, the force vector describes a pattern of forces in the FORCE domain, and the result vector describes a change of state in the patient's properties. Verbs like PUSH and MOVE then refer to the force vector and the result vector of such an event, respectively. Finally, prepositions such as ABOVE and TOWARDS can be interpreted as regions and directions in physical space [181, Chapter 11].

Design criteria for conceptual regions

Even under the constraint of convexity, there are still infinitely many ways to partition a given similarity space into different conceptual regions. Douven and Gärdenfors [137] have listed several design criteria for a good conceptual system. More specifically, they consider parsimony (being memory efficient), informativeness (good coverage of the underlying domain), representation (availability of representative prototypes), contrast (difference between concepts), and learnability (being learnable from a small number of examples). Douven and Gärdenfors argue that convexity by itself covers parsimony and representation, and to some extent also learnability. However, in order to also cover informativeness and contrast, a second principle called *well-formedness* needs to be introduced, which essentially requires a high similarity of observations belonging to the same concept and a low similarity between observations belonging to different concepts. Douven and Gärdenfors illustrate using the color space that the constraints of convexity and well-formedness can together with asymmetries in the perceptual similarity space determine an optimal partition of this similarity space with respect to their design criteria.

Two types of learning processes

Gärdenfors [182] identifies two different types of learning processes in his framework. He argues that the first learning process consists of constructing the dimensions and domains of the conceptual space, whereas the second learning process focuses on finding meaningful regions in this space.

Learning domains and dimensions

Gärdenfors [182, 183] argues that the domains of a conceptual space are based on invariants of the sensory input that help to drastically reduce its dimensionality. The process of learning these domains involves a transformation from rapidly changing raw sensations to more stable, invariant representations of the environment. For instance, an egocentric representation of physical space is invariant with respect to the position of one's own head. An allocentric representation of physical space is in addition to that independent of one's own location in physical space. Gärdenfors argues that these invariants are the basis for identifying and learning domains. We would like to emphasize that the separation of domains seems to be at least partially based on learning and not completely innate. This is for example indicated by the observation that young infants have difficulties in separating domains such as SHAPE and COLOR from each other [298, Chapter 9].

Learning conceptual regions

With respect to the process of learning conceptual regions, Gärdenfors argues that perceptual information is not random but comes in clusters: There are certain combinations of properties which tend to co-occur in objects (cf. the discussion on correlations in Section 1.1.2). Therefore,

objects tend to form clusters in the conceptual space. By generalizing these sets of points to regions, one can learn the concepts describing the underlying categories. In his book [179] Gärdenfors has illustrated this learning process with a supervised algorithm, but later [182] he has also argued for unsupervised methods like k-means clustering (both to be described in more detail in Chapter 5).

1.2.2 Relation to Psychology

Gärdenfors argues that the convexity requirement allows us to relate the conceptual spaces framework to the prototype theory of concepts. Recall from Section 1.1.1 that the prototype view states that concepts can be described by a prototype and that concept membership is based on similarity to this prototype. This can explain why some members of a category are deemed to be more typical than others. Gärdenfors [179, Section 3.8] now argues that if concepts are represented by convex regions, one can assign a degree of centrality to each of the points in this region by measuring its distance from the center of the region. Thus, a prototype (in the "best example" sense) can be obtained by computing the center of gravity for the conceptual region. Conversely, Gärdenfors [179, Section 3.9] shows that by assuming a prototype-based representation, one can easily generate convex region. For instance, if color properties such as RED and ORANGE are represented by their prototypical points in COLOR space (e.g., their corresponding focal colors), one can partition the overall space into convex regions by assigning each point in the space to its closest prototype. This way of partitioning a space is called a *Voronoi tessellation* and is used by Gärdenfors for most of his arguments and illustrations. Using this prototype-based interpretation of conceptual spaces, one can easily model concept learning by defining the prototype as an average across all examples seen for the corresponding concept.

Recently, Lewis and Lawry [253] have argued that such a Voronoi tessellation of the space has, however, the disadvantage that each individual point in the conceptual space has to be assigned to exactly one prototype and hence to exactly one concept. If instead points are assigned to a concept if their distance to this concept's prototype is below a given threshold, one can also model cases where an observation can be described by multiple concepts at once or by none at all.

Strößner [393] has furthermore highlighted the similarities and differences between conceptual spaces and frames: Both approaches analyze concepts in terms of attributes and values. Frames use a symbolic representation of these values, which can be quite restrictive in comparison to the quantitative information represented in conceptual spaces. However, frames allow for complex and recursive structures, while the structure of conceptual spaces is limited to domains and dimensions. Strößner emphasizes that combining frames and conceptual spaces may create many synergies. For instance, one can use conceptual spaces for modeling individual properties, and then define a frame-like structure

Conceptual spaces as a spatial variant of prototype theory

Limitations of Voronoi tessellations

Conceptual spaces and frames

Observation	Classical View	Prototype View	Exemplar View	Knowledge View	Conceptual Spaces
Typicality Effects	×	✓	✓	–	✓
Exemplar Effects	×	×	✓	–	–
Correlations	×	–	✓	✓	✓
Hierarchies	×	✓	×	–	✓
Basic Level	–	–	×	–	–
Induction	×	✓	×	✓	✓
Concept Combination	×	✓	×	✓	✓
Knowledge Effects	–	–	–	✓	–

Table 1.2: Overview of empirical observations about concept learning and concept usage, and their relation with different psychological theories, including the conceptual spaces framework ("✓" means "predicted", "×" means "conflicting", and "–" means "neutral").

on top of this in order to represent fully fleshed concepts which may also include recursive part-whole structures. Moreover, conceptual spaces may provide a useful way for modeling prototype effects within a frame-based approach.

Conceptual spaces as a theory of concepts

Let us now discuss to which extent conceptual spaces are capable of modeling the empirically observed effects from the psychological literature as discussed in Section 1.1.2. Table 1.2 shows an overview of the different effects, the psychological models from Section 1.1.1, and the conceptual spaces framework.

Typicality effects

As argued above, conceptual spaces can be seen as a geometric variant of prototype models. They are thus able to explain *typicality effects* based on distances in the similarity space.

Exemplar effects

While *exemplar effects* are not easily explained by Gärdenfors' account, one can imagine to extend his framework by adding a small number of exemplars to the representation of both properties and concepts. A concrete example for such an extension will be discussed below.

Correlations

Correlations between domains are explicitly included as an important component of the definition of a concept. Although Gärdenfors does not specify how exactly these correlations should be represented, some extensions of his work (e.g., [329]) have proposed concrete mechanisms for representing these correlations.

Conceptual hierarchies and basic level

As noted above, *conceptual hierarchies* are implicitly represented in conceptual spaces through the subsethood relation of the respective conceptual regions. The *basic level* of concepts can be modeled by translating informativeness and distinctiveness into spatial terms, namely, a low intra-class distance and a high inter-class distance, respectively. Essentially, the work by Douven and Gärdenfors [137] discussed in Section 1.2.1 tries to capture this with their proposed well-formedness

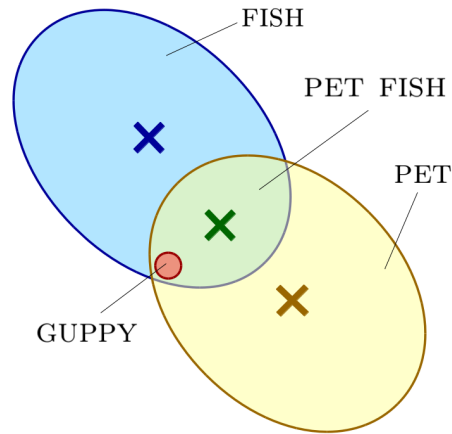


Figure 1.2: Simple spatial representation of the pet fish example.

principle. The basic level then corresponds to a set of conceptual regions which optimize the trade-off between being small and being separated from each other by a clear margin. As we will see in Chapter 7, the underlying principle can be related to clustering techniques from machine learning.

All *induction* effects based on typicality and similarity can also be explained with conceptual spaces by resorting to the distances in the underlying similarity spaces. Recently, Osta-Vélez and Gärdenfors [311] have demonstrated how this can be achieved with a relatively simple mathematical model, which is able to account for many empirically observed effects.

Induction

The conceptual spaces framework also provides a geometric grounding for many *concept combinations*. Lieto et al. [260] use the pet fish example by Osherson and Smith [309] to illustrate how concept combinations in the conceptual spaces framework are capable of preserving typicality structures: A *GUPPY* is neither a typical *PET* nor a typical *FISH*, but it is considered to be a very typical *PET FISH*. In a conceptual space, one can define typicality based on the distance to the center of the respective region. Moreover, in a simplified version of the framework, we can define the region describing *PET FISH* as the intersection of the regions representing *PET* and *FISH* (see Figure 1.2). As we can see, *GUPPY* is a subset of both *FISH* and *PET*, but it lies quite far away from the center of both regions (marked by crosses). However, it is located quite close to the center of the *PET FISH* region. Thus, the geometric representation of conceptual knowledge provides an intuitive way of explaining the pet fish example. We will take a closer look at concept combination in conceptual spaces in Chapter 3 in the context of our mathematical formalization of the framework.

Concept combination in conceptual spaces

Knowledge effects are not considered in conceptual spaces. One can, however, imagine that background knowledge can be incorporated in the form of external constraints on the way in which conceptual regions are formed and combined. One way of incorporating such external constraints is offered by the salience weights which define the importance of the individual domains in the current context. Knowledge

Knowledge effects

effects will resurface in Chapters 4 and 7 in the context of defining concept similarity and learning conceptual regions, respectively.

Taking stock

Overall, one can see that conceptual spaces are able to accommodate almost all of the effects described in Section 1.1.2. This is mainly caused by conceptual spaces being a geometric variant of the prototype theory.

The Dual-PECCS system

Lieto et al. [258, 264, 265, 266] have used the conceptual spaces framework in order to build a computational model of conceptual categorization which unifies prototype theory, exemplar approaches, and the classical view of concepts. They made use of the distinction of cognitive processes into two types (cf. [217]): System 1 processes are unconscious, parallel, fast, and based on associations, while system 2 processes are conscious, sequential, slow, and based on explicit rule following. Lieto et al. mapped both exemplar and prototype theories to commonsense reasoning strategies (which are a system 1 process) and the classical view of concepts to deductive reasoning (i.e., a system 2 process). In their Dual-PECCS system, they used a hybrid knowledge base which employed conceptual spaces for representing information about prototypes and exemplars and which used the OpenCyc ontology¹ [281] to represent classical definitional information about concepts.

The conceptual space of Dual-PECCS

Their example implementation focuses on finding the correct animal category (e.g., CAT) for short linguistic descriptions such as "the mice hunter with whiskers and long tail". In order to prepare their knowledge base accordingly, they extracted information about nine domains (namely, SIZE, SHAPE, COLOR, LOCATION, FEEDING, LOCOMOTION, HASPART, PARTOF, and MANRELATIONSHIP [262]) from ConceptNet [267]. Essentially, they scanned ConceptNet for relevant associations which were then mapped onto the individual dimensions of the conceptual space, using in some cases a translation dictionary to map symbolic terms such as RED onto a numeric representation such as coordinates in the color space [266]. The points obtained for basic level categories such as CAT were interpreted as prototypes for the respective concept, while points obtained for subordinate categories such as CANADIAN SPHYNX were interpreted as exemplars for their associated basic level category.

Classification procedure

In order to find the correct category for a given linguistic description, Lieto et al. first extracted the relevant pieces of information from the query using standard tools for natural language processing. The extracted information was then fed into their system 1 process which operates on the prototypes and exemplars in the conceptual space. If an exemplar was close enough to the given query point (taking only into account dimensions for which there is information on the query point), the category associated to this exemplar was used as system 1 response. If no matching exemplar was found, the system picked the closest match among all prototypes and exemplars as system 1 response. This preference for exemplars over prototypes had been motivated by findings reported in the psychological literature [287]. The output of

¹ OpenCyc was a publicly available subset of the commercial Cyc ontology (see <https://cyc.com/>), which has been discontinued in 2017.

system 1 was then validated by a system 2 process, which essentially checked whether the query and the category identified by system 1 matched the definition from the ontology. One can thus say that prototypes and exemplars were used to generate candidate responses while the ontology was used to filter them.

Lieto et al. evaluated their system on a dataset of 112 linguistic descriptions [265]. In a preliminary study with human subjects, Lieto et al. established that 56 of these descriptions were interpreted as referring to a prototype while the categorization of the remaining 56 descriptions was based on an exemplar. Lieto et al. were able to show that their system identified the correct category more frequently than standard search engines such as Google, Bing, and Wolfram Alpha when considering their first ten responses [266]. In most cases, Dual-PECCS used the expected representation (prototype or exemplar) for making its response. The most confusions with respect to the representation type arose when a prototype-based response was expected, but the system used an exemplar to come to its conclusion. This can, however, easily be explained by the preference given to exemplars when determining the system 1 response. Since the implementation of this preference in their cognitive model resulted in outputs that were not in line with human responses, this observation can be seen as a contradiction to the psychological data that motivated this design choice. It therefore urges for more psychological research on this topic.

The Dual-PECCS system has successfully been integrated into four cognitive architectures [265, 266], namely, ACT-R [13], CLARION [398], SOAR [239], and LIDA [160]. Each of these cognitive architectures has its own set of underlying assumptions about human cognition. For example, the SOAR architecture assumes that cognition can be explained mainly by symbol manipulation while CLARION emphasizes the tight integration of symbolic and connectionist approaches in each of its modules. Since Lieto et al. were able to integrate their representation and reasoning mechanisms in all of these architectures despite their different underlying assumptions, they argue that their approach can be used as a common ground for knowledge representation and reasoning in cognitive architectures in general (cf. also [260]). Finally, there was a recent proposal for replacing the ontology used in Dual-PECCS with a theory-based approach [258]. The resulting updated system then computes a degree of consistency of the system 1 response with its background knowledge in the form of theories instead of making a binary decision based on necessary and sufficient criteria. Thus, in this extension of the Dual-PECCS system, the classical view is discarded in favor of the knowledge view.

The work by Lieto et al. makes an important contribution to concept research: They propose a concrete way of combining different views on concepts, considering both their representational mechanisms and their classification procedures. Moreover, their system provides an actual implementation of this proposal which makes it easy to validate theoretical claims by practical experiments. Finally, their system shows the value of conceptual spaces as a modeling tool for cognitive AI.

Evaluation results

*Integration into
cognitive
architectures*

Relevance

1.2.3 The Classical Layers of Representation

*A general dichotomy
of models*

Generally speaking, models of human cognition can be distinguished into two broad classes, namely symbol systems and connectionist systems [190]. In the field of artificial intelligence, this is reflected by the dichotomy of knowledge representation into the symbolic and the subsymbolic layer. Gärdenfors [179, Chapter 2] notes that in biological systems, the distinction into symbolic and subsymbolic representations corresponds to different ways of describing the same object, based on different levels of granularity. In artificial systems, however, they usually map to separate modules or layers. In the following, we will describe these two classical layers in more detail.

Subsymbolic models

The *subsymbolic approach* (often also called *connectionism*) views cognition as dynamic patterns of activity in a complex network of individual nodes [190]. This definition originally also encompassed graph-based representations where events and concepts are represented as nodes and activation is spread among their connections [179, Chapter 2]. In the recent past, it has, however, mostly been equated with artificial neural networks which are a mathematical model for learning associations between a given set of inputs and outputs [179, Chapter 2]. The input to these neural networks is usually based directly on sensory input [260] and the computations involved are mostly concerned with pattern recognition [179, Chapter 7].

*Strengths and
weaknesses of
subsymbolic models*

Connectionist models are capable of learning patterns by applying a relatively small class of algorithms to a wide variety of tasks [190]. They are therefore quite flexible and relatively robust to noise [179, Chapter 2]. Depending on the actual model, they can also be mapped to the structure and activity of human brains (see e.g., [106, 443]). However, connectionist models are in general incapable of modeling systematic symbolic properties such as compositionality [260] which seem to be important for many cognitive tasks [190]. As not all cognitive operations reduce to pattern learning, connectionism can therefore not account for all cognitive phenomena. Moreover, artificial neural networks tend to require large amounts of training data [179, Chapter 2] and typically result in black box solutions, which are hard to interpret [260].

Symbolic models

The *symbolic approach* interprets the human mind as a symbol system and the process of cognition as symbol manipulation [190]. A symbol system is based on a set of arbitrary atomic tokens (called symbols) which can be combined with each other based on certain rules in order to construct composite symbol strings. These rules are exclusively based on the syntactical structure of these symbol strings. All of these entities (i.e., atomic symbols, composite symbol strings, and rules) are semantically interpretable in the sense that one can systematically assign a meaning to them based on the underlying syntax. More generally speaking, symbolic representations tend to be based on logical formalisms [260] and are typically processed through recursion, the application of rules, tree traversal, and search algorithms [179, Chapter 7].

Symbolic models have the full computational power of Turing machines [190] and are well-suited for complex reasoning tasks [260], e.g., the computation of logical consequences [179, Chapter 2]. Moreover, they are able to model compositionality [260], which is for example required when processing language. Symbolic AI systems such as SOAR [239], but also ontologies from the semantic web [171] can be seen as practical implementations of this approach [260]. Overall, the symbolic approach has strong ties to the classical view on concepts due to their common dependence on formal logics.

Strengths of symbolic models

However, symbolic models also come with severe limitations. For instance, they are often ad-hoc in the sense that the rules and symbols need to be handcrafted for the specific task [190]. Furthermore, they cannot account for typicality effects (such as ROBIN being a more typical example of the BIRD category than PENGUIN) [260]. Moreover, the individual symbols are mostly specified a priori and cannot be changed [179, Chapter 2], which limits their flexibility. Finally, the symbols used in symbolic models do not have any connection to the real world and are thus subject to the symbol grounding problem [190].

Weaknesses of symbolic models

Inspired by Searle's Chinese room argument [356], Harnad [190] describes the *symbol grounding problem* as the task of learning Chinese by only using a Chinese-Chinese dictionary. This dictionary contains a definition for each Chinese word in terms of other Chinese words. However, it is arguably impossible to learn the underlying meaning of the Chinese words only based on this dictionary. Harnad argues that purely symbolic systems face a similar problem since their symbols are only defined in terms of each other without a reference to the external world. The symbol grounding problem can thus be formulated as the following question:

The symbol grounding problem

How is symbol meaning to be grounded in something other than just more meaningless symbols? [190, Section 2.2]

One may note that most symbolic systems can still be interpreted by humans in practice. However, this is usually only possible because the symbols have been chosen in such a way the human interpreter can easily connect them to their own perceptual experience. If the symbols in the system are for example named RED, SWEET, and APPLE, then an interpretation of this system inadvertently triggers the sensory experiences associated with this specific color, taste, and object, respectively. This enables humans to interpret rules such as $APPLE(x) \wedge RED(x) \Rightarrow SWEET(x)$ and to judge their meaningfulness. However, if the symbols in such a system were consistently replaced with meaningless tokens (such that for example RED would be replaced everywhere in the system with 42XH8QW), then the corresponding rule $42XH8QW(x) \wedge 4C89(x) \Rightarrow 29F4QUZ(x)$ would be completely opaque to a human interpreter. Even with access to all rules in the system, one would not be able to understand the meaning of atomic symbols such as 42XH8QW, let alone complex symbol strings.

Interpretability of symbolic models depends on the symbol names

Continued relevance Although there have been several concrete proposals for solving the symbol grounding problem, it still remains highly relevant today and cannot yet be considered to be solved in its entirety (see e.g., [85, 109, 249, 404]).

Towards a hybrid approach As one can see from this brief discussion, both classical approaches have complementary advantages and disadvantages. Both from the perspective of cognitive science in general (which aims for a holistic understanding of the human mind) and of artificial intelligence in specific (where one aims for good and robust performance in a variety of tasks), it is thus desirable to combine of both approaches with the aim to conserve their strengths and to eliminate their weaknesses.

Harnad's solution to the symbol grounding problem Harnad [190] does not only note and describe the symbol grounding problem, but he also sketches a possible solution. This solution is based on the two fundamental cognitive processes of discrimination and identification. *Discrimination* corresponds to judging whether two stimuli are the same or different and to determine how much they differ from each other. Harnad argues that this essentially requires a way of judging the similarity of two stimuli. *Identification* on the other hand corresponds to assigning a unique response (i.e., a name or a symbol) to a class of inputs, treating them as equivalent in the current context. Essentially, this corresponds to classification.

Iconic and categorical representations Based on these two cognitive processes, Harnad argues for two types of representations. *Iconic representations* (or icons) are simple analog transformations of the raw sensory information. They are sufficient for determining similarity between stimuli by simply superimposing them and determining their degree of disparity. Iconic representations can thus in principle support discrimination. As the sensory apparatus of humans is fixed, iconic representations can be assumed to be mostly innate. Icons are, however, in general not sufficient for identification as there are too many of them which can blend into each other continuously. Also Murphy [298, Chapter 2] has argued that a clear separation of objects into different categories would only be possible if the world consisted "distinct clumps of objects", i.e., if no continuous blending between stimuli could occur. Therefore, Harnad [190] postulates categorical representations as a second representational type. *Categorical representations* are the output of category-specific feature detectors which compress the iconic representations in such a way that the invariant features of the respective category are emphasized while irrelevant variations are filtered out. Using these invariant features, categorical representations thus provide a way to distinguish category members from non-members. As the categories which may be relevant in a given environment cannot be anticipated by evolution, categorical representations can be assumed to be mostly learned.

Support from psychology The distinction into iconic and categorical representations can also be supported from a psychological point of view: Native speakers of a language with only two color terms (distinguishing light from dark colors) have a similar perception and memory of colors as native speakers of languages with a larger inventory of color terms [298, Chapter

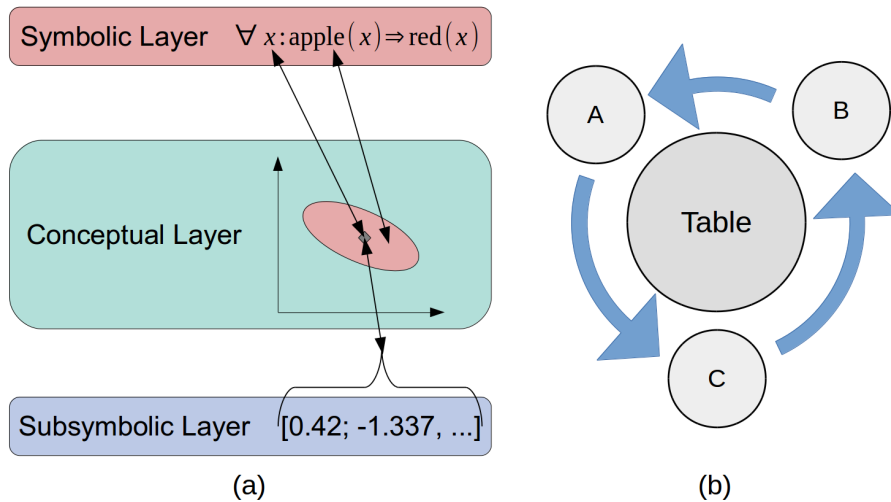


Figure 1.3: (a) Conceptual spaces as an intermediate layer for translating between symbolic and subsymbolic representations. (b) Mental image of three people sitting at a small round table and a modified TO THE RIGHT OF relation.

11]. It thus seems that the overall perception of color is shared among different cultures (i.e., same iconic representation) while different color categories have been learned (i.e., different categorical representations).

Harnad [190] now argues that the categorical representation can be used to ground atomic symbols of a symbol system such as HORSE and STRIPED. The meaning of these symbols then corresponds to the category selected by the corresponding categorical representation, i.e., to the set of stimuli which are members of the given category. Thus, atomic symbols are linked to categories of perceived objects. The symbol system can then operate on top of these atomic symbols by using rules to express more complex symbols for which no perceptual grounding is available. For instance, one could define the symbol ZEBRA as follows: $\text{ZEBRA}(x) :\Leftrightarrow \text{HORSE}(x) \wedge \text{STRIPED}(x)$. The symbol ZEBRA is then indirectly grounded in perception through the categorical representations of HORSE and STRIPED. Thus, a person who had never seen a zebra could nevertheless identify one based on this definition and the prior sensory experiences of horses and striped things.

Harnad proposes to use connectionist approaches for learning the categorical representations by pattern matching. His overall proposal thus provides a principled way of using both symbolic systems and connectionism while combining their advantages and largely eliminating their individual weaknesses.

1.2.4 Using the Conceptual Layer to Solve the Symbol Grounding Problem

The general solution to the symbol grounding problem sketched by Harnad [190] is still quite abstract, since it does for instance not specify how categorical representations look like. A more concrete proposal has

Symbol grounding with categorical representations

Summary

Conceptual spaces as an intermediate layer of representation

been made by Gärdenfors [179] who proposes to use conceptual spaces as an intermediate layer of representation, the so-called *conceptual layer* (see Figure 1.3a). Individual observations, which correspond to high-dimensional activation vectors in the subsymbolic layer, are represented by points in the lower-dimensional conceptual space and can be mapped onto constants and variables from the symbolic layer. Predicates from the symbolic layer (such as APPLE and RED) can be mapped onto concepts and properties in the conceptual layer. Each predicate is thus mapped onto one or more convex regions in the conceptual space which describe sets of observations that are similar to each other and dissimilar from other observations. One can therefore easily see that the symbols from the symbolic layer can be indirectly grounded in subsymbolic perception through the conceptual layer.

*Conceptual spaces
and Harnad's
proposal*

According to Gärdenfors [179, Chapter 2], this overall architecture can be roughly aligned with Harnad's proposal [190] by mapping the iconic representation to the subsymbolic layer and the categorical representation to the conceptual layer. We would, however, like to point out that this is only a rough alignment since the underlying mechanisms are of different nature. Harnad's iconic representation mainly targets similarity judgments between pairs of stimuli. As similarity is measured as inverse distance in Gärdenfors' framework, the iconic representation can also be mapped onto *points* in the conceptual layer. Harnad's categorical representation on the other hand focuses on concept membership and is thus closely related to *regions* in the conceptual space. Even though the two proposals do not align perfectly, they clearly have a strong relationship to each other.

*Advantages of
conceptual spaces*

It has been argued that this intermediate layer with geometric representations has several advantages over both symbolic and subsymbolic approaches [179, 181, 260]. In addition to providing a connection between the two classical layers, the conceptual spaces approach thus has its own merit in modeling and explaining certain aspects of cognition.

*Advantages over
purely connectionist
models*

When comparing the conceptual layer to the subsymbolic layer, one can note that the number of dimensions in the conceptual space is considerably lower than the dimensionality of a neural network's activation vector. When learning classification boundaries for concepts, one therefore needs to estimate a considerably smaller amount of free parameters, which can lead to faster learning and better generalization [179, Chapter 7]. Moreover, the artificial neural networks used in the subsymbolic layer often tend to be black box models which are hard to interpret by humans. In many domains (e.g., medicine), it is, however, not only desirable to make correct predictions, but also to give an explanation for this prediction. This is the focus of the growing body of research in explainable AI [113, 133, 274]. By lifting information from the subsymbolic to the conceptual layer, where we have interpretable dimensions and a meaningful domain structure, AI systems can become more interpretable. Lieto et al. [260] argue that in principle, one can interpret the individual layers of an artificial neural network as a conceptual space, where the activation of each individual

node determines the coordinate with respect to one of the coordinate axes. This way, conceptual spaces can be used as a way to analyze and interpret the inner workings of artificial neural networks such as self-organizing maps or RBF networks.

Most advantages of the conceptual layer become apparent when comparing it to the symbolic layer [181, Chapter 14]: As we have seen in Section 1.2.1, a conceptual hierarchy does not need to be explicitly coded in conceptual spaces, but it emerges based on the subsethood relation between regions. Also the identity of two concepts or objects emerges naturally based on the identity of the underlying regions or points. Moreover, also property characteristics (such as "GREEN is a COLOR" or "nothing is both completely RED and completely GREEN") emerge directly from the structure of the underlying conceptual space (namely, the region describing GREEN being defined only on the COLOR domain, and RED and GREEN being represented by disjoint regions). All three of these effects need to be explicitly modeled in symbolic systems which considerably increases the modeling effort associated with these approaches. Furthermore, Gärdenfors argues that in the conceptual layer no symbolic inference engine is needed since all relevant pieces of information are encoded geometrically and can thus be accessed by straightforward geometric computations. Another key advantage of conceptual spaces over symbolic systems is the inherent representation of semantic similarity which allows for approximate reasoning.

Lieto et al. [260] note that conceptual spaces provide also a unifying framework for many diagrammatic and analogical representations. These types of representations typically involve mental images or mental models of specific contexts and can be more intuitive than symbolic approaches especially in spatial domains. For example, the relation TO THE RIGHT OF is usually assumed to be transitive. However, one can easily imagine three people sitting at a small round table, where the TO THE RIGHT OF relation is reinterpreted based on a counterclockwise order. By visualizing the scene as a mental model (see Figure 1.3b), this violation of transitivity can be easily predicted, while formal logics require additional complex assertions to capture this case. Lieto et al. argue that the geometric nature of the conceptual spaces approach provides a way to easily embed these mental images.

Since the conceptual layer offers advantages over both the symbolic and the subsymbolic layer and is capable of integrating diagrammatic approaches, Lieto et al. [260] have argued that it should be seen as a "Lingua Franca" for cognitive architectures. As Gärdenfors [181, Chapter 14], however, points out, the main limitation of the conceptual spaces framework is that one needs to correctly identify and describe its underlying domains and dimensions.

One of the first applications of conceptual spaces in artificial intelligence was the work by Chella et al. [95, 96, 97, 98] on perceptual anchoring in robotics. Perceptual anchoring is the problem of linking symbols for individual objects to the corresponding sensory data and maintaining this connection over time [95]. As such, perceptual anchor-

Advantages over purely symbolic models

Diagrammatic and analogical models

Conceptual spaces as "Lingua Franca"

Conceptual spaces for perceptual anchoring in robotics

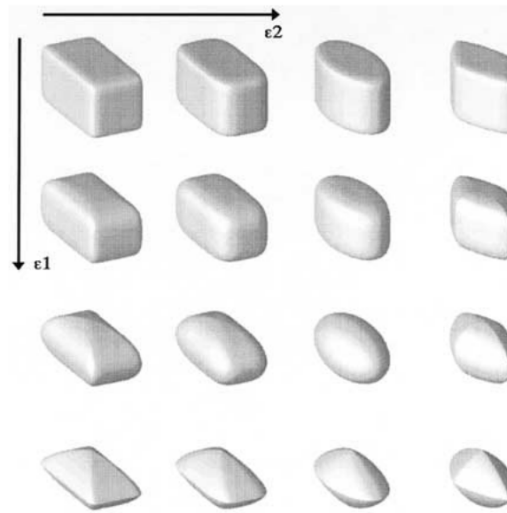


Figure 1.4: Different superquadrics based on varying form factors ϵ_1 (rows, ranging from 0.2 to 1.5) and ϵ_2 (columns, ranging from 0.2 to 1.5). Figure taken from [97], reprinted with permission.

ing can be seen as a special case of the symbol grounding problem. For example, if the symbol `CUP-22` is used by a symbolic planner in the context of preparing a breakfast table, this symbol needs to be connected to the sensory input in order to identify the cup's location and in order to generate appropriate motor commands for manipulating it. Chella et al. used handcrafted conceptual spaces as a way for translating between sensory input and symbolic descriptions. They distinguish three types of different conceptual spaces [96]:

Perceptual space:
static information

The *perceptual space* represents static information about the scene, where each object is represented by one point. Its dimensions are based on COLOR information (using the HSV COLOR space), position information in the SPATIAL domain, and SHAPE information [95]. The dimensions of the SHAPE domain are based on the parameters of so-called *superquadrics* [97], which describe three-dimensional shapes based on their width, height, and depth, as well as two form factors (see Figure 1.4). Objects with multiple parts are represented as sets of points in the perceptual space with one point per part. The perceptual space is connected to the sensory system through specialized feature extractors which extract the coordinates (e.g., the shape parameters) from the raw perceptual input.

Situation space:
dynamic information

The *situation space* captures dynamic information and is built on top of the perceptual space. In the situation space, each point represents a simple motion, i.e., a continuous ongoing movement of one simple object. Again, a complex movement involving multiple parts can be represented by a set of points in the situation space [98]. The situation space can be connected to situation calculus [283], which is a symbolic formalism for describing the state of a given scene.

Action space:
changes in movement

Finally, the *action space* represents changes in movement. Each point in the action space corresponds to a pair of points in the situation space, describing the situation immediately before and immediately after the change. For instance, if the movement slows down, accelerates,

or changes its direction, the corresponding point in the situation space moves instantaneously to a different location. This "scattering" is captured in the action space. Processes such as moving a finger can then be represented by two actions [98], one for the beginning of the movement and a second one for the end of the movement.

Chella et al. [95] have devised various anchoring functions in this setting, which includes establishing an anchor both in a top-down and a bottom-up manner (based on a symbol that needs to be grounded, and a perception that needs to be classified, respectively). Their work also covers functions for updating the anchors over time and reacquiring them if necessary (e.g., because the object was occluded in the camera image for a short amount of time). In these anchoring functions, they use a mapping of properties from the symbolic layer (where CUP-22 is for example annotated with the property RED) to a region in the conceptual space (in this case to the COLOR domain of the perceptual space) in order to constrain the set of candidate observations that can be matched to the given symbol. They do, however, not explicitly require these conceptual regions to be convex. Chella et al. have also applied their approach to imitation learning [96], where they use k-means clustering to discover prototypes for different concepts. In this case, they follow Gärdenfors' proposal of representing conceptual regions by a single prototypical point.

*Anchoring with
conceptual spaces*

The work by Chella et al. has given a first proof of concept for the usefulness of the conceptual spaces framework in artificial intelligence, especially with respect to the symbol grounding problem. We will revisit their work in Chapters 2 and 10, where we discuss their proposals for representing part-whole relations and the shape domain in more detail.

Relevance

1.2.5 Relation to Machine Learning

Classification problems in machine learning are often interpreted as the search for good classification boundaries between two or more classes in a *feature space*. In such a setting, each observation is annotated with its values for a fixed set of features and its correct class assignment. If one interprets each feature as a dimension, each observation can be interpreted as a point in a high-dimensional space spanned by these features. Any decision rule for separating the observations belonging to different classes can then be visualized as a decision boundary in this feature space. Solving the classification problem is thus equivalent to partitioning the feature space into different regions corresponding to the different classes. Some machine learning algorithms such as k nearest neighbor compute the distance between points in this feature space to make their classifications [295, Chapter 8], other algorithms such as linear support vector machines attempt to separate the classes with a hyperplane [73, 110].

*Geometric
interpretation of
classification
problems*

*Conceptual spaces as
a special kind of
feature space*

One can easily see that a conceptual space can thus be interpreted as a special kind of feature space, which is spanned by semantically meaningful features and where conceptual regions are required to be convex. The process of obtaining the dimensions of a conceptual space for a particular application can therefore be linked to feature engineering procedures in machine learning, while concept learning is just a constrained version of learning classification boundaries in a feature space. This strong link between conceptual spaces and feature spaces gives further motivation to our research which tries to make conceptual spaces usable for artificial intelligence applications.

*Conceptual spaces
and feature spaces
have different
objectives*

One should, however, note that despite their similarities, there are a number of differences between conceptual spaces and feature spaces. Chella et al. [97] have argued that the objective of feature spaces is to enable the classification of objects into a set of given categories, whereas conceptual spaces aim to give a cognitive grounding of symbolic representations on perceptual data. Therefore, the dimensions of a feature space serve the main function of allowing for good discrimination, whereas the dimensions of a conceptual space also allow for the generation of a rich symbolic description. Also Sileno et al. [370] argue that the discriminatory power of a given quality dimension with respect to a given concept should be reflected by its respective salience weight rather than being inherent in the conceptual space itself. Banaee et al. [26, 27], however, have put a stronger emphasis on the discriminative power of the quality dimensions by considering only highly discriminative features as candidates for quality dimensions.

*Conceptual spaces
and feature spaces
use different levels of
abstraction*

Chella et al. [97] furthermore observe that feature spaces are often based on low level features which are closely related to sensory input. Conceptual spaces on the other hand are based on cognitively salient dimensions which might be more abstract. Moreover, feature spaces typically treat objects as wholes without being able to represent their individual parts. Conceptual spaces by themselves are also limited to whole objects, but there have been extensions to the framework making it possible to represent part-whole relations [97, 158].

*Conceptual spaces
have a domain
structure, feature
spaces typically don't*

Another important difference in our opinion is the structure of the spaces. Feature spaces from machine learning are typically assumed to form one big, unstructured Euclidean space, where distance is computed with the Euclidean metric. Conceptual spaces, on the other hand, group their dimensions into domains. In order to compute the distance between two points in the conceptual space, one first computes their Euclidean distance within each domain before combining these intra-domain distances with the Manhattan distance (i.e., by simply summing them up). These different notions of distance may result in different notions of similarity and thus in different classification behavior.

*A first example
application*

A first example application of conceptual spaces in machine learning is given by Banaee et al. [26, 27], who consider the bottom-up process of creating linguistic descriptions of numerical data. Their work consists of two important contributions, namely the extraction of conceptual spaces from machine learning datasets, and the generation of linguistic

descriptions based on conceptual spaces. We will now give a brief high-level summary of their work, which will be discussed in more detail in later chapters of this dissertation

In order to extract a conceptual space from a machine learning dataset, Banaee et al. assume that not all features from the dataset are necessary and only a subset is needed for generating meaningful descriptions. In order to identify this subset of features, they measure the predictive power of each feature for each of the different classes. A domain is then defined as a set of features which have a high predictive power for a set of classes. Banaee et al. iteratively select such sets of features until at least one such domain has been identified for each class. Now, the domain structure of the conceptual space has been identified and all remaining features can be discarded. Banaee et al. have shown on two example datasets (leaves and time series) that the resulting domains tend to group together features with a common underlying meaning (e.g., various features describing the convexity of a leaf). The examples from the dataset are then mapped onto points in the conceptual space by removing the unused features. Banaee et al. then construct for each class a convex hull of its example data points.

Obtaining a conceptual space from a machine learning dataset

In order to extract a linguistic description of a given observation, Banaee et al. first project this observation into the conceptual space and then analyze it with respect to the different domains. In each domain, they first check whether the point is contained in any conceptual region. If this is the case, they annotate the observation with the respective class label. Otherwise, they use annotations with respect to the individual dimensions (such as `ELONGATED` referring to the upper end of the `ASPECT RATIO` feature of the leaves dataset). After having collected a list of such annotations, Banaee et al. use handcrafted templates to generate an overall linguistic description. For both of their datasets, Banaee et al. found that human subjects were successful in identifying the correct observation based on the generated description. Moreover, for most of the incorrect responses the incorrectly selected observation was quite close to the target observation in the conceptual space which provides an intuitive explanation for the observed human behavior.

Creating linguistic descriptions with conceptual spaces

The work by Banaee et al. provides bottom-up procedures for connecting the subsymbolic to the conceptual layer through the creation of conceptual domains and for connecting the conceptual layer to the symbolic layer through the generation of linguistic descriptions. Moreover, it highlights the strong relation between conceptual spaces and machine learning. We will revisit their work in Chapters 3 and 6, in the context of concept combination and machine learning, respectively.

Outlook

Another important research contribution in the context of machine learning has been made by Derrac and Schockaert [122, 123], who use the conceptual spaces framework as a way to implement commonsense reasoning strategies. *Commonsense reasoning* is here seen in contrast to classical deductive reasoning, which often fails in contexts where only insufficient knowledge is available. Commonsense reasoning aims to draw plausible conclusions which may be unsound from a strictly

Commonsense reasoning with conceptual spaces

logical perspective, but which nevertheless are useful in practice. In their work, Derrac and Schockaert extract conceptual spaces from text corpora, identify interpretable directions in these spaces, define several commonsense reasoning strategies based on geometric relationships, and empirically evaluate classifiers based on these strategies.

*Extracting
conceptual spaces
from textual data*

Derrac and Schockaert base their conceptual spaces on textual data, which is generally available in large quantities. They consider three domains, namely, MOVIE, PLACE TYPE, and WINE. For a set of candidate entities in each domain (i.e., individual movies, place types, and wine variants), they collected bag of words representations (i.e., a list of all words associated with the respective entity) based on review texts (for movies and wines) and image tags (for place types). These bag of words representations are then transformed into coordinates in a similarity space, using among others the technique of multidimensional scaling, which will be described in more detail in Chapter 8. Essentially, multidimensional scaling represents each entity as a point in a low-dimensional space and arranges these points in such a way that their Euclidean distances accurately reflect the pairwise dissimilarities (which are in this case based on the bag of words representation). As a result, Derrac and Schockaert obtain one conceptual space for each of the three domains under consideration.

*Identifying
interpretable
directions*

These conceptual spaces reflect the pairwise similarities of the underlying entities, but the coordinate axes of their coordinate system are not necessarily interpretable. In order to identify interpretable directions in these conceptual spaces, Derrac and Schockaert identify a set of candidate terms (e.g., adjectives such as "funny" in the MOVIE domain) and try to find directions in the similarity space which separate all entities associated with the given candidate term from all other entities.

*Defining different
commonsense
reasoning strategies*

After having extracted these conceptual spaces, Derrac and Schockaert provide geometric definitions for several commonsense reasoning strategies: Similarity-based reasoning (which generalizes properties from similar observations) can directly be based on distances in the conceptual space – the smaller the distance between two entities, the more similar they can be considered to be. In addition to this, Derrac and Schockaert also formalize a degree of betweenness for triples of points (e.g., quantifying to which extent WINE SHOP is conceptually between GOURMET SHOP and LIQUOR STORE) in order to allow for interpolative reasoning (such as concluding that if both a GOURMET SHOP and a LIQUOR STORE have to pay a certain tax, a WINE SHOP will have to do so as well). In order to support analogy-based reasoning of the type "MEDICAL SCHOOL is to SANATORIUM AS MILITARY SCHOOL is to MILITARY BARRACKS", they also propose different ways of measuring the parallelism of directions.

*Implementing
commonsense
reasoning as
classifiers*

All of these definitions are put to use in classifiers targeting commonsense reasoning. Derrac and Schockaert identify similarity-based reasoning with a k nearest neighbor classifier. Moreover, they devise a betweenness-based classifier which is based on the assumption that conceptual regions are convex. If the betweenness-based classifier is asked to classify a point b , it tries to find two points a and c which have an identical class label and for which b has a high degree of betweenness.

Also the analogy-based reasoning can be used for a classifier as follows: Given a query point a , find three points b, c, d such that \vec{ab} has a high degree of parallelism with \vec{cd} . By analyzing the class labels assigned to b, c , and d , one can infer a class label for a (e.g., if both b and d are classified as ACTION MOVIES and c is labeled as COMEDY, then by analogy also a should be classified as COMEDY). Finally, Derrac and Schockaert propose to implement a fortiori reasoning of the type "if x is more scary than THE SHINING, then x is a HORROR MOVIE" by extracting such rules based on the ranking of the entities with respect to the interpretable directions they had identified before.

In several experiments, Derrac and Schockaert found that for the conceptual spaces of PLACE TYPE and WINE, where only a limited amount of observations is available, betweenness-based and analogy-based classifiers seem to perform best, defeating similarity-based and a fortiori reasoning as well as standard machine learning techniques. On the other hand, on the conceptual space of the MOVIE domain, where many observations are available, the betweenness-based and analogy-based classifiers cannot be efficiently used due to a combinatorial explosion of candidate tuples to consider. Here, a standard support vector machine outperformed the a fortiori classifier.

The work by Derrac and Schockaert makes several valuable contributions to the research area of conceptual spaces and will resurface in later chapters of this dissertation. Their definition of conceptual betweenness as a meaningful relation for commonsense reasoning will be incorporated into our formalization of conceptual spaces in Chapter 4. Moreover, we will discuss their approach towards extracting a conceptual space from textual data and for identifying interpretable directions in this space in Chapter 8 in the general context of multidimensional scaling. Finally, their experiments with respect to machine learning in conceptual spaces will be reviewed in more detail in Chapter 7.

Experimental results

Relevance

1.2.6 Neural Grounding

The idea that neurons in the human brain encode spatial structures has been proposed by various authors [25, 144, 191]. In the following, we summarize three recent proposals, which may provide some neural grounding of the conceptual spaces framework.

Balkenius and Gärdenfors [25] claim that psychological similarity spaces naturally emerge from neural activity spaces through the process of dimensionality reduction. In order to illustrate their hypothesis, they note that individual neurons in the motor cortex seem to be tuned to particular movement directions. The activity of such a neuron is strongly correlated to the degree to which the movement matches this neuron's preferred direction. This preferred direction can be interpreted as a prototype. If one considers a whole population of such neurons, and if the activation of each of these neurons is represented as a real number, then the overall activation of the neural population can be interpreted

*Neural populations
can encode
psychological
similarity spaces*

as a vector of individual activation values. This activation vector lies in a space which one could call the *neuron space* or the *activation space*. Based on the activation vector and the preferred directions of the individual neurons, it is possible to reconstruct the original movement direction (which can be represented as a vector in three-dimensional physical space): One simply needs to compute a weighted sum over the neurons' preferred directions, using their activation as a weight. Balkenius and Gärdenfors therefore argue that the population coding in the brain is just a high-dimensional, redundant representation of a low-dimensional space. The large amount of redundancy within a neural population makes it more robust against both noise and the deletion of individual neurons. Balkenius and Gärdenfors note that population coding implicitly ensures that similar activation vectors correspond to similar low-dimensional psychological vectors. This means that similarity relations are preserved – which is quite important for the framework of conceptual spaces.

*More examples:
physical space,
emotions, and colors*

Balkenius and Gärdenfors make several other examples that illustrate how psychological spaces can be mapped onto brain activities: External physical space seems to be represented by the activity of so-called place cells and grid cells, a three-dimensional emotion space can be grounded in the neurotransmitters serotonin, dopamine, and noradrenaline, and the psychological color space is based on so-called opponent channels (black-white, red-green, and blue-yellow) which can be grounded in the three types of color cones in the human retina.

*Important
mathematical
properties*

Balkenius and Gärdenfors also argue that a spatial representation of perceptions and actions is useful from a cognitive point of view. If one views cognition as a mapping function from perception to action, then this mapping function should according to Balkenius and Gärdenfors fulfill the following three properties:

- *Monotonicity*: An increase in the perceptual variable always leads to an increase in the action variable.
- *Continuity*: Small changes in the perceptual variable only lead to small changes in the action variable.
- *Convexity*: Closed regions in perceptual space are mapped onto closed regions in action space.

*Making the learning
problem feasible*

According to Balkenius and Gärdenfors, these restrictions on the mapping function constrain it in such a way that learning such a mapping becomes feasible. The three restrictions especially help to generalize to unseen perceptions by extrapolation (monotonicity), similarity (continuity), and interpolation (convexity). This relates to the different commonsense reasoning strategies discussed by Derrac and Schockaert [123] (cf. Section 1.2.5). Moreover, Balkenius and Gärdenfors argue that if both perceptions and actions are spatially represented, then continuity implies both monotonicity and convexity. Taken together, their work provides an argument that the conceptual spaces framework is not only psychologically and cognitively useful, but that it also can

be connected to activity in the human brain.

Also the semantic pointer architecture (SPA), a cognitive architecture by Eliasmith [144], is based on the hypothesis that populations of neurons encode a single cognitive dimension. Also Eliasmith makes a distinction between the *state space* (which describes the mathematical object that is being represented) and the *neuron space* (which describes the activity of the neural population representing this object). The central notion of the SPA is the notion of a semantic pointer: A *semantic pointer* is a high-dimensional vector in state space which is represented by a neural population in neuron space. Eliasmith assumes that these semantic pointers are a compressed version of the full representation of a given observation or concept. For example, one could use the activation of the highest level of a deep artificial neural network as a semantic pointer which in some sense "summarizes" the original visual input. The key idea here is that the semantic pointer can be "dereferenced" or "decoded" into lower-level information, e.g., by reconstructing the lower-level input that led to the observed high-level activation.

The semantic pointer architecture

Eliasmith assumes that all semantic pointers have unit length, and that each concept is associated with a prototypical vector as well as a certain *attractor region*: All vectors that fall within this attractor region are classified as belonging to the corresponding concept. Eliasmith visualizes this as a *conceptual golf ball*: The surface of this golf ball corresponds to a conceptual similarity space, the dimples represent the concepts, and points on the surface of the ball correspond to individual observations. While this conceptual space also allows us to compute similarities between observations and between concepts, it uses the angular distance between the vectors instead of an Euclidean or Manhattan distance of the points.

The conceptual golf ball

In order to combine information from multiple domains and modalities, Eliasmith introduces a binding operation $*$ which can be used to combine multiple semantic pointers in the same space. He shows that if circular convolution is used as binding operation, then one can also reconstruct the individual constituents from a composite. Eliasmith proposes that one can then encode concepts as follows:

Combining multiple domains

$$\begin{aligned} \text{DOG} = & \text{PERCEPTUAL} * (\text{VISUAL} * \text{DOG}_{\text{VISUAL}} \\ & + \text{AUDITORY} * \text{DOG}_{\text{AUDITORY}} \\ & + \text{TACTILE} * \text{DOG}_{\text{TACTILE}} + \dots) \\ & + \text{ISA} * \text{MAMMAL} + \dots \end{aligned}$$

Here, $*$ denotes the binding operation and $+$ denotes the vector addition. One can think of the resulting composite concept as a frame-like structure, where the vectors PERCEPTUAL , VISUAL , etc. are the slot names, which are then connected with the semantic pointer which fills this slot (e.g., $\text{DOG}_{\text{VISUAL}}$). While this is in some degree reminiscent of the domain structure of conceptual spaces, it is important to note that all computations here take place in the same space: All the semantic pointers involved in the DOG concept lie in the same space as the DOG

Relation to frames and conceptual spaces

concept itself. In the conceptual spaces framework, however, properties are defined in individual domains whereas concepts span multiple domains and thus inhabit a different space.

Relevance

Although the semantic pointer architecture is not a direct implementation of the conceptual spaces theory, it makes use of spatial representations and connects them to neural activity. As Eliasmith has demonstrated, the SPA is capable of reproducing various psychological and neuroscientific results, offering thus a way to bridge those two disciplines. It adds support to the general idea that cognition can be expressed through similarity spaces and it provides concrete ideas about how such similarity spaces can arise from neural computations.

Tensor product binding

The SPA is heavily inspired by the earlier work of Smolensky [380], who also considered the variable binding problem. More specifically, he raised the question, how symbolic data structures such as lists and trees can be mapped into a vector space. His proposed solution is the so-called *tensor product binding*. Smolensky defines a symbolic structure as a set of roles with possible fillers, and an object as a conjunction of concrete filler/role bindings. He points out, that conjunction can be represented in connectionist systems by simple vector addition, which is both associative and commutative. He then proposes to represent both roles and fillers as vectors in a given vector space, and to represent the variable binding of a given role vector $\vec{r} \in \mathbb{R}^n$ and a given filler vector $\vec{f} \in \mathbb{R}^m$ with the tensor product $A_{ij} = \vec{r}_i \cdot \vec{f}_j$.

Unbinding

Smolensky also introduces an unbinding mechanism for reconstructing \vec{f} , given \vec{r} and A : He assumes, that the set of role vectors $\vec{r}^{(k)}$ spans a subspace of the overall vector space \mathbb{R}^n , and that one can find a dual basis consisting of vectors $\vec{u}^{(k)}$, such that $\vec{r}^{(k)} \cdot \vec{u}^{(k)} = 1$, while $\vec{r}^{(k)} \cdot \vec{u}^{(l)} = 0$ for $k \neq l$. Then, the filler value $\vec{f}^{(k)}$ associated with the role $\vec{r}^{(k)}$ can be obtained as $\vec{f}^{(k)} = s \cdot \vec{r}^{(k)}$ with a scaling factor s . Smolensky argues, that by using $\vec{r}^{(k)}$ instead of $\vec{u}^{(k)}$ for the unbinding, one obtains a superimposition of all fillers based on the respective cosine similarity between $\vec{r}^{(k)}$ and $\vec{r}^{(l)}$. This allows for graceful degradation, where a potentially infinite number of roles can be represented in a finite-dimensional vector space. Both the binding and unbinding mechanism can be implemented with so-called *sigma-pi units*, which are specialized artificial neurons. Unfortunately, the number of these units quickly becomes prohibitively large with increasing problem size.

Recent usage

Recently, Jiang et al. [210] have applied Smolensky's ideas to modern deep learning approaches. Considering the use case of abstractive summarization, they modified the transformer architecture [417] (which is the basis of modern large language models such as GPT-3 [83]) based on Smolensky's proposal. They report better results with respect to both abstractive summarization and other related tasks such as named entity recognition and part of speech tagging.

Relation to SPA and conceptual spaces

Overall, the tensor product binding approach by Smolensky can be seen as a precursor to the semantic pointer architecture discussed above. Its main strength lies in the representation of part-whole struc-

tures. While Smolensky's approach combines fillers and roles into more complex product spaces (and can do so recursively), the SPA confines itself to a single space, which considerably reduces representational complexity. This is achieved by using circular convolution rather than the tensor product for filler/role bindings, coming at the disadvantage of a higher potential for noise in the unbinding process. In comparison to Smolensky's proposal, conceptual spaces offer interpretable dimensions, intuitive notions of conceptual betweenness and semantic similarity, and a clear cognitive grounding.

Recently, Hawkins et al. [191] have made an argument for spatial representations in the human brain based on grid cells. They start their discussion by noting that grid cells have been associated with the representation of an animal's location in the environment. In difference to other neurons, which typically exhibit their maximal activation only for one specific prototypical input, grid cells seem to have a grid of such prototypical inputs which are roughly equally spaced. A grid cell fires if the input is close to any of these prototypical points. When representing physical space, these prototypical points correspond to specific locations of the animal in a given environment. The output of a single grid cell does by itself not identify the exact location of the animal. However, if one combines the output of multiple grid cells, one can identify the animal's position – there is usually only a single location in physical space that excites a given set of grid cells simultaneously.

Hawkins et al. suggest that such grid cells exist everywhere the neocortex and that they are used to represent also other properties in the form of locations in a semantic space. Since neuroscientific research has indicated that there is a unique representation for each different environment (e.g., each room in a building), Hawkins et al. postulate that also each object has its own representation. Instead of moving in the environment, we move our sensors (e.g., our eyes) over the object of interest (e.g., a pen) where we can again identify different locations (e.g., the cap). Thus, also the properties of objects can be represented in a spatial way, for instance through grid cells.

According to their proposal, part-whole relations can be described with displacement vectors. For instance, if a logo is somewhere on a coffee cup, then a particular location can be represented either with respect to the cup or with respect to the logo. As the difference between these two locations is always constant, their difference vector can be used to describe the spatial arrangement of parts. This hypothesis of representing part-whole relations through displacement information on parts can be related to extensions of the conceptual spaces framework that use a similar approach for representing the configuration of parts [158] (see also Chapter 2 for a discussion of part-whole relations in conceptual spaces). Hawkins et al. hypothesize that this displacement vector can be learned with so-called displacement cells which are usually employed to compute the difference between two locations in the same space (namely three-dimensional physical space) for navigation, but which can also be used to compute the difference between two

*Grid cells for
representing physical
space*

*Grid cells for
representing
semantic spaces*

*Part-whole relations
as displacement
vectors*

non-physical location in two different spaces. Also recent ideas by Goldowsky [168] in the context of place cells go in a similar direction by considering the encoding of landmarks in two different brain areas (which are linked with a so-called Laplace transformation) and the computation of differences in order to model trajectories.

Relation to the exemplar view

Hawkins et al. furthermore argue that there is no single central model of a concept (such as CUP), but that there exist hundreds of decentralized models for each concept, each one based on slightly different perceptions. A new observation is then classified as belonging to a given concept, if all or most models agree on the classification. This can be related to the exemplar view on concepts which also emphasizes that there is no central abstract representation for concepts.

Comparison to the SPA

The work by Hawkins et al. differs from the work by Eliasmith in several ways: While Eliasmith represents a concept by a single semantic pointer, Hawkins et al. propose the usage of hundreds of distributed models. Moreover, they use grid cells as basis of their analysis, whereas Eliasmith focuses on population coding. Finally, Hawkins et al. do not explicitly differentiate between domains, which Eliasmith does when using the binding operator to construct composite representations. While the proposal by Hawkins et al. is at the current point in time quite speculative, it nevertheless adds independent support to the claims by Balkenius and Gärdenfors by proposing an alternative mechanism by which psychological spaces could be represented in the human brain.

1.3 CONTRIBUTIONS OF THIS DISSERTATION

Existing applications of conceptual spaces

The framework of conceptual spaces has already been applied in a wide variety of contexts, ranging from linguistics [70, 69] over the semantic web [4, 129, 180] and psychology [238] to cognitive science in general [124, 128, 136, 369, 370] (cf. also [218, 454]). As we have seen in Section 1.2, the conceptual spaces framework has also been successfully used in various areas of artificial intelligence. Table 1.3 summarizes the four main applications discussed in Section 1.2. As one can see, despite using the same underlying framework, these approaches differ with respect to their concrete interpretation of conceptual spaces, e.g., concerning the grounding of the domains and dimensions. Moreover, each of these applications tends to have some idiosyncratic assumptions, preventing it from being easily extended to other use cases. Our research attempts to fill this gap by providing general tools for applying conceptual spaces in artificial intelligence. In Section 1.3.1, we describe our overall envisioned AI system, which serves as a motivation for the concrete research contributions presented in this dissertation. While our overall motivation is strongly tied to the symbol grounding problem, we do not confine ourselves to any specific subtopic.

The three core problems

We can divide the overall problem of using conceptual spaces for artificial intelligence into three subproblems: Firstly, we need a proper mathematical formalization of the framework which lends itself to-

	Lieto et al. [258, 264, 265, 266] (Section 1.2.2)	Chella et al. [95, 96, 97, 98] (Section 1.2.4)	Banaee et al. [26, 27] (Section 1.2.5)	Derrac and Schockaert [122, 123] (Section 1.2.5)
Application	Classification	Perceptual Anchoring	Description Generation	Common- sense Reasoning
Data Source	Structured Linguistic Resources	Sensory Data	Machine Learning Datasets	Textual Data (Tags and Reviews)
Grounding of Conceptual Space	Handcrafted Feature Extractors	Handcrafted Feature Extractors	Feature Selection	Multi- dimensional Scaling
Repre- sentation of Concepts	Prototype, Exemplars, Definition	Arbitrary Region / Prototype	Convex Hull of Exemplars	Exemplars

Table 1.3: Overview of four applications of conceptual spaces in the area of artificial intelligence.

wards an actual implementation in software. This first requirement is motivated in more detail in Section 1.3.2. Secondly, one needs to determine the dimensions spanning the conceptual space and relate them to sensory input in order to map individual perceptual observations into the conceptual space. We will give a brief overview over this research problem in Section 1.3.3. Thirdly, we need to specify how these individual observations can give rise to conceptual regions. A short introduction into cognitively plausible ways of learning concepts is given in Section 1.3.4. These three central requirements for a successful application of the conceptual spaces framework in artificial intelligence will then resurface in the three parts of this dissertation:

In Part I of this dissertation, we provide a thorough mathematical formalization of the conceptual layer along with a publicly available implementation of this formalization. Since many applications of conceptual spaces use an ad-hoc implementation of the framework, our work can be used as a common basis for future applications, making it possible to integrate them with each other. In Chapter 2, we develop a formal description of conceptual regions, before providing several operations for creating new concepts based on existing ones (Chapter 3) and for quantifying relations between concepts (Chapter 4).

Part II of this dissertation then provides a thorough overview of several approaches in machine learning and optimization which are relevant to the two learning processes identified by Gärdenfors [182] (cf. Section 1.2.1). This includes a general introduction to important concepts in machine learning in Chapter 5 and a more detailed discussion of artificial neural networks for representation learning in Chapter 6. Moreover, we describe how dissimilarity ratings elicited in psychological studies can give rise to psychological similarity spaces

*Formalizing the
conceptual layer*

*Machine learning
and optimization
background*

in Chapter 8, where we introduce the technique of multidimensional scaling. Finally, we comment on several research strands aiming for a more human-like way of learning concepts in Chapter 7.

*Experimental studies
on domain
grounding*

In Part III of this dissertation, we then report the results of two experimental studies aimed at automatically extracting the dimensions of a conceptual space from a given dataset of observations. More specifically, we explore a hybrid approach which combines the psychological grounding provided through multidimensional scaling with the generalization capabilities of artificial neural networks. In Chapter 9, we describe a first feasibility study for this hybrid approach, which was conducted on a dataset of novel and unknown objects. We then put our focus on the domain of shapes, whose internal structure is still poorly understood: In Chapter 10, we provide a brief overview of ways for modeling the shape domain in different scientific disciplines. We furthermore introduce a psychological dataset on shape perception and provide a first analysis of the raw data. In Chapter 11, we then apply multidimensional scaling on this dataset to extract psychological similarity spaces, which are then thoroughly analyzed with respect to three core predictions of the conceptual spaces framework. Finally, in Chapter 12, we apply artificial neural networks to learn a mapping from raw images into the extracted shape similarity spaces, considering both transfer learning and multi-task learning settings based on both a classification and a reconstruction task.

1.3.1 The Envisioned AI System

Motivation

In this section, we sketch an envisioned AI system which motivates the research presented in this dissertation. Please note that in the course of this dissertation, we will not be able to build such a complete system. However, we lay important groundwork for the practical implementation of such an approach. The individual contributions made by this dissertation can then be combined with each other as well as with other existing tools and approaches to implement a system similar to the one sketched in this section.

Scope of this research

Before we describe the overall envisioned architecture, we need to comment on its intended scope. Our research is intended to focus primarily on physical objects. This means that we will not consider abstract concepts like FRIENDSHIP OR ILLEGAL which do not directly correspond to observable objects in the real world. Moreover, we will not take into account temporal concepts including actions and events. Finally, while we understand our work in the context of cognitive AI, neither our envisioned system nor the foundational research presented in this dissertation attempt to make any claims about the inner workings of the human mind.² They should therefore not be interpreted as a faithful psychological model of human cognition, but rather as a cognitively inspired approach for machine intelligence.

² Thanks to Igor Douven for urging me to explicitly clarify this.

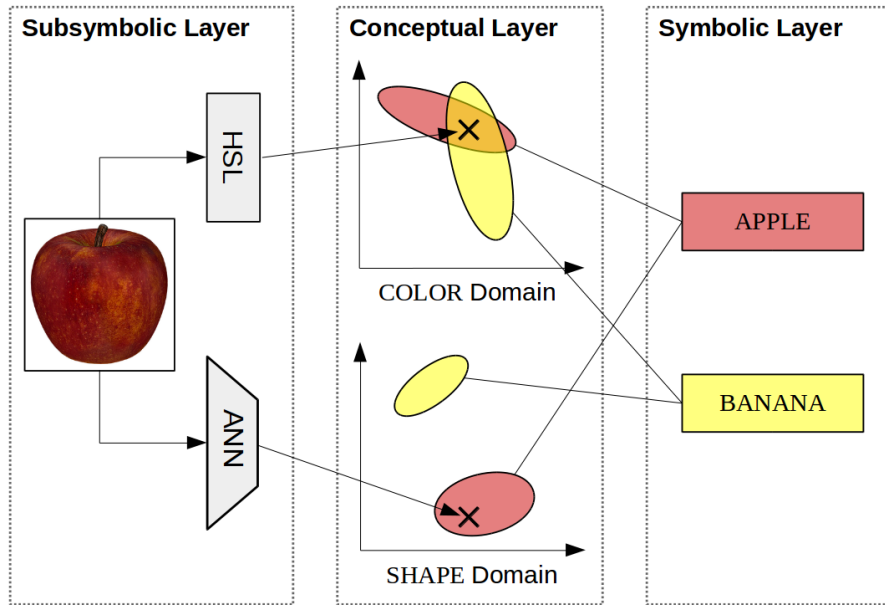


Figure 1.5: Illustration of the envisioned AI system.

Figure 1.5 illustrates our envisioned overall system. It is comprised of the three layers introduced in Section 1.2 as well as concrete ways of translating between them. Overall, we target a bottom-up process, but top-down connections can be established as well.

For the sake of simplicity, we show only two domains, namely COLOR and SHAPE. COLOR can be represented by the HSL space from Figure 1.1 using the three dimensions HUE, SATURATION and LIGHTNESS. This information can be extracted from an input image by using a specialized hard-coded procedure. The structure of the SHAPE domain on the other hand is less well understood. We therefore assume that it needs to be learned, for example by using an artificial neural network. The activation of the output layer of this network can be used as dimensions of the shape domain.

Both example concepts APPLE and BANANA are available as symbols in the symbolic layer, but can also be described by their respective regions in the COLOR and SHAPE domain. Intuitively, APPLE covers red, yellow, and green tones in the color domain, while BANANA includes shades of green, yellow, and brown. This is illustrated in Figure 1.5 by overlapping regions in the COLOR domain. In the SHAPE domain, however, we expect little overlap between APPLE (which tends to be round) and BANANA (which is typically elongated and curved).

After having described the overall structure of our envisioned system, let us now consider some of its potential dynamics. More specifically, we will now sketch how classification, incremental concept learning, nonmonotonic reasoning, and top-down processes can be implemented.

If the system makes a new observation (e.g., an apple as depicted in Figure 1.5), it can convert this observation into a point in the conceptual space by applying the translation mechanisms outlined above. For the

The envisioned system in a nutshell

Two example domains

Two example domains

System dynamics

Classification

COLOR domain, this is done by a hard-coded conversion to the HSL COLOR space. Moreover, the observation is fed into the neural network, which predicts the coordinates for the SHAPE domain. In order to classify this observation, the system then needs to check whether the resulting data point is contained in any of the conceptual regions. In our example, the data point belongs to the APPLE region in both the COLOR and the SHAPE domain and is thus classified as an apple.

Incremental learning

Each new observation can potentially also lead to an update of the respective conceptual regions. If the observation was classified as APPLE, but it is not close to the center of the APPLE region in one of the domains, this region might be enlarged or moved by a small amount, such that the observation is better matched by the concept description. If the observation does not match any of the given concepts at all, a new concept might be created. Please note that the updates considered above only concern the connections between the conceptual and the symbolic layer. The connections between the subsymbolic and the conceptual layer are expected to remain fixed in our architecture. The neural network thus only serves as a preprocessing step in our approach: It is trained before the overall system is used and remains unchanged afterwards. Simultaneous updates of both the neural network and the concept description might be desirable, but would probably introduce a great amount of additional complexity.

Nonmonotonic reasoning

As already mentioned at various points in Section 1.2, the conceptual layer lends itself towards different types of nonmonotonic reasoning, including similarity and typicality judgments as well as concept combinations and commonsense reasoning. All of these processes can be included in our envisioned architecture as well: The classification of an observation with respect to the classes from the symbolic layer can be annotated with typicality information based on distances in the conceptual layer. The concepts from the symbolic layer can be combined with each other by combining their respective regions in an appropriate way. In simple cases such as RED APPLE, this corresponds to restricting the conceptual region of APPLE in the color domain to red colors, which can be geometrically implemented by an intersection of regions. Finally, approximate reasoning can be implemented based on the relations between conceptual regions: If we have the symbolic annotation that Alice likes to eat apples, and if the conceptual regions representing APPLE and PEAR are quite close to each other in all domains, then it may be plausible to assume that Alice will also enjoy eating a pear. This conclusion cannot be drawn from symbolic knowledge alone, but is based on the spatial representation in the conceptual layer.

Top-down processes

So far, we have only considered bottom-up processes and processes involving only the conceptual layer. We would, however, like to point out that also top-down processes can be incorporated into our envisioned AI system. For instance, *imagination* can be seen as a translation from the symbolic to the subsymbolic layer: One starts for instance with the APPLE concept in the symbolic layer, activates its spatial representation in the conceptual layer, picks a point representative of the respective regions

(e.g, the centroid) and feeds it through the reverted conversion processes to arrive at a subsymbolic representation. Depending on the type of neural network used, such a reconstruction of raw input based on a compressed representation can be easily obtained. Another important top-down process is *zero shot learning*, where a new concept together with its grounding can be learned without any observations but only based on a logical description in the symbolic layer. This corresponds to the zebra example from Harnad [190] discussed in Section 1.2.3. In our case, information from the symbolic layer is used to construct a representation in the conceptual layer from existing conceptual regions.

One can see that our overall envisioned system is basically an implementation of the three layer architecture introduced by Gärdenfors [179]. It is capable to include many different types of learning and reasoning processes and can thus potentially unify the individual existing applications of conceptual spaces to artificial intelligence in a single system. However, building such a system in practice is quite challenging as many different parts and their interactions need to be properly specified. Especially the concrete setup of the conceptual layer and its relations to the subsymbolic and the symbolic layer is of crucial importance. Only if this overall setup is sound, a successful integration of the different learning and reasoning mechanisms is possible. This dissertation focuses on providing a comprehensive formalization of the conceptual layer as well as principled ways of connecting it to the subsymbolic and the symbolic layer. It therefore lays the groundwork for practical applications of the conceptual spaces framework which can then use this overall setup to implement the different cognitive processes described above.

Outlook

1.3.2 A Comprehensive Formalization of the Framework

In order to implement the conceptual spaces framework for practical AI applications, one needs to specify how concepts are represented. Gärdenfors argues that concepts should be modeled as convex regions, but this restriction still leaves many degrees of freedom. For an actual implementation, we need a parametric description of conceptual regions such that each conceptual region can be represented by a small number of parameters. Moreover, developing a thorough mathematical formalization and a practical implementation forces one to be explicit about many details of the framework. This can provide further insight into theoretical relationships between different aspects of the framework.

Motivation

There have already been several proposals for a mathematical formalization of conceptual spaces [3, 10, 11, 253, 327, 329], all of which, however, have certain limitations and shortcomings. In our opinion, a thorough formalization of the conceptual spaces framework should fulfill the following criteria:

Criteria for a good formalization

- Concepts and properties should be described in a parametric way in order to allow for a straightforward implementation.

- Concepts and properties should be represented with the same formalism. This makes it easier to devise mechanisms for learning and reasoning that are applicable to both properties and concepts without major modifications.
- The formalization should provide a concrete way for representing cross-domain correlations, because these correlations contain important information about the concept.
- Conceptual boundaries should be imprecise in order to reflect borderline cases.
- The formalization should come with a publicly available implementation such that other researchers can use it right away for their own research projects.
- There should be a large set of operations that can be applied within this formalization in order to support both learning and reasoning processes.

The need for a new formalization

Unfortunately, none of the existing formalizations satisfies all of these requirements and combining different formalizations with different strengths and weaknesses is not easily possible. This justifies the development of a new formalization aiming to fulfill all of the above mentioned desiderata.

Concepts as star-shaped regions

Starting from the desire to represent correlations between domains in a geometric way, we notice a problem with Gärdenfors' convexity requirement in combination with the usage of the Manhattan metric for combining domains in Chapter 2. In order to resolve this problem, we loosen the convexity requirement by replacing it with star-shapedness. This leads us to a definition of concepts as star-shaped fuzzy sets based on axis-parallel cuboids.

Operations for concept creation

In Chapter 3, we enrich our formalization with several operations that can be used to create new concepts based on existing ones. More specifically, we discuss the intersection, union, negation, and projection of concepts, as well as an operation for splitting a concept into two parts. Each of these concept creation operations is expected to return a valid concept according to our parametric definition. However, the naive set intersection of two star-shaped sets is for example not necessarily star-shaped. We therefore propose to employ a repair mechanism reflecting the psychological effect of overextensions (cf. Section 1.1.2).

Relations between concepts

In Chapter 4, we then put our focus on different measures for specifying various relations between concepts. This includes concept size, subsethood, implication, similarity, and betweenness. Many of the remaining measures are directly or indirectly based on the size of a concept, which can be defined quite easily by equating it with the integral over the concept's membership function. However, as we are interested in a practical implementation of the framework, this level of abstraction is not satisfactory: We need a concrete procedure for computing this integral. We therefore derive a closed formula for the size of a concept

which can be evaluated significantly faster than approximating the integral with numerical optimization algorithms.

Overall, we provide a fairly comprehensive formalization of the conceptual spaces framework, taking into account not only representational desiderata, but also providing a plethora of useful operations and measures. Our publicly accessible implementation of this formalization not only serves as a proof of concept, but provides other researchers with the opportunity to use our formalization off the shelf for their own experiments. Throughout Part I of this dissertation, we compare our definitions to prior formalizations of the framework. Moreover, we sketch how the different operations can support a variety of learning and reasoning mechanisms.

Summary

1.3.3 Obtaining the Dimensions of a Conceptual Space

One general issue that arises when defining a conceptual space concerns the number of its dimensions. A low-dimensional space corresponds to a very compact representation which is efficient with respect to both storage and computation. Constraining the space to a small number of dimensions can furthermore help to filter out irrelevant noise through information compression. A high-dimensional space on the other hand is usually capable of representing finer nuances of cognitive similarity. Following Occam's razor, one therefore generally aims to optimize this trade off by selecting as many dimensions as necessary but as few as possible. Determining the optimal number of dimensions is often based on analyzing the interpretability of the space and its capability of accurately representing similarities in a spatial manner.

How many dimensions do we need?

From a machine learning perspective, this is related to feature engineering: If the feature space contains not enough features, a classifier might not be able to make important distinctions between classes. If the feature space is, however, too large and only sparsely populated, the classifier might be led astray by irrelevant features, confusing noise for valuable information. We will revisit this issue in Chapter 5, when talking about the "curse of dimensionality" and various dimensionality reduction techniques.

The machine learning perspective

In order to define the dimensions of a conceptual space, one needs, however, not only to define how many dimensions there should be, but also how these dimensions should be interpreted and how their values can be extracted from raw sensory data.

Grounding dimensions in perception

Overall, our research focuses on obtaining a conceptual space for the SHAPE domain. It is well known that the shape of objects contains important information for their classification [298, Chapter 10] and that children exhibit a strong shape bias when making generalization [212, 242]. Hence, shape information can be expected to be crucial for any artificial agent with visual input. However, despite its importance, the internal structure of the SHAPE domain is still not understood very well [146]. Using the SHAPE domain as an example, we investigate different

The domain of shapes

ways of learning a shape space in this dissertation. Our proposed procedures are formulated in a general way which is expected to generalize to other domains as well, hence providing general tools for obtaining conceptual spaces for poorly understood domains.

Three principled approaches

In his book [179, Sections 1.7 and 6.5], Gärdenfors identifies three basic ways for constructing a conceptual space: Handcrafting, machine learning, and multidimensional scaling. We will summarize them shortly in the following, discussing also their individual strengths and limitations.

Handcrafting a conceptual space

Handcrafting a conceptual space usually consists in manually defining the dimensions of the conceptual space based on the available sensors. Moreover, it entails specifying a mapping function from sensory input to values on these dimensions [179, Section 1.7]. This approach was for example taken in the anchoring system by Chella et al. [96] (cf. Section 1.2.4). Similar proposals have been made in the area of cognitive robotics by Thosar et al. [407] and Jäger et al. [214] who investigate perceptually grounded knowledge bases for robots, using both physical and functional features. Although their work is not directly linked to the conceptual spaces approach by the authors, it would be relatively straightforward to transform the contents of their data bases into a conceptual space. Both Lieto et al. [266] and Banaee et al. [27] made use of such pre-existing data bases for defining their conceptual spaces (cf. Sections 1.2.2 and 1.2.5).

Advantages of handcrafting

A clear advantage of the handcrafting approach is that the resulting dimensions are interpretable, since they have been manually defined by domain experts. Moreover, by manually defining the mapping function, one can ensure a high quality, for instance by using smoothing techniques to reduce sensor noise. Another advantage relevant in the field of AI is the ability to generalize to unseen inputs: If the sensors produce previously unseen values (e.g., a camera image that has not been seen before), then the mapping function can easily convert this novel observation into a point in the conceptual space. The system using this conceptual space is therefore not limited to a fixed set of inputs and is potentially able to generalize to new situations.

Disadvantages of handcrafting

The handcrafting approach does, however, also come with some drawbacks: First of all, it can be quite time consuming to manually define the mapping function for complex input, for instance when working with camera images as input modality. The mapping function might in this case involve different computer vision algorithms, requiring a certain expertise in this domain. Scaling this approach up to multiple complex domains might thus not be possible due to the prohibitive amount of work involved. Furthermore, the dimensions defined by a domain expert might be useful to solve the problem at hand, but by themselves they cannot claim any psychological validity. Moreover, this approach is only applicable to domains with a well understood internal structure such as the COLOR domain. However, handcrafting becomes quite difficult for more complex domains such as SHAPE that are based on complex sensors such as cameras. As we focus our research on the SHAPE domain,

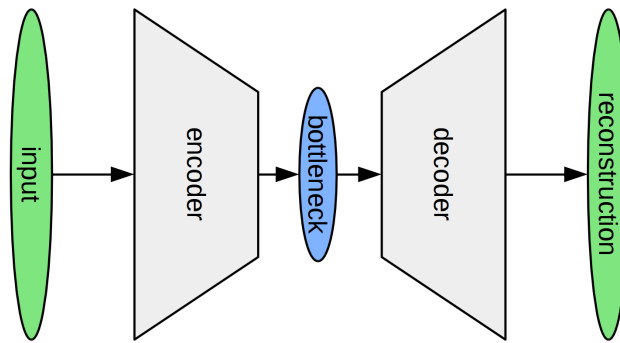


Figure 1.6: Structure of an autoencoder.

we will therefore not consider the handcrafting approach in more detail.

The second approach for obtaining the dimensions of a conceptual space uses *machine learning techniques for dimensionality reduction*. Gärdenfors [179, Section 6.5] argues that raw perceptual input is too rich and too unstructured for direct processing. It is thus necessary to lift the input to a more economic form of representation, which typically involves a drastic reduction in the number of dimensions. There exists a large variety of dimensionality reduction algorithms in the machine learning field. Our focus in this dissertation lies on artificial neural networks (ANNs) which are capable of conducting a multi-layered and non-linear dimensionality reduction.

Machine learning techniques for dimensionality reduction

One example for such a neural network is the structure of an autoencoder [172, Chapter 14] illustrated in Figure 1.6. The high-dimensional input of the network (e.g., the raw pixel values of an image) is compressed in multiple steps by the encoder network until it reaches the bottleneck layer which consists of a very small number of neurons. This compressed representation from the bottleneck layer is then subsequently decompressed by the decoder network until it reaches the output layer which has the same size as the input layer. Based on a large dataset of observations, the parameters of the individual compression and decompression steps in this network are adjusted in such a way that the activation of the output layer is a faithful reconstruction of the original input. If the network succeeds in this reconstruction task, then it has learned a low-dimensional representation of the high-dimensional input. One can then interpret this bottleneck layer as a conceptual space by interpreting each of the neurons as one dimension and its activation for a given stimulus as the value of this stimulus on this dimension.

Autoencoders

Training such a neural network for extracting a low-dimensional representation usually requires only little prior knowledge about the domain. It is thus applicable if handcrafting specific features is not feasible due to a lack of domain knowledge. Moreover, once they are trained, neural networks are able to generalize to unseen inputs, which is an important requirement for practical AI applications. Since neural networks are in principle agnostic about the type of input data, they can furthermore be applied to a wide variety of domains.

Advantages of dimensionality reduction

Disadvantages of dimensionality reduction

However, a certain expertise in defining the exact structure of the network and the appropriate training scheme is crucial for success. Moreover, adjusting the parameters of the neural network typically requires large amounts of data and computation time. The most serious drawback of neural network models in the context of conceptual spaces is, however, their lack of interpretability and psychological grounding.

In Chapter 5, we provide a more thorough introduction to machine learning and dimensionality reduction, and in Chapter 6, we survey research in artificial neural networks for the purpose of representation learning. In this context, we especially pay attention to recently proposed network structures such as InfoGAN [101] and β -VAE [196] which claim to learn interpretable features from unlabeled data.

Multidimensional scaling on psychological dissimilarity ratings

A third way of obtaining a conceptual similarity space is based on *psychological dissimilarity ratings* [179, Section 1.7]. These dissimilarity ratings are collected for a fixed set of stimuli in a psychological experiment. They are then fed into an algorithm called *multidimensional scaling* (MDS) which computes an n -dimensional geometric representation of the stimulus set, where geometric distances between pairs of stimuli reflect their psychological dissimilarity [71]. While typically used in the area of psychophysics to understand and visualize the similarity of perceptual stimuli, the research by Derrac and Schockaert [123] (cf. Section 1.2.5) has shown that MDS can also be applied to other data sources such as text corpora.

Advantages of multidimensional scaling

The similarity spaces obtained through MDS have the advantage of being psychologically grounded since distances in the spaces relate to the findings from psychological experiments. Multidimensional scaling is especially useful for exploratory studies in domains which are poorly understood since the resulting arrangement of points in the similarity space can provide valuable insight about potentially relevant features.

Disadvantages of multidimensional scaling

Multidimensional scaling has, however, two considerable drawbacks: On the one hand, the data collection process through psychological studies is quite time-consuming as it requires carefully designed experiments with a sufficiently large number of participants. Moreover, the similarity spaces obtained through MDS are limited to the given stimulus set. This means that novel stimuli cannot be easily mapped into the similarity space unless additional dissimilarity ratings are collected. Finally, the individual coordinate axes of an MDS solution are not necessarily interpretable.

A hybrid approach combining MDS and ANNs

In Chapter 8, we explain in more detail how conceptual spaces can be obtained through multidimensional scaling, considering the collection of psychological data, different MDS algorithms, and ways of evaluating the quality of the resulting similarity spaces. Moreover, since both neural networks and multidimensional scaling allow us to obtain similarity spaces for poorly understood domains, we propose a hybrid procedure which combines these two approaches with each other. Essentially, we propose to initialize the similarity space by applying MDS to psychological dissimilarity ratings and to train a neural network afterwards on the task of mapping raw stimuli into this psychological

similarity space. This way, we combine the psychological grounding of MDS with the generalization capabilities of artificial neural networks. We report results of a first feasibility study in Chapter 9, where we apply our proposed approach to a dataset of novel objects. Afterwards, we conduct a psychological study on the domain of shapes in Chapter 10, and analyze the resulting shape spaces with respect to their alignment with the conceptual spaces theory in Chapter 11. Finally, we train different neural networks on the mapping task in Chapter 12.

1.3.4 Learning Conceptual Regions

Once the domains and dimensions of a conceptual space have been defined, learning regions in this space is very similar to classical machine learning problems. These can be broadly divided into two classes: *Supervised* approaches make use of labeled datasets where each training example is annotated with its correct classification. Their main target is to maximize classification accuracy which can be achieved by partitioning the feature space into multiple regions, each one belonging to one of the given classes. *Unsupervised* approaches on the other hand can also be applied to unlabeled datasets, where no classification information is given for the individual observations. Clustering algorithms typically sort the observations into multiple clusters by maximizing the similarity within clusters while minimizing the similarity between clusters.³ Just like in conceptual spaces, similarity is assumed to be inversely related to distance in the feature space. As each cluster of observations is just a set of points in the feature space, one can easily generalize from this set of points to a region (e.g., by computing their convex hull).

In both cases, the parameters of the respective model are usually estimated based on a large dataset of observations which is processed at once in a batch-like manner. Moreover, the parameters are optimized in such a way that they implicitly or explicitly optimize a given target function (e.g., minimizing the number of misclassifications).

While the task of learning concepts in a conceptual space is quite related to these machine learning problems, it has an additional constraint. Namely, the regions being learned should be convex (if one follows Gärdenfors' original approach) or star-shaped (following our own formalization of conceptual spaces). Moreover, as our research is seen in the context of cognitive AI, we are interested in cognitively plausible learning processes. On the one hand, we therefore favor incremental processes over batch-learning. On the other hand, we are interested in scenarios where only scarce or indirect feedback about the correct classification is available.

Concept learning as a machine learning problem

Properties of typical machine learning approaches

Constraints on the learning procedure

³ Again, we would like to point out the strong connection to the basic level of categorization [298, Chapter 7] (cf. Section 1.1.2) and the well-formedness criterion proposed by Douven and Gärdenfors [137] (cf. Section 1.2.1)

Making machine learning more cognitively plausible

In Chapter 7, we focus on several problems and approaches especially relevant to our envisioned application of conceptual spaces in artificial intelligence. These include knowledge-based constraints, concept formation, and language games.

Learning with knowledge-based constraints

Standard machine learning techniques learn their class boundaries from a blank slate, i.e., in a purely inductive manner. The incorporation of additional *knowledge-based constraints*, e.g., in the form of background knowledge from the symbolic layer, can be interesting in the context of conceptual spaces, since it provides a way to account for the knowledge view on concepts. We focus our discussion of this idea on the framework of *logic tensor networks* (LTNs) [21, 358] which combines supervised machine learning with logical constraints in the form of rules. We sketch a path towards conceptual logic tensor networks, i.e., a fruitful combination of LTNs and conceptual spaces.

Learning in an incremental and unsupervised way

While most approaches in machine learning (including LTNs) rely on a batch-processing of labeled datasets, *concept formation* [164] aims at incrementally creating a meaningful hierarchical categorization based on unlabeled observations. Concept formation implicitly solves the symbol grounding problem through a bottom-up process, since the emerging categories (symbols) are generalizations of actual observations. We will introduce COBWEB [159], ART [90], and SUSTAIN [270] as an important examples of concept formation algorithms.

Learning through communication

Finally, since concepts need to be shared among a population of agents in order to enable efficient communication, we will furthermore discuss *language games* [389] as an interesting avenue of research: In this approach, a group of artificial agents is simulated which repeatedly interact with each other in a given common environment. Each agent has its own conceptualization of the world as well as a dictionary mapping its concepts to words. Both the conceptualization and the dictionary are updated based on the communicative success in the simulated interactions, where typically one agent "describes" a target object in a jointly observed scene, while the other agent needs to correctly identify this target object based on the utterance. Through the indirect feedback from these interactions, the population of agents can converge to a common conceptualization of the world. While this is certainly interesting from the viewpoint of language evolution, it can also be interpreted as an incremental machine learning process which uses only indirect feedback.

Outlook

Although this dissertation does not provide any crucial new insights into cognitively plausible concept learning (neither in a theoretical nor in an experimental way), it gives a good overview of several relevant topics. This overview can then serve as a useful starting point for future research endeavors.

Part I

FORMALIZING CONCEPTUAL SPACES

In the first part of this dissertation, we present our thorough mathematical formalization of the conceptual spaces framework, which is able to encode correlations between domains in a geometric way. In Chapter 2, we define concepts as fuzzy star-shaped regions, before introducing various operations for creating novel concepts based on existing ones in Chapter 3. Finally, in Chapter 4, we provide several ways for quantifying relations between concepts.

2 | DEFINING CONCEPTUAL REGIONS

2.1	The Original Conceptual Spaces Framework	54
2.1.1	The Structure of a Conceptual Space	54
2.1.2	Conceptual Regions	56
2.2	An Argument Against Convexity	60
2.2.1	Convex Sets Cannot Encode Cross-Domain Correlations	60
2.2.2	Potential Solutions	62
2.3	A Parametric Definition of Concepts	65
2.3.1	Formalizing Conceptual Similarity Spaces	66
2.3.2	Crisp Conceptual Regions	67
2.3.3	Fuzzy Conceptual Regions	69
2.3.4	Implementation and Example: Fruit Space	74
2.4	Related Work	79
2.4.1	Related General Ideas	79
2.4.2	Prior Formalizations of Conceptual Spaces	82
2.4.3	Composite Concepts	86
2.5	Detailed Comparison to Other Formalizations	88
2.5.1	Comparison to Adams and Raubal	91
2.5.2	Comparison to Rickard	93
2.5.3	Comparison to Lewis and Lawry	95
2.6	Summary	98

In this chapter, we lay the groundwork for our mathematical formalization of the conceptual spaces framework and its open source implementation. The research presented in this chapter has been previously published in [41, 42, 46].

We begin by presenting more details about the representational aspects of the conceptual spaces framework in Section 2.1. In Section 2.2, we then argue that Gärdenfors’ convexity requirement prohibits a geometric representation of correlations between domains. We present different solution approaches and argue that a relaxation of the convexity criterion is the most promising choice. Based on this insight, we provide a parametric definition of concepts as fuzzy star-shaped regions in Section 2.3. This forms the basis of our mathematical formalization of the overall framework. In Section 2.4, we provide an overview of relevant literature and show how it relates to our work. We then give a more thorough comparison of our proposed formalization to three prior formalization proposals in Section 2.5. Finally, in Section 2.6, we summarize the contributions of our work and sketch possible directions for future research.

Overview

2.1 THE ORIGINAL CONCEPTUAL SPACES FRAMEWORK

This section is largely based on the book "Conceptual Spaces: The Geometry of Thought" by Peter Gärdenfors [179], where the cognitive framework of conceptual spaces is described in great detail. We provide here only a short summary of the formal representational aspects. We first consider the overall structure of the similarity space in Section 2.1.1 before discussing conceptual regions in Section 2.1.2.

2.1.1 The Structure of a Conceptual Space

Quality dimensions

Please recall from Section 1.2.1 that a conceptual space is a similarity space spanned by so-called *quality dimensions*. Each of these dimensions represents an interpretable and cognitively meaningful way of judging the similarity of two given observations. Examples for quality dimensions include TEMPERATURE, WEIGHT, TIME, PITCH, and HUE. We assume that each of these dimensions is equipped with a distance measure such that we can compute the semantic difference of two stimuli with respect to this dimension. In machine learning terms, each of the dimensions represents an interpretable feature, and together they span a meaningful feature space.

Integral dimensions and separable dimensions

A pair of dimensions can be either called integral or separable. *Integral dimensions* are dimensions that inherently belong together: If one assigns a value to one of the dimensions, one also has to assign a value to the other dimension as well – they are perceived holistically. This also means that focusing on only one of the dimensions while ignoring the other is difficult. If a pair of dimensions is not integral, it is called *separable*. In this case, one can selectively attend to one of the dimensions – the two dimensions are perceived as being independent from each other. For instance, the dimensions of PITCH and VOLUME are integral, because they cannot be perceived separately from each other. On the other hand, the dimensions of PITCH and BRIGHTNESS are separable, as one can perceive PITCH without perceiving BRIGHTNESS (e.g., when hearing a bell ringing while having one's eyes closed).

Domains

This distinction between integral and separable dimensions is used to introduce the notion of a domain: A domain is a set of integral dimensions that are separable from all other dimensions. Each perceptual modality (like COLOR, SHAPE, SOUND, OR TASTE) is represented by such a domain. In other words, the dimensions of one domain jointly describe one meaningful aspect of the world. The COLOR domain for instance consists of the three dimensions HUE, SATURATION, and BRIGHTNESS. Grouping dimensions into domains also provides a logical structure for the overall conceptual space – something that is typically absent from the feature spaces used in machine learning (cf. Section 1.2.5).

The weighted Minkowski metric

When measuring the overall distance between two points within the conceptual space, one needs to aggregate the dimension-wise distances.

Gärdenfors advocates the use of the *weighted Minkowski metric* (with $r > 0$) for this purpose:

Definition 2.1 (Weighted Minkowski Metric)

Let $x, y \in CS$. Their distance according to the *weighted Minkowski metric* of order $r > 0$ with weights w_i is defined as follows:

$$d_r(x, y) = \left(\sum_{i=1}^n w_i \cdot |x_i - y_i|^r \right)^{\frac{1}{r}}$$

The fixed positive weights w_i represent the importance of the respective dimensions in the current context. A large w_i corresponds to an important dimension whereas a small w_i indicates a less important dimension. Both the *Euclidean metric* d_E and the *Manhattan metric* d_M are special cases of d_r (for $r = 2$ and $r = 1$, respectively).

Euclidean metric and Manhattan metric

$$d_E(x, y) = d_2(x, y) = \sqrt{\sum_{i=1}^n w_i \cdot |x_i - y_i|^2}$$

$$d_M(x, y) = d_1(x, y) = \sum_{i=1}^n w_i \cdot |x_i - y_i|$$

Gärdenfors argues based on psychological evidence [19, 211, 367, 368] that integral dimensions should be combined with the weighted Euclidean metric, whereas separable dimensions should be combined with the Manhattan metric. Intuitively, the (weighted) Manhattan metric is just a (weighted) sum of the individual distances and is thus well suited for combining separable dimensions which can be perceived separately. The (weighted) Euclidean metric on the other hand cannot be conceptualized as a (weighted) sum of distances and thus can be thought of as computing a more holistic distance. This seems to be a better fit for integral dimensions which are perceived holistically.

Euclidean distance within domains, Manhattan distance between domains

Please note that the usage of $r = 2$ for integral dimensions and of $r = 1$ for separable dimensions is of course a simplification of the actual results obtained in psychological studies. Usually a value of $r \in (1, 2)$ is found to best reflect the similarity judgments made by humans [367]. However, one typically observes that the optimal value of r for separable dimensions is much smaller than the optimal value of r for integral dimensions. In order to get a more faithful approximation of the psychological results, one could also allow arbitrary values of r for combining dimensions. However, then one would need to specify this value r for each possible combination of dimensions in the conceptual space, increasing the number of parameters in the overall model dramatically. We therefore follow Gärdenfors' broad distinction by using $r = 2$ for all sets of integral dimensions and $r = 1$ for all sets of separable dimensions.

Reality may be more complex

A central aspect of the conceptual spaces framework is the notion of *semantic similarity*. The similarity of two points in a conceptual space is inversely related to their distance within this space: The smaller the distance between two points in the conceptual space, the larger

Semantic similarity

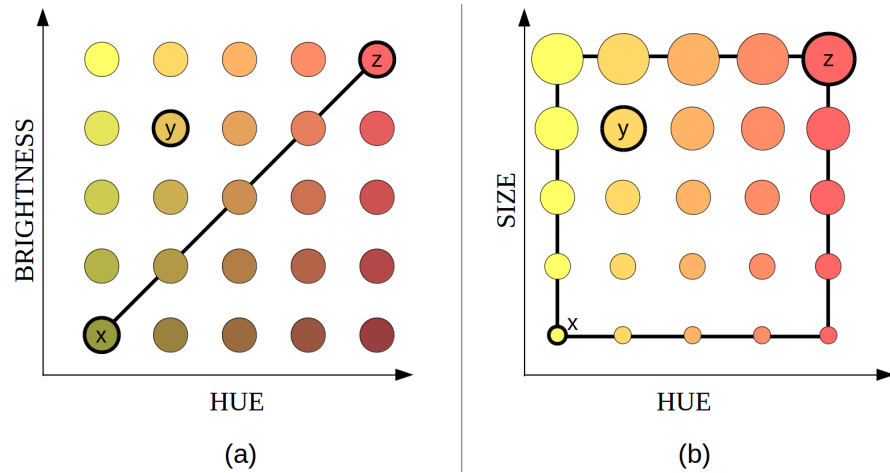


Figure 2.1: (a) Euclidean betweenness for stimuli on integral dimensions. (b) Manhattan betweenness for stimuli on separable dimensions.

the semantic similarity of the observations they represent. In order to formalize the connection between distance and similarity, Gärdénfors proposes to use an exponentially decaying function:

$$Sim(x, y) = e^{-c \cdot d(x, y)}$$

*Sensitivity parameter
and psychological
evidence*

The parameter $c > 0$ is a general *sensitivity parameter* and controls the rate of the exponential decay. Shepard [368] has argued that this exponential relation between similarity and distance can be found for a wide variety of stimulus sets and that it is not only confined to human perception but also applies to other species such as pigeons.

*Relation to artificial
intelligence*

This geometrical and intuitive definition of semantic similarity is a key feature of the conceptual spaces framework that makes it an interesting complement for traditional logic-based AI approaches which are incapable of representing such similarities. The definition of similarity as inverse distance is also in line with the feature spaces in machine learning, where this assumption is used for instance in clustering algorithms (see Chapter 7).

2.1.2 Conceptual Regions

*Geometrical
betweenness*

Gärdénfors bases the definition of conceptual regions on the notion of *geometrical betweenness*. Betweenness is a logical predicate $B(x, y, z)$ that is true if and only if y is geometrically between x and z . It can be defined based on a given metric d :

Definition 2.2 (Betweenness)

Let $x, y, z \in CS$ and d be a metric on CS . The point y is said to lie between x and z (denoted as $B_d(x, y, z)$) if and only if $d(x, y) + d(y, z) = d(x, z)$.

*Betweenness under
the Euclidean
distance*

A point y is considered to be between two other points x and z , if the path from x through y to z is not longer than the direct path from x to z . The Euclidean metric measures the distance of two points as the length of the straight line segment connecting them. One can easily

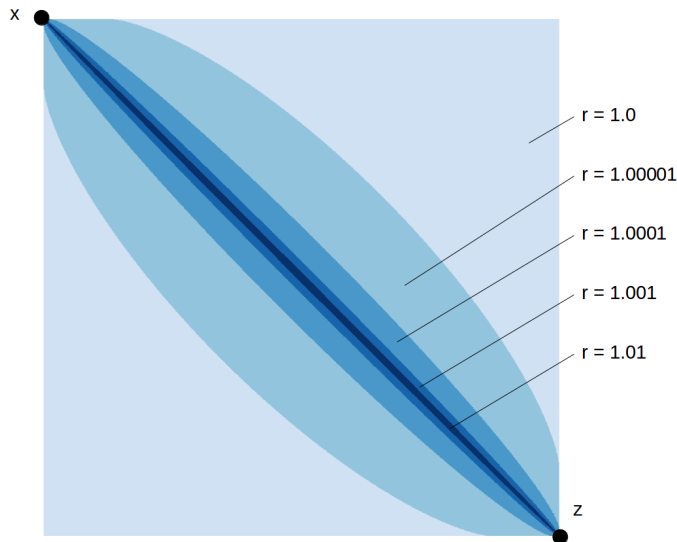


Figure 2.2: Betweenness under the Minkowski metric for different values of r .

see that $B_{d_E}(x, y, z)$ is true if and only if y lies on the straight line segment connecting x and z . Figure 2.1a shows stimuli which differ on the integral dimensions of HUE and BRIGHTNESS. The stimulus labeled y can therefore not be considered to lie between x and z .

The Manhattan metric on the other hand measures the distance of two points by simply adding up the distances with respect to the individual dimensions. This also means that all points in the bounding box spanned by x and z can be considered to lie between x and z . Figure 2.1b shows stimuli that differ with respect to two separable dimensions, namely SIZE and HUE. As SIZE and HUE are separable dimensions (i.e., they belong to different domains), they should be combined with the Manhattan distance. In this example, the stimulus y can therefore be treated as an intermediate case between x and z .







As we can see in Figure 2.1, betweenness corresponds to a linear interpolation between the two endpoints in both cases. In the Euclidean case, this interpolation takes place in the overall space, i.e., considering all dimension jointly. In the Manhattan case, the interpolation takes place for each dimension individually. This corresponds to the intuitive notion that one can attend to separable dimensions individually while integral dimensions are always processed jointly.

As one can see in Figure 2.2, intermediate values of $r \in (1, 2)$ result in intermediate versions of betweenness that lie between the Euclidean ($r = 2$) and Manhattan ($r = 1$) cases. Interestingly, although $r = 1.01$ is quite close to $r = 1$, the betweenness relation for $r = 1.01$ is much more similar to the one for $r = 2$ than to the one for $r = 1$. We can also see in Figure 2.2 that there exists a certain subset relation for the betweenness relations for different values of r : If a point y is between points x and z for a Minkowski metric with $r \in [1, 2]$, then y is also between x and z for any $r' \in [1, r)$. In the following, we focus our discussion on the cases of $r = 1$ and $r = 2$, which are much easier to model than $r \in (1, 2)$.

*Betweenness under
the Manhattan
distance*

*Betweenness as
interpolation*

*Intermediate notions
of betweenness*

Euclidean betweenness	Convex			
	Star-shaped, but not convex	does not exist		
	Not star-shaped	does not exist	does not exist	
		Convex	Star-shaped, but not convex	Not star-shaped

Manhattan betweenness

Figure 2.3: Illustration of convexity and star-shapedness as based on the Euclidean vs. the Manhattan metric.

Convexity and star-shapedness

Based on the betweenness relation, we can now define two criteria that can be used to constrain conceptual regions in a meaningful way, namely *convexity* and *star-shapedness*.

Definition 2.3 (Convexity)

A set $C \subseteq CS$ in a conceptual space CS is convex under a metric $d \Leftrightarrow$

$$\forall x \in C, z \in C, y \in CS : (B_d(x, y, z) \rightarrow y \in C)$$

Definition 2.4 (Star-Shapedness)

A set $S \in CS$ in a conceptual space CS is star-shaped under a metric d with respect to a set $P \subseteq S \Leftrightarrow$

$$\forall p \in P, z \in S, y \in CS : (B_d(p, y, z) \rightarrow y \in S)$$

Interpretation of the definitions

If we require a conceptual region C to be convex, we require that all interpolations between two members $x, z \in C$ of the concept also belong to this concept. If we only require a conceptual region S to be star-shaped, then we only require interpolations between any prototypical example $p \in P$ and any other concept member $z \in S$ to also belong to this concept. As one can easily see, convexity is a special case of star-shapedness (where $P = S$) and is thus a stronger constraint on the conceptual region.

Convexity and star-shapedness depend on the distance metric

As both the notions of convexity and star-shapedness are based on the notion of betweenness (which is in turn based on the metric being used), we can also distinguish Euclidean versions of convexity and star-shapedness from their Manhattan-based counterparts. Figure 2.3 illustrates the differences between these notions with some examples. We would like to point out that only axis-parallel cuboids are convex under the Manhattan metric. Moreover, since Euclidean betweenness is a special case of Manhattan betweenness (cf. Figure 2.2), all sets that are convex (star-shaped) in the Manhattan sense are also convex (star-shaped) in the Euclidean sense: Consider any two points $x, z \in C$.

If C is convex under d_M , then all points y with $B_{d_M}(x, y, z)$ must also belong to C . As we have seen in Figure 2.2, $B_{d_r}(x, y, z) \Rightarrow B_{d_M}(x, y, z)$ for $r \in [1, 2]$. If we include all y with $B_{d_M}(x, y, z)$ in C , then we also include all points for which the special case $B_{d_r}(x, y, z)$ holds true.

It is a traditional assumption in cognitive science that any observation or concept can be described by a combination of different properties (e.g., Minsky's frames [294] or the schema-based variant of prototype theory, cf. Section 1.1.2). For instance, an APPLE could be described as being RED, ROUND, and SWEET. Gärdenfors' hypothesis is that most of these properties can be described as geometric regions within a single conceptual domain. In our example, the property RED can be described as a region in the COLOR domain, whereas ROUND and SWEET belong to the domains of SHAPE and TASTE, respectively. Gärdenfors furthermore argues that these regions are expected to be convex – if two points in the COLOR space are classified as RED, then any other point between them should also be classified as RED. As domains consist of integral dimensions, we apply the Euclidean sense of convexity. This leads to Gärdenfors' definition of a *property*:

Definition 2.5 (Criterion P)

A natural property is a convex region of a domain in a conceptual space.

As stated above, *concepts* can be expressed as a combination of properties from different domains. However, not all of these properties might be of equal importance to a given concept. For instance, the SMELL of an apple is less prominent than its COLOR and SHAPE. This can be reflected by so-called *salience weights*: Every domain that is part of the respective concept has an associated weight. Larger weights indicate important domains, whereas smaller weights indicate less crucial domains. These salience weights can of course be influenced by the current context. For instance, when eating an apple, the TASTE domain becomes much more prominent than the SHAPE domain.

Moreover, the different properties of a concept are not completely independent of each other: A RED APPLE tends to be SWEET, while a GREEN APPLE tends to be SOUR. These correlations between different domains are an important aspect of concepts [288]. Recall from Section 1.1.2 that systems of such correlations can significantly aid learning processes [66] and play an important role in reasoning [298, Chapter 8]. For instance, if we know that a given BANANA is GREEN, we can use the correlation between the COLOR and the TASTE domain to infer that it is probably not SWEET. From a linguistic perspective, these cross-domain correlations can be indicated by natural language patterns such as "The dish was excellent but not expensive": They involve properties from different domains and state that a certain expected correlation between these domains (in this example between the quality and the price of a dish) have been violated [455].

Gärdenfors uses these three components (properties, weights, and correlations) to define a concept as follows:

Properties as convex sets under the Euclidean metric

Concepts and salience weights

Correlations between domains

Definition of concepts

Definition 2.6 (Criterion C)

A natural concept is represented as a set of convex regions in a number of domains together with an assignment of salience weights to the domains and information about how the regions in different domains are correlated.

2.2 AN ARGUMENT AGAINST CONVEXITY

Overview

Based on the notion of convexity under the Manhattan distance, we encounter a problem when trying to encode cross-domain correlations through the form of the conceptual region. We introduce this problem in Section 2.2.1, before discussing several potential approaches towards solving it in Section 2.2.2.

2.2.1 Convex Sets Cannot Encode Cross-Domain Correlations

*Concepts as regions
in the overall
conceptual space*

In the following, we will assume that concepts can be represented as regions in the overall conceptual space. This assumption is never mentioned in the original work by Gärdenfors. However, we think that it is quite intuitive from the perspective of knowledge representation and machine learning: Properties are regions within particular domains. If concepts are represented by regions in the overall space, then they can be represented with the same formalism as properties. This in turn enables us to use the same learning and reasoning mechanisms for both properties and concepts. The only difference between concepts and properties is then the space on which they are defined (i.e., the overall space or a single domain).¹

*Representing
correlations in a
geometric way*

Gärdenfors [179] argues for the importance of cross-domain correlations, but he does not propose any concrete way for representing them. If we assume that concepts are represented as regions in the overall space, it is natural to represent the correlations between different domains in a geometric way. Consider the sketch in Figure 2.4a. In this example, we consider two dimensions, AGE and HEIGHT, in order to define the concepts of CHILD and ADULT. We expect a strong correlation between AGE and HEIGHT for children, but no such correlation for adults. An intuitive sketch of these concepts results in the two elliptical regions shown in Figure 2.4a.² As one can see, the values of AGE and HEIGHT constrain each other: For instance, if the value on the AGE dimension is low and the point lies in the CHILD region, then also the value on the HEIGHT dimension must be low.

*A representation
with convex regions*

Please note that AGE and HEIGHT are separable dimensions: They can be perceived independently of each other, unlike e.g., HUE and SATURATION. We should therefore combine them by using the Manhattan metric. As argued in Section 2.1.2, convex regions under the Manhattan metric are axis-parallel cuboids. If we assume conceptual regions to

¹ Thanks to Peter Brüssel for challenging me to explain this (previously tacit) assumption.

² Please note that this is a very simplified, artificial example to illustrate our main point.

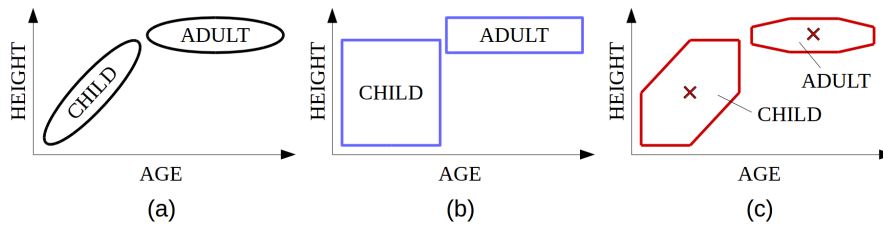


Figure 2.4: (a) Intuitive way to define regions for the concepts of *ADULT* and *CHILD* in a conceptual space spanned by the dimensions *AGE* and *HEIGHT*. (b) Representation of these concepts as convex regions. (c) Representation of these concepts as star-shaped regions.

be convex (as proposed by Gärdenfors), we have to represent the two concepts by axis-parallel cuboids. This is illustrated in Figure 2.4b.

In our point of view, this representation is, however, highly problematic: All information about the correlation of *AGE* and *HEIGHT* in the *CHILD* concept is lost in this representation – the values of *AGE* and *HEIGHT* do not constrain each other at all. According to the convex set representation, a *CHILD* of age 2 with a height of 1.80 m would be totally conceivable – which does not make any intuitive sense. This example illustrates that we cannot geometrically represent correlations between domains if we assume that concepts are convex and that the Manhattan metric is used. We think that our example is not a pathological one and that similar problems will occur quite frequently when representing concepts. For instance, there is an obvious correlation between a *BANANA*'S *COLOR* and its *TASTE*. If one replaces the *AGE* dimension with *HUE* and the *HEIGHT* dimension with *SWEETNESS* in Figure 2.4, one can observe similar encoding problems for the *BANANA* concept as for the *CHILD* concept.

Manhattan convexity cannot express correlations

Also Hernández-Conde has argued against the convexity constraint in conceptual spaces, although from a different point of view [195]. His arguments are mainly of a theoretical and indirect nature, essentially criticizing that there is currently no sufficient support for the claim that conceptual regions are convex. Moreover, his criticism is at least partially based on misunderstandings of the conceptual spaces theory.³ In contrast to that, our example highlights a representational deficit of the original theory from the perspective of artificial intelligence and machine learning: The most natural way of representing cross-domain correlations in a geometric setting is not possible when using convex sets under the Manhattan distance.

Other criticisms of convexity

³ For example, he argues that some objects (such as apples) have a non-convex shape, hence the region of the *APPLE* concept in the *SHAPE* domain cannot be convex. This argument, however, misses the main point of the quality dimensions used to span the *SHAPE* space: They do not correspond to physical dimensions, but to meaningful shape-related features of an observation, such as its aspect ratio or degree of curvature (see Chapter 10 for more information on how the *SHAPE* domain can be represented).

2.2.2 Potential Solutions

*First option:
replacing convexity
with star-shapedness*

One potential remedy for our problem is the notion of star-shapedness: If we require conceptual regions only to be star-shaped instead of convex, we can represent the correlation of AGE and HEIGHT for the CHILD concept in a geometric way. Figure 2.4c illustrates this approach: The sketched sets are star-shaped under the Manhattan metric with respect to a central point, illustrated as a cross in the center of the conceptual region.⁴ Although this representation does not exactly correspond to our intuitive sketch in Figure 2.4a, it is a clear improvement over the convexity-based representation from Figure 2.4b. The weaker requirement of star-shapedness allows us to "cut out" some corners from the rectangular region. This enables us to geometrically represent correlations through the form of the conceptual region. Therefore, by relaxing the convexity requirement, a geometric representation of cross-domain correlations becomes feasible.

*Connected regions
are too unconstrained*

One could argue at this point that instead of star-shapedness, we should consider an even weaker constraint, for instance *connectedness*. A set C is connected if for all points $x, z \in C$, we can find an arbitrarily shaped path from x to z such that all points on this path also lie in C .⁵ If the regions representing concepts only have to be connected, then we can define the concepts of ADULT and CHILD as in Figure 2.4a – which would be even closer to our intuition. However, the notion of connectedness is relatively weak and permits e.g., also ring-shaped sets that have a "hole" in the middle. This, however, does not seem convincing for representing concepts. Star-shapedness prevents such "holes". Star-shaped regions furthermore have a well defined central point or region that can be interpreted as a prototype. Thus, the connection that Gärdenfors [179] established between the prototype theory of concepts and the framework of conceptual spaces (cf. Section 1.2.2) is preserved. Replacing convexity with star-shapedness is therefore only a minimal departure from the original framework while enabling us to geometrically represent correlations between domains.

*Second option: using
only the Euclidean
distance*

The problem illustrated in Figure 2.4 could also be resolved by replacing the Manhattan metric with a different distance function for combining domains. A natural choice would be to use the Euclidean distance everywhere.⁶ We think, however, that this would be a major departure from the original framework.

⁴ Please note that although the sketched sets are still convex under the Euclidean metric, they are star-shaped but not convex under the Manhattan metric. Please also note that the region for the ADULT concept in Figure 2.4a is star-shaped under the Manhattan distance. Thanks to Luciano Serafini for pointing out the latter observation.

⁵ Please note that all convex sets are connected (in this case the path between x and z needs to be a shortest path) and that also all star-shaped sets are connected (here, we fix an intermediate point $p \in P$, which must lie on the path from x to z and require the subpaths from x to p and from p to z to be shortest paths).

⁶ One could of course also replace the Manhattan metric with some non-Euclidean metric (e.g., the Mahalanobis distance). However, there is currently no strong evidence supporting the usage of any particular other metric.

In conceptual spaces, the distance between two points in the overall space does not only depend on their coordinates with respect to the individual dimensions, but also on the way that these dimensions are grouped into domains: The overall distance in the space is computed using the (weighted) Manhattan metric on the (weighted) Euclidean distances within the respective domains. If we replace the Manhattan distance with the Euclidean distance, then we get a (weighted) Euclidean distance based on (weighted) Euclidean distances. This is, however, equivalent to computing a single weighted Euclidean distance on the overall space (see Lemma A.1 in Appendix A.1). In this case, the structure of the conceptual space loses its importance for computing distances. It is therefore not quite clear whether using domains to structure the conceptual space would be useful any more – it seems that the framework would lose one of its central parts.

Moreover, there exists some psychological evidence [19, 211, 367, 368] which indicates that human dissimilarity ratings are reflected better by the Manhattan metric than by the Euclidean metric if separable dimensions are involved (e.g., stimuli differing in both SIZE and BRIGHTNESS, cf. also Figure 2.1). Since a psychologically plausible representation of similarity is one of the core principles of the conceptual spaces framework, these findings should be taken into account. Furthermore, the Manhattan metric provides a better relative contrast between close and distant points in high-dimensional spaces than the Euclidean metric [7]. If we expect a large number of domains (each consisting only of a small number of dimensions), this also supports the usage of the Manhattan metric from a computational point of view.

Also multivariate Gaussians might seem to be a good candidate for representing concepts that could solve our representational problem: They contain both a prototypical element (the mean μ) and information about correlations (the positive semi-definite covariance matrix Σ). The membership function of a multivariate Gaussian can be expressed as a specific type of similarity to the mean μ :

$$Sim(x, \mu) = e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)}$$

Mathematically speaking, multivariate Gaussians compute an exponentially decaying similarity to a prototype based on the squared Mahalanobis distance $d_S(x, y) = \sqrt{(x-y)^T \Sigma^{-1}(x-y)}$. The Mahalanobis distance, however, corresponds to the Euclidean metric in a transformed space: Since the covariance matrix Σ is positive semi-definite, one can write $\Sigma^{-1} = G^T G$ for some matrix G [167, 236]. We can now make the following transformations:

$$\begin{aligned} d_S(x, y) &= \sqrt{(x-y)^T \Sigma^{-1}(x-y)} = \sqrt{(x-y)^T (G^T G)(x-y)} \\ &= \sqrt{(G(x-y))^T (G(x-y))} = \sqrt{(Gx - Gy)^T (Gx - Gy)} \end{aligned}$$

One can easily see that $d_S(x, y) = d_E(Gx, Gy)$. Overall, we thus get that $Sim(x, \mu) = e^{-\frac{1}{2} \cdot d_E(Gx, G\mu)^2}$. Both the implicit transformation of the

A representational argument for keeping the Manhattan distance

Psychological and computational arguments for keeping the Manhattan distance

Third option: representing conceptual regions as multivariate Gaussians

The Mahalanobis distance and conceptual spaces

Squared distance in a transformed space

similarity space and the usage of squared distances are in our opinion, however, not in line with the original conceptual spaces framework.

*Fourth option:
considering
intermediate cases of
the Minkowski
metric*

Our representational problem also disappears if we assume the two dimensions of age and height are not completely separable.⁷ So instead of using the Manhattan metric (where $r = 1$), we could use the Minkowski metric with $r > 1$. In Section 2.1.1, we have already noted that values of $r \in (1, 2)$ are the usual case in psychological analyses. As we have seen in Figure 2.2, even a value of $r = 1.01$ already leads to a betweenness relation that is much more similar to the Euclidean case than to the Manhattan case. When replacing $r = 1$ with $r > 1$, we can either use a different r for each pair of separable dimensions (which adds many free parameters to the framework, cf. Section 2.1.1) or use a single global value $r > 1$. The latter option seems to be preferable to the first one as it implies less additional complexity. However, even using the same $r > 1$ for all pairs of separable dimensions has a major disadvantage: While betweenness under both the Euclidean and the Manhattan metric can be described in a very concise way (as a straight line segment and as an axis-parallel cuboid, respectively), this is not easily possible for intermediate cases with $r \in (1, 2)$. The subsequent analysis of convexity and star-shapedness as well as the parametric description of conceptual regions might thus become quite complex. As we have seen in Figure 2.2, betweenness for $r \in (1, 2]$ is a special case of betweenness for $r = 1$. This means that any set that is star-shaped/convex for $r = 1$ is also star-shaped/convex for $r \in (1, 2]$. By basing our formalization on the Manhattan metric, we therefore take into account the limiting case of $r = 1$.

*Fifth option:
redefining
betweenness*

Finally, there is a potential solution that allows us to keep both the constraint of convexity and the usage of the Manhattan metric: If we define betweenness always in the Euclidean sense (i.e., as the straight line segment connecting two points), then the problem sketched in Section 2.2.1 does not arise any more, since the regions sketched in Figure 2.4a would already be considered convex under any distance metric.⁸ This approach, however, breaks the tight connection of betweenness and distance (and thus similarity) in a conceptual space and is therefore a quite strong modification of the original framework. By using Euclidean betweenness also for separable dimensions, we can no longer say that stimulus y in Figure 2.1b is between stimuli x and z . As we can see by this example, the redefinition of the betweenness relation also has direct effects on points in the conceptual space, i.e., on the way individual stimuli are treated. As mentioned already for several other candidate solutions, such side-effects are also to be expected when changing the underlying notion of distance in the space.

Summary

Overall, it seems that from a representational point of view, the

⁷ Thanks to Peter Gärdenfors for this helpful comment.

⁸ Thanks to the audience at the DCLPS colloquium in Düsseldorf for this idea.

relaxation of the convexity criterion is the cleanest solution – it causes only a small departure from the original framework with no unintended side effects or major complications.

Please note that the example given above is intended to highlight representational problems of conceptual spaces in the context of artificial intelligence, if a geometric representation of correlations is desired. We do not make any claims that star-shapedness is a psychologically plausible extension of the original framework and we do not know about any psychological data that could support such a claim.⁹ Moreover, our example does not preclude that properties in a conceptual space are convex sets within a single domain.¹⁰ The convexity requirement only caused problems for concepts which are defined across multiple domains, not for properties within a single domain. As all convex sets are also star-shaped, properties may be convex regions without contradicting our formalization – they would simply constitute a special case. For the sake of simplicity, we will use the same type of star-shaped sets for representing both concepts and properties in the remainder of this dissertation. Investigations about the psychological plausibility of our formalization would be certainly interesting, but lie outside of the scope of this dissertation.

Finally, one should mention that one can circumvent the representational problem described above by representing correlations between domains in a non-geometric way: Concepts can be represented by a list of individual properties instead of being represented by a region in the overall space (cf. frames and schema-based variants of prototype theory, Sections 1.1.1 and 1.2.1). Then, all regions are defined as convex regions within individual domains. Correlations between these domains can then be represented in a different way, e.g., as co-occurrence statistics of properties [329] (see also Sections 2.4.2 and 2.5.2). However, as already stated in Section 2.2.1, we think that a geometric representation of correlations in the overall space is more intuitive. Moreover, this approach allows us to treat properties and concepts in the same way. This is especially useful if we aim to learn the conceptual regions with machine learning algorithms.

Disclaimer: this is not about psychology

Non-geometric representation of correlations

2.3 A PARAMETRIC DEFINITION OF CONCEPTS

Based on our insights from Section 2.2, we now provide a formalization of conceptual similarity spaces (Section 2.3.1) and star-shaped conceptual regions both in a crisp and a fuzzy variant (Sections 2.3.2 and 2.3.3). We then introduce our open source implementation of this formalization in Section 2.3.4.

Overview

⁹ Thanks to Igor Douven for pointing this out to me.

¹⁰ Strößner [395] has recently argued, that convexity is empirically supported mostly for single-domain concepts, but not for multi-domain concepts, which may, however, be more crucial for human cognition.

2.3.1 Formalizing Conceptual Similarity Spaces

Formalizing
dimensions

Before we can derive a mathematical definition of conceptual regions, we first need to formalize the underlying conceptual space. Let D be the set of quality dimensions $d \in D$ that span the conceptual space. Then, the overall conceptual space CS can be defined as the product space of all these dimensions. We denote the distance between two points x and y with respect to a dimension d as $|x_d - y_d|$ (or, equivalently, $|y_d - x_d|$).¹¹

Formalizing domains

These dimensions are now grouped together into domains. Each domain δ is defined as subset of D . Taken together the set of all domains Δ is a partition of D (i.e., each dimension d belongs to exactly one domain δ). Distance within a domain $\delta \subseteq D$ is measured by the weighted Euclidean metric d_E^δ (cf. Definition 2.1):

$$d_E^\delta(x, y, W_\delta) = \sqrt{\sum_{d \in \delta} w_d \cdot |x_d - y_d|^2}$$

Dimension weights

The parameter W_δ contains positive weights $w_d > 0$ for all dimensions $d \in \delta$, representing their relative importance within the domain. These weights are assumed to sum to one, i.e., $\sum_{d \in \delta} w_d = 1$. This normalization requirement prevents that the number of dimensions in a given domain has a great impact on the numeric value of the distances. It is intended to make distances within small and large domains comparable.

Combined metric in
the overall space

Now let Δ be the set of all domains δ that are part of the conceptual space. Distance within the overall conceptual space is measured by the weighted Manhattan metric of the intra-domain distances (henceforth called the *combined metric*):

Definition 2.7 (Combined Metric)

Let CS be a conceptual space based on dimensions $d \in D$, which are partitioned into domains $D \supseteq \delta \in \Delta$. Let W_Δ be the set of positive domain weights w_δ for all $\delta \in \Delta$ with $\sum_{\delta \in \Delta} w_\delta = |\Delta|$. Let moreover W_δ be the set of positive domain weights w_d for all $d \in \delta$ with $\sum_{d \in \delta} w_d = 1$. Let furthermore $x, y \in CS$. Their distance according to the combined metric is defined as follows, where $W = \langle W_\Delta, \{W_\delta\}_{\delta \in \Delta} \rangle$:

$$d_C^\Delta(x, y, W) = \sum_{\delta \in \Delta} w_\delta \cdot d_E^\delta(x, y, W_\delta) = \sum_{\delta \in \Delta} \left(w_\delta \cdot \sqrt{\sum_{d \in \delta} w_d \cdot |x_d - y_d|^2} \right)$$

Domain weights

The parameter $W = \langle W_\Delta, \{W_\delta\}_{\delta \in \Delta} \rangle$ contains both the dimension weights as introduced above and a set of domain weights W_Δ . The weights in W are not globally constant, but they (and thus also the notion of distance) depend on the current context. Observations and

¹¹ For linear dimensions, this distance can be simply computed by taking the absolute value of the difference between x_d and y_d . If the underlying dimension is circular (e.g., representing a rotation angle in $[0^\circ, 360^\circ)$, the distance computation needs to take into account this circularity, such that for example the distance between 10° and 350° equals 20° rather than 340° .

concepts that are defined on a larger number of domains can have a larger psychological distance from each other than observations and concepts defined on a small number of domains. For example, points in the conceptual space can differ from the property RED only with respect to their COLOR. If we compare them, however, to the APPLE concept, then also differences with respect to SHAPE, TASTE, and SIZE matter. By requiring $\sum_{\delta \in \Delta} w_\delta = |\Delta|$ instead of $\sum_{\delta \in \Delta} w_\delta = 1$, we can enforce this intuition mathematically.

Lemma 2.1 (Metric Properties of d_C^Δ)
 d_C^Δ with a fixed parameter W is a metric.

Proof. See Appendix A.2. □

We call this overall metric d_C^Δ for the conceptual space the *combined metric*, because it is a combination of the Euclidean and the Manhattan metric. Note that both d_M and d_E are special cases of d_C^Δ : If every domain contains only a single dimension, then d_C^Δ is equivalent to d_M . On the other hand, if there is only a single domain which contains all dimensions, then d_C^Δ is equivalent to d_E .

Relation to Euclidean and Manhattan distance

2.3.2 Crisp Conceptual Regions

Our mathematical formalization of concepts is based on the following insight, which relates convex regions to star-shaped regions:

A union of intersecting convex regions is star-shaped

Lemma 2.2 (Union of Convex Sets)

Let C_1, \dots, C_m be convex sets under some metric d and let $P = \bigcap_{i=1}^m C_i$. If $P \neq \emptyset$, then $S = \bigcup_{i=1}^m C_i$ is star-shaped under d with respect to P .

Proof. See Appendix A.3. □

Lemma 2.2 states that every union of convex sets with a nonempty intersection is star-shaped with respect to this intersection. Moreover, one can also approximate any star-shaped set by a union of convex sets with a nonempty intersection (cf. [245, 378]):

The reverse direction

If the original set S is star-shaped with respect to the region P , each of the convex sets C_i to be constructed will be a superset of P . Define for each point x on the surface of S a minimal convex set C_x that contains both P and x . As S is star-shaped, all the points in C_x are also contained in S . The union over all C_x is thus a subset of S . Moreover, this union of all C_x covers all points in S : Every point $y \in S$ is either a surface point or lies between a surface point x and the central region P . We can thus for every $y \in S$ find a C_x such that $y \in C_x$. Thus, $\bigcup C_x = S$. Please note that in practical applications, one is limited to a finite number of convex sets C_x , hence only enabling an *approximation*, but not a perfect representation of S .

Approximating star-shaped regions as union of convex regions

If we ensure that our sets C_1, \dots, C_m from Lemma 2.2 are convex under the Manhattan distance d_M , we are guaranteed that they are also convex under d_r with $r \in [1, 2]$ (cf. Section 2.1.2). Then, the resulting set S is not only star-shaped under d_M , but also under d_r . Using convexity

Manhattan distance as limiting case

under d_M thus can be considered a limiting case (cf. Section 2.2.2) – if in practice the actual value of r for a given conceptual space lies in the open interval $(1, 2)$, a formalization based on convexity and star-shapedness under d_M is still applicable.

Axis-parallel cuboids
as building blocks

Please recall from Sections 2.1.2 and 2.2.1 that convex sets under the Manhattan distance correspond to axis-parallel cuboids. They can be formally defined in the following way:¹²

Definition 2.8 (Axis-Parallel Cuboid)

We describe an axis-parallel cuboid¹³ C as a triple $\langle \Delta_C, p^-, p^+ \rangle$. C is defined on the domains $\Delta_C \subseteq \Delta$, i.e., on the dimensions $D_C = \bigcup_{\delta \in \Delta_C} \delta$. We call p^-, p^+ the support points of C and require that $p^+, p^- \in D_C$. This means that p^+ and p^- have an entry p_d^+, p_d^- for each dimension $d \in D_C$. For all $d \in D \setminus D_C$, we can for convenience assume that $p_d^+ = +\infty$ and $p_d^- = -\infty$. If for any $d \in D_C$ we have $p_d^- > p_d^+$, then the cuboid C is empty, i.e., $C = \emptyset$.

Then, we define the cuboid C in the following way:

$$C = \{x \in CS \mid \forall d \in D : p_d^- \leq x_d \leq p_d^+\}$$

Interpretation

Intuitively, the interval $[p_d^-, p_d^+]$ describes the set of admissible values for a given dimension d . A cuboid C thus restricts the values on the dimensions in Δ_C into intervals specified by p^- and p^+ .

Lemma 2.3 (Cuboids are Convex)

Cuboids are convex
under d_C

A cuboid C is convex under $d_C^{\Delta_C}$, given a fixed set of weights W .

Proof. See Appendix A.3. □

Since d_M and d_E are special cases of d_C , cuboids are also convex under both d_M and d_E .

Conceptual regions
as union of
intersecting cuboids

By coming Lemma 2.2 with Lemma 2.3, we can see that any union of intersecting cuboids is star-shaped under d_C^{Δ} . We use this insight to define crisp *cores* for our conceptual regions (see Figure 2.5 for an illustrative example):

Definition 2.9 (Core of a Conceptual Region)

We describe a core S as a tuple $\langle \Delta_S, \{C_1, \dots, C_m\} \rangle$ where $\Delta_S \subseteq \Delta$ is a set of domains on which the cuboids $\{C_1, \dots, C_m\}$ (and thus also S) are defined. Again, we can derive $D_S = \bigcup_{\delta \in \Delta_S} \delta$, i.e., the set of all dimensions on which S is defined. We further require that the central region $P = \bigcap_{i=1}^m C_i \neq \emptyset$. Then the core S is defined as follows:

$$S = \bigcup_{i=1}^m C_i$$

How many cuboids
per core?

We have defined a core S as a union of a (potentially very large) number of cuboids with nonempty intersection. In an actual implementation, however, we might need to restrict the number of cuboids m in order to

¹² All of our definitions and propositions hold for any number of dimensions.

¹³ We will drop the modifier "axis-parallel" from now on.

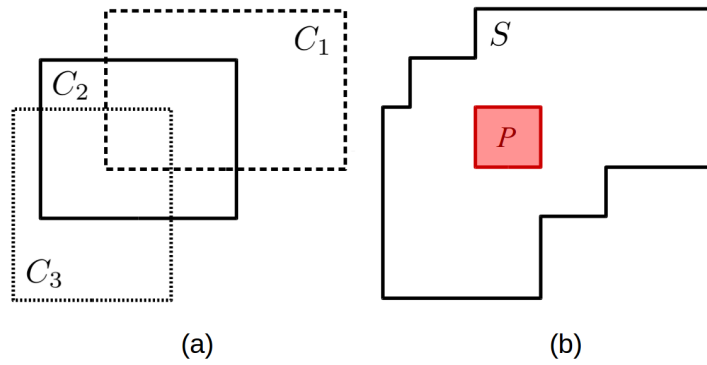


Figure 2.5: Illustration of a core. (a) Three cuboids C_1, C_2, C_3 with nonempty intersection. (b) Core S based on C_1, C_2, C_3 with central region P .

limit both computation time and memory requirements. One could for instance base this on the number of domains or dimensions on which S is defined, i.e., $m \propto |\Delta_S|$ or $m \propto |D_S|$. As a potential restriction on the number of cuboids does not influence our mathematical definitions, we leave this decision to the concrete application. We will, however, revisit this point in Sections 3.1.3, 4.1.3, and 4.5.3, when considering the runtime of different operations in dependence on the number of cuboids used to represent each concept.

Since cores are defined as a union of cuboids, we can interpret them as consisting of several sub-concepts (the individual cuboids) which are defined based on ranges of possible feature values. Requiring a non-empty intersection of the cuboids thus means that these ranges of admissible values need to overlap for each dimension: There must be at least one observation that falls into the range of possible values for all cuboids. A correlation between domains and dimensions is also expressed through the cuboid structure: The ranges of admissible feature values may co-vary, such that certain combinations of sub-ranges exist while others do not.

*Cuboids as
sub-concepts*

2.3.3 Fuzzy Conceptual Regions

In practice, it is often not possible to define clear-cut boundaries for conceptual regions. As Murphy [298, Chapter 2] argues, clear-cut conceptual boundaries are only useful if the world has "distinct clumps of objects" (cf. Section 1.2.3). However, if the world consists of shadings and if objects can have a rich mixture of different kind of properties, then one will encounter borderline cases with no clear concept membership. Figure 2.6a shows a simple example from the `COLOR` domain for the property `RED`. No matter where the crisp conceptual boundary is put, it is always possible to find two points $x \neq x'$ that are very close to each other perceptually, but which lie on different sides of the boundary. They represent borderline cases that lie close to the decision boundary. One can then argue that these two specific colors are very similar to

*Crisp set cannot deal
well with borderline
cases*

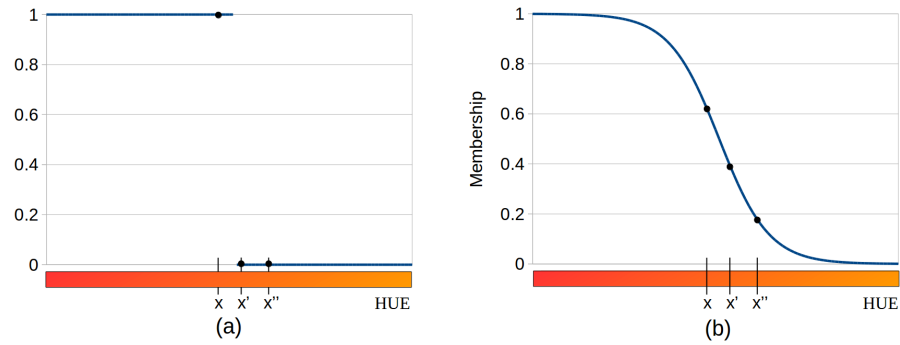


Figure 2.6: (a) Crisp conceptual boundaries lead to abrupt changes in membership. (b) Fuzzy conceptual boundaries allow for smooth changes in membership.

each other (as $d_C^{\Delta C}(x, x', W)$ is very small) and that the decision to call x RED and x' NOT RED is quite arbitrary. If we decide to solve this problem by moving the conceptual boundary in such a way that both x and x' belong to the conceptual region of RED, then we immediately face a similar problem with respect to x' and x'' .

Crisp membership is not continuous

The underlying problem is that the membership to the property RED is crisp (i.e., a given point either is a full member of the conceptual region, or it is not a member at all). At the conceptual boundary, we thus have a sudden jump from full membership to no membership at all. The membership function which maps points in the conceptual space to their membership to the property RED is therefore not continuous: A small change in the input (i.e., the coordinates of the point) can lead to a large change in the output (i.e., membership to the property RED).

Fuzzy sets use continuous degrees of membership

The theory of *fuzzy sets* [448] tries to solve this problem by allowing intermediate degrees of membership: Points are also allowed to be *partial* members of a region. As we can see in Figure 2.6b, the membership function can now become continuous – small changes in the input only lead to small changes in the output. The sudden jump between x and x' has been replaced with a smoother transition between members and non-members. As we do no longer have a clear threshold separating members from non-members, we can say that the conceptual boundary has become imprecise. Please note that using fuzzy sets to represent imprecise knowledge about conceptual boundaries has already a long tradition in cognitive science (cf. [55, 136, 310, 340, 450]).

Definition 2.10 (Fuzzy Set)

A fuzzy set \tilde{A} on CS is defined by its membership function $\mu_{\tilde{A}} : CS \rightarrow [0, 1]$. For each $x \in CS$, $\mu_{\tilde{A}}(x)$ is interpreted as degree of membership of x in \tilde{A} , with $\mu_{\tilde{A}}(x) = 1$ indicating full membership and $\mu_{\tilde{A}}(x) = 0$ indicating complete non-membership.

Relation to crisp sets

Fuzzy sets contain crisp sets as a special case where $\mu_{\tilde{A}} : CS \rightarrow \{0, 1\}$. In analogy to the crisp power set $\mathcal{P}(CS)$, we denote the set of all fuzzy sets on CS as $\mathcal{F}(CS)$ and call this the fuzzy power set of CS .

Definition 2.11 (α -Cut)

Given a fuzzy set \tilde{A} on CS , its α -cut \tilde{A}^α for $\alpha \in [0, 1]$ is defined as follows:

$$\tilde{A}^\alpha = \{x \in CS \mid \mu_{\tilde{A}}(x) \geq \alpha\}$$

The special case of \tilde{A}^1 is called the core of \tilde{A} .

Intuitively, an α -cut returns a crisp set of all the points that have a certain minimal membership to the underlying fuzzy set. Definition 2.11 also gives a hint why the crisp conceptual regions from Section 2.3.2 are called cores: The points inside these regions will be the ones that receive a full degree of membership in our fuzzy conceptual regions.

α -cuts turn fuzzy sets into crisp sets based on a threshold

The notion of convexity can be generalized from crisp sets to fuzzy sets as follows, resulting in a property that is commonly called *quasi-concavity* [394, 395, 411] (see also [179, Chapter 3.5]):

Quasi-concavity as fuzzy version of convexity

Definition 2.12 (Quasi-Concavity)

A fuzzy set \tilde{A} on CS is called quasi-concave under a metric d , if for all points $x, y, z \in CS$ with $B_d(x, y, z)$, we have $\mu_{\tilde{A}}(y) \geq \min(\mu_{\tilde{A}}(x), \mu_{\tilde{A}}(z))$.

Equivalently, each α -cut of \tilde{A} is either empty or convex under d .

Also the notion of star-shapedness can be generalized from crisp sets (see Definition 2.4) to fuzzy sets by using α -cuts:

Fuzzy star-shapedness

Definition 2.13 (Fuzzy Star-Shapedness)

A fuzzy set \tilde{A} on CS is called star-shaped under a metric d with respect to a crisp set $P \Leftrightarrow$

$$\forall \alpha \in [0, 1] : (\tilde{A}^\alpha = \emptyset \text{ or } \tilde{A}^\alpha \text{ is star-shaped under } d \text{ with respect to } P).$$

We can now define concepts as fuzzy sets based on the crisp cores from Section 2.3.2: The membership of a point $x \in CS$ to a concept \tilde{S} is based on the maximal similarity of x to any point $y \in S$ in the core S .

Concepts as fuzzified cores

Definition 2.14 (Concept as Fuzzy Region)

A concept \tilde{S} is described by a quadruple $\langle S, \mu_0, c, W \rangle$. The components of this quadruple are the following:

- $S = \langle \Delta_S, \{C_1, \dots, C_m\} \rangle$ is a non-empty core as described in Definition 2.9.
- The parameter $\mu_0 \in (0, 1]$ controls the highest possible membership to \tilde{S} and is usually set to 1.
- The sensitivity parameter $c > 0$ controls the rate of the exponential decay in the similarity function and thus the overall fuzziness of \tilde{S} .
- Finally, $W = \langle W_{\Delta_S}, \{W_\delta\}_{\delta \in \Delta_S} \rangle$ contains positive weights for all domains in Δ_S and all dimensions within these domains. These weights are used when computing the combined metric $d_C^{\Delta_S}$ and reflect the relative importance of the respective domains and dimensions. As argued in Section 2.3.1, we assume that $\sum_{\delta \in \Delta_S} w_\delta = |\Delta_S|$ and that $\forall \delta \in \Delta_S : \sum_{d \in \delta} w_d = 1$.

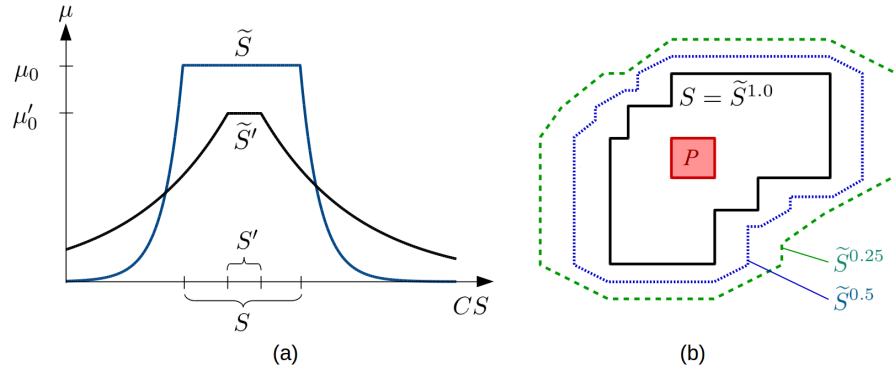


Figure 2.7: (a) Two concepts \tilde{S} and \tilde{S}' in a one-dimensional conceptual space. (b) A concept \tilde{S} in a two-dimensional conceptual space along with two of its α -cuts.

The membership function of \tilde{S} is then defined as follows:

$$\mu_{\tilde{S}}(x) = \mu_0 \cdot \max_{y \in S} \left(e^{-c \cdot d_C^{\Delta S}(x,y,W)} \right)$$

Concepts are fuzzy star-shaped regions

One can show that concepts as defined in Definition 2.14 are star-shaped under $d_C^{\Delta S}$ in the sense of Definition 2.13. This requires the insight that each α -cut of \tilde{S} is equivalent to an ϵ -neighborhood of S .

Lemma 2.4 (α -Cut is an ϵ -Neighborhood)

Let $\tilde{S} = \langle S, \mu_0, c, W \rangle$ be a concept and let $\alpha \leq \mu_0$. Then, the α -cut \tilde{S}^α is equivalent to an ϵ -neighborhood of S with $\epsilon = -\frac{1}{c} \cdot \ln\left(\frac{\alpha}{\mu_0}\right)$.

Proof. See Appendix A.4. □

Proposition 2.1 (Concepts are Fuzzy Star-Shaped)

Any concept $\tilde{S} = \langle S, \mu_0, c, W \rangle$ is star-shaped with respect to $P = \bigcap_{i=1}^m C_i$ under $d_C^{\Delta S}$.

Proof. See Appendix A.4. □

Example concepts

Figure 2.7a illustrates the membership functions of two concepts \tilde{S} and \tilde{S}' in a one-dimensional conceptual space. Here, \tilde{S} has a large core S , a high maximal membership value μ_0 , and a large sensitivity parameter c , while the respective parameters of \tilde{S}' (namely, S' , μ'_0 , and c') are relatively small. Figure 2.7b shows a two-dimensional concept \tilde{S} that is based on the core S from Figure 2.5. In this illustration, the x and the y axis belong to different domains, and are thus combined with the Manhattan metric. Moreover, the x axis has a lower salience weight than the y axis.

Controlling the fuzziness

As we can see in Figure 2.7, both the weights W and the sensitivity parameter c directly influence the fuzziness of \tilde{S} : The sensitivity parameter c controls the overall fuzziness of \tilde{S} by determining how fast the membership drops to zero. Larger values c cause steeper membership functions (i.e., "crisper" fuzzy sets). This can be seen in Figure 2.7a. The

weights W on the other hand represent not only the relative importance of the respective domain or dimension to the represented concept, but they also influence the relative fuzziness with respect to this domain or dimension: If the weight of a given dimension¹⁴ d is relatively large in comparison to the weights of another dimension d' , the membership function with respect to d will be steeper than with respect to d' . In short: Larger weights cause less fuzziness. We can see this in Figure 2.7b, where the x axis receives a lower salience weight than the y axis and consequently the membership function drops faster on the y axis than on the x axis.

How can these salience weights be determined? There are at least three possible ways:

Due to the effect of the weights on the fuzziness of the resulting concept, one may want to base the salience weights on the size of the underlying crisp set S in the respective dimension: If S is already relatively "large" (i.e., it covers a relatively large part of the space) in a given dimension d , this means that many different values on d can lead to a full membership in the corresponding concept \tilde{S} . We would then also expect many different values on d to lead to high partial memberships (e.g., membership values of 0.8 and higher). This means that the respective α -cuts should also be large, i.e., the membership function should decay relatively slowly with respect to d . This effect can be achieved by using a small salience weight w_d for this dimension – which also indicates that d is not very important for determining concept membership. Analogously, we can link a small size of S to a steep drop in the membership function and a large salience weight.

Alternatively, one can use the *diagnosticity* of each dimension to determine its salience weight: If the given concept has a large distance to other concepts with respect to a given dimension d , then the values on d are very informative for making a classification. Therefore, the salience weight w_d should be large. Sileno et al. [370] have made a concrete proposal along these lines: In their work, all concepts of the same hierarchy level (e.g., APPLE, PEAR, LEMON) are contrasted with their superordinate concept (e.g., FRUIT) by computing the direction in which the given concept differs from typical examples of the superordinate. The dimensions in which this difference is large receive a larger salience weight than the dimensions with only small differences.

Finally, one can also jointly optimize all parameters of a given concept in order to achieve an optimal fit to a given data set. This approach treats the salience weights as yet another model parameter in a machine learning context and does not make use of a particular intuitive motivation. However, one may expect that also in this case the weights are tuned in a way that optimizes classification performance, hence resembling somewhat the diagnosticity-based approach.

Basing salience weights on the size of the core

Obtaining salience weights from diagnosticity

Estimating salience weights with machine learning

¹⁴ For better readability, we only refer to dimension weights here. The same applies of course also to domain weights.

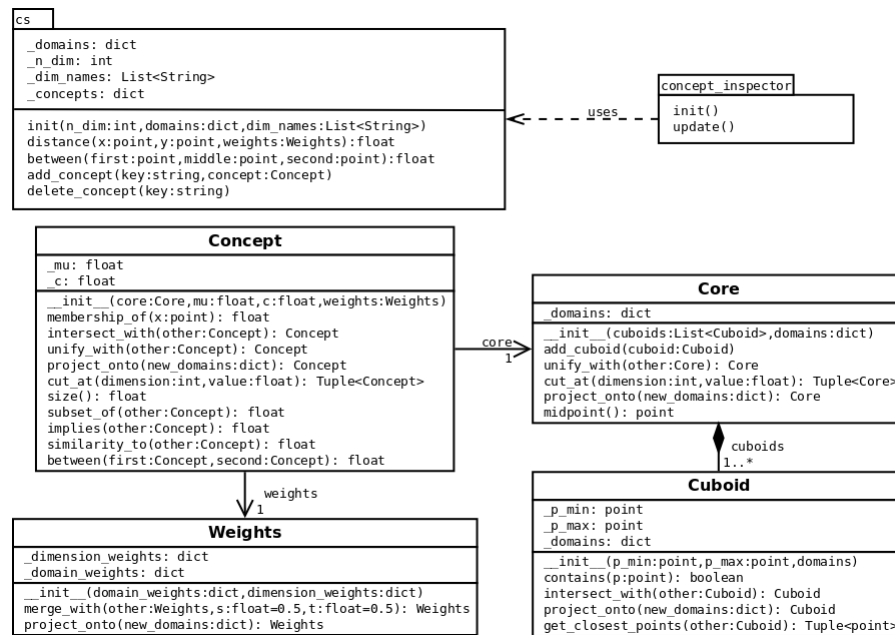


Figure 2.8: Class diagram of our implementation.

Properties and concepts can use the same formalism

Note that if $|\Delta_S| = 1$, then \tilde{S} represents a property (since it is only defined on a single domain), and if $|\Delta_S| > 1$, then \tilde{S} represents a concept. This way, we can represent both properties and concepts with the same formalism. As we only discovered problems with the convexity criterion for the combination of multiple domains, we might still want to follow Gärdenfors' convexity assumption for regions within a single domain (i.e., for properties). This, however, forces us to represent properties with a single cuboid, which might be too coarse-grained for practical applications. Again, we leave this decision up to the specific application scenario.

2.3.4 Implementation and Example: Fruit Space

Overview of the implementation

We have implemented our formalization in Python and have made it publicly available on GitHub¹⁵ [41, 51]. Figure 2.8 shows a class diagram illustrating the overall structure of our implementation. Each of the components from our definition (i.e., weights, cuboids, cores, and concepts) is represented by an individual class.¹⁶ Moreover, the `cs` module contains the overall domain structure of the conceptual space (represented as a dictionary mapping from domain identifiers to sets of dimensions) along with some utility functions (e.g., computing distance and betweenness of points). The `concept_inspector` package contains a visualization tool that displays 3D and 2D projections of the concepts stored in the `cs` package.

¹⁵ See <https://github.com/lbechberger/ConceptualSpaces>.

¹⁶ Most of the methods implemented for the `Concept` class refer to operations from our formalization which will be introduced in Chapter 3 and 4.

In order to illustrate the definitions made in this chapter, we introduce a very simplified conceptual space for fruit. This space consists of only three domains, each containing only a single dimension. We restrict ourselves to three dimensions in total in order to visualize the conceptual structures sufficiently well. Using only one-dimensional domains might be a considerable limitation of this example's expressiveness, but we hope that it nevertheless helps to complement this chapter's formal results with some concrete examples. We will return to this example fruit space in Chapters 3 and 4, where we will use it to illustrate operations on concepts.

A three-dimensional fruit space

Our overall conceptual space for fruits can be defined based on the following domains and dimensions:

Domains and dimensions

$$\Delta = \{\delta_{color} = \{d_{hue}\}, \delta_{shape} = \{d_{round}\}, \delta_{taste} = \{d_{sweet}\}\}$$

The three dimensions of this space have the following semantics:

- d_{HUE} describes the HUE of the observation's COLOR, with focal values of 0.00 for purple, 0.25 for blue, 0.50 for green, 0.75 for yellow, and 1.00 for red.¹⁷
- d_{ROUND} is computed by calculating the bounding circle of the object and measuring the percentage of its area that is covered by the object. Its values range thus from 0.00 (e.g., for a line segment) to 1.00 (for a perfect circle).
- d_{SWEET} represents the relative amount of sugar contained in the fruit. We assume that it is normalized to a range between 0.00 (no sugar) and 1.00 (highest sugar content).

In our implementation, we can define this conceptual space as follows:

```
domains = {'color':[0], 'shape':[1], 'taste':[2]}
dimension_names = ['hue', 'round', 'sweet']
space.init(3, domains, dimension_names)
```

Table 2.1 contains definitions for several concepts and properties. When giving coordinates for points, we assume that the dimensions are ordered like this: $d_{HUE}, d_{ROUND}, d_{SWEET}$.

Some example fruit concepts

Note that we have assigned the domain weights according to the prominence of the respective domain to the respective concept. For instance, the TASTE of a LEMON sets it clearly apart from all the other fruit concepts (it is not sweet at all), while its COLOR is not that unique (e.g., also a BANANA or an APPLE can be yellow). Therefore, in \tilde{S}_{LEMON} , the TASTE domain receives a much higher weight than the COLOR domain. As all domains are one-dimensional, the dimension weights w_d are always equal to 1, thus identical for all concepts, and therefore not listed in Tables 2.1. In the code, we can define concepts as follows:

Domain and dimension weights

```
c_pear = Cuboid([0.5, 0.4, 0.35], [0.7, 0.6, 0.45], domains)
s_pear = Core([c_pear], domains)
```

¹⁷ HUE should actually be viewed as a circular dimension. For the sake of simplicity, we, however, treat it here as linear dimension.

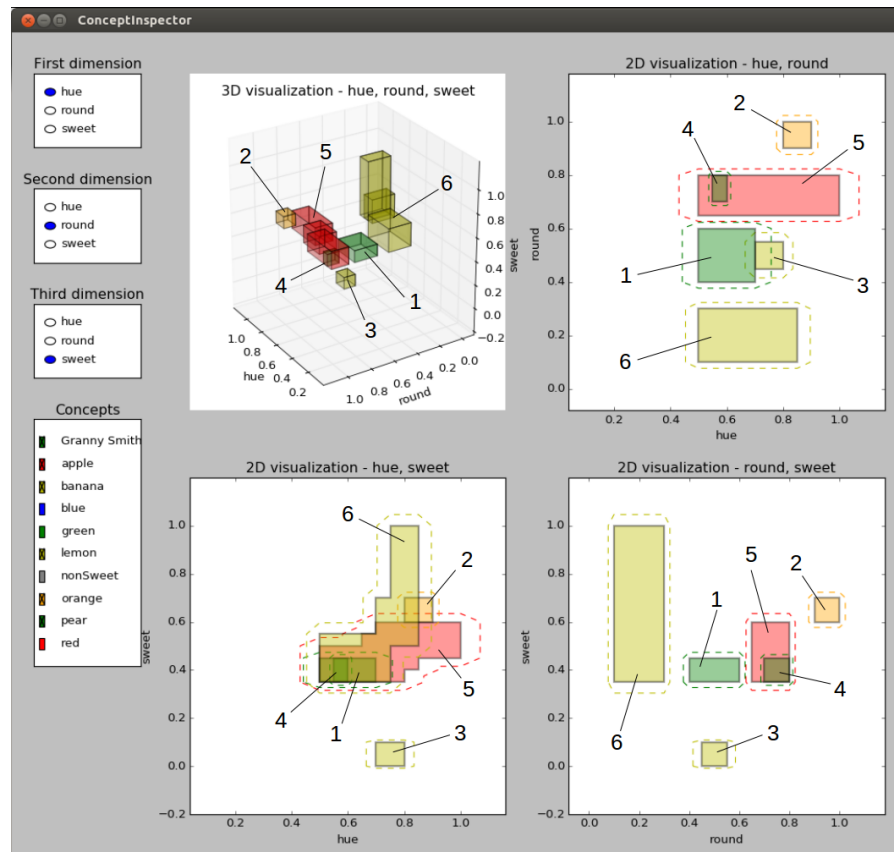


Figure 2.9: Screenshot from the ConceptInspector tool illustrating the fruit concepts in our example space. The concepts are labeled as follows: PEAR (1), ORANGE (2), LEMON (3), GRANNY SMITH (4), APPLE (5), BANANA (6).

Concept	Δ_S	p^-	p^+	μ_0	c	W		
						$w_{\delta_{\text{COLOR}}}$	$w_{\delta_{\text{SHAPE}}}$	$w_{\delta_{\text{TASTE}}}$
PEAR	Δ	(0.50, 0.40, 0.35)	(0.70, 0.60, 0.45)	1.0	24.0	0.50	1.25	1.25
ORANGE	Δ	(0.80, 0.90, 0.60)	(0.90, 1.00, 0.70)	1.0	30.0	1.00	1.00	1.00
LEMON	Δ	(0.70, 0.45, 0.00)	(0.80, 0.55, 0.10)	1.0	40.0	0.50	0.50	2.00
GRANNY SMITH	Δ	(0.55, 0.70, 0.35)	(0.60, 0.80, 0.45)	1.0	50.0	1.00	1.00	1.00
APPLE	Δ	(0.50, 0.65, 0.35) (0.65, 0.65, 0.40) (0.70, 0.65, 0.45)	(0.80, 0.80, 0.50) (0.85, 0.80, 0.55) (1.00, 0.80, 0.60)	1.0	20.0	0.50	1.50	1.00
BANANA	Δ	(0.50, 0.10, 0.35) (0.70, 0.10, 0.50) (0.75, 0.10, 0.50)	(0.75, 0.30, 0.55) (0.80, 0.30, 0.70) (0.85, 0.30, 1.00)	1.0	20.0	0.75	1.50	0.75
RED	$\{\delta_{\text{COLOR}}\}$	(0.90, $-\infty$, $-\infty$)	(1.00, $+\infty$, $+\infty$)	1.0	40.0	1.00	–	–
GREEN	$\{\delta_{\text{COLOR}}\}$	(0.45, $-\infty$, $-\infty$)	(0.55, $+\infty$, $+\infty$)	1.0	40.0	1.00	–	–
BLUE	$\{\delta_{\text{COLOR}}\}$	(0.20, $-\infty$, $-\infty$)	(0.30, $+\infty$, $+\infty$)	1.0	40.0	1.00	–	–
NONSWEET	$\{\delta_{\text{TASTE}}\}$	($-\infty$, $-\infty$, 0.00)	($+\infty$, $+\infty$, 0.20)	1.0	14.0	–	–	1.00

Table 2.1: Definitions of several fruit concepts and properties.

```

w_pear = Weights({'color':0.50, 'shape':1.25, 'taste':1.25},
                 {'color':{0:1.0}, 'shape':{1:1.0},
                  'taste':{2:1.0}})
pear = Concept(s_pear, 1.0, 12.0, w_pear)

```

Because all single-cuboid concepts are not only star-shaped, but also convex, their central region P is identical to their cuboid C . For the concepts of apple and banana, however, these central regions need to be computed:

Central regions

$$\begin{aligned}
P_{\text{APPLE}} &= \bigcap_{i=1}^3 C_{\text{APPLE},i} = \langle \Delta, p_{\text{APPLE}}^- = (0.70, 0.65, 0.45), \\
&\quad p_{\text{APPLE}}^+ = (0.80, 0.80, 0.50) \rangle \\
P_{\text{BANANA}} &= \bigcap_{i=1}^3 C_{\text{BANANA},i} = \langle \Delta, p_{\text{BANANA}}^- = (0.75, 0.10, 0.50), \\
&\quad p_{\text{BANANA}}^+ = (0.75, 0.30, 0.55) \rangle
\end{aligned}$$

While this numeric representation is clearly well suited for automated processing in a computer, it is intuitively hard to grasp for a human. Figures 2.9 and 2.10 therefore show screenshots of the ConceptInspector tool for the concepts and properties from Table 2.1, respectively. Each figure shows both the crisp cores in a three-dimensional space and three two-dimensional visualizations of both the cores and the 0.5-cuts.

Visualization

We would finally like to note that NONSWEET is a rather artificial property – it would be more natural to define SWEET as a property on the TASTE domain. However, for the examples we will make in Chapters 3 and 4, we will need a TASTE property which intersects with the LEMON concept. As we only use the SWEETNESS dimension to define the TASTE domain, we hence introduced NONSWEET for illustrative purposes.

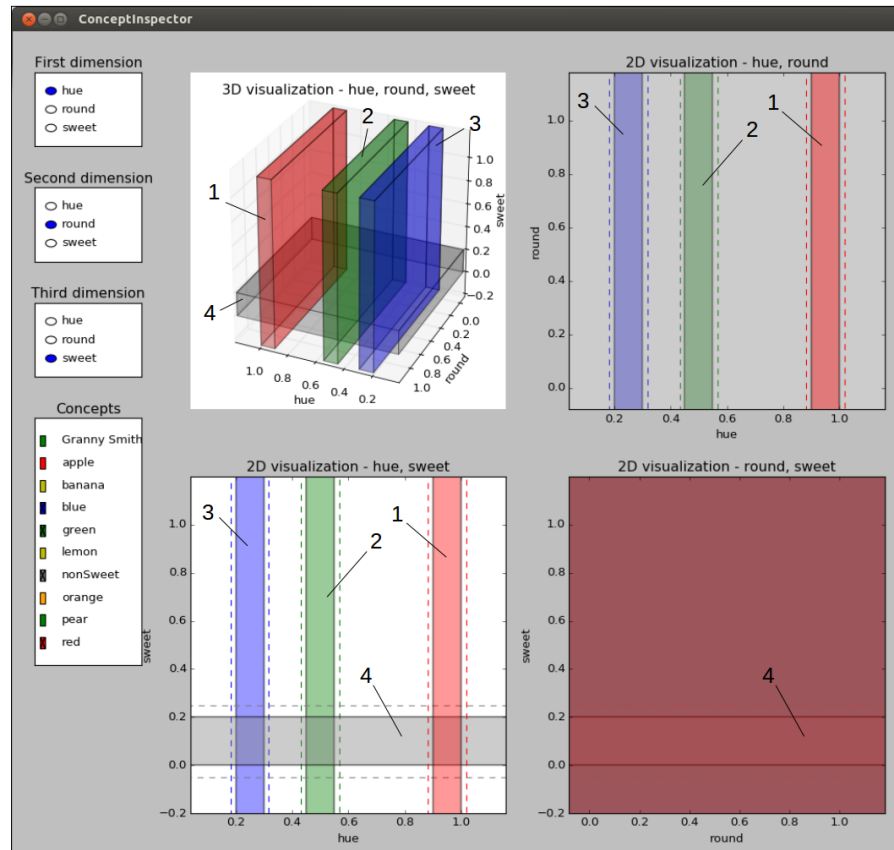


Figure 2.10: Screenshot from the Concept Inspector tool illustrating the properties in our example space. The properties are labeled as follows: RED (1), GREEN (2), BLUE (3), NONSWEET (4).

2.4 RELATED WORK

Our proposed formalization relates of course to a large body of research about conceptual spaces. In Section 2.4.1, we first show how our formalization relates to general ideas proposed by other researchers, before providing a high-level comparison to other formalizations in Section 2.4.2. Finally, we will comment on two proposals for representing composite concepts in Section 2.4.3 and discuss their possible incorporation into our work.

Overview

2.4.1 Related General Ideas

The need for *imprecise conceptual boundaries* has been articulated and analyzed by Douven et al. [136]. They propose an extension of the conceptual spaces framework in order to explicitly consider borderline cases. In Gärdenfors' original proposal [179], each concept is represented by a single prototypical point (cf. Section 1.2.2). Conceptual regions are then created by assigning each point in the space to its closest prototype. Douven et al. argue that the resulting Voronoi diagram comes with a poor representation of borderline cases: Only the points that have the exact same distance to two or more prototypes lie on such a border line. They then propose to extend Gärdenfors' original proposal by representing prototypes not as single points but as *sets* of points. They then aggregate all possible Voronoi diagrams that can be created by picking one point from each prototypical region. Borderline cases are now represented as points that belong to different conceptual regions for different Voronoi diagrams.

Imprecise concept boundaries

Although Douven et al. provide an account of vagueness in conceptual spaces, their approach still makes a sharp distinction between borderline cases and non-borderline cases. This shortcoming has later been addressed by Decock et al. [118], who provide a fuzzy degree of concept membership for points in the borderline area. We will revisit this approach in more detail in Chapter 7. Moreover, they keep the assumption that concepts can be interpreted as Voronoi tessellations of the conceptual space. While this may be true for properties (e.g., every point in the COLOR space supposedly belongs to a certain COLOR property), it seems more likely that the overall conceptual space is populated much more sparsely (cf. Section 1.2.2). In our opinion, there are points in the overall conceptual space that are far away from all concepts and should not be classified as members of any of those concepts – they rather represent an unusual or exceptional observation [395, 453]. Moreover, Voronoi tessellations cannot represent overlapping conceptual regions, which are needed to encode conceptual hierarchies such as CHESTNUT being a particular shade of BROWN.

Limitations of Voronoi-based approaches

Imprecise concept boundaries can of course also be represented by fuzzy sets that are not based on Voronoi tessellations. As already mentioned in Section 2.3.3, a popular fuzzy generalization of the convexity constraint is the notion of *quasi-concavity*, which has for instance been

Non-partitional approaches

proposed by Ströβner [394, 395] and Tull [411]. In his work, Tull [411] extends the framework by Bolt et al. [69] (which combines conceptual spaces with category theory) from crisp sets to fuzzy sets, using a special type of quasi-concave membership functions. Also some prior formalizations of the conceptual spaces framework (to be discussed in Sections 2.4.2 and 2.5) explicitly consider fuzzy concept boundaries.

*Contrast in
conceptual spaces*

From a somewhat different perspective, also Dessalles [124] has criticized the usage of a Voronoi tessellations. He argues that such a categorization rule does not provide a fine-grained judgment of concept membership: It only allows to refuse membership to a given concept if the observation is closer to a prototype of a different concept. Moreover, he argues that justifications for membership or non-membership in a given category are essential for the categorization process. By looking at examples like "This is a BOOK, because it has been properly published" and "This is not a BOOK, because it is too thin", Dessalles motivates the need for a *contrast operation* for comparing an observation to a prototype. In a conceptual space, this contrast can be simply represented as the vector denoting the difference between two points. The direction of this contrast vector then indicates which domains and dimensions are relevant. One can use this contrast vector then to select the property used to describe the observation. For example, a RED FACE can be interpreted as a FACE which differs from a prototypical face mostly by its COLOR (which is redder than usual). In a similar way, one can interpret BIG FLEA and SMALL ELEPHANT. This provides an elegant account of relative properties and differs from the one proposed originally by Gärdenfors [179], which will be introduced in the context of concept combination in Chapter 3.

*Concept membership
based on contrast*

Moreover, Dessalles [124] notes that also concept membership can be expressed based on contrast vectors: An observation can be defined as being a member of a concept if the length of the contrast vector is less than a given threshold. This supports the idea that concepts should be star-shaped: If a point x is a member of a concept with prototype p , then the contrast vector \vec{px} is shorter than some threshold ϵ . This means that for any point y between x and p , \vec{py} is shorter than \vec{px} and thus y is also a member of the concept. If we now allow for a *set* of prototypical points and if we use a *fuzzy degree* of membership (where shorter contrast vectors yield larger memberships), we arrive at a description of concepts quite similar to our formalization. In our case, the set of prototypical points is the core S of a concept, the length of the contrast vector is determined by the combined metric d_C^Δ , and the degree of membership is an exponentially decaying function of this length.

*An egg-yolk model of
conceptual regions*

As Sileno et al. [370] have argued, the approach based on predication and contrast vectors has a number of additional advantages: While the standard approach to conceptual spaces typically requires a distance computation over all possible dimensions, the predicative approach does not need such a holistic perspective. It moreover does not need definite regions to describe concepts – prototypical points and some rough regional information are sufficient. Sileno et al. [370] have formal-

ized the idea of contrast vectors using the egg-yolk model: A concept is described by its prototypical point p and two threshold vectors σ and ρ . If the distance of a given observation o to the prototypical point p of a concept is less than σ_i for each dimension i , then it belongs to the *yolk* region and is considered a typical member of the concept. If the distance between o and p is less than ρ_i , then o belongs to the *egg* region and is considered a member of the concept, although not a very typical one. All observations with a distance of more than ρ_i are not associated with the given concept at all. This representation makes it possible to distinguish membership from typicality: All elements in the egg region are full members of the concept, but only the ones in the yolk region are considered typical. One can in some sense compare the yolk to the prototypical region P in our formalization and the egg to the core S . Our formalization adds the ability to also express imprecise boundaries by allowing for degrees of membership.

A third issue that relates our formalization to other work from the literature is the representation of correlations in conceptual spaces. For instance, Derrac and Schockaert [123] (whose work has been introduced in Section 1.2.5) do not assume that interpretable directions in the conceptual spaces they extract from textual data are orthogonal to each other. Also Jameel et al. [207] use a conceptual space with non-orthogonal dimensions for illustrating their approach. If the quality dimensions of a conceptual space are not orthogonal to each other, this means that changing a value on one of the dimensions also induces changes to the values on other dimensions. Especially when the dimensions of the conceptual space are not linearly independent (i.e., if we have n quality dimensions but they only span an n' -dimensional space with $n' < n$), we get such interdependency effects.

Encoding correlations directly in the structure of the conceptual space asserts that the given correlation exists for *all* concept in the space to an equal degree. This makes sense if one considers a certain group of concepts for which this correlation holds. For example, if we limit ourselves to `FRUIT` concepts, then encoding a correlation between `COLOR` and `TASTE` at a global level is both meaningful and efficient: The encoded correlation is expected to hold for all concepts we might ever describe in this conceptual space and storing it only once is more elegant than storing it locally for each concept. However, if our conceptual space does not only include `FRUIT` concepts but also other food items (e.g., different types of `ICE CREAM`) or other household objects (such as `PLATE` and `BOWL`), then the globally encoded correlation might not hold for all concepts and might be even misleading in some cases.

Moreover, computing the distance between two points in a conceptual space (and thus their similarity) is a very basic operation. If our conceptual space consists of orthogonal dimensions, then we can simply use the combined metric d_C as defined in this chapter. However, if the dimensions of the conceptual space are not orthogonal to each other and maybe not even linearly independent, computing distances might become more complicated.

Correlations through non-orthogonal dimensions

Representational advantages and disadvantages

Potential problems with distance computations

*Correlations as
co-occurrence
statistics*

There is only one other formal discussion of correlations in conceptual spaces that we are aware of, namely, the mathematical formalization of the framework by Rickard [329]. Essentially, he represents correlations between domains through co-occurrence statistics about different properties involved in the concept. We will discuss his non-geometric approach of encoding correlations in Sections 2.4.2 and 2.5.2.

2.4.2 Prior Formalizations of Conceptual Spaces

*Criteria for a good
formalization*

In order to compare different prior formalizations of the conceptual spaces framework to our own proposal, we use the following list of criteria, which has already been briefly mentioned in Section 1.3.2:

- *Parametric description of conceptual regions*
Concepts and properties should be described in a parametric way, i.e., through a mathematical formula with a clear set of parameters. This parametric description of concepts is a crucial requirement for an actual implementation of the framework.
- *Properties are a special case of concepts*
Concepts and properties should be represented with the same formalism. This makes it easier to devise mechanisms for learning and reasoning that are applicable to both properties and concepts without major modifications.
- *Correlations between domains*
The formalization should provide a concrete way for representing cross-domain correlations, because these correlations contain important information about the concept.
- *Imprecise concept boundaries*
Conceptual boundaries should be imprecise in order to reflect borderline cases and a continuous degree of membership.
- *Implementation*
The formalization should come with a publicly available implementation such that other researchers can use it right away for their own research projects.

In the following, we briefly introduce various formalizations from the literature and argue to which extent they fulfill our criteria. Table 2.2 gives an overview of our results. The three strongest competitors to our own work will be reviewed in more detail in Section 2.5.

*Aisbett and Gibbon:
too abstract for
practical use*

An early and quite thorough formalization was developed by Aisbett and Gibbon [11]. They used pointed metric spaces and a generalized form of betweenness as a basis for their formalization. Like we, they considered concepts to be regions in the overall conceptual space. However, they kept Gärdenfors' assumption of convexity. Although their formalization targets the interplay of symbols and geometrically

Formalization	Parametric Description	Properties are Concepts	Correlations	Imprecise Boundaries	Implementation
Aisbett and Gibbon [11]	–	✓	–	–	–
Raubal [327]	–	–	–	–	–
Ahlqvist [10]	–	✓	–	✓	–
Rickard [329]	✓	–	✓	✓	–
Rickard et al. [330]	✓	–	✓	✓	–
Adams and Raubal [3]	✓	✓	–	–	–
Lewis and Lawry [253]	✓	✓	–	✓	–
Our Formalization	✓	✓	✓	✓	✓

Table 2.2: Overview of different formalizations of the conceptual spaces framework based on our list of criteria ("✓" means "fulfilled" and "–" means "not fulfilled").

represented concepts, it is still relatively abstract. For instance, they did not define concepts in a parametric way, which prevents a direct implementation. They also did not comment on the issues of imprecise concept boundaries and cross-domain correlations. Taken together, their formalization seems to be quite incomplete with respect to our list of criteria.

In Raubal's formalization [327], the normalization of all dimensions via the z-transformation (i.e., by subtracting the mean and dividing through the standard deviation) is highlighted as important preprocessing step. Raubal defines a conceptual space as a vector space spanned by domains and dimensions. For calculating semantic distance, he uses only the Euclidean metric in contrast to Gärdenfors' original proposal of combining the Euclidean and the Manhattan metric. Moreover, properties and concepts, which are very important parts of the conceptual spaces framework, are not formalized in Raubal's proposal, which makes it quite incomplete. Overall, it does not fulfill any of our criteria.

Raubal: too incomplet to count as a formalization

Ahlqvist [10] bases his formalization of conceptual spaces on *rough fuzzy sets*, which are characterized by two membership functions serving as a lower and upper bound for concept membership. Concepts are represented by one rough fuzzy set per domain. Ahlqvist's formalization does not consider correlations between different domains. Moreover, the class of rough fuzzy sets is not restricted to a parametrically describable subclass, which limits the direct implementability of his proposal. However, the usage of rough fuzzy sets provides a powerful way for describing imprecise conceptual boundaries. Moreover, properties can be treated as special cases of concepts.

Ahlqvist: concepts as rough fuzzy sets

Also Rickard [329] provides a formalization based on the idea of

Rickard: concepts as co-occurrence statistics of properties

fuzziness. In his notation, each concept is represented as a graph: Nodes in this graph correspond to properties which are represented as convex regions in their respective domains. Directed edges between these nodes indicate the co-occurrence of the respective properties. Weights on these edges represent the strength of the respective co-occurrence. The overall graph can also be seen as a matrix of edge weights. By concatenating the rows of this matrix, Rickard obtains a vector that can be interpreted as a point in a hypercube. This point is then assumed to represent a fuzzy set which is defined on the universe of ordered property pairs. The observed co-occurrence of a pair of properties corresponds to this pair's membership in the concept.

*Limitations of
Rickard's
formalization*

Rickard's representation nicely captures the correlations between different properties, but these correlations are not represented geometrically: Rickard first discretizes the domains (by defining properties) and then computes co-occurrence statistics between these properties. Depending on the discretization, this might lead to a relatively coarse-grained notion of correlation. Moreover, as properties and concepts are represented in different ways, one has to use different learning and reasoning mechanisms for them. Furthermore, Rickard does not restrict properties to a parametrically describable subclass, again limiting the direct implementability of this proposal. Finally, this formalization has the disadvantage that it is not easy to work with due to the complex mathematical transformations involved.

*Rickard et al.:
combining Rickard
with Aisbett and
Gibbon*

Rickard et al. [330] combine the approaches of Rickard [329] and Aisbett and Gibbon [11]. They define properties as fuzzy sets and describe concepts by a quadratic matrix of property associations. These property associations correspond to Rickard's co-occurrence statistics. Again, there is no parametrically describable subclass of fuzzy sets that is considered. Moreover, no constraint like convexity or star-shapedness is enforced on the concepts and properties. In general, the same criticism that applied to Rickard [329] is also applicable to Rickard et al. [330].

*Adams and Raubal:
concepts as sets of
convex polytopes*

Adams and Raubal [3] provide another important formalization of conceptual spaces. They define a concept as a set of properties which are represented as convex regions on their respective domains. These convex regions are represented by convex polytopes, i.e., sets of linear inequations which define their confining hyperplanes. This allows for efficient computations while being potentially more expressive than our cuboid-based approach. Adams and Raubal use the Manhattan metric to combine different domains. However, correlations between different domains are not taken into account as each convex polytope is only defined on a single domain. Also imprecise conceptual boundaries are not a part of their formalization.

*Limitations of their
work*

One could generalize their approach by using polytopes that are defined on the overall space and that are convex under the Euclidean and star-shaped under the Manhattan metric. However, as we will discuss in more detail in Section 2.5.1, we have found that this requires additional constraints in order to ensure star-shapedness. The number

of these constraints grows exponentially with the number of dimensions. Each modification of a concept's description would then involve a large constraint satisfaction problem, rendering this representation unsuitable for learning processes.

Lewis and Lawry [253] formalize conceptual spaces using random set theory. A random set can be characterized by a set of prototypical points P and a threshold ϵ . Observations that have a distance of at most ϵ to the prototypical set are considered to be elements of the set. The threshold ϵ is, however, not exactly determined – only its probability distribution δ is known. Based on this uncertainty, a membership function $\mu(x)$ can be defined that corresponds to the probability $\mathbb{P}_\delta(d(x, P) \leq \epsilon)$. Lewis and Lawry define properties as random sets within single domains and concepts as random sets in a boolean space whose dimensions indicate the presence or absence of properties. In order to define this boolean space, a single property is taken from each domain. This is in some respect similar to the approach of Rickard [329] where concepts are also defined on top of existing properties. However, while Rickard uses two separate formalisms for properties and concepts, Lewis and Lawry use random sets for both – only the underlying space differs.

Lewis and Lawry show that under some assumptions concepts can be described as weighted sums of properties and that concept combination can also be formalized as weighted sums of concepts. They illustrate how their mathematical formalization is capable of reproducing some effects from the psychological concept combination literature. However, they do not develop a way of representing correlations between domains (such as "a RED APPLE is typically SWEET, while a GREEN APPLE is typically SOUR"). One possible way to do this within their framework would be to define two separate concepts REDAPPLE and GREENAPPLE and then define on top of them a disjunctive concept APPLE = REDAPPLE \vee GREENAPPLE. This, however, is a quite indirect way of defining correlations. Nevertheless, their approach is similar to ours in using a distance-based membership function to a set of prototypical points while using the same representational mechanisms for both properties and concepts.

Many practical applications of conceptual spaces (e.g., [98, 123, 129, 327]) use only partial ad-hoc implementations of the conceptual spaces framework which usually ignore some important aspects (e.g., the domain structure). As they do not come close to a formal mathematical treatment, we do not consider them any further.

Moreover, all formalizations discussed so far come without an implementation. The only publicly available implementation of the conceptual spaces framework that we are currently aware of is provided by Lieto et al. [262, 265] for their Dual-PECCS system discussed in Section 1.2.2. It is, however, not based on a thorough mathematical formalization. In Dual-PECCS, each concept is represented in the conceptual space by a single prototypical point and a number of exemplar points. Correlations between domains can therefore only be encoded through the selection of appropriate exemplars. In contrast to our work, the current

*Lewis and Lawry:
concepts as random
sets*

*Limitations of this
formalization*

*Many applications
are not based on
formalizations*

*Implementations of
conceptual spaces*

implementation of their system¹⁸, however, comes without any publicly available source code¹⁹ due to the dependence on third-party code in the form of the ACT-R cognitive architecture [13].

Our own formalization

Finally, let us comment on our own formalization with respect to our desiderata: Both concepts and properties are represented through a parametric description of fuzzy conceptual regions with imprecise borders. By using star-shapedness instead of convexity, we are able to encode cross-domain correlations. As described in Section 2.3.4, we also provide a proof-of-concept implementation of our formalization. Therefore, all criteria outlined above are fulfilled.

2.4.3 Composite Concepts

Why do part-whole structures matter?

Many objects encountered in the world can be viewed not only as wholes, but also as a configuration of parts. For instance, a chair can be regarded as one overall object, or it can be viewed as an assemblage of its legs, seat, and back. Without doubt, the ability to focus on individual parts of an overall object is quite important for both learning and reasoning. The presence or absence of certain parts might be crucial for distinguishing different categories (e.g., birds have a beak, but mammals do not). It is therefore desirable that a formal theory of concepts offers ways to represent the part-whole structure of composite concepts.

Part-whole structures and conceptual spaces

The original framework of conceptual spaces as presented by Gärdenfors [179, 181] (and as formalized in this chapter) does not make any statements about part-whole structures. Also none of the formalizations discussed in Section 2.4.2 (including our own) considers composite concepts in a general way. In this section, we review the proposals by Fiorini [157, 158] and Chella et al. [97] and discuss how they fit in with our formalization.

Holistic and structural processes

Fiorini [158] argues that psychological findings (e.g., [152]) indicate the existence of two complementary cognitive processes in object recognition: a *holistic process* which focuses on the whole object and a *structural process* which focuses on the object's parts. Moreover, within the structural process, the shape of parts and their arrangement seem to be processed separately.

Additional spaces for part-whole relations

Based on this, Fiorini proposes to extend the conceptual spaces framework as illustrated in Figure 2.11: The conceptual space that is used to represent a concept such as APPLE in a holistic way (i.e., based on its overall SHAPE, SIZE, COLOR, WEIGHT, etc.) is called the *holistic space*. This is the conceptual space described in Gärdenfors's original framework. The concept representation is enriched by another space, called the *structure space*. This structure space contains for each part a separate *part space*, which encodes the properties of the part, along with a *structure*

18 See <http://www.dualpeccs.di.unito.it/download.html>.

19 The source code of an earlier and more limited version of their system can be found here: http://www.di.unito.it/~lieto/cc_classifier.html.

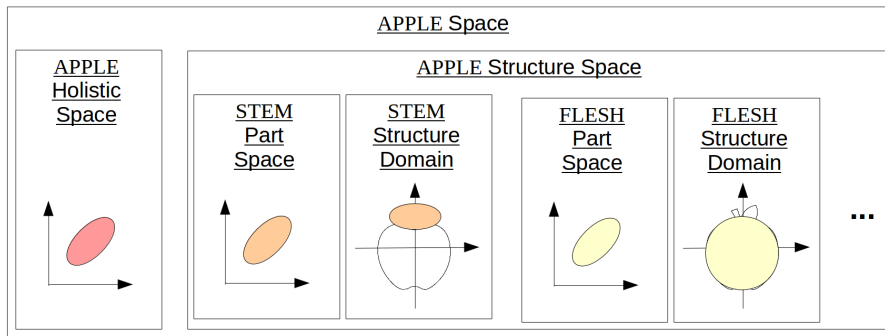


Figure 2.11: Illustration of Fiorini's proposal for representing part-whole relations in conceptual spaces.

domain, which encodes this part's role within the overall concept (e.g., by providing information about the part's typical location within the whole object). In Figure 2.11, we only consider two parts of an *apple*, namely its *STEM* and its *FLESH*. The *SHAPE*, *SIZE*, *COLOR*, etc. of an *APPLE*'S *STEM* is described as a region in the *STEM part space*. The typical location of an *APPLE*'S *STEM* with respect to the whole *APPLE* is depicted as a region in the *STEM structure domain*.

So in a nutshell, Fiorini proposes to represent concepts by three types of information: information about the properties of the overall object (holistic space), information about the properties of the object's parts (part spaces), and information about how these parts are arranged (structure domains).

A problem may arise if the parts themselves are again composite concepts that consist of smaller parts: For instance, the flesh of an apple consists of a certain amount of cells arranged in a specific way, each of these cells consists of molecules, and each molecule consists of atoms. If all these part-whole structures were imported to the *APPLE* concept, the number of dimensions involved would grow infeasibly large and the concept would thus very quickly become unusable. In order to avoid these problems, Fiorini proposes so-called *dimensional filters*. These dimensional filters define which aspects of a part (i.e., which domains) are used in the definition of the overall concept. In the example made above, a dimensional filter applied to the concept of an *APPLE*'S *FLESH* could for instance only keep the *TASTE* and *COLOR* domain while removing all other domains (including the information about its parts).

A slightly different, but related approach for representing composite objects was used by Chella et al. [97] when applying conceptual spaces to computer vision (cf. Section 1.2.4). They describe each simple object by a single point in the conceptual space and composite objects by a *set* of such points in the conceptual space. Each of the points in this set can be thought of as representing one part of the overall object. Because their conceptual space also contains a domain representing *DISPLACEMENT* information, Chella et al. implicitly encode the arrangement of the parts

Three types of information

Recursive part-whole relations and dimensional filters

Composite objects as sets of points

through their respective coordinates in the `DISPLACEMENT` domain. In contrast to Fiorini, they do not explicitly separate the part space from the structure domain. While Fiorini allows the different part spaces to differ (e.g., by involving different domains), Chella et al. use the same conceptual space for all parts. Moreover, their proposal does not explicitly consider holistic information. Overall, the approach proposed by Chella et al. therefore seems to be a special case of Fiorini’s proposal with more limited capabilities.

*Part-whole relations
and our
formalization*

Fiorini’s proposal [158] can be easily integrated in our formalization: He simply introduces additional domains in order to represent the properties and configurations of parts. As our formalization works with any number of domains, we can use all the definitions made in this chapter on this extended space as well. One might, however, need to distinguish the different types of domains in order to interpret them accordingly. For instance, a structure domain might need to be interpreted differently than a regular domain when reasoning about concepts. Fiorini’s dimensional filters can be easily implemented by removing irrelevant domains from the part concepts before attaching them to the whole. In Chapter 3, we will define a projection operation that can be used for this purpose. Therefore, introducing part-whole structures to our formalism would not require a major rework, but only some manageable additions.

*Correlations and
part-whole structures*

Fiorini mentions that the domains of different parts might be correlated, but he does not fully develop this idea. A formalization of his proposal in our formalism would provide means to also specify correlations between wholes, their parts, and the configuration of these parts: For instance, by using our definition of a concept, we would be able to encode that cars with scissor doors usually have a powerful engine (think of expensive sports cars), and that cars with sliding doors in the rear usually have a large body (think of mini vans).

2.5 DETAILED COMPARISON TO OTHER FORMALIZATIONS

The competitors

Three of the formalizations presented in Section 2.4.2 deserve a more detailed discussion: The formalization by Adams and Raubal [3] proposes a parametrized definition of concepts as geometric regions, but fails to account for correlations between domains. The formalization by Rickard [329] puts its focus exactly on such correlations, but does not offer a parametrizable geometric representation of concepts. Finally, the formalization by Lewis and Lawry [253] uses random set theory and a set of prototypical points for defining concepts. However, it also does not consider cross-domain correlations.

*A modified fruit
space*

In order to contrast these three approaches with our own, we consider a variant of the fruit space introduced in Section 2.3.4. This modified

Concept	Δ_S	p^-	p^+	μ_0	c
RED	$\{\delta_{color}\}$	$(0.70, -\infty, -\infty)$	$(1.00, +\infty, +\infty)$	1.0	40.0
YELLOW	$\{\delta_{color}\}$	$(0.40, -\infty, -\infty)$	$(0.60, +\infty, +\infty)$	1.0	40.0
GREEN	$\{\delta_{color}\}$	$(0.00, -\infty, -\infty)$	$(0.30, +\infty, +\infty)$	1.0	40.0
SOUR	$\{\delta_{taste}\}$	$(-\infty, 0.50, 0.00)$	$(+\infty, 1.00, 0.40)$	1.0	14.0
SWEET	$\{\delta_{taste}\}$	$(-\infty, 0.00, 0.50)$	$(+\infty, 0.40, 1.00)$	1.0	14.0
APPLE	Δ	$(0.10, 0.50, 0.10)$ $(0.30, 0.30, 0.40)$ $(0.45, 0.10, 0.45)$	$(0.55, 0.90, 0.50)$ $(0.70, 0.60, 0.55)$ $(0.90, 0.50, 0.90)$	1.0	20.0

Table 2.3: Definitions of conceptual regions of several fruit concepts and properties for comparing different formalizations.

Concept	W				
	$w_{\delta_{COLOR}}$	$w_{\delta_{TASTE}}$	$w_{d_{HUE}}$	$w_{d_{SOUR}}$	$w_{d_{SWEET}}$
RED	1.00	–	1.00	–	–
YELLOW	1.00	–	1.00	–	–
GREEN	1.00	–	1.00	–	–
SOUR	–	1.00	–	0.70	0.30
SWEET	–	1.00	–	0.30	0.70
APPLE	0.67	1.33	1.00	1.00	1.00

Table 2.4: Definitions of domain and dimension weights of several fruit concepts and properties for comparing different formalizations.

space consists of two domains, namely COLOR and TASTE. The COLOR domain is modeled by a single dimension (HUE), whereas the TASTE domain is modeled by two dimensions (SWEETNESS, measurable as sugar content, and SOURNESS, measurable as acidity). We again assume that all dimensions are scaled between 0 and 1.

We will now consider how the different formalisms represent the concept of APPLE. Let us start with our own formalism and let us assume that the dimensions are ordered as follows: $d_{HUE}, d_{SOUR}, d_{SWEET}$. Tables 2.3 and 2.4 show the definition of three COLOR properties, two TASTE properties, and the APPLE concept. The definition of the APPLE concept encodes that a RED APPLE tends to be SWEET, whereas a GREEN APPLE tends to be SOUR. Moreover, an APPLE tend not to be SWEET and SOUR at the same time. Our encoding thus expresses correlations within and between domains. Figure 2.12 illustrates this representation.

The APPLE concept

In the following, we contrast our own representation to the ones used by Adams and Raubal [3] (Section 2.5.1), Rickard [329] (Section 2.5.2), and Lewis and Lawry [253] (Section 2.5.3).

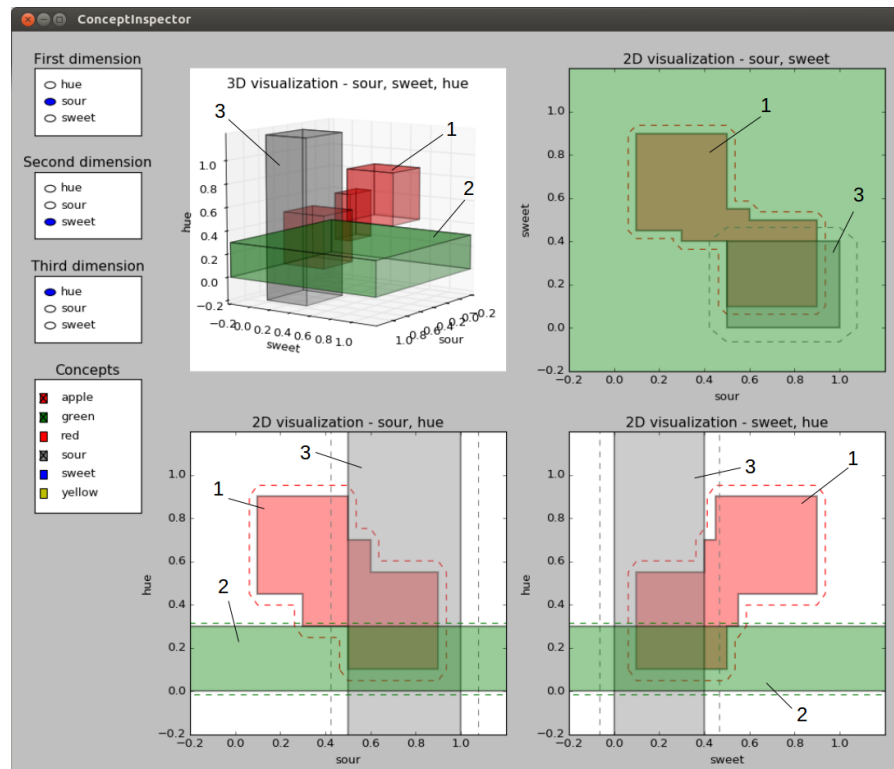


Figure 2.12: Screenshot of the ConceptInspector tool for our proposed encoding of the APPLE concept. There are correlations within the TASTE domain as well as between the TASTE and the COLOR domain. The intersection of APPLE (1) with GREEN (2) has a large overlap with SOUR (3), indicating that green apples tend to be sour.

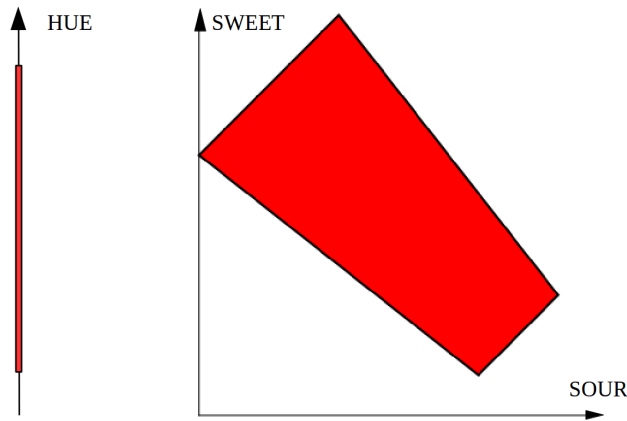


Figure 2.13: Illustration of the APPLE concept encoded under the formalization of Adams and Raubal [3]. There are correlations within the taste domain, but not between the taste and the color domain.

2.5.1 Comparison to Adams and Raubal

Let us first consider the formalization by Adams and Raubal [3]: In their formalism, every concept γ is defined as a pair $\langle \diamond, P \rangle$ with a finite set \diamond of convex regions and a prototypical member P . Each convex region $\diamond \in \diamond$ is defined as a *convex polytope* within a single domain. Convex polytopes can be expressed by a set of linear inequalities. Let us for our current example define the polytopes of the apple concept as follows:

$$\diamond_{color} = \begin{pmatrix} x_{color} \leq 0.9 \\ x_{color} \geq 0.1 \end{pmatrix}$$

$$\diamond_{taste} = \begin{pmatrix} 11 \cdot x_{sour} + 14 \cdot x_{sweet} \geq 910 \\ 7 \cdot x_{sour} + 0.5 \cdot x_{sweet} \leq 795 \\ -x_{sour} + x_{sweet} \leq 65 \\ -x_{sour} + x_{sweet} \geq -60 \end{pmatrix}$$

Figure 2.13 illustrates this encoding by showing the individual polytopes on their respective domains. As one can see by comparing Figure 2.13 to the two-dimensional visualizations of our encoding in Figure 2.12, convex polytopes offer a much more fine-grained way of describing the regions associated to a concept within a single domain. Our cuboid-based approach is less elegant in this respect. However, the formalization of Adams and Raubal is not capable of representing correlations between different domains. It provides no way of expressing the fact that a GREEN APPLE tends to be SOUR while a RED APPLE is expected to be SWEET. The main reason for this is the fact that concepts are not represented as a single region in the overall conceptual space, but as a set of regions in the individual domains. If we visualize the overall conceptual space as the product space of the domains (see Figure 2.14), the lack of cross-domain correlations becomes apparent immediately.

Regions as convex polytopes

Convex polytopes are more fine-grained, but limited to individual domains

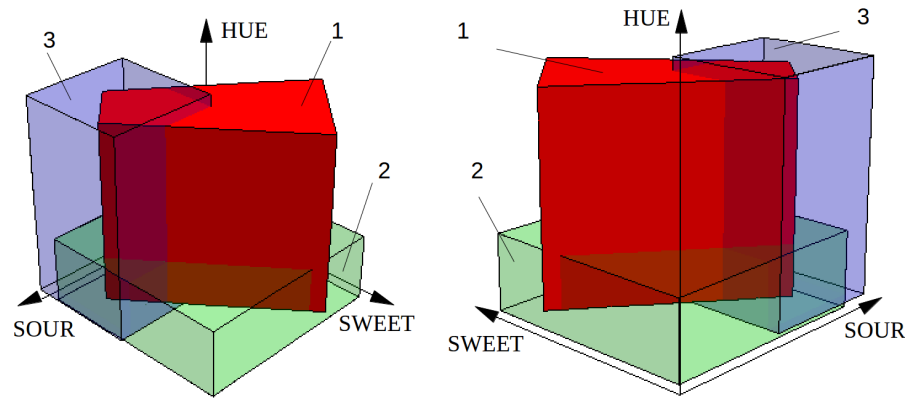


Figure 2.14: Visualization of the overall conceptual space as product space of the domains from Figure 2.13. The intersection of APPLE (1) with GREEN (2) has only a small overlap with SOUR (3), highlighting that this representation cannot encode cross-domain correlations.

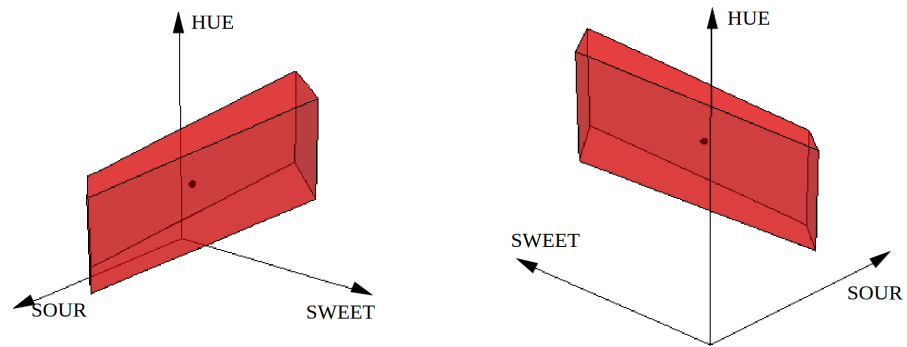


Figure 2.15: Three-dimensional illustration of the encoding of the APPLE concept using a modified version of the formalization by Adams and Raubal [3]. The overall concept is defined as a polytope that is star-shaped with respect to a prototypical point under the combined distance.

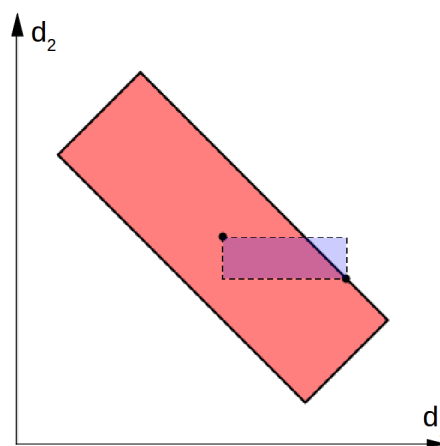


Figure 2.16: A polytope defined by a number of inequalities that is not star-shaped under the Manhattan metric.

The formalization by Adams and Raubal could be generalized by defining each concept as a polytope on the overall conceptual space. As argued in Section 2.2, one should then only require star-shapedness in order to encode correlations. This is illustrated in Figure 2.15, where the APPLE concept is encoded as a three-dimensional polytope that is star-shaped with respect to a central point under the combined metric. This representation clearly allows to encode cross-domain correlations such as a GREEN APPLE being SOUR. However, one needs additional constraints in order to ensure star-shapedness under the combined metric. Figure 2.16 shows a polytope in a two-dimensional space. One can easily see that this region is not star-shaped with respect to its central point under the Manhattan metric – if it were, then the blue rectangle would be completely contained inside the set. In order to ensure star-shapedness, one needs to formulate constraints on the inequalities that define the polytope. Unfortunately, the number of necessary constraints grows very fast as the number of dimensions increases. Updating not only a set of inequalities during a learning process but also taking into account a large number of constraints might quickly become infeasible.

A possible generalization for incorporating correlations

Overall, it thus seems that the approach by Adams and Raubal is inherently limited in its capabilities of representing cross-domain correlations. Its definition of conceptual regions as polytopes is more expressive than our cuboid-based approach, while our formalization can express cross-domain correlations, also includes imprecise concept boundaries, and comes with a publicly available implementation.

Summary

2.5.2 Comparison to Rickard

Let us now look at the formalization by Rickard [329]. This formalization requires properties to be crisp convex regions, but does not propose a parametric description. For the sake of simplicity, we will use the cores of the properties RED, YELLOW, GREEN, SOUR, and SWEET as we have defined them above in the context of our own formalization. According to Rickard, a concept is defined by a matrix of co-occurrence values of different properties. The co-occurrence value $C(S_1, S_2)$ indicates for two properties S_1 and S_2 how often S_2 occurs if S_1 is observed. These co-occurrence values can be understood as conditional probabilities $\mathbb{P}(S_2 | S_1)$. They can be estimated based on observed frequencies in a given set of observations. If we start from a geometric representation like ours, one could also define $C(S_1, S_2)$ as the degree to which S_2 is a subset of S_1 . We will revisit this thought in Chapter 4, when we define a degree of subsethood for fuzzy concepts.

Co-occurrence statistics of properties

Let us assume we have the following co-occurrence values for the APPLE concept based on the properties defined above:

Example representation of the APPLE concept

$$\begin{aligned} C(S_{\text{SOUR}}, S_{\text{RED}}) &= 0.0 & C(S_{\text{RED}}, S_{\text{SOUR}}) &= 0.0 \\ C(S_{\text{SOUR}}, S_{\text{YELLOW}}) &= 0.4 & C(S_{\text{RED}}, S_{\text{SWEET}}) &= 1.0 \\ C(S_{\text{SOUR}}, S_{\text{GREEN}}) &= 0.6 & C(S_{\text{YELLOW}}, S_{\text{SOUR}}) &= 0.5 \end{aligned}$$

$$\begin{aligned}
C(S_{\text{SWEET}}, S_{\text{RED}}) &= 0.6 & C(S_{\text{YELLOW}}, S_{\text{SWEET}}) &= 0.5 \\
C(S_{\text{SWEET}}, S_{\text{YELLOW}}) &= 0.3 & C(S_{\text{GREEN}}, S_{\text{SOUR}}) &= 1.0 \\
C(S_{\text{SWEET}}, S_{\text{GREEN}}) &= 0.1 & C(S_{\text{GREEN}}, S_{\text{SWEET}}) &= 0.0
\end{aligned}$$

Co-occurrences may
be asymmetric

Note that in general $C(S_1, S_2) \neq C(S_2, S_1)$. For example, we have $C(S_{\text{RED}}, S_{\text{SWEET}}) = 1 > 0.6 = C(S_{\text{SWEET}}, S_{\text{RED}})$. This reflects that all red apples are sweet, but only 60 % of the sweet apples are red. If we assume that $C(S_1, S_1) = 1$ and that different properties within the same domain cannot co-occur, we can write the APPLE concept as a matrix:

$$C = \left(\begin{array}{ccc|cc}
1.0 & 0.0 & 0.0 & 0.0 & 1.0 \\
0.0 & 1.0 & 0.0 & 0.5 & 0.5 \\
0.0 & 0.0 & 1.0 & 1.0 & 0.0 \\
\hline
0.0 & 0.4 & 0.6 & 1.0 & 0.0 \\
0.6 & 0.3 & 0.1 & 0.0 & 1.0
\end{array} \right)$$

Interpreting the
concept matrix

Each entry C_{ij} of this *concept matrix* can be interpreted as $C(S_i, S_j)$, assuming that the properties are ordered as follows: RED, YELLOW, GREEN, SOUR, SWEET. Please note that the first and the fourth quadrant of the matrix C correspond to identity matrices and express that different properties within the same domain do not co-occur. Rickard argues that we can interpret the entries of this matrix as membership values of the respective property pair to the concept under consideration. For example, the property pair SWEET-YELLOW would be assigned a membership of 0.3 to the APPLE concept, indicating that a SWEET APPLE may be YELLOW, but that this is not a very typical expectation.

Representation of
correlations

This representation nicely captures the correlation between different properties. For instance, if we consider a GREEN APPLE, we can simply select the row in the matrix corresponding to the property GREEN to obtain the expected correlation of GREEN with SWEET and SOUR in the context of the APPLE concept.

Computing concept
membership

However, this representation of concepts is not geometrical and computing membership values for observations is therefore more complex. If we would like to judge whether a given observation (represented as a point in the overall conceptual space) is a member of the APPLE concept, we cannot simply check whether it is contained in the overall conceptual region. We first need to classify it with respect to the given properties and then somehow compare the resulting vector of classifications to the matrix given above. In a nutshell, Rickard proposes to compute the similarity of the given point to all properties involved in the concept²⁰ and then to aggregate these similarities to individual properties into similarities to *pairs* of properties. These can then be used as entries for an *observation matrix*. This observation matrix has the same shape as the concept matrix. Concept membership of the given observation is then defined as the fuzzy subethood of the observation matrix and the concept matrix, i.e., by checking to which extent the entries of

²⁰ If properties were defined as fuzzy sets, this would correspond to simply computing the respective membership values.

the observation matrix are smaller than the corresponding entries of the concept matrix. Overall, this membership computations involves multiple complicated transformations. Therefore, checking for concept membership is not as intuitively graspable in Rickard's formalization as the geometrical inclusion used by most other works (including our own). We think that this additional complexity is the main drawback of Rickard's formalization.

One can also criticize Rickard's assumption that different properties from the same domain cannot co-occur. This assumption may be valid when using crisp sets for defining properties. However, if we represent properties with fuzzy sets, we can easily imagine that the fuzzy borderline regions of neighboring properties such as GREEN and YELLOW overlap. In this case, a point lying in this borderline region would receive a partial membership to both properties. We think, however, that this limitation of Rickard's formalization can be overcome: If we allow properties from the same domain to co-occur, then the parts of the matrix C that used to correspond to identity matrices might also contain non-zero elements off the diagonal.

Co-occurrence is limited to properties from different domains

Overall, the work by Rickard is the only formalization from the literature that explicitly represents correlations between domains. However, the representation used in this formalization makes the simple task of computing the membership of an observation to a concept quite complicated. We therefore think that our geometric representation of correlations is superior, especially since it allows us to treat properties and concepts in the same way.

Summary

2.5.3 Comparison to Lewis and Lawry

Let us now discuss the formalization of Lewis and Lawry [253]. As stated in Section 2.4.2, they define properties to be random sets within a given domain. A *random set* is characterized by a set P of prototypical points and a probability distribution δ of the distance threshold ϵ . Each point x in the underlying space is assigned a membership value to the random set based on the probability that it is closer to the prototypical set P than the threshold ϵ :

Random sets

$$\mu(x) = \mathbb{P}(d(x, P) \leq \epsilon) = \int_{d(x, P)}^{\infty} \delta(\epsilon) d\epsilon \quad (2.1)$$

Lewis and Lawry do not provide any strict constraints on the prototypical region P and the probability distribution δ . If we identify P with a core S from our definition, then we can obtain our definition of a concept by using $\delta(\epsilon) = c \cdot e^{-c\epsilon}$, which is a probability density function on the interval $[0, \infty)$, since all values are non-negative and its integral equals one. By inserting this definition into Equation 2.1, we get:

Relation to our membership function

$$\mu(x) = \mathbb{P}(d(x, S) \leq \epsilon) = \int_{d(x, S)}^{\infty} \delta(\epsilon) d\epsilon = \int_{d(x, S)}^{\infty} c \cdot e^{-c\epsilon} d\epsilon$$

$$\begin{aligned}
&= [-e^{-c \cdot \epsilon}]_{\epsilon=d(x,S)}^{\epsilon \rightarrow \infty} = 0 - (-e^{-c \cdot d(x,S)}) = e^{-c \cdot d(x,S)} \\
&= e^{-c \cdot \min_{y \in S} d(x,y)} = \max_{y \in S} (e^{-c \cdot d(x,y)})
\end{aligned}$$

Our formalization as a special case

This is equivalent to the definition of $\mu(x)$ in Definition 2.14 if we assume that $\mu_0 = 1$ and that the combined distance d_C is used. If we only consider properties, i.e., regions within a single domain, our formalization is thus a special case of the proposal by Lewis and Lawry. In the following, we can therefore use the properties from Tables 2.3 and 2.4 for illustrating their definition of concepts.

The combination space

In order to represent concepts, Lewis and Lawry construct a binary *combination space*: From each domain δ , a single property \tilde{S}_δ is selected. This property is used to define a binary dimension z_δ in the combination space, where a value of 1 indicates that the respective property is present, whereas a value of 0 encodes the absence of this property. Since properties are represented by random sets, any given observation will match any given property only with a certain probability (which is expressed by the membership function described above). So in turn, observations will not map onto points in the combination space, but to probability distributions in this space. It is important to emphasize that the only possible values on the dimensions of the combination space are zero and one. If a given observation x_δ has a membership value of $\mu(x_\delta) = 0.7$ to a given property S_δ , this is therefore *not* translated to a coordinate value of 0.7 on the respective dimension z_δ in the combination space. It is rather reflected by specifying that $\mathbb{P}(z_\delta = 1 \mid x_\delta) = 0.7$ and $\mathbb{P}(z_\delta = 0 \mid x_\delta) = 0.3$

Concepts as random sets in the combination space

Lewis and Lawry now define concepts as random sets in this combination space, i.e., by a prototypical point and a probability distribution of the concept's threshold ϵ . In order to compute the membership of a given observation x in a given concept α , Lewis and Lawry propose to use the following formula:

$$\mu_\alpha(x) = \sum_z \left(\mu_\alpha(z) \cdot \prod_{\delta \in \Delta} \mathbb{P}(z_\delta \mid x_\delta) \right)$$

Interpretation

For each possible point z in the combination space, we multiply its membership to the concept α (which can be computed with Equation 2.1) with the probability of observing this point z . This probability is based on the membership values of the observation x_δ in the respective properties z_δ for the different domains δ .

Concept combination as weighted sum

Lewis and Lawry show that under some assumptions concepts can be described as weighted sums of properties and that concept combination can also be formalized as weighted sums of concepts. They illustrate how their mathematical formalization is capable of reproducing some effects from the psychological concept combination literature.

Fruit space example

In our fruit space example, we have to pick a single property from each domain in order to define the combination space. Let us choose

\tilde{S}_{RED} and \tilde{S}_{SWEET} . Then, the concept α_{APPLE} can be defined using the prototype $(1, 1)$ in this combination space (i.e., both properties are required to be present), and a distribution of the concept's threshold with $\delta(\epsilon) = c \cdot e^{-c \cdot \epsilon}$. For this example, we pick $c = 2$ and assume that both dimensions in the combined space are weighted equally.

If we make an observation $x = (0.8, 0.2, 0.6)$ ²¹, one can easily see that $\mu_{\tilde{S}_{\text{RED}}}(x) = \mu_{\tilde{S}_{\text{SWEET}}}(x) = 1$, since this point lies inside the cores S_{RED} and S_{SWEET} . Therefore, the observation x is translated into a point $z = (1, 1)$ in the combination space with a probability of one. As this is equivalent to the prototype of the `APPLE` concept, we get that $\mu_{\alpha_{\text{APPLE}}}(x) = 1$. Hence, x is a perfect example of the `APPLE` concept.

However, for $x' = (0.2, 0.6, 0.5)$, which lies in the core S_{GREEN} and between the cores S_{SWEET} and S_{SOUR} , we get $\mu_{\tilde{S}_{\text{RED}}}(x') \approx 2.06 \cdot 10^{-9}$ and $\mu_{\tilde{S}_{\text{SWEET}}}(x') \approx 0.2158$. Therefore, the different vectors z' in the combination space have the following probabilities:

$$\begin{aligned} \mathbb{P}((0, 0) | x') &= (1 - \mu_{\text{RED}}(x')) \cdot (1 - \mu_{\text{SWEET}}(x')) \approx 0.7842 \\ \mathbb{P}((0, 1) | x') &= (1 - \mu_{\text{RED}}(x')) \cdot \mu_{\text{SWEET}}(x') \approx 0.2156 \\ \mathbb{P}((1, 0) | x') &= \mu_{\text{RED}}(x') \cdot (1 - \mu_{\text{SWEET}}(x')) \approx 1.62 \cdot 10^{-9} \\ \mathbb{P}((1, 1) | x') &= \mu_{\text{RED}}(x') \cdot \mu_{\text{SWEET}}(x') \approx 4.45 \cdot 10^{-10} \end{aligned}$$

For the membership of x' in α_{APPLE} we now get the following result, computing $\mu_{\alpha_{\text{APPLE}}}$ according to Equation 2.1:

$$\begin{aligned} \mu_{\alpha_{\text{APPLE}}}(x) &= \mathbb{P}((0, 0) | x') \cdot \mu_{\alpha_{\text{APPLE}}}((0, 0)) \\ &\quad + \mathbb{P}((0, 1) | x') \cdot \mu_{\alpha_{\text{APPLE}}}((0, 1)) \\ &\quad + \mathbb{P}((1, 0) | x') \cdot \mu_{\alpha_{\text{APPLE}}}((1, 0)) \\ &\quad + \mathbb{P}((1, 1) | x') \cdot \mu_{\alpha_{\text{APPLE}}}((1, 1)) \\ &\approx 0.7852 \cdot 0.0591 + 0.2156 \cdot 0.1353 \\ &\quad + 1.62 \cdot 10^{-9} \cdot 0.1353 + 4.45 \cdot 10^{-10} \cdot 1.0000 \\ &\approx 0.0756 \end{aligned}$$

Since we were forced to take only a single property from each domain for defining the `APPLE` concept, we cannot represent that apples are either `RED` and `SWEET` (like x) or `GREEN` and somewhat `SOUR` (like x'). There are two potential remedies for this problem:

On the one hand, we can simply define the combination space based on *all* properties from *all* domains and allow concepts to have multiple prototypical points in this combination space. This would, however, be a relatively strong modification of the work by Lewis and Lawry. It is especially unclear whether their theoretical results regarding concept combination would still hold under these conditions.

On the other hand, one could define two separate concepts `REDAPPLE` and `GREENAPPLE` and combine them disjunctively. This approach is somewhat reminiscent of our own formalization, where the individual cuboids can be interpreted as sub-concepts, whose union makes up the

A red and sweet apple has a high membership

A green and sour apple has a low membership

Combining the individual probabilities

No encoding of correlations

Taking all properties from all domains

Representing correlations with sub-concepts

21 Please recall that the dimensions are ordered as follows: $d_{\text{HUE}}, d_{\text{SOUR}}, d_{\text{SWEET}}$.

crisp core of the overall concept (cf. Section 2.3.2). Lewis and Lawry propose to use higher-level combination spaces for combining concepts (i.e., two-dimensional spaces based on the memberships to the two original concepts). However, they again are limited to a single prototypical point for defining a combined concept. This limits their formalization to conjunctive concept combinations, since a disjunctive combination of concepts would require at least two prototypical points, namely $(0, 1)$ and $(1, 0)$ (and arguably also $(1, 1)$ in some cases). A potential workaround for this would be to choose $(0, 0)$ as a prototypical point and to invert the membership function. This, however, would be again a major departure from their original proposal.

Summary

Although their formalization is not capable of representing correlations, Lewis and Lawry are able to reproduce a variety of psychological findings on conjunctive concept combination. Moreover, their usage of random sets seems to be a fairly general approach for defining membership functions (including our own membership function as a special case)– an aspect that will resurface in Chapter 7, where we will discuss several membership functions from a machine learning perspective.

2.6 SUMMARY

*Lessons learned:
fuzzified star-shaped
sets ...*

In this chapter, we proposed a new formalization of the conceptual spaces framework. We aimed to geometrically represent correlations between domains, which led us to consider the more general notion of star-shapedness instead of Gärdenfors' favored constraint of convexity. We defined concepts as fuzzy sets based on intersecting cuboids and a similarity-based membership function.

*... and an
open-source
implementation*

Moreover, we introduced the implementation of this formalization, which is publicly available and can be used by any researcher interested in conceptual spaces. We think that our implementation can be a good foundation for practical research on conceptual spaces and that it will considerably facilitate research in this area.

*Open ends:
part-whole
structures, ...*

Although we argued that our formalization is superior to any of its predecessors, there are certain ways in which it can be further improved: We have sketched in Section 2.4.3 how part-whole structures could be included in our formalization, but we have not given a full mathematical treatment of this topic. Future work could consist in formalizing Fiorini's proposal [158] in the context of our approach.

... salience weights, ...

Moreover, we have not specified how conceptual regions and salience weights can be obtained. We will consider this problem to some extent in Chapter 7. Future research could compare different ways of deriving salience weights for existing conceptual regions and their respective advantages and shortcomings. As mentioned in Section 2.3.3, the work by Sileno et al. [370] on contrast in conceptual spaces seems to be a promising avenue of research.

... and typicality

Finally, our formalization currently does not make any difference

between concept membership and typicality. However, as for instance Hampton [188] has argued, a given observation can be a full member of a concept without being very typical (e.g., a penguin is definitely a BIRD, but not a very typical one). Another possible extension of our formalization would thus be the definition of a typicality function, which could for instance be based on the distance to the concept's central area P . This could to some extent be based on the egg-yolk model proposed by Sileno et al. [370].

So far, we are limited to *representing* concepts – with the formalization laid out in this chapter we are not yet able to manipulate and combine different concepts, or talk about their relations to each other. This is the content of Chapters 3 and 4, respectively.

Outlook

3 | OPERATIONS FOR COMBINING CONCEPTS

- 3.1 Intersection 102
 - 3.1.1 Intersection of Cores 103
 - 3.1.2 Intersection of Concepts 107
 - 3.1.3 Implementation and Example 109
- 3.2 Union 116
 - 3.2.1 Definition 116
 - 3.2.2 Implementation and Example 118
- 3.3 Negation 122
- 3.4 Subspace Projection 124
 - 3.4.1 Definition 125
 - 3.4.2 Implementation and Example 126
- 3.5 Axis-Parallel Cut 127
 - 3.5.1 Definition 131
 - 3.5.2 Implementation and Example 132
- 3.6 Supported Applications 135
 - 3.6.1 Concept Formation 135
 - 3.6.2 Concept Combination 136
- 3.7 Comparison to Other Formalizations 142
- 3.8 Summary 145

Any representation of concepts is only useful if it can be applied in cognitive tasks. This requires that there are certain operations which can be applied to these concepts. When giving our definition of concepts as fuzzy star-shaped sets in Chapter 2, we have already provided a way of determining the membership of a given observation x in a given concept \tilde{S} , namely, by evaluating the concept's membership function $\mu_{\tilde{S}}(x)$. This is, however, clearly not sufficient, since we also need ways to modify, create, and combine concepts. In this chapter, we provide mathematical definitions for several operations aiming at creating new concepts based on existing ones.

Motivation

Some conjunctive concept combinations such as GREEN BANANA can be represented by intersecting the conceptual regions of GREEN and BANANA. In Section 3.1, we propose a method for ensuring that the intersection of two star-shaped regions is again star-shaped. This method is then used in conjunction with numerical optimization methods to define an intersection operation for concepts.

Intersection

Just as multiple observations can give rise to a concept such as ORANGE, a set of concepts from the same abstraction level (e.g., ORANGE, LEMON, and GRAPEFRUIT) can be used to construct higher-level categories (e.g., CITRUS FRUIT). In Section 3.2, we provide a definition of the union operation in order to enable concept creation processes of this type.

Union

Negation Since intersection and union can be related to logical conjunction and disjunction, it seems worthwhile to also provide a geometric definition for the logical negation. In Section 3.3, we discuss that the set complement as most obvious candidate in our geometrical setting can, however, not result in valid conceptual regions.

Projection When using concepts in a given cognitive task, it may sometimes be necessary to focus on some specific subset of domains. For instance, when seeing only the shadow of an object, the information from the COLOR domain is irrelevant and can be ignored. The salience weights introduced in Section 2.3.3 can be used to reflect this to some extent, but they cannot be set to zero. In Section 3.4, we therefore introduce an operation for projecting a given concept onto a subset of its domains.

Cut Finally, we consider the need for splitting a concept into two sub-concepts. This may be necessary if a given concept is too general for the given task and needs to be divided into finer-granular subordinate concepts. In Section 3.5, we consider the most straightforward case where a concept is split into two parts based on a threshold with respect to a single dimension. For example, the concept CITRUS FRUIT could be split based on the SWEETNESS dimension in order to distinguish the sub-concept ORANGE from the sub-concept LEMON.

*Examples,
application scenarios,
and related work*

For each of these operations, we provide a formal mathematical definition, which is accompanied by illustrative examples based on the fruit space from Section 2.3.4. We furthermore illustrate the usefulness of these operations by sketching two application scenarios (namely, concept formation and concept combination) in Section 3.6. In Section 3.7, we compare our approach to other formalizations of the conceptual spaces framework, before concluding this chapter in Section 3.8.

The research contributions presented in this chapter have previously been published in [41, 42, 46].

3.1 INTERSECTION

*Intersection as
logical conjunction*

The intersection of two concepts can be interpreted as the geometric equivalent of the logical conjunction: Intersecting GREEN with BANANA should result in a concept for GREEN BANANA. If concepts are represented by convex regions, then the intersection of two concepts is guaranteed to be convex as well. However, as we will see in Section 3.1.1, the intersection of two star-shaped regions is not necessarily star-shaped. We therefore propose a repair mechanism in order to ensure that the resulting region is a valid core. In Section 3.1.2, we then also specify how to determine the remaining parameters μ_0 , c , and W of the intersection result. Finally, we describe our implementation of the intersection in Section 3.1.3 and provide an illustrative example using the fruit space from Section 2.3.4.

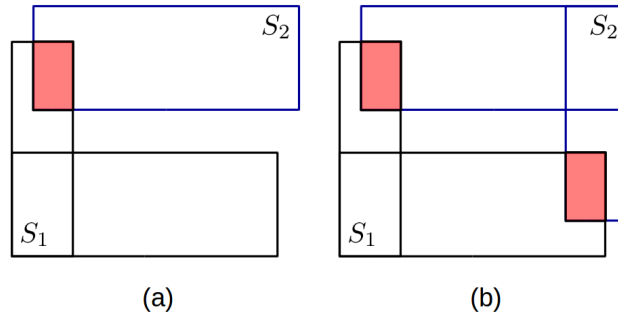


Figure 3.1: (a) The intersection of two cores is not necessarily star-shaped or even connected. (b) $P_1 \cap P_2 \neq \emptyset$ is not a necessary condition for a star-shaped intersection result.

3.1.1 Intersection of Cores

Let us first attempt to define the intersection of two cores (i.e., crisp sets) before generalizing this to concepts (i.e., fuzzy sets). One can easily see that the intersection of two star-shaped sets is not necessarily star-shaped (see e.g., Figure 3.1a). The intersection of two cores is therefore not necessarily a valid core. However, we can show the following:

Intersection of cores is not necessarily star-shaped

Lemma 3.1 (Intersection of Cores is Union of Cuboids)

Let $S_1 = \langle \Delta_{S_1}, \{C_1^{(1)}, \dots, C_{m_1}^{(1)}\} \rangle$ and $S_2 = \langle \Delta_{S_2}, \{C_1^{(2)}, \dots, C_{m_2}^{(2)}\} \rangle$ be two cores. Then $S = S_1 \cap S_2$ can be written as union of cuboids, namely, $S = \bigcup_{i \in I} C_i$.

Proof. See Appendix B.1. □

In order for the intersection result to be a valid core, its cuboids also need to have a nonempty intersection. One can easily show that the following condition is sufficient:

When does the intersection result in a valid core?

Lemma 3.2 (Sufficient Condition for Star-Shaped Intersection Result)

Let $S_1 = \langle \Delta_{S_1}, \{C_1^{(1)}, \dots, C_{m_1}^{(1)}\} \rangle$ and $S_2 = \langle \Delta_{S_2}, \{C_1^{(2)}, \dots, C_{m_2}^{(2)}\} \rangle$ be two cores with central regions P_1 and P_2 , respectively. Let furthermore $S = S_1 \cap S_2 = \bigcup_{i \in I} C_i$, where C_i are the cuboids of the intersection result. If $P_1 \cap P_2 \neq \emptyset$, then $P = \bigcap_{i \in I} C_i \neq \emptyset$

Proof. See Appendix B.1. □

Corollary 3.1 (Intersection of Orthogonal Cores)

If two cores S_1 and S_2 are defined on completely different domains (i.e., $\Delta_{S_1} \cap \Delta_{S_2} = \emptyset$ and therefore $D_{S_1} \cap D_{S_2} = \emptyset$), then $P_1 \cap P_2 \neq \emptyset$ and $S_1 \cap S_2$ is a core.

While $P_1 \cap P_2 \neq \emptyset$ is a sufficient condition for $S_1 \cap S_2$ being star-shaped, it is not a necessary condition.¹ Consider for example the two cores shown in Figure 3.1b: Their intersection consists of a single cuboid, which means that it is star-shaped by definition. However, the central regions of the original cores do not intersect.

Sufficiency and necessity

¹ Thanks to Martha Lewis for pointing this out to me.

The case of empty
central regions

In general, it might happen that $S = S_1 \cap S_2$ has an empty central region P and is therefore not star-shaped and consequently not a valid core. This is for example illustrated in Figure 3.1a. In order to ensure that the intersection of two cores results in a valid core (i.e., in a star-shaped set based on cuboids), we thus need to apply some sort of "repair mechanism" in these cases.

Two principled repair
mechanisms

In principle, there are three ways of ensuring star-shapedness, namely, by removing cuboids, by extending cuboids, and by treating the intersection result as multiple independent cores.

Removing cuboids
causes
underextensions

If cuboids are removed, the result is still guaranteed to be a proper subset of both original cores. However, there are points that belong to both cores and are not included in the intersection result. This effect is called *underextension* in the psychological literature [298, Chapter 12] and reported only infrequently for conjunctive concept combination [187]. Moreover, removing cuboids might involve some arbitrary choices. For instance, in Figure 3.1a, it is unclear which of the two cuboids should be removed from the intersection result.

Extending cuboids
causes
overextensions

On the other hand, if cuboids are extended, then all points that belong to both cores also belong to the intersection results. However, the intersection result also may contain points that did not belong to one or both of the original cores. This effect is called *overextension* in the psychological literature [298, Chapter 12] and much more common than underextensions in the context of concept combination [187]. In Section 1.1.2, we gave the example of CHESS not being a SPORT, but a SPORT WHICH IS A GAME. Also the PET FISH example from Section 1.2.2 can be seen in this context.

Returning multiple
cores may be
unintuitive

Finally, returning multiple cores (each one potentially only based on a single cuboid) as an intersection result ensures that we only return star-shaped sets. However, it may be unintuitive to receive multiple results from a single intersection operation, especially if these results are geometrically speaking not connected.

Using overextension

For our current purpose, we would like to ensure that a single star-shaped set is returned as a result. We choose to extend cuboids rather than removing them in order to be more in line with the psychological literature. When merging multiple cuboids C_1, \dots, C_m (all defined on Δ_S or equivalently D_S) into a single core, we furthermore aim for an extension of the cores that minimizes the size of the resulting core.

Minimal extension
by including a
central point

It is intuitively clear that requiring the cuboids C_i to contain a region P^* leads to a larger resulting core than requiring them to meet in a single point $p^* \in CS$. We can obtain the extended version C'_i for every cuboid C_i by defining its new support points like this based on p^* :

$$\forall d \in D : p_{id}^{-'} = \min(p_{id}^-, p_d^*), \quad p_{id}^{+'} = \max(p_{id}^+, p_d^*)$$

The intersection of the resulting C'_i contains at least p^* , i.e., it is not empty. This means that the resulting set $S' = \bigcup_{i \in I} C'_i$ is a core.

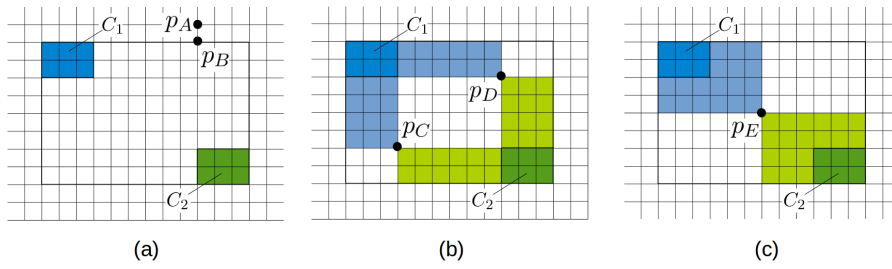


Figure 3.2: Finding the minimal star-shaped hull of two cuboids C_1 and C_2 . (a) A point p_A outside of the bounding cuboid is dominated by another point p_B on the border of the bounding cuboid. (b) Two optimal solutions p_C and p_D which minimize the resulting volume. (c) Using p_E , the midpoint of the cuboids' centers.

We will now argue that there exists such a point p^* which minimizes the size of the resulting core. For the sake of simplicity, we confine our argument to two cuboids C_1 and C_2 . For this purpose, let us define a size function $f(p)$ that returns for each potential midpoint p the size of the resulting core:

$$f(p) = f(C_1, p) + f(C_2, p) - f(C_1 \cap C_2, p)$$

$$\text{with } f(C, p) = \prod_{d \in D} (\max(p_d^+, p_d) - \min(p_d^-, p_d))$$

The size $f(p)$ of the resulting core is computed based on the inclusion-exclusion formula $|A \cup B| = |A| + |B| - |A \cap B|$, using the sizes $f(C, p)$ for a cuboid C that has been extended to include the point p . Figure 3.2a shows two cuboids C_1 and C_2 and also two candidate points p_A and p_B . As one can easily see, the resulting volume of using p_A is larger than the resulting volume of using p_B , i.e., $f(p_A) > f(p_B)$. This is a property that holds true in general: For any point outside of the bounding cuboid around all given cuboids (i.e., the cuboid with $p_d^- = \min_{i=1}^m p_{id}^-$ and $p_d^+ = \max_{i=1}^m p_{id}^+$), we can find a point on the surface of this bounding cuboid that results in a smaller size of the resulting core. So if there is a point p^* that minimizes the size of the resulting star-shaped set, it cannot lie outside of the bounding cuboid.

Moreover, $f(p)$ is a continuous function, because all operations used for computing $f(p)$ (i.e., maximum, minimum, subtraction, multiplication, and addition) are continuous. Therefore, the extreme value theorem (see e.g., [153]) tells us that there is at least one point p^* in the bounding cuboid (which is a compact set) for which $f(p)$ is minimized.

In general, we do, however, not know whether p^* is unique. Figure 3.2b shows two points $p_C \neq p_D$ with $f(p_C) = f(p_D) = 36$. Both points are optimal solutions as there is no other point p' with $f(p') < 36$. Since we would like to return a single core as intersection result, we are now forced to arbitrarily pick one of the two minima for defining the intersection result. This arbitrary choice heavily influences the shape of the resulting core, which is somewhat unsatisfactory.

Existence of an optimal solution

Interpretation of $f(p)$

Applying the extreme value theorem

Uniqueness of the optimal solution

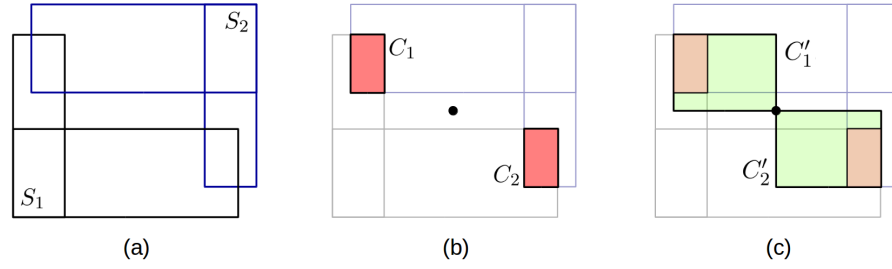


Figure 3.3: (a) Two cores S_1 and S_2 . (b) Their standard set intersection $S = S_1 \cap S_2$ and the midpoint p of this intersection's cuboids. (c) Extended cuboids C'_1 and C'_2 , resulting in S' .

Picking the central point heuristically

An alternative to this optimization-based approach uses a heuristic for identifying a central point p . This heuristic will most likely not give an optimal result, but it returns a unique and well-defined solution that is easy to compute. We propose to use the weighted average of the cuboid's centers as such a heuristic:

$$p = \frac{\sum_{i=1}^m \omega_i \cdot \frac{(p_i^- + p_i^+)}{2}}{\sum_{i=1}^m \omega_i}$$

Weights of the heuristic

If the weights ω_i are all set to 1, this results in an unweighted average. This case is illustrated in Figure 3.2c, where the heuristically chosen point p_E causes the resulting core to have a size of $f(p_E) = 48$. On the other hand, the weights ω_i can be defined based on the size of the respective cuboid C_i . If the weights are inversely proportional to the size of the cuboid, the central point is closer to smaller cuboids than to larger ones, causing larger cuboids to grow more than smaller cuboids. Conversely, if larger cuboids receive larger weights, the central point lies closer to the larger cuboids, causing smaller cuboids to grow more than larger cuboids.

Modified intersection of cores

In our formalization, we use the following definition of the *modified intersection* of cores based on the proposed heuristic with identical weights for all cuboids:

Definition 3.1 (Modified Intersection of Cores)

Let $S_1 = \langle \Delta_{S_1}, \{C_1^{(1)}, \dots, C_{m_1}^{(1)}\} \rangle$ and $S_2 = \langle \Delta_{S_2}, \{C_1^{(2)}, \dots, C_{m_2}^{(2)}\} \rangle$ be two cores. Let furthermore be $S_1 \cap S_2 = \bigcup_{i \in I} C_i$ a cuboid-based representation of their set intersection. Their modified intersection $I(S_1, S_2) = \langle \Delta_{S_1} \cup \Delta_{S_2}, \{C'_1, \dots, C'_{m'}\} \rangle$ is based on the cuboids C'_i , which have been obtained from the corresponding cuboids C_i using the heuristic to obtain a central point p :

$$p = \frac{1}{m'} \cdot \sum_{i=1}^m \frac{(p_i^- + p_i^+)}{2}$$

$$\forall d \in D : p_{id}^- = \min(p_{id}^-, p_d), \quad p_{id}^+ = \max(p_{id}^+, p_d)$$

It is trivial to see that $I(S_1, S_2)$ is a valid core. Figure 3.3 visualizes the modified intersection.

Please note that the modified intersection operation is no longer associative, so in general $I(I(S_1, S_2), S_3) \neq I(S_1, I(S_2, S_3))$. This loss of associativity is caused by our proposed repair mechanism. If we want to intersect more than two cores, we should therefore first intersect all cuboids from all cores, and apply the repair mechanism only once in the end in order to arrive at a well-defined result.

Loss of associativity

We would also like to point out that the repair mechanism of extending cuboids is only applied if the standard set intersection does not result in a valid core. Moreover, both the standard set intersection and the repair mechanism are always computed on the domains $\Delta_{S'} = \Delta_{S_1} \cup \Delta_{S_2}$, i.e., with respect to the dimensions $D_{S'} = D_{S_1} \cup D_{S_2}$.

Final notes

3.1.2 Intersection of Concepts

Now let us look at the intersection of fuzzy *concepts*. Every concept \tilde{S} is defined based on a core S and additional parameters that control its fuzziness, namely, the sensitivity parameter c , the domain and dimension weights W , and the highest possible membership value μ_0 . If we want to express the result of intersecting two concepts \tilde{S}_1, \tilde{S}_2 as another concept \tilde{S}' , we thus need to both construct a new core S' and to infer new parameter settings μ'_0, c', W' . We propose the following definition for the intersection of concepts:

Intersection of concepts

Definition 3.2 (Modified Intersection of Concepts)

Let $\tilde{S}_1 = \langle S_1, \mu_0^{(1)}, c^{(1)}, W^{(1)} \rangle$ and $\tilde{S}_2 = \langle S_2, \mu_0^{(2)}, c^{(2)}, W^{(2)} \rangle$ be two concepts. We define their modified intersection as $\tilde{S}' = I(\tilde{S}_1, \tilde{S}_2) = \langle S', \alpha', c', W' \rangle$ with the following components:

- $\alpha' = \max \left\{ \alpha \in [0, 1] : \tilde{S}_1^\alpha \cap \tilde{S}_2^\alpha \neq \emptyset \right\}$
- $S' = I \left(\tilde{S}_1^{\alpha'}, \tilde{S}_2^{\alpha'} \right)$
- $c' = \min (c^{(1)}, c^{(2)})$
- W' with weights w'_δ, w'_d defined as follows (using interpolation factors $s, t \in [0, 1]$):

$$\begin{aligned} \forall \delta \in \Delta_{S_1} \cap \Delta_{S_2} : & \left(\left(w'_\delta = s \cdot w_\delta^{(1)} + (1-s) \cdot w_\delta^{(2)} \right) \right. \\ & \left. \wedge \forall d \in \delta : \left(w'_d = t \cdot w_d^{(1)} + (1-t) \cdot w_d^{(2)} \right) \right) \\ \forall \delta \in \Delta_{S_1} \setminus \Delta_{S_2} : & \left(\left(w'_\delta = w_\delta^{(1)} \right) \wedge \forall d \in \delta : \left(w'_d = w_d^{(1)} \right) \right) \\ \forall \delta \in \Delta_{S_2} \setminus \Delta_{S_1} : & \left(\left(w'_\delta = w_\delta^{(2)} \right) \wedge \forall d \in \delta : \left(w'_d = w_d^{(2)} \right) \right) \end{aligned}$$

Let us take a look at the individual components of Definition 3.2. We first identify the largest value of α for which the α -cuts of \tilde{S}_1 and \tilde{S}_2 have a nonempty intersection. Since the membership values are always positive, we know that such an α' must exist. Moreover, both α -cuts

Interpretation of the definition

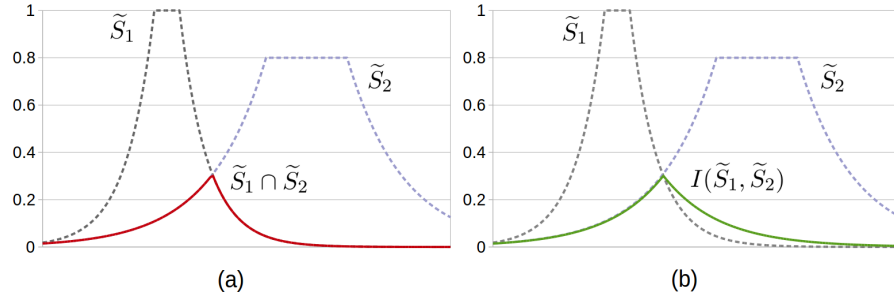


Figure 3.4: (a) The intersection $\mu_{\tilde{S}_1 \cap \tilde{S}_2}(x) = \min(\mu_{\tilde{S}_1}(x), \mu_{\tilde{S}_2}(x))$ of two concepts \tilde{S}_1 and \tilde{S}_2 . (b) Our modified intersection $I(\tilde{S}_1, \tilde{S}_2)$.

$\tilde{S}_1^{\alpha'}$ and $\tilde{S}_2^{\alpha'}$ are guaranteed to be star-shaped sets (cf. Lemma 2.4 and Proposition 2.1 from Section 2.3.3).

Obtaining a new core

We then use the modified intersection from Definition 3.1 in order to obtain a new core S' for the result of our fuzzy intersection. Both the repair mechanism and the final representation of this modified intersection of cores require cuboids as building blocks, but neither $\tilde{S}_1^{\alpha'}$ nor $\tilde{S}_2^{\alpha'}$ nor their intersection can necessarily be represented as a union of cuboids. Therefore, the result of their standard set intersection needs to be approximated with bounding boxes. We will introduce the details of obtaining both α' and S' in Section 3.1.3.

The sensitivity parameter

When combining two somewhat imprecise concepts, the result should not be more precise than any of the original concepts. As the sensitivity parameter c is inversely related to fuzziness, we define $c' = \min(c^{(1)}, c^{(2)})$.

Merging the weights

Finally, we combine the domain and dimensions weights of the two original concepts. If a weight is defined for both sets, we take a convex combination, and if it is only defined for one set, we simply copy it. The importance of each domain and dimension to the new concept thus lies somewhere between its importance with respect to the two original concepts. In some cases, the normalization constraint of the resulting domain weights (i.e., $\sum_{\delta \in \Delta_{S'}} w_\delta = |\Delta_{S'}|$ and $\forall \delta \in \Delta_{S'} : \sum_{d \in \delta} w_d = 1$, cf. Definition 2.7) might be violated after computing the convex combination. We therefore manually normalize the weights in order to enforce these constraints.

Our modified intersection and the standard intersection of fuzzy sets

Please note that in the standard fuzzy set theory, the intersection of two fuzzy sets \tilde{A} and \tilde{B} is defined by using the minimum over their membership functions:

Definition 3.3 (Fuzzy Intersection)

Let \tilde{A}, \tilde{B} be two fuzzy sets on CS . Then, the membership function of their fuzzy intersection is defined as follows:

$$\forall x \in CS : \mu_{\tilde{A} \cap \tilde{B}}(x) = \min(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x))$$

Figure 3.4 illustrates with two one-dimensional concepts that $\tilde{S}_1 \cap \tilde{S}_2$ yields slightly different results than our modified intersection $I(\tilde{S}_1, \tilde{S}_2)$.

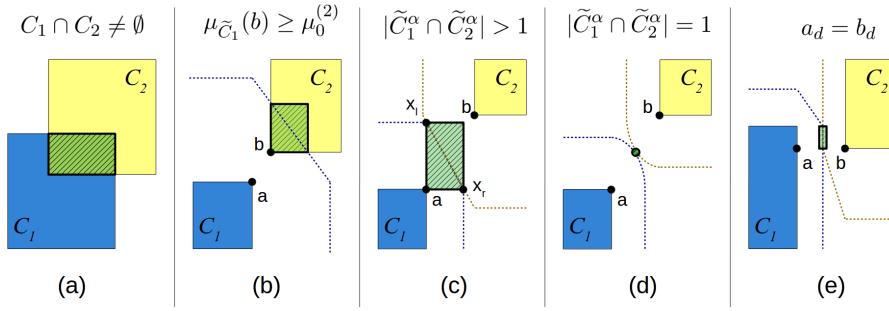


Figure 3.5: Possible results of intersecting two fuzzy cuboids.

This difference is caused by the repair mechanism as well as the aggregation of the sensitivity parameter c and the weights W .

The maximally attainable membership to the resulting concept is set to the highest value of α for which the α -cuts of both sets intersect. This is the maximal degree of membership that any point in the conceptual space can have to *both* original concepts. Of course, one can manually reset μ'_0 to 1 after the intersection. This way, one can model overextension effects also for fuzzy conceptual region (e.g., a GUPPY is neither a typical PET nor a typical FISH, but a typical PET FISH – cf. [187, 309, 310, 450] and Sections 1.1.2, 1.2.2, and 3.1.1).

Fuzzy overextension

3.1.3 Implementation and Example

The key challenge with respect to the intersection of two concepts \tilde{S}_1 and \tilde{S}_2 as presented in Definition 3.2 is to find the highest value α' for which the α -cuts of \tilde{S}_1 and \tilde{S}_2 intersect. In an actual implementation, we therefore need a concrete way of obtaining α' . We simplify this problem by iterating over all combinations of cuboids $C_1 \in S_1, C_2 \in S_2$ and by looking at each pair of cuboids individually. This requires the notion of a *fuzzified cuboid* which can be defined based on Definition 2.14:

Fuzzified cuboids

Definition 3.4 (Fuzzified Cuboid)

Let $\tilde{S} = \langle S, \mu_0, c, W \rangle$ be a concept and $C \in S$ be a cuboid from its core. Then, the membership to the fuzzified cuboid \tilde{C} is defined as follows:

$$\mu_{\tilde{C}}(x) = \mu_0 \cdot \max_{y \in C} \left(e^{-c \cdot d_C^\Delta(x, y, W)} \right)$$

It is obvious that $\mu_{\tilde{S}}(x) = \max_{C \in S} \mu_{\tilde{C}}(x)$ (cf. Definition 2.14). Algorithm 3.1 shows how the intersection of two fuzzified cuboids \tilde{C}_1 and \tilde{C}_2 can be computed based on a distinction into the following cases, which are illustrated in Figure 3.5:

Computing the intersection

In the most straightforward case, the underlying crisp cuboids C_1 and C_2 have a nonempty intersection and both concepts have the same maximal membership $\mu_0^{(1)} = \mu_0^{(2)}$ (lines 1 and 2, Figure 3.5a). In this case, we simply compute their intersection as defined in Section 3.1.1. The α -value of this intersection is equal to $\mu_0^{(1)}$ and $\mu_0^{(2)}$.

First case: crisp intersection

Algorithm 3.1: Finding the highest non-empty α -cut of two fuzzified cuboids.

Input: Fuzzified cuboid \tilde{C}_1 , fuzzified cuboid \tilde{C}_2

Output: Highest value of α for which \tilde{C}_1 and \tilde{C}_2 intersect, crisp cuboid approximation C of $\tilde{C}_1^\alpha \cap \tilde{C}_2^\alpha$

```

1 if  $\mu_0^{(1)} = \mu_0^{(2)} \wedge C_1 \cap C_2 \neq \emptyset$  then
2   |  $\alpha = \min(\mu_0^{(1)}, \mu_0^{(2)})$ ,  $C = C_1 \cap C_2$ 
3 else
4   | Find closest points  $a \in C_1, b \in C_2$ 
5   | if  $\mu_{\tilde{C}_1}(b) \geq \mu_0^{(2)}$  then
6     |  $\alpha = \mu_0^{(2)}$ ,  $C =$  cuboid approximation of  $\tilde{C}_1^{\mu_0^{(2)}} \cap C_2$ 
7   | else if  $\mu_{\tilde{C}_2}(a) \geq \mu_0^{(1)}$  then
8     |  $\alpha = \mu_0^{(1)}$ ,  $C =$  cuboid approximation of  $\tilde{C}_2^{\mu_0^{(1)}} \cap C_1$ 
9   | else
10    | Find  $x^* = \arg \max_{x \in CS}(\mu_{\tilde{C}_1}(x))$  with  $\mu_{\tilde{C}_1}(x) = \mu_{\tilde{C}_2}(x)$ 
11    |  $\alpha = \mu_{\tilde{C}_1}(x^*)$ 
12    | if
13      |  $\exists t \in R : \forall d \text{ with } a_d \neq b_d : w_{\delta(d)}^{(1)} \cdot \sqrt{w_d^{(1)}} = t \cdot w_{\delta(d)}^{(2)} \cdot \sqrt{w_d^{(2)}}$ 
14      | then
15        | for  $i = 1$  to  $|\{d : a_d \neq b_d\}| - 1$  do
16          | Find all  $x$  on the  $i$ -faces of the bounding box
17          | spanned by  $a$  and  $b$  with  $\mu_{\tilde{C}_1}(x) = \mu_{\tilde{C}_2}(x) = \alpha$ 
18          | if found at least one  $x$  then
19            |  $C =$  cuboid-approximation of the set of all  $x$ 
20            | break
21          | end
22        | end
23      | else
24        |  $C =$  trivial cuboid consisting of  $x^*$ 
25      | end
26    | end
27    | Extrude  $C$  in all dimensions  $d$  where  $a_d$  and  $b_d$  can vary
28  | end
29 return  $\alpha, C$ 

```

If this is not the case, we compute the pair $a \in C_1, b \in C_2$ of closest points from the two cuboids (line 4), i.e., $\forall x \in C_1, y \in C_2 : d(a, b) \leq d(x, y)$. If there are multiple possible choices for a and b (like in Figure 3.5e), we pick a single pair of closest points and store the dimensions in which a and b may vary.

Obtaining closest points

If the μ_0 parameters of the two concepts are different and the $\mu_0^{(i)}$ -cut of \tilde{C}_j intersects with C_i (lines 5 to 8, Figure 3.5b), we need to intersect $\tilde{C}_j^{\mu_0^{(i)}}$ with C_i and approximate the result with a cuboid. The α -value of this intersection is equal to $\mu_0^{(i)}$.

Second case: α -cut intersects with crisp cuboid

In all other cases, the intersection does not involve any of the crisp cuboids. We therefore use a numerical optimization algorithm² to find a point x^* between a and b with equal maximal membership to both \tilde{C}_1 and \tilde{C}_2 (line 10). The α -value of the resulting intersection is then just the membership of x^* to any of the fuzzified cuboids (line 11). We now need to distinguish two remaining cases:

Finding one point in the α -cut intersection

On the one hand, the intersection of the two fuzzified cuboids may consist of a *set* of points (Figure 3.5c). This can only happen if the α -cut boundaries of both fuzzified cuboids are parallel to each other, which requires multiple domains to be involved and the weights of both concepts to be linearly dependent.³ This condition is checked in line 12. If this is the case, we look for additional points x on the surface of the bounding box spanned by the points a and b , such that $x \in \tilde{C}_1^\alpha$ and $x \in \tilde{C}_2^\alpha$. In Figure 3.5c, this would be the points x_l and x_r . In Algorithm 3.1, we iteratively look at the edges ($i = 1$), faces ($i = 2$), etc. of the bounding box until we find such points (lines 13-19)⁴ and then approximate the overall set of points with a cuboid.

Third case: intersection yields a set of points

In the final remaining case, the intersection of the two fuzzified cuboids consists of a single point x^* lying between a and b (line 21, Figure 3.5d). Here, we define the intersection to be a trivial cuboid with $p^- = p^+ = x^*$.

Fourth case: intersection yields a single point

Finally, in line 24 we extrude the identified cuboid in all dimensions d where both a_d and b_d can vary (cf. Figure 3.5e).

Final step: extrusion

Algorithm 3.1 provides us with a way of computing the intersection of a pair of fuzzified cuboids. We can now apply this algorithm to all pairs of cuboids $C_1 \in S_1, C_2 \in S_2$. Next, we remove all intersection results with non-maximal α . If the remaining set of cuboids has an empty intersection, we perform the repair mechanism as defined in

The intersection of two fuzzy concepts

² Our implementation uses the `scipy.optimize` package.

³ As we will discuss in Section 4.1.1, the shape of a fuzzified cuboid's α -cut depends on the metric for the underlying space: If the Euclidean metric is used, its borders have the form of an ellipse, while the Manhattan metric causes diamond-shaped corners. Parallel boundaries can thus only happen under the Manhattan metric, i.e., in the presence of multiple domains. Since the salience weights control how much the α -cut grows in each direction (and thus also how stretched the diamonds are), we furthermore need linearly dependent weights.

⁴ Again, we use numerical optimization algorithms from the `scipy.optimize` package.

Number n of Dimensions	Runtime in Milliseconds
1	1.1169
2	3.7356
4	11.4794
8	47.9072
16	263.3106
32	1737.8090

Table 3.1: Average runtime of the intersection operation (averaged across 1,000 pairs of randomly created single-cuboid concepts).

Section 3.1.1 in order to obtain a valid core S' . Based on Definition 3.2, we can then compute the remaining parameters c' and W' in order to arrive at the overall intersection result \tilde{S}' . In our implementation, we use $s = t = 0.5$ when computing the convex combination of the original weights from W_1 and W_2 .

Runtime experiments

In order to evaluate the runtime of the fuzzy intersection as defined in Definition 3.2 (which involves Algorithm 3.1 for finding both α' and S'), we have tested it with 1,000 pairs of randomly created concepts. We have ensured that the set of weights is always identical for both concepts as this is likely to trigger the case of Figure 3.5c, which involves the largest amount of computations. This way, the numbers obtained in our runtime experiments are skewed more towards the worst case. Runtime was analyzed for different numbers of dimensions n in the conceptual space and for different numbers of cuboids m per concept. For each pair of concepts, a new conceptual space with a random grouping of the dimensions into domains was created (using maximally 5 dimensions per domain). Runtime was measured on a laptop with an Intel Core i5-6440HQ CPU (2.60GHz quad-core) processor and 8 GB main memory.

Runtime and the number of dimensions

Let us first look at runtime as a function of the number n of dimensions. We can see in Table 3.1 and Figure 3.6 that runtime seems to grow in a superlinear way: Doubling the number of dimensions leads to a runtime which is more than twice as long. Moreover, this growth seems to accelerate: For example, doubling the number of dimensions from two to four leads to a runtime increase by a factor of three, while an increase from four to eight dimensions increases runtime by a factor of four. This hints at an exponential trend, which can be explained by the numerical optimization algorithms used at various points in our implementation. Increasing the number of dimensions leads to an increase in the number of free variables to optimize when attempting to find a point x^* with identical maximal membership to both concepts. This in turn can be expected to lead to an increase in the runtime of the optimization algorithm and hence our overall intersection operation. Already for a 32-dimensional space, the average runtime of an intersection is approximately 1.7 seconds. This highlights

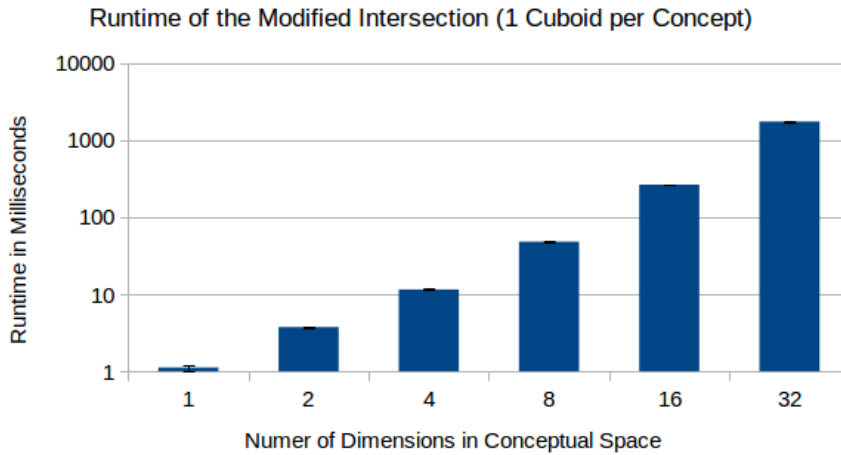


Figure 3.6: Average runtime of the intersection operation for single-cuboid concepts as a function of the number of dimensions in the conceptual space.

Number m of Cuboids	Runtime in Milliseconds
1	11.4794
2	45.5935
4	179.6660
8	708.3379

Table 3.2: Average runtime of the intersection operation (averaged across 1,000 pairs of randomly created concepts in a four-dimensional conceptual space).

that more efficient implementations or approximations are needed in order to make our formalization usable for practical applications.

Let us now investigate how the number m of cuboids per concept influences the runtime of the intersection operation. In Table 3.2 and Figure 3.7, we observe that also increasing the number of cuboids per concept considerably increases the runtime of the algorithm: Doubling the number of cuboids per concept while keeping the number of dimensions constant increases the runtime of the intersection operation by a factor of four. Interestingly, this factor seems to be relatively constant, while we observed an accelerating growth when increasing the number of dimensions. From an intuitive point of view, one may have expected an exponential trend, since we need to consider all possible pairs of cuboids when computing the intersection. Increasing the number of cuboids thus should lead to a combinatorial explosion. Such an accelerating growth of runtime may be observed in practice for even larger number of cuboids and/or higher-dimensional conceptual spaces. Since a larger number of cuboids allows for a more fine-granular representation of conceptual regions, one may need to find a reasonable trade-off between representational power and runtime considerations

Runtime and the number of cuboids

Runtime of the Modified Intersection (4 Dimensions in the Conceptual Space)

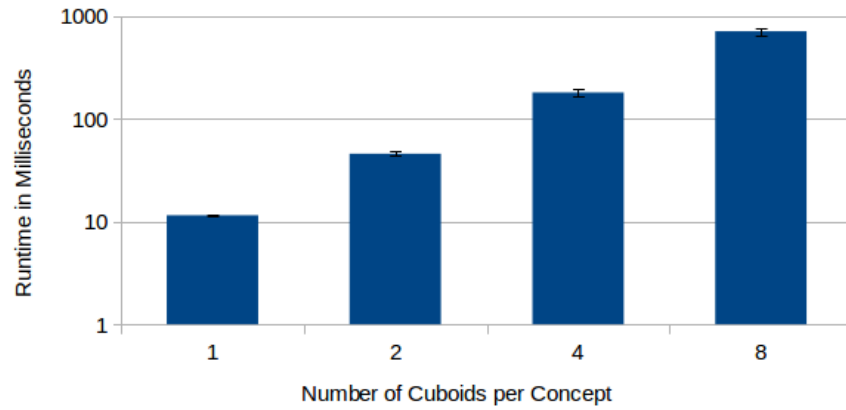


Figure 3.7: Average runtime of the intersection operation as a function of the number of cuboids per concept in a four-dimensional conceptual space.

Concept	Δ_S	p^-	p^+	μ_0	c	W		
						$w_{\delta_{\text{COLOR}}}$	$w_{\delta_{\text{SHAPE}}}$	$w_{\delta_{\text{TASTE}}}$
PEAR	Δ	(0.50, 0.40, 0.35)	(0.70, 0.60, 0.45)	1.0	24.0	0.50	1.25	1.25
APPLE	Δ	(0.50, 0.65, 0.35)	(0.80, 0.80, 0.50)	1.0	20.0	0.50	1.50	1.00
		(0.65, 0.65, 0.40)	(0.85, 0.80, 0.55)					
		(0.70, 0.65, 0.45)	(1.00, 0.80, 0.60)					

Table 3.3: Definition of the concepts APPLE and PEAR for the intersection example.

in practical applications (cf. also Sections 2.3.2, 4.1.3, and 4.5.3). Again, these observations urge for a more efficient implementation or approximation of the intersection operation in future versions of our code.

Fruit space example

In order to illustrate the intersection of concepts, let us intersect the concepts of APPLE and PEAR from the fruit space defined in Section 2.3.4. Their definition is re-printed in Table 3.3. The intersection result can be interpreted as the set of objects that fit both concepts to some degree and might be called APPLE-PEAR.

Obtaining the new core

Because $S_{\text{APPLE}} \cap S_{\text{PEAR}} = \emptyset$ and $\mu_0^{(\text{APPLE})} = \mu_0^{(\text{PEAR})} = 1$, we look for an $x^* \in CS$ with maximal equal membership in both concepts (line 10 in Algorithm 3.1). Using numerical optimization, we find $x^* = (0.500, 0.625, 0.350)$ with $\alpha = \mu_{\tilde{S}_{\text{APPLE}}}(x^*) = \mu_{\tilde{S}_{\text{PEAR}}}(x^*) \approx 0.4724$. As the weights of the two concepts are not linearly dependent, we are not in the case depicted in Figure 3.5c. However, we deal with a case like in Figure 3.5e: Because the two concepts overlap with respect to the HUE and SWEETNESS dimensions, we need to extrude our point x^* in these dimensions (line 24 of Algorithm 3.1). One can easily see that this results in $S' = \langle \Delta, \{C'\} \rangle$ with C' being defined as follows:

$$C' = \langle \Delta, p^- = (0.500, 0.625, 0.350), p^+ = (0.700, 0.625, 0.450) \rangle$$

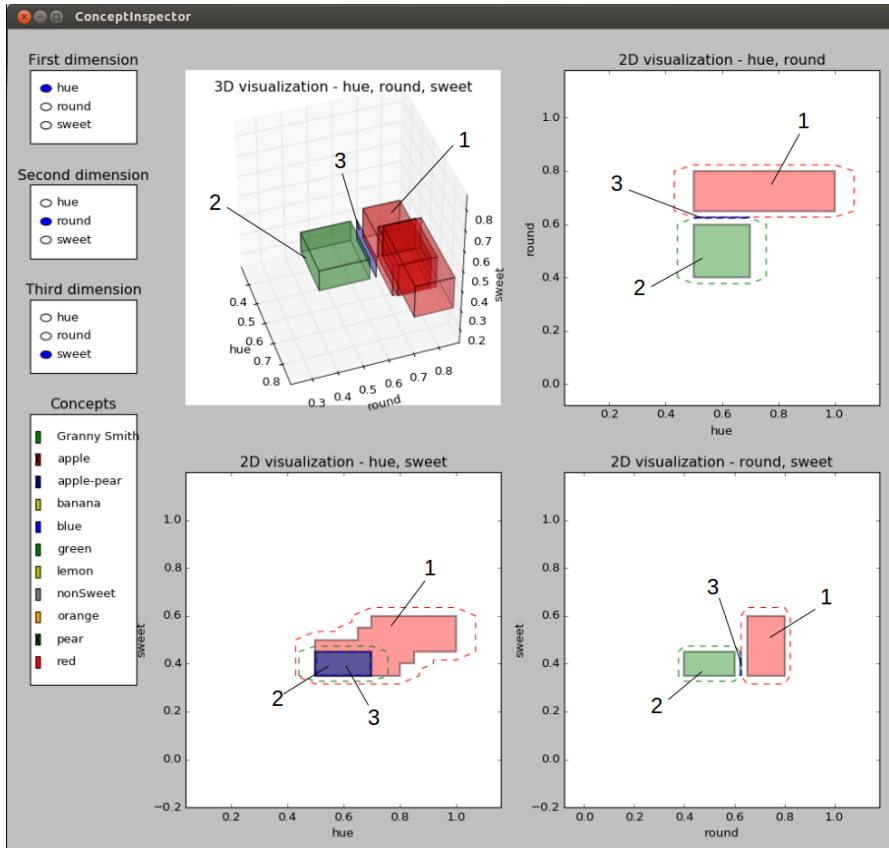


Figure 3.8: Screenshot of the ConceptInspector tool, illustrating the intersection of APPLE (1) and PEAR (2) in the three-dimensional fruit space, resulting in the APPLE-PEAR (3) concept.

Finally, we need to compute c' as well as the new weights W' . Because both concepts are defined on Δ (i.e., the whole conceptual space), we need to interpolate between the weights. In this case, we choose $s = t = 0.5$ and obtain the following results:

Sensitivity parameter and salience weights

$$c' = \min(c^{(\text{APPLE})}, c^{(\text{PEAR})}) = \min(20.00, 24.00) = 20.00$$

$$W' = \langle \{w_{\delta_{\text{COLOR}}} = 0.50, w_{\delta_{\text{SHAPE}}} = 1.375, w_{\delta_{\text{TASTE}}} = 1.125\}, W_{\text{dim}} \rangle$$

The overall result of this intersection is then $\tilde{S}' = \langle S', 0.4724, 20.00, W' \rangle$. In our implementation, this whole computation happens behind the scenes and one can obtain this result by a simple function call, whose result is illustrated in Figure 3.8:

The intersection result

```
>>> print(apple.intersect_with(pear))
core: [[0.5, 0.625, 0.35]-[0.7, 0.625, 0.45]]
mu: 0.4723665527
c: 20.0
weights: <{'color': 0.5, 'taste': 1.125, 'shape': 1.375},
{'color': {0: 1.0}, 'taste': {2: 1.0}, 'shape': {1: 1.0}}>
```

3.2 UNION

Union as logical disjunction

A union of two or more concepts is the geometric equivalent of the logical disjunction and can be used to construct higher-level concepts. For instance, the concept of CITRUS FRUIT can be obtained by computing the union of LEMON, ORANGE, ORANGE, and GRAPEFRUIT. Generally speaking, the union of any pair of conceptual regions is not necessarily connected. If conceptual regions are convex, a valid output concept can be obtained by using a convex hull of the standard set union. In our formalization, we use the repair mechanism from Section 3.1.1 in order to obtain a star-shaped hull of the given cores. We first give a mathematical definition of the union of two concepts in Section 3.2.1, before using two examples from our fruit space to illustrate the union operator in Section 3.2.2.

3.2.1 Definition

Union of cores

Let $S_1 = \langle \Delta_{S_1}, \{C_1^{(1)}, \dots, C_{m_1}^{(1)}\} \rangle$ and $S_2 = \langle \Delta_{S_2}, \{C_1^{(2)}, \dots, C_{m_2}^{(2)}\} \rangle$ be two cores. We can write the union of S_1 and S_2 as follows (where S is defined on $\Delta_S = \Delta_{S_1} \cup \Delta_{S_2}$):

$$S = S_1 \cup S_2 = \left(\bigcup_{i_1=1}^{m_1} C_{i_1}^{(1)} \right) \cup \left(\bigcup_{i_2=1}^{m_2} C_{i_2}^{(2)} \right)$$

The need for a repair mechanism

Obviously, we can represent S as a union of cuboids. This new set S is star-shaped if and only if the regions P_1 and P_2 have a nonempty intersection (i.e., there are some points that are contained in all $C_{i_1}^{(1)}$ and all $C_{i_2}^{(2)}$). In general, this is, however, not necessarily the case. Therefore, we again need to apply a repair mechanism in order to restore star-shapedness. In order to maintain consistency, we propose to use the same repair mechanism that has also been introduced for the intersection in Section 3.1.1. We denote the modified union of two cores as $S' = U(S_1, S_2)$.

Definition 3.5 (Modified Union of Cores)

Let $S_1 = \langle \Delta_{S_1}, \{C_1^{(1)}, \dots, C_{m_1}^{(1)}\} \rangle$ and $S_2 = \langle \Delta_{S_2}, \{C_1^{(2)}, \dots, C_{m_2}^{(2)}\} \rangle$ be two cores. Their modified union $U(S_1, S_2) = \langle \Delta_{S_1} \cup \Delta_{S_2}, \{C'_1, \dots, C'_{m'}\} \rangle$ is based on the cuboids C'_i , which have been obtained from the corresponding cuboids C_i from S_1 and S_2 , using the heuristic from Section 3.1.1 to obtain a central point p :

$$p = \frac{1}{m'} \cdot \sum_{i=1}^m \frac{(p_i^- + p_i^+)}{2}$$

$$\forall d \in D : p_{id}^- = \min(p_{id}^-, p_d), \quad p_{id}^+ = \max(p_{id}^+, p_d)$$

Loss of associativity

Just as the modified intersection, the modified union is not expected to be associative. If we would like to combine three or more cores with the union operator, we should therefore first collect *all* cuboids from all cores and then apply the repair mechanism only once in the end.

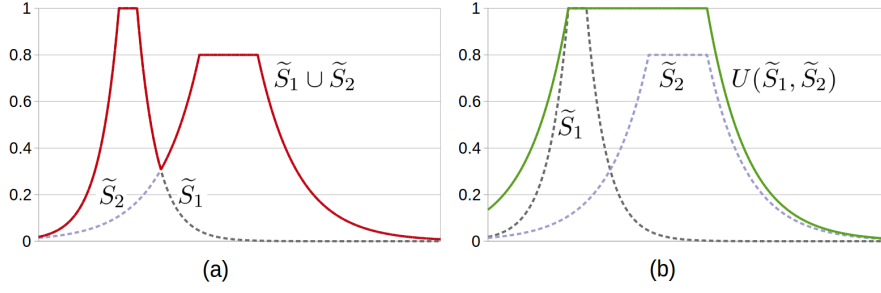


Figure 3.9: (a) The union $\mu_{\tilde{S}_1 \cup \tilde{S}_2}(x) = \max(\mu_{\tilde{S}_1}(x), \mu_{\tilde{S}_2}(x))$ of two concepts \tilde{S}_1 and \tilde{S}_2 . (b) Our modified union $U(\tilde{S}_1, \tilde{S}_2)$.

Now let us look at the union of concepts. The most straightforward approach to define the union $\tilde{S}' = U(\tilde{S}_1, \tilde{S}_2)$ of two concepts \tilde{S}_1 and \tilde{S}_2 uses the maximum over $\mu_0^{(1)}$ and $\mu_0^{(2)}$, the modified union of the cores S_1 and S_2 , and the same computations for c' and W' as the intersection operation (cf. Definition 3.2):

Union of concepts

Definition 3.6 (Modified Union of Concepts)

Let $\tilde{S}_1 = \langle S_1, \mu_0^{(1)}, c^{(1)}, W^{(1)} \rangle$ and $\tilde{S}_2 = \langle S_2, \mu_0^{(2)}, c^{(2)}, W^{(2)} \rangle$ be two concepts. We define their modified union as $\tilde{S}' = U(\tilde{S}_1, \tilde{S}_2) = \langle S', \mu'_0, c', W' \rangle$ with the following components:

- $\mu'_0 = \max(\mu_0^{(1)}, \mu_0^{(2)})$
- $S' = U(S_1, S_2)$
- $c' = \min(c^{(1)}, c^{(2)})$
- W' with weights w'_δ, w'_d defined as follows (using interpolation factors $s, t \in [0, 1]$):

$$\begin{aligned} \forall \delta \in \Delta_{S_1} \cap \Delta_{S_2} : & \left(\left(w'_\delta = s \cdot w_\delta^{(1)} + (1-s) \cdot w_\delta^{(2)} \right) \right. \\ & \left. \wedge \forall d \in \delta : \left(w'_d = t \cdot w_d^{(1)} + (1-t) \cdot w_d^{(2)} \right) \right) \\ \forall \delta \in \Delta_{S_1} \setminus \Delta_{S_2} : & \left(\left(w'_\delta = w_\delta^{(1)} \right) \wedge \forall d \in \delta : \left(w'_d = w_d^{(1)} \right) \right) \\ \forall \delta \in \Delta_{S_2} \setminus \Delta_{S_1} : & \left(\left(w'_\delta = w_\delta^{(2)} \right) \wedge \forall d \in \delta : \left(w'_d = w_d^{(2)} \right) \right) \end{aligned}$$

Figure 3.9 shows a visualization of this modified union for two one-dimensional concepts, comparing it to the standard union from fuzzy set theory, which is based on the maximum across the two original membership functions:

Our modified union and the standard union of fuzzy sets

Definition 3.7 (Fuzzy Union)

Let \tilde{A}, \tilde{B} be two fuzzy sets on CS . Then, the membership function of their fuzzy union is defined as follows:

$$\forall x \in CS : \mu_{\tilde{A} \cup \tilde{B}}(x) = \max(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x))$$

Concept	Δ_S	p^-	p^+	μ_0	c	W		
						$w_{\delta_{\text{COLOR}}}$	$w_{\delta_{\text{SHAPE}}}$	$w_{\delta_{\text{TASTE}}}$
PEAR	Δ	(0.50, 0.40, 0.35)	(0.70, 0.60, 0.45)	1.0	24.0	0.50	1.25	1.25
APPLE	Δ	(0.50, 0.65, 0.35)	(0.80, 0.80, 0.50)	1.0	20.0	0.50	1.50	1.00
		(0.65, 0.65, 0.40)	(0.85, 0.80, 0.55)					
		(0.70, 0.65, 0.45)	(1.00, 0.80, 0.60)					

Table 3.4: Definition of the concepts APPLE and PEAR for the union example.

*Modified union as
superset of standard
fuzzy union*

One can see in Figure 3.9a that the standard union does in general not result in a valid concept. However, under certain circumstances, it is a fuzzy subset of $U(\tilde{S}_1, \tilde{S}_2)$, i.e., $\forall x \in CS : \mu_{\tilde{S}_1 \cup \tilde{S}_2}(x) \leq \mu_{U(\tilde{S}_1, \tilde{S}_2)}(x)$:

Proposition 3.1 (Modified Union as Superset of Standard Union)

Let $\tilde{S}_1 = \langle S_1, \mu_0^{(1)}, c^{(1)}, W^{(1)} \rangle$ and $\tilde{S}_2 = \langle S_2, \mu_0^{(2)}, c^{(2)}, W^{(2)} \rangle$ be two concepts. If we assume that $\Delta_{S_1} = \Delta_{S_2}$ and $W^{(1)} = W^{(2)}$, then $\tilde{S}_1 \cup \tilde{S}_2 \subseteq U(\tilde{S}_1, \tilde{S}_2) = \tilde{S}'$, i.e., $\forall x \in CS : \mu_{\tilde{S}_1 \cup \tilde{S}_2}(x) \leq \mu_{U(\tilde{S}_1, \tilde{S}_2)}(x)$.

Proof. See Appendix B.2. □

3.2.2 Implementation and Example

Implementation

The union operation is quite straightforward to implement: One simply needs to collect all cuboids from S_1 and S_2 , apply the repair mechanism to this overall set of cuboids to obtain S' , and compute all remaining parameters as defined in Definition 3.6. We again use $s = t = 0.5$ for computing the new weights. Since this does not involve any complex computations (such as for example numerical optimization), the runtime of the union operation has not been analyzed.

*Fruit space example:
union of APPLE and
PEAR*

In order to illustrate the union operation, let us again make use of the concepts of APPLE and PEAR (re-printed in Table 3.4 and illustrated in Figure 3.10). In order to find the new core S' , we first take the union of all cuboids from the original cores and then apply the repair mechanism from Section 3.1.1 to ensure star-shapedness. We define p^* to be the arithmetic mean of the midpoints of all cuboids, yielding $p^* = (0.7125, 0.66875, 0.45625)$. Extending each cuboid such that it includes at least p^* results in the following extended cuboids:

$$\begin{aligned}
 C'_1 &= \langle \Delta, p_1'^- = (0.50, 0.40, 0.35), \\
 &\quad p_1'^+ = (0.71250, 0.66875, 0.45625) \rangle \\
 C'_2 &= \langle \Delta, p_2'^- = (0.50, 0.65, 0.35), p_2'^+ = (0.80, 0.80, 0.50) \rangle \\
 C'_3 &= \langle \Delta, p_3'^- = (0.65, 0.65, 0.40), p_3'^+ = (0.85, 0.80, 0.55) \rangle \\
 C'_4 &= \langle \Delta, p_4'^- = (0.70, 0.65, 0.45), p_4'^+ = (1.00, 0.80, 0.60) \rangle \\
 S' &= \langle \Delta, \{C'_1, C'_2, C'_3, C'_4\} \rangle
 \end{aligned}$$

*Considering the new
core*

As one can see, the three cuboids that originally belonged to the APPLE concept are not modified at all because $p^* \in P_{\text{APPLE}}$. Only the cuboid

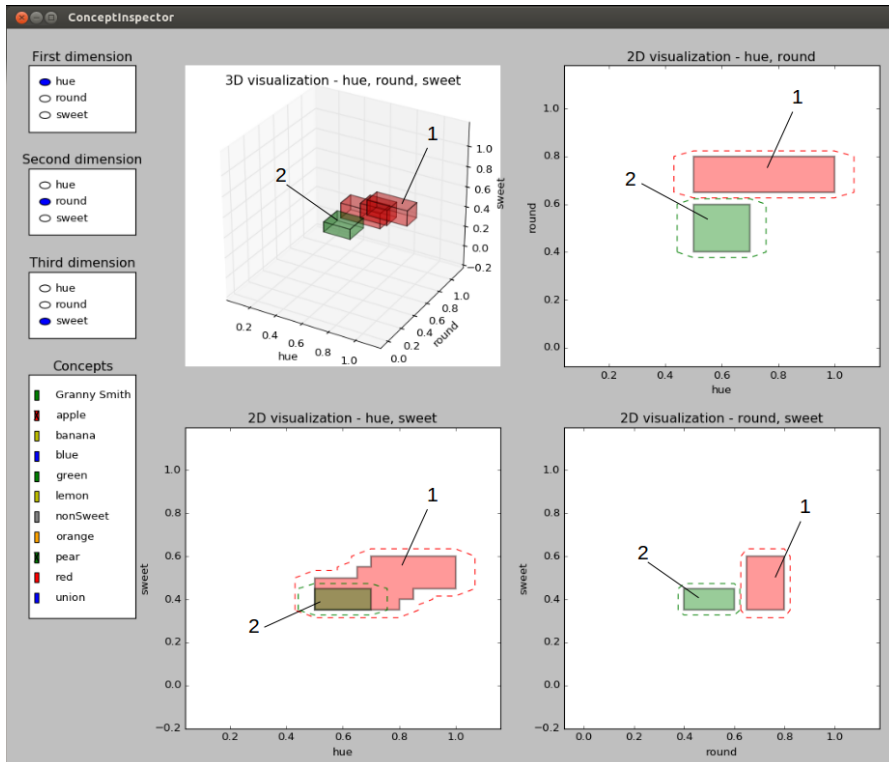


Figure 3.10: Screenshot of the ConceptInspector tool, illustrating the concepts of APPLE (1) and PEAR (2) in the three-dimensional fruit space.

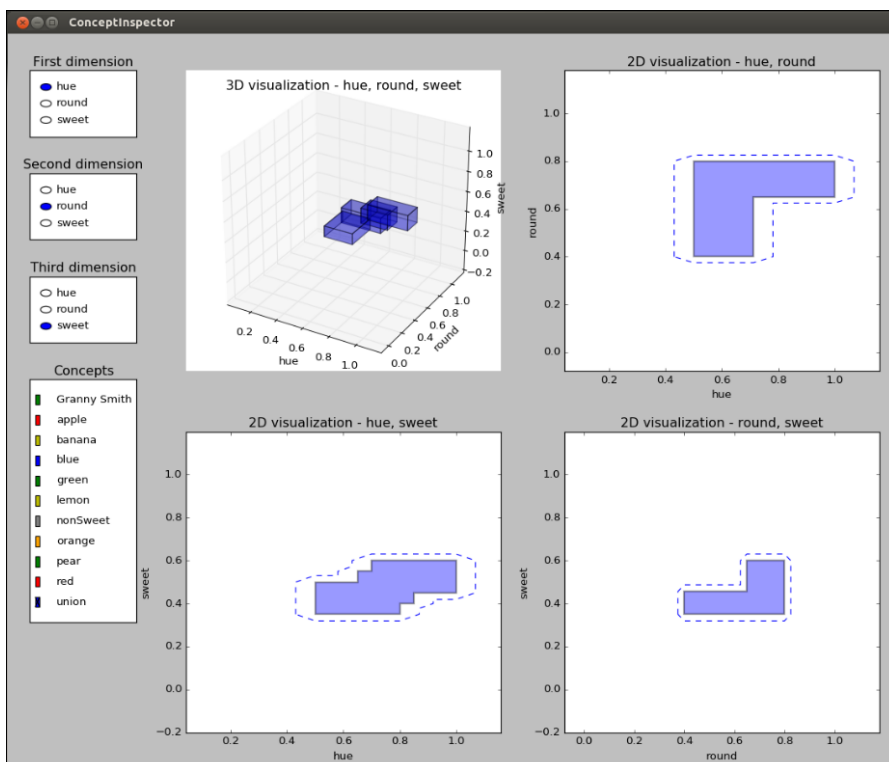


Figure 3.11: Screenshot of the ConceptInspector tool, illustrating the union of APPLE and PEAR in the three-dimensional fruit space.

from the PEAR concept has been extended in order to include p^* . The central region of the new core S' can be described as $P' = \langle \Delta, p'^- = (0.70, 0.65, 0.45), p'^+ = (0.7125, 0.66875, 0.45625) \rangle$.

Sensitivity parameter and salience weights

Both c' and W' are computed in exactly the same way as for the intersection. Therefore, we get the same values as in Section 3.1.3:

$$c' = \min \left(c^{(\text{APPLE})}, c^{(\text{PEAR})} \right) = \min (20.00, 24.00) = 20.00$$

$$W' = \langle \{w_{\delta_{\text{color}}} = 0.50, w_{\delta_{\text{shape}}} = 1.375, w_{\delta_{\text{taste}}} = 1.125\}, W_{\text{dim}} \rangle$$

Highest possible membership

Finally, the highest possible membership is computed as follows:

$$\mu'_0 = \max \left(\mu_0^{(\text{APPLE})}, \mu_0^{(\text{PEAR})} \right) = \max (1.00, 1.00) = 1.00$$

Visualization

The modified union $\tilde{S}' = \langle S', \mu'_0, c', W' \rangle$ of APPLE and PEAR is illustrated in Figure 3.11. In our implementation, the union operation can be executed as follows:

```
>>> print(apple.union_with(pear))
core: {[0.5, 0.65, 0.35]-[0.8, 0.8, 0.5],
       [0.65, 0.65, 0.4]-[0.85, 0.8, 0.55],
       [0.7, 0.65, 0.45]-[1.0, 0.8, 0.6],
       [0.5, 0.4, 0.35]-[0.7125, 0.6687500000000001,
       0.45625000000000004]}
mu: 1.0
c: 20.0
weights: <{'color': 0.5, 'taste': 1.125, 'shape': 1.375},
          {'color': {0: 1.0}, 'taste': {2: 1.0}, 'shape': {1: 1.0}}>
```

Fruit space example: union of ORANGE and LEMON

For an additional demonstration, let us consider the union of ORANGE and LEMON (corresponding to the more abstract concept of CITRUS FRUIT). In this case, the result looks as follows:

```
>>> print(lemon.union_with(orange))
core: {[0.7, 0.45, 0.0]-[0.8, 0.725, 0.35],
       [0.8, 0.725, 0.35]-[0.9, 1.0, 0.7]}
mu: 1.0
c: 30.0
weights: <{'color': 0.75, 'taste': 1.5, 'shape': 0.75},
          {'color': {0: 1.0}, 'taste': {2: 1.0}, 'shape': {1: 1.0}}>
```

Interpretation

In this example, the two extended cuboids C'_1 and C'_2 meet exactly in one point, namely $p^* = (0.80, 0.725, 0.35)$. Therefore, $P' = \{p^*\}$. Figures 3.12 and 3.13 illustrate this procedure. Because the concepts of ORANGE and LEMON are quite far apart in the given fruit space, their union is considerably larger than the original concepts. Moreover, one can see that their union intersects other concepts such as APPLE. Note that this is, however, not a weakness of our union operation – the observed effect is merely a limitation of the given fruit space. If the fruit space contained additional dimensions (e.g., indicating the latitude of the region where a particular fruit was grown), the concepts of LEMON and ORANGE would presumably much closer to each other and their union would not intersect any of the other concepts. This highlights that in practical applications an accurate definition of the similarity space with its domains and dimensions is of critical importance.

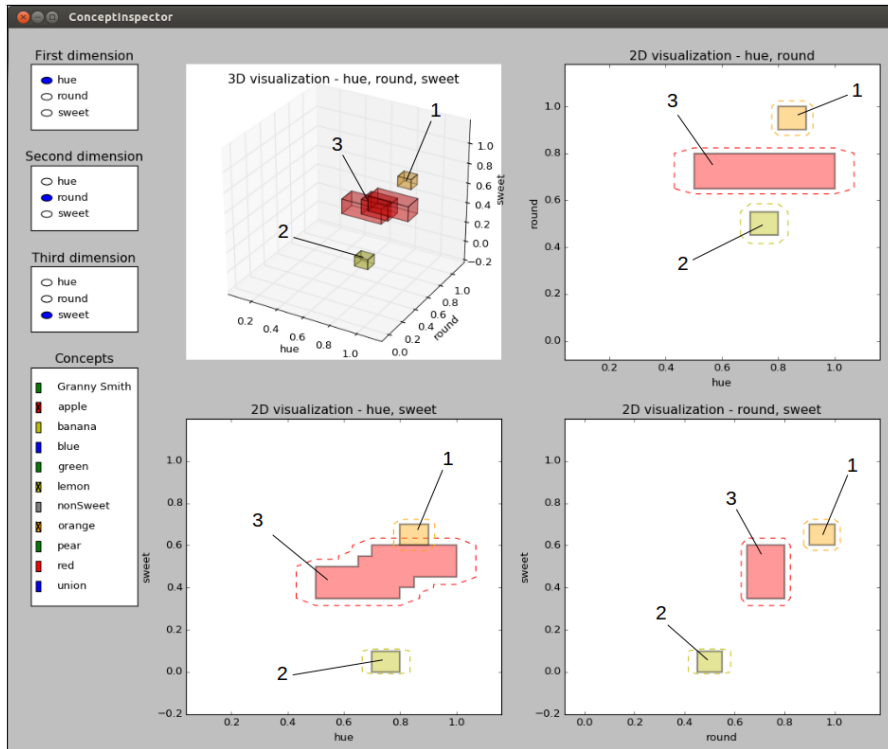


Figure 3.12: Screenshot of the ConceptInspector tool, illustrating the concepts of ORANGE (1), LEMON (2), and APPLE (3) in the three-dimensional fruit space.

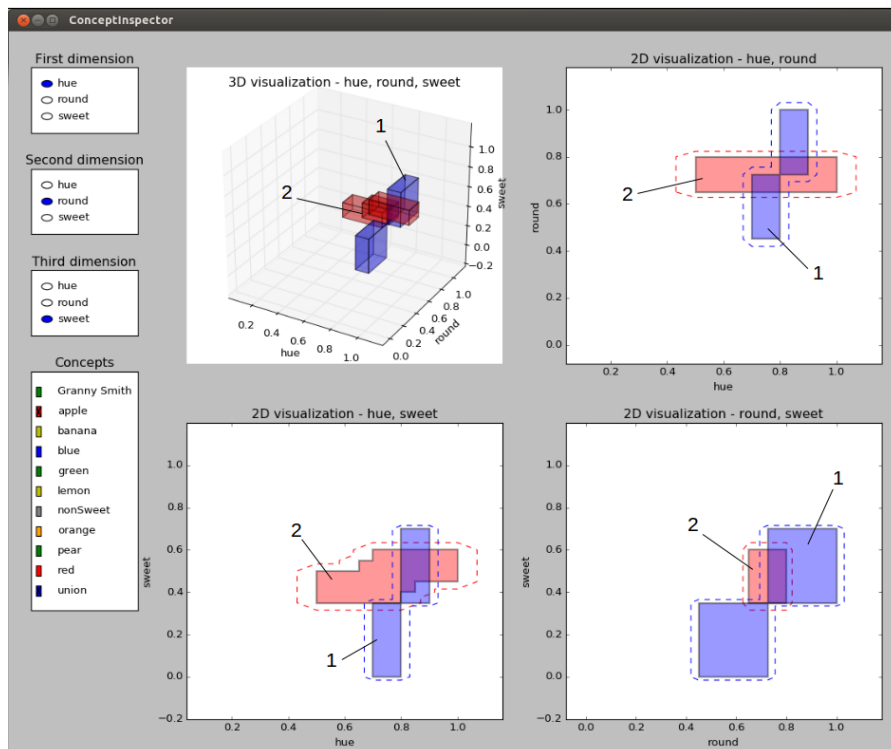


Figure 3.13: Screenshot of the ConceptInspector tool, illustrating the CITRUS FRUIT concept (1) (defined as the union of ORANGE and LEMON) and the APPLE concept (2).

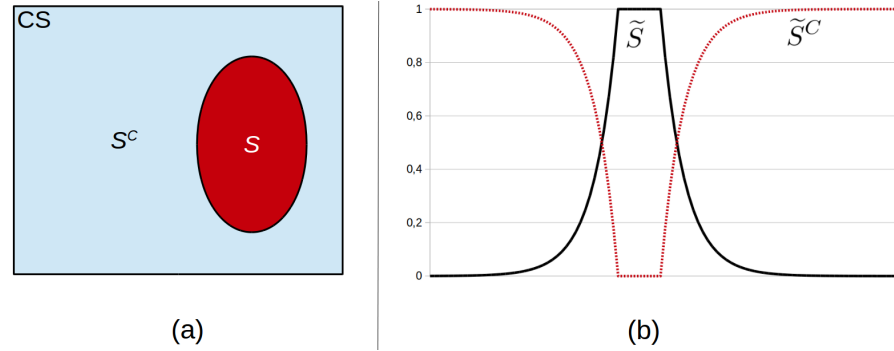


Figure 3.14: (a) Set complement of a crisp conceptual region S in a two-dimensional space. (b) Set complement of a fuzzy conceptual region \tilde{S} in a one-dimensional space.

3.3 NEGATION

Negation as set complement

The intersection and union operations defined in the previous sections can be interpreted as logical conjunction and disjunction. What is still missing for a complete set of logical operators is the operation of *negating* a concept. Since we define concepts as (fuzzy) regions in the conceptual space, the negation of a concept can be intuitively mapped to the set complement. In the crisp case, the complement S^C of a set S is defined as $S^C = \{x \in CS \mid x \notin S\}$ (see Figure 3.14a). For fuzzy sets \tilde{S} , the set complement \tilde{S}^C is typically defined as follows (illustrated in Figure 3.14b):

$$\forall x \in CS : \mu_{\tilde{S}^C}(x) = 1 - \mu_{\tilde{S}}(x)$$

The set complement does not produce valid conceptual regions

As we can see in Figure 3.14, the set complement does not result in a valid concept, since the resulting region is neither convex nor star-shaped – we are not able to identify a clear prototypical region P . In both cases, the set complement contains a "hole", namely the original conceptual region. As we have already argued in Section 2.2.2 in the context of connectedness⁵, such holes in the geometric representation are, however, quite problematic. The set complement yields a valid conceptual region only if the original concept is empty, contains the whole conceptual space, or is a half-space (see Figure 3.15). The latter case might hold for a property like COLD which can be defined as the lower end of the TEMPERATURE dimension (i.e., a half-open interval) – here, NOT COLD is also a valid concept. However, already a COLOR property like YELLOW cannot be simply negated by taking the set complement: AS YELLOW is represented as a region in the three-dimensional COLOR space, we face a situation similar to Figure 3.14a. The same problem also arises for concepts like APPLE which involve multiple domains.

Repair mechanisms are not helpful

One could try to work around this problem by proposing a repair mechanism as in Sections 3.1.1 and 3.2.1. However, extending the resulting set in order to make it star-shaped (as for intersection and

⁵ While S^C in Figure 3.14a is still connected, none of the α -cuts of \tilde{S}^C in Figure 3.14b is connected.

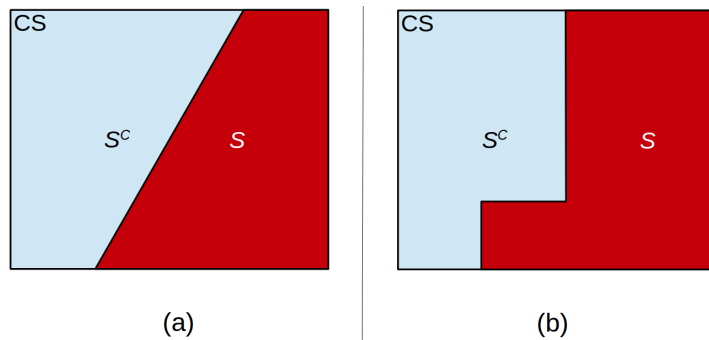


Figure 3.15: The set complement yields valid conceptual regions for half-spaces. (a) Convex half-space defined by a hyperplane. (b) Star-shaped half-space defined by half-open cuboids.

union) would force us to include the original conceptual region in the negation: The only star-shaped completion of the set complements S^C and \tilde{S}^C from Figure 3.14 is equal to the overall conceptual space CS . On the other hand, shrinking the resulting region in order to ensure star-shapedness requires us to decide which subregion of S^C and \tilde{S}^C to exclude, respectively. Any choice seems quite arbitrary, especially in symmetric scenarios. Therefore, in the case of negation, neither an overextension nor an underextension seems to be helpful.

Please note that these problems with respect to conceptual negation are inherent in the conceptual spaces framework in general and not just our formalization. Other forms of spatial representation such as word embeddings (to be introduced in Chapter 6) define the similarity of words based on the angle between the vectors representing them. This corresponds to using polar rather than Cartesian coordinates. These approaches are less limited with respect to negation: Negation of individual vectors can be implemented by considering the subspace orthogonal to this vector [252] in order to model expressions like "rock NOT BAND" (which indicates that the geological concept rock and not the music genre is being referred to) [434]. Conceptual regions can in this context be modeled as convex cones, and their negation can be implemented by using the polar cone (i.e., all points with an angle of at least 90 degrees to any vector in the concept) [312]. It therefore seems that the negation problem for conceptual spaces is rooted in one of its fundamental assumptions, namely, the usage of Euclidean and Manhattan distances as a measure of semantic dissimilarity.

Negation in other spatial representations

We would furthermore like to point out that especially for full-fledged concepts like APPLE, the semantics of a negation is somewhat unclear. For instance, the set of things that we would classify as NOT APPLE (such as BANANA, CHAIR, FRIENDSHIP, MICKEY MOUSE, BLASPHEMY, and PI) do not necessarily have a shared set of attributes, let alone a prototype. One needs either to restrict this set of alternatives (e.g., by only including

Semantics of negated concepts

fruits or tangible objects) or to acknowledge that negation does not lead to valid concepts.

*Matching bias:
negation only denies
a statement*

Shaikh et al. [362] distinguish two interpretations of negation in natural language: The *matching bias account* states that negation in natural language conveys denial rather than assertion of a proposition. If we adopt this view, we may assume that negation is a quite abstract operation and that it is thus only defined in the symbolic layer, but has no strict equivalent in the conceptual and the subsymbolic layer.⁶ Under this view, the phrase "not an apple" does not describe a concept, but rather a condition that can or cannot be met.

*Contrast classes:
negation targets
plausible alternatives*

The *contrast classes account* on the other hand assumes that negation in natural language targets plausible alternatives. In order to reflect this set of plausible alternatives, one can augment the literal logical negation with a so-called "worldly context", for instance by taking into account the conceptual hierarchy of the negated word [335]. In our example from above, NOT APPLE would thus refer to other sub-concepts of FRUIT SUCH AS PEAR OR BANANA.

Summary

Since the negation operation does not seem to be very useful for the direct creation of novel concepts, it is not implemented in our formalization. We thus implicitly adopt the matching bias account of negation. If we need to negate a concept as an intermediate step for concept combination (e.g., for phrases such as "a bird that is not yellow"), we can use the set complement as a temporary construct.

3.4 SUBSPACE PROJECTION

*Projection for
ignoring domains*

Projecting a concept onto a subspace corresponds to focusing on certain domains while completely ignoring others. For instance, projecting the concept APPLE onto the COLOR domain results in a property which may include different shades of RED, GREEN, and YELLOW. Also the dimensional filters proposed by Fiorini [158] (cf. Section 2.4.3), which remove all irrelevant domains from a part in the context of a composite concept, can be implemented through such a projection.

*Projection works on
the level of domains*

Please note that we always assume that the projection takes place on the level of domains and not on the level of individual dimensions. This assumption is based on the fact that domains are defined as sets of integral dimensions that are separable from all other dimensions (cf. Section 2.1.1). This means that the dimensions within a domain are either perceived jointly or not at all, but not individually.

In the following, we give a formal definition of the projection operation for cuboids, cores, and concepts (Section 3.4.1), before illustrating with an example from our fruit space (Section 3.4.2).

⁶ Thanks to Achim Stephan for pointing this out to me.

3.4.1 Definition

Let us begin by considering how to project a cuboid C onto a subspace given by the domains $\Delta_{C'} \subseteq \Delta_C$. This can be achieved by simply removing any constraints with respect to the dimensions $d \in \Delta_C \setminus \Delta_{C'}$ from its support points p^+ and p^- .

Projecting a single cuboid

Definition 3.8 (Projection of a Cuboid)

Let $C = \langle \Delta_C, p^-, p^+ \rangle$ be a cuboid defined on Δ_C . The projection $P(C, \Delta_{C'})$ of this cuboid onto a subspace $\Delta_{C'} \subseteq \Delta_C$ is defined as the cuboid $C' = \langle \Delta_{C'}, p'^-, p'^+ \rangle$, whose support points are defined as follows (using $D_C = \bigcup_{\delta \in \Delta_C} \delta$ and $D_{C'} = \bigcup_{\delta \in \Delta_{C'}} \delta$):

$$\begin{aligned} \forall d \in D_{C'} : p_d'^- &= p_d^- \wedge p_d^+ = p_d^+ \\ \forall d \in D \setminus D_{C'} : p_d'^- &= -\infty \wedge p_d'^+ = +\infty \end{aligned}$$

We can now define the projection $P(S, \Delta_{S'})$ of a core S onto domains $\Delta_{S'} \subseteq \Delta_S$ by simply projecting all of its cuboids individually.

Projecting a core

Definition 3.9 (Projection of a Core)

Let $S = \langle \Delta_S, \{C_1, \dots, C_m\} \rangle$ be a core. Let $C'_i = P(C_i, \Delta_{S'})$ be the projection of C_i onto the domains $\Delta_{S'} \subseteq \Delta_S$. Then, $S' = P(S, \Delta_{S'}) = \langle \Delta_{S'}, \{C'_1, \dots, C'_m\} \rangle$ is the projected version of S .

One can easily show that $P' = \bigcap_{i=1}^m C'_i \neq \emptyset$ is the projected version of $P = \bigcap_{i=1}^m C_i \neq \emptyset$. Moreover, projecting any geometric region onto two complementary subspaces and then intersecting these projections again in the original space yields in general a superset of the original region. We can show that the projection of cores together with our intersection operation from Section 3.1.1 also respects this general observation:

Intersecting complementary projections of cores

Proposition 3.2 (Intersection of Projections of Cores)

Let $S = \langle \Delta_S, \{C_1, \dots, C_m\} \rangle$ be a core, and let $S_1 = P(S, \Delta_1)$ and $S_2 = P(S, \Delta_2)$ be its projections, where $\Delta_1 \cup \Delta_2 = \Delta_S$ and $\Delta_1 \cap \Delta_2 = \emptyset$. Then, $S \subseteq S' = I(S_1, S_2)$.

Proof. See Appendix B.3. □

In order to project a concept \tilde{S} , we also need to decide how to update the weights W , the sensitivity parameter c , and the maximal membership μ_0 . We propose to apply only minimal changes:

Projecting a concept

Definition 3.10 (Projection of a Concept)

Let $\tilde{S} = \langle S, \mu_0, c, W \rangle$ be a concept and $\Delta_{S'} \subseteq \Delta_S$ a subset of its domains. The projection of \tilde{S} on $\Delta_{S'}$ is defined as $P(\tilde{S}, \Delta_{S'}) = \langle S', \mu_0, c, W' \rangle$ with

$$S' = P(S, \Delta_{S'}) \text{ and } W' = \left\langle \left\{ |\Delta_{S'}| \cdot \frac{w_\delta}{\sum_{\delta' \in \Delta_{S'}} w_{\delta'}} \right\}_{\delta \in \Delta_{S'}}, \{W_\delta\}_{\delta \in \Delta_{S'}} \right\rangle.$$

In Definition 3.10, we obtain the new core by a projection of the original core, and we leave the maximal membership μ_0 , the sensitivity parameter c , and the dimension weights W_δ unchanged. Only the domain weights are updated in such a way that their normalization constraint is fulfilled.

Interpretation

Our projection and the standard projection of fuzzy sets

Our definition of the projection operation differs from the usual approach to projecting a fuzzy set \tilde{A} from a space $d_1 \times \dots \times d_n$ to a subspace $d_1 \times \dots \times d_l$ (with $l < n$). In the standard notion of projection, one takes the minimum over all possible coordinates on the dimensions to be removed:

Definition 3.11 (Fuzzy Projection)

Let \tilde{A} be a concept defined on a space $d_1 \times \dots \times d_n$. Its projection $\tilde{A} \downarrow (d_1 \times \dots \times d_l)$ to a subspace $d_1 \times \dots \times d_l$ (with $l < n$) is defined as follows:

$$\mu_{\tilde{A} \downarrow (d_1 \times \dots \times d_l)}(x_1, \dots, x_l) = \max_{\substack{(x_{l+1}, \dots, x_n) \\ \in d_{l+1} \times \dots \times d_n}} \mu_{\tilde{A}}(x_1, \dots, x_l, x_{l+1}, \dots, x_n)$$

A subsethood relation

Despite this difference, we can show that our definition of projection always results in a subset of the standard fuzzy set projection.

Proposition 3.3 (Modified Projection as a Subset of Standard Projection)

Let $\tilde{S} = \langle S, \mu_0, c, W \rangle$ be a concept. Let $\Delta_{S'} \subseteq \Delta_S$ and let $P(\tilde{S}, \Delta_{S'}) = \langle S', \mu_0, c, W' \rangle$ be the projection of \tilde{S} onto $\Delta_{S'}$. Let furthermore $\tilde{S} \downarrow_{\Delta_{S'}}$ be the standard fuzzy set projection of \tilde{S} onto the domains $\Delta_{S'}$ as defined above. Then, $P(\tilde{S}, \Delta_{S'}) \subseteq \tilde{S} \downarrow_{\Delta_{S'}}$, i.e., $\forall x \in CS : \mu_{P(\tilde{S}, \Delta_{S'})}(x) \leq \mu_{\tilde{S} \downarrow_{\Delta_{S'}}}(x)$.

Proof. See Appendix B.3. □

Intersecting complementary projections of concepts

Proposition 3.2 states that projecting a core onto two complementary subspaces and then intersecting these projections in the original space results in a superset of the original core. We can generalize Proposition 3.2 to concepts under one additional constraint:

Proposition 3.4 (Intersection of Projections of Concepts)

Let $\tilde{S} = \langle S, \mu_0, c, W \rangle$ be a concept, and let $\tilde{S}_1 = P(\tilde{S}, \Delta_1)$ and $\tilde{S}_2 = P(\tilde{S}, \Delta_2)$ be its projections with $\Delta_1 \cup \Delta_2 = \Delta_S$ and $\Delta_1 \cap \Delta_2 = \emptyset$. Let furthermore $\tilde{S}' = I(\tilde{S}_1, \tilde{S}_2)$ as defined in Definition 3.2. If $\sum_{\delta \in \Delta_1} w_\delta = |\Delta_1|$ and $\sum_{\delta \in \Delta_2} w_\delta = |\Delta_2|$, then $\tilde{S} \subseteq \tilde{S}' = I(\tilde{S}_1, \tilde{S}_2)$, i.e., $\forall x \in CS : \mu_{\tilde{S}}(x) \leq \mu_{I(\tilde{S}_1, \tilde{S}_2)}(x)$.

Proof. See Appendix B.3. □

3.4.2 Implementation and Example

Implementation

The subspace projection can be easily implemented by projecting the concept's cuboids onto the given subspace (see Definition 3.8) and updating the domain weights according to Definition 3.10. We do not provide a runtime analysis, since the implementation is computationally straightforward.

Fruit space example

Let us use the BANANA concept from Section 2.3.4 in order to illustrate the projection operation. The BANANA concept is defined on $\Delta = \{\delta_{\text{COLOR}}, \delta_{\text{SHAPE}}, \delta_{\text{TASTE}}\}$. We will consider projections onto $\Delta_1 = \{\delta_{\text{COLOR}}, \delta_{\text{TASTE}}\}$ and $\Delta_2 = \{\delta_{\text{SHAPE}}\}$.

Concept	Δ_S	p^-	p^+	μ_0	c	W		
						$w_{\delta_{\text{COLOR}}}$	$w_{\delta_{\text{SHAPE}}}$	$w_{\delta_{\text{TASTE}}}$
BANANA	Δ	(0.50, 0.10, 0.35) (0.70, 0.10, 0.50) (0.75, 0.10, 0.50)	(0.75, 0.30, 0.55) (0.80, 0.30, 0.70) (0.85, 0.30, 1.00)	1.0	20.0	0.75	1.50	0.75
$P(\text{BANANA}, \Delta_1)$	Δ_1	(0.50, $-\infty$, 0.35) (0.70, $-\infty$, 0.50) (0.75, $-\infty$, 0.50)	(0.75, $+\infty$, 0.55) (0.80, $+\infty$, 0.70) (0.85, $+\infty$, 1.00)	1.0	20.0	1.00	–	1.00
$P(\text{BANANA}, \Delta_2)$	Δ_2	($-\infty$, 0.10, $-\infty$)	($+\infty$, 0.30, $+\infty$)	1.0	20.0	–	1.00	–
Intersection	Δ	(0.50, 0.10, 0.35) (0.70, 0.10, 0.50) (0.75, 0.10, 0.50)	(0.75, 0.30, 0.55) (0.80, 0.30, 0.70) (0.85, 0.30, 1.00)	1.0	20.0	1.00	1.00	1.00

Table 3.5: Definition of the BANANA concept, its projections onto $\Delta_1 = \{\delta_{\text{COLOR}}, \delta_{\text{TASTE}}\}$ and $\Delta_2 = \{\delta_{\text{SHAPE}}\}$ as well as the intersection of these projections.

Table 3.5 shows the definition of the BANANA concept, its projections, and their intersection. Figures 3.16, 3.17 and 3.18 illustrate these concepts with screenshots of the ConceptInspector tool. As one can see by looking at the salience weights, the intersection of the projections is not identical to the original concept, even though their cores are identical. Moreover, Proposition 3.4 is not applicable because its precondition with respect to the domain weights is violated. In the code, the projections and the subsequent intersection can be executed as follows:

```
>>> projection_1 = banana.project_onto({'color':[0], 'taste':
    :[2]})
>>> projection_2 = banana.project_onto({'shape':[1]})
>>> intersection = projection_1.intersect_with(projection_2)
>>> print(intersection)
core: {[0.5, 0.1, 0.35]-[0.75, 0.3, 0.55],
      [0.7, 0.1, 0.5]-[0.8, 0.3, 0.7],
      [0.75, 0.1, 0.5]-[0.85, 0.3, 1.0]}
mu: 1.0
c: 20.0
weights: <{'color': 1.0, 'taste': 1.0, 'shape': 1.0},
          {'color': {0: 1.0}, 'taste': {2: 1.0}, 'shape': {1: 1.0}}>
```

*Projecting the
BANANA concept*

3.5 AXIS-PARALLEL CUT

In a concept formation process, it might happen that over-generalized concepts are learned (e.g., a single concept that represents both dogs and cats). If it becomes apparent that a finer-granular conceptualization is needed, the system needs to be able to split its current concepts into multiple parts. This could be achieved by partitioning the cuboids from the concept's core into multiple subsets, because these cuboids can be thought of as building blocks of the conceptual region (cf. Section 2.3.2). However, since these cuboids have a non-empty intersection (namely, the central region P), this approach always results in overlapping

*Axis-parallel cut for
creating sub-concepts*

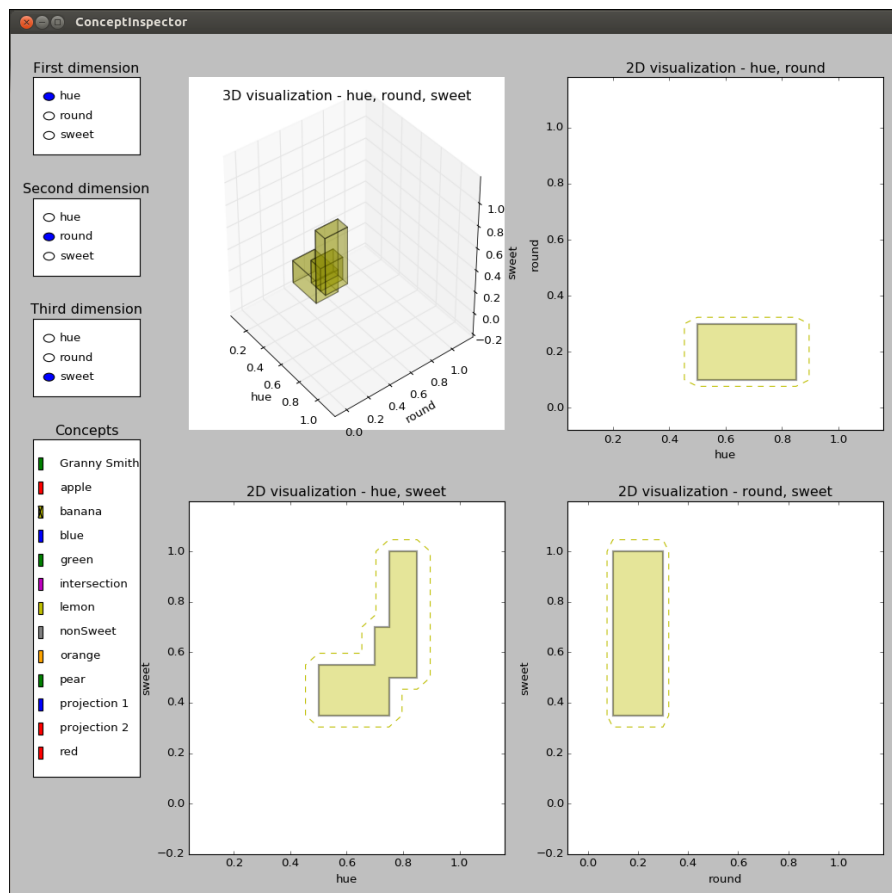


Figure 3.16: Screenshot of the ConceptInspector tool, illustrating the BANANA concept in the example fruit space.

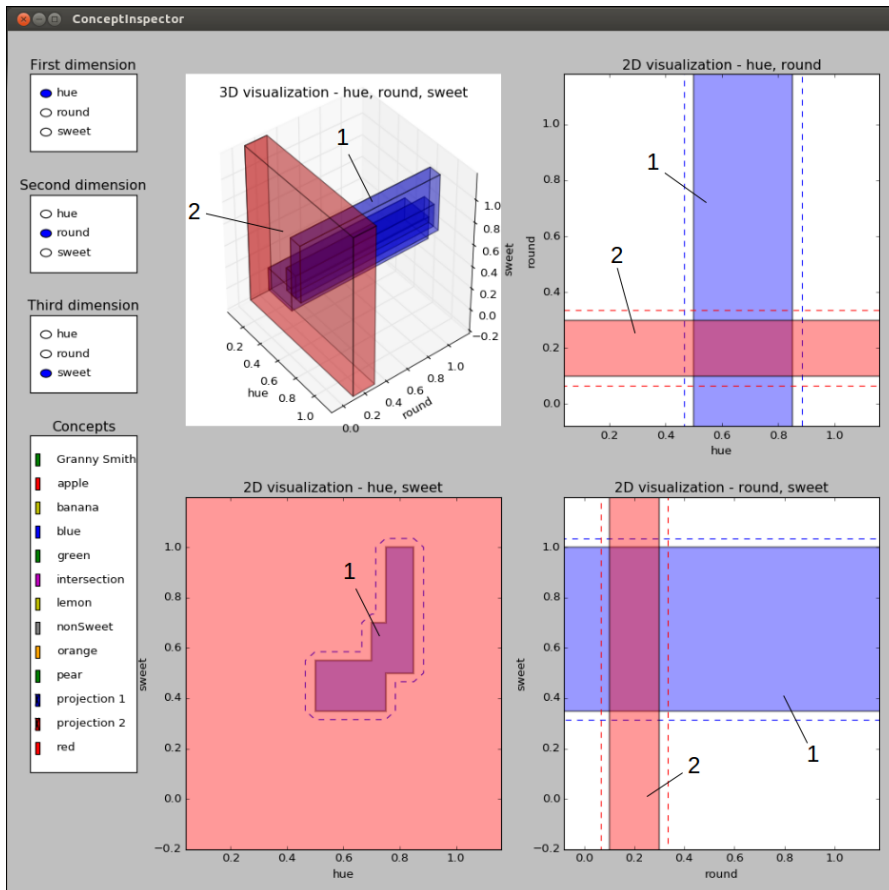


Figure 3.17: Screenshot of the ConceptInspector tool, illustrating the projections of the BANANA concept onto the sets of domains $\Delta_1 = \{\delta_{\text{COLOR}}, \delta_{\text{TASTE}}\}$ (1) and $\Delta_2 = \delta_{\text{SHAPE}}$ (2).

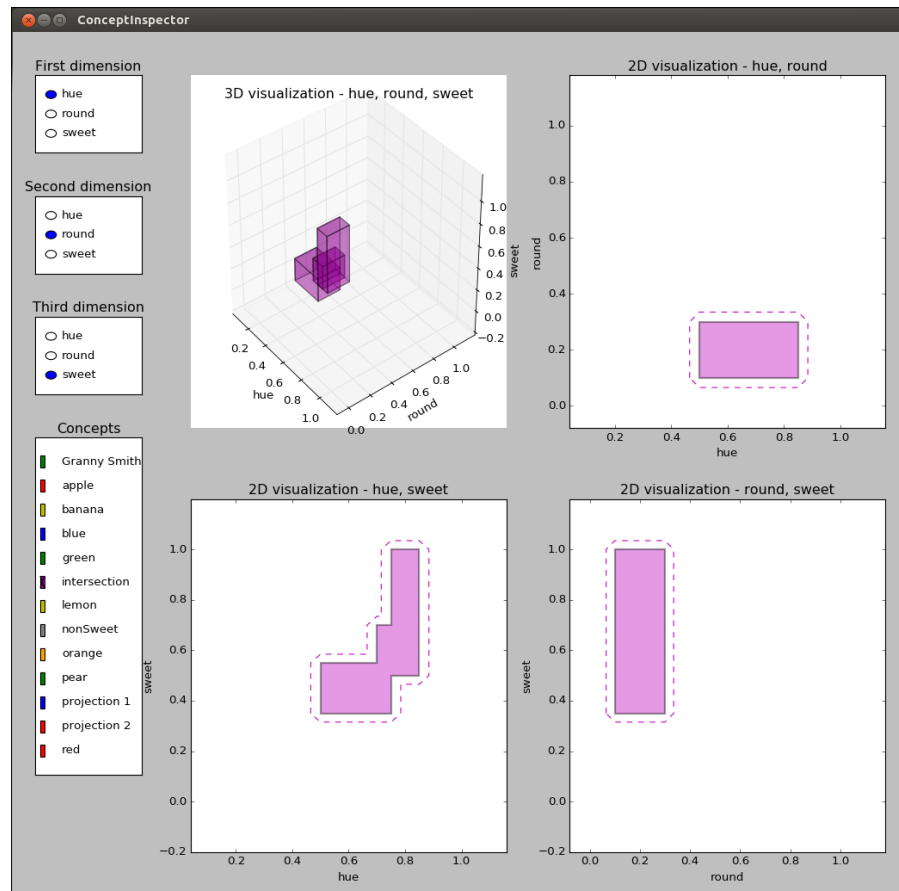


Figure 3.18: Screenshot of the ConceptInspector tool, illustrating the intersection of the two projections of the BANANA concept.

sub-concepts. In some cases, it may, however, be desirable to obtain sub-concepts with non-overlapping cores. In order to accommodate this need, we consider another relatively straightforward way of splitting a concept, namely, by using a threshold on a single dimension. In Section 3.5.1, we show that this results in two valid concepts and in Section 3.5.2 we provide an illustrative example.

3.5.1 Definition

Let us first assume that we would like to split a given core into two parts based on a threshold value v on a given dimension d^* .

Axis-parallel cut of a core

Definition 3.12 (Axis-Parallel Cut of a Core)

Let $S = \langle \Delta_S, \{C_1, \dots, C_m\} \rangle$ be a core, $d^* \in D_S$ be any dimension and v be any value on this dimension. The axis-parallel cut of S based on a threshold value v for a dimension d^* is given by $S^- = \{x \in S \mid x_{d^*} \leq v\}$ and $S^+ = \{x \in S \mid x_{d^*} \geq v\}$.

We denote this cut operation as $S^-, S^+ = C(S, d^*, v)$. One can show that both S^- and S^+ are valid cores:

Results are valid cores

Proposition 3.5 (Cut of a Core Results in Cores)

Let S be a core and $S^-, S^+ = C(S, d^*, v)$ its cut at a threshold value v on a dimension $d^* \in D_S$. Then, both S^- and S^+ are valid cores.

Proof. See Appendix B.4. □

Please note that the cores S^- and S^+ have a slight overlap: We had to include $x \in S$ with $x_{d^*} = v$ into both S^- and S^+ in order to obtain valid cuboids, since cuboids are defined based on closed intervals. If necessary, our definition could also be augmented by a separation distance $\epsilon > 0$, leading to a re-definition of $S^- = \{x \in S \mid x_{d^*} \leq v - \frac{\epsilon}{2}\}$ and $S^+ = \{x \in S \mid x_{d^*} \leq v + \frac{\epsilon}{2}\}$. For a small enough ϵ , this does not influence the validity of our proof. Since we simply cut the core S into two parts without applying any repair mechanism afterwards, the union of the two parts is equal to the original set, i.e., $S^- \cup S^+ = S$ for $S^-, S^+ = C(S, d^*, v)$.

The resulting cores overlap

We can now define the axis-parallel cut $\tilde{S}^+, \tilde{S}^- = C(\tilde{S}, d^*, v)$ of a concept \tilde{S} with respect to a dimension d^* and a threshold v as follows:

Axis-parallel cut of a concept

Definition 3.13 (Axis-Parallel Cut of a Concept)

Let $\tilde{S} = \langle S, \mu_0, c, W \rangle$ be a concept. The axis-parallel cut $\tilde{S}^+, \tilde{S}^- = C(\tilde{S}, d^*, v)$ based on a threshold value v for a dimension d^* is given by $\tilde{S}^- = \langle S^-, \mu_0, c, W \rangle$ and $\tilde{S}^+ = \langle S^+, \mu_0, c, W \rangle$, where $S^-, S^+ = C(S, d^*, v)$.

Please note that most parameters of \tilde{S} remain unchanged, only the cores are updated. One can therefore easily see that $U(\tilde{S}^-, \tilde{S}^+) = \tilde{S}$.

Concept	Δ_S	p^-	p^+	μ_0	c	W		
						$w_{\delta_{\text{COLOR}}}$	$w_{\delta_{\text{SHAPE}}}$	$w_{\delta_{\text{TASTE}}}$
APPLE	Δ	(0.50, 0.65, 0.35) (0.65, 0.65, 0.40) (0.70, 0.65, 0.45)	(0.80, 0.80, 0.50) (0.85, 0.80, 0.55) (1.00, 0.80, 0.60)	1.0	20.0	0.50	1.50	1.00
\tilde{S}^-	Δ	(0.50, 0.65, 0.35) (0.65, 0.65, 0.40) (0.70, 0.65, 0.45)	(0.80, 0.80, 0.50) (0.80, 0.80, 0.55) (0.80, 0.80, 0.60)	1.0	20.0	0.50	1.50	1.00
\tilde{S}^+	Δ	(0.80, 0.65, 0.40) (0.80, 0.65, 0.45)	(0.85, 0.80, 0.55) (1.00, 0.80, 0.60)	1.0	20.0	0.50	1.50	1.00

Table 3.6: Definition of the APPLE concept as well as its two sub-concepts \tilde{S}^- and \tilde{S}^+ obtained by applying an axis-parallel cut on d_{HUE} .

3.5.2 Implementation and Example

Implementation

Again, the implementation is quite straightforward and basically follows Definition 3.13 and the proof of Proposition 3.5: If applicable, the cuboids of the concept's core are split, while all other parameters remain unchanged. A runtime analysis is therefore again not expected to result in any interesting insights and was therefore omitted.

Fruit space example

In order to illustrate the cut operation, let us assume that the concept of APPLE should be split up because separate concepts GREENAPPLE and REDAPPLE are more useful in the given application context. Using the axis-parallel cut, we can split the concept \tilde{S}_{APPLE} at a HUE value of 0.8, resulting in the two child-concepts \tilde{S}^- and \tilde{S}^+ , which are defined in Table 3.6 and illustrated in Figures 3.19 and 3.20.

Interpretation

As one can see, two cuboids of the original concept are split, while the third one remains intact. The cores of the two concepts \tilde{S}^- and \tilde{S}^+ touch at $d_{\text{HUE}} = 0.8$. If the concepts are updated in the future (e.g., by a clustering algorithm), their cores might, however, start to move away from each other. In our implementation, the cut operation can be invoked as follows:

```
>>> s1, s2 = apple.cut_at(0, 0.8)
>>> print(s1)
core: {[0.5, 0.65, 0.35]-[0.8, 0.8, 0.5],
        [0.65, 0.65, 0.4]-[0.8, 0.8, 0.55],
        [0.7, 0.65, 0.45]-[0.8, 0.8, 0.6]}
mu: 1.0
c: 20.0
weights: <{'color': 0.5, 'taste': 1.0, 'shape': 1.5},
          {'color': {0: 1.0}, 'taste': {2: 1.0}, 'shape': {1: 1.0}}>
>>> print(s2)
core: {[0.8, 0.65, 0.4]-[0.85, 0.8, 0.55],
        [0.8, 0.65, 0.45]-[1.0, 0.8, 0.6]}
mu: 1.0
c: 20.0
weights: <{'color': 0.5, 'taste': 1.0, 'shape': 1.5},
          {'color': {0: 1.0}, 'taste': {2: 1.0}, 'shape': {1: 1.0}}>
```

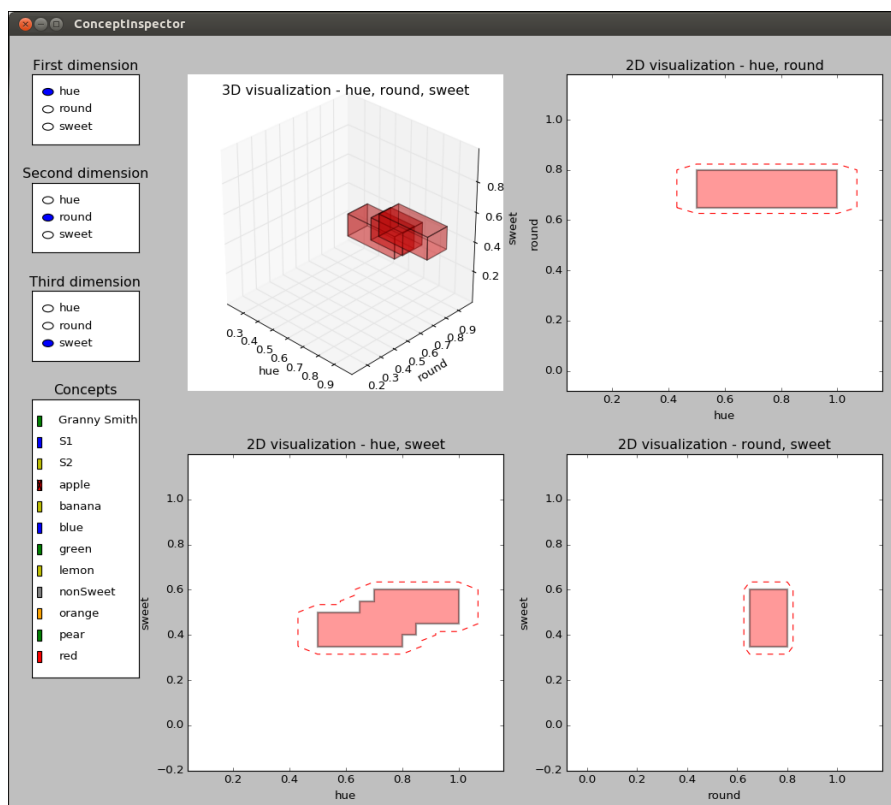


Figure 3.19: Screenshot of the ConceptInspector tool, illustrating the APPLE concept in the example fruit space.

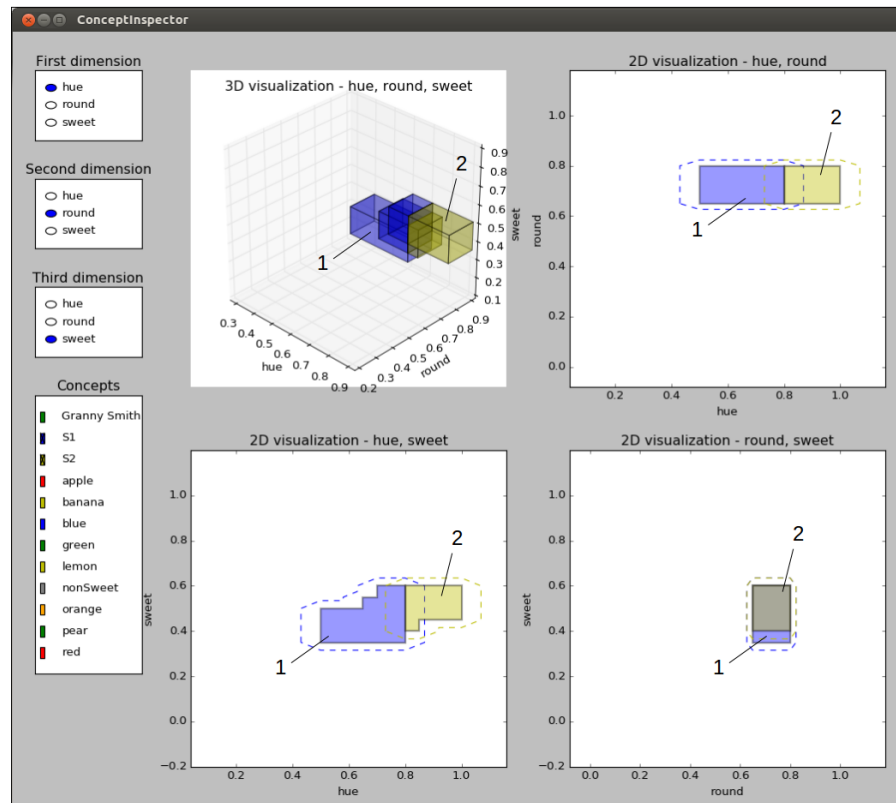


Figure 3.20: Screenshot of the ConceptInspector tool, illustrating the two resulting sub-concepts \tilde{S}^- (1) and \tilde{S}^+ (2) after applying the cut operation on the HUE dimension.

3.6 SUPPORTED APPLICATIONS

We now sketch how the operations defined in this chapter can be applied in cognitive tasks. In Section 3.6.1, we consider concept formation (to be introduced more thoroughly in Chapter 7) as an important learning process. In Section 3.6.2, we then elaborate on concept combination, which can be interpreted as a reasoning process.

Overview

3.6.1 Concept Formation

Concept formation is an incremental clustering process, which extracts a conceptual hierarchy from individual observations [164]: The system is faced with a stream of unlabeled observations and tries to find meaningful concepts by grouping these observations into clusters. These clusters are usually based on the similarity of the observations they contain. In our case, each cluster can be represented as a fuzzy concept according to Definition 2.14. After each observation, the set of clusters is updated. In the following, we consider different types of incremental modifications and argue that they are supported by our formalization.

Concept formation as incremental clustering

In the initial state of the concept formation algorithm, there are not yet any clusters. As soon as some observations have been made that are reasonably similar to each other, they can be replaced by a summary description, i.e., a cluster. In our case, one would need to create a new concept $\tilde{S} = \langle S, \mu_0, c, W \rangle$ based on this set of observations. The core S of this new concept can be initialized with a single cuboid which is the bounding box of the observations. The maximal membership μ_0 can be set to 1 and both the sensitivity parameter c and the weights W can be estimated based on the distribution of the observations in the conceptual space (cf. Section 2.3.3).

Creating a new cluster

Moreover, the clustering algorithm might erroneously create a cluster for a group of observations which are only outliers. At some point, it might become clear that this cluster is irrelevant and should be deleted, for instance, because it has not played any role for classifying novel observations for a long period of time. The deletion can be trivially achieved by removing an existing concept from the list of concepts under consideration.

Deleting an old cluster

Whenever a new observation is made, one or more clusters may be updated by moving them, resizing them, or by changing their form. These modifications of existing clusters are performed to ensure that the clusters reflect the distribution of observations well. In our formalization, this can be done by moving and resizing a concept's cuboids C_i (more specifically their support points p_i^- and p_i^+), and by adding or removing cuboids. During these modifications, one needs of course to ensure that the intersection of the cuboids stays non-empty. One can also change a concept's sensitivity parameter c in order to control the overall degree of fuzziness. In addition, a concept's weights W for the different domains and dimensions can be modified to control its narrowness with respect

Adjusting the location, form, or size of a cluster

to different domains and dimensions. While all of these modifications are easily possible, there is no dedicated operation that specifies *which* exact modifications should be applied under which circumstances.

*Splitting a cluster
into sub-clusters*

If a cluster grows too large, it might become too general to be useful. For instance, a single cluster may be used to represent both LEMON and ORANGE. If it becomes apparent that a more fine-granular distinction is needed (e.g., because predictions made based on the cluster membership are not accurate or fine-grained enough), the given cluster may be split into two sub-clusters. One can identify a suitable dimension and a suitable threshold on that dimension and apply the axis-parallel cut introduced in Section 3.5.

*Merging neighboring
clusters*

Also the opposite case might occur in practice: The current clustering might be too fine-grained, e.g., by distinguishing different subtypes of APPLE even though it suffices for the current application context to make distinctions on a higher level, e.g., between APPLE and PEAR. In this case, the clustering can be improved by merging neighboring clusters and obtaining a single, more coarse-grained cluster. Merging neighboring clusters is supported by the union operation as defined in Section 3.2.

Outlook

As we can see from this brief discussion, many important update steps are in principle supported by our formalization. In addition to direct modifications of a concept's parameters, especially the cut and union operations can be useful for controlling the concept inventory's overall level of granularity. What is still missing, however, are guidelines on when to apply which type of modification. We will comment on this issue in Chapter 4 in the context of formal ways for measuring relations between concepts. We will furthermore introduce three example concept formation algorithms from the literature in Chapter 7 with their respective control strategies.

3.6.2 Concept Combination

*Concept combination
à la Gärdenfors*

As already argued in Section 1.1.2, the process of combining different concepts into novel ones is an important aspect of concept usage [189]. In Sections 2.4.2 and 2.5.3, we have encountered the formalization of the conceptual spaces framework by Lewis and Lawry [253], which explicitly targets conjunctive concept combinations. In the following, we will, however, focus on the original proposals by Gärdenfors [179, Section 4.4] for modeling adjective-noun combinations like GREEN APPLE OR BLUE BANANA within the conceptual spaces framework. He argues that they can be expressed by combining properties with concepts. In the following, we argue that his approach can be easily implemented with our formalization.

*Adjective and noun
are compatible*

In examples like GREEN BANANA, BROWN APPLE, and YELLOW BOOK, adjective and noun are *compatible*, since they do not contain contradictory information: The cores of GREEN and BROWN have a nonempty intersection with the projections of the cores of BANANA and APPLE onto the COLOR

domain, respectively, and `BOOK` is not defined on the `COLOR` domain at all. In these cases, `GREEN`, `BROWN`, and `YELLOW` narrow down the `COLOR` information associated with the concepts `BANANA`, `APPLE`, and `BOOK`, respectively. Based on the correlations encoded in the concepts representing the noun, this may result in further updates to other domains (e.g., a `GREEN BANANA` is not expected to have a `SWEET TASTE`). In our formalization, this "narrowing down" can be implemented by simply intersecting the property with the concept, using the modified intersection from Section 3.1.2.

If adjective and noun are *incompatible*, they contain conflicting information on at least one domain. This includes examples like `BLUE BANANA`: The projection of the core of the `BANANA` concept onto the `COLOR` domain does not intersect with the core of the property `BLUE`. Moreover, their fuzzy intersection is expected to have a rather low value for μ_0 . In this case, we should *replace* the `COLOR` region of the `BANANA` concept with the `BLUE` region. In our formalization, we can achieve this by first removing the `COLOR` domain from the `BANANA` concept (through a subspace projection onto all domains but the `COLOR` domain, cf. Section 3.4.1) and by then intersecting this intermediate result with `BLUE`, using our modified intersection from Section 3.1.2.

Adjective and noun are incompatible

In examples like `TALL JOCKEY`, the interpretation of the adjective depends on the context in which it is used: A `TALL JOCKEY` is in general still shorter than a `SHORT BASKETBALL PLAYER`. The properties `TALL` and `SHORT` refer to the upper and the lower parts of the `HEIGHT` dimension, but they need to be adapted to their so-called *contrast class*. In order to apply `TALL` to `JOCKEY`, we thus first need to resize the height scale (and thus also the region of `TALL`) to the area on which `JOCKEY` is defined. Only after this adaption, we can compute the intersection of these two regions. Please note that this explicit resizing of the `HEIGHT` dimension differs from the approach of Dessalles [124], who proposes to compute the contrast vector between a given observation and the prototype of the respective concept (cf. Section 2.4.1). If this contrast vector has a large positive entry for the `HEIGHT` dimension, then the property `TALL` is applicable. Since Dessalles' approach considers points rather than regions, we follow Gärdenfors' original proposal which is more directly applicable to our region-based formalization.

The adjective is inherently relative

In examples such as `RED WINE` or `RED HAIR`, the adjective `RED` is not used in an absolute sense: `RED WINE` is actually `PURPLE` and `RED HAIR` is actually `COPPER`. Gärdenfors argues that in these cases, `RED` should be interpreted as a relative adjective. This implies that the same resizing as for `TALL` would need to happen – now, however, for the whole `COLOR` domain. Once it is resized to fit the region occupied by `WINE` or `HAIR`, we can again compute the intersection of these two regions. One may criticize that it remains somewhat unclear in which cases an adjective should be interpreted as being relative (e.g., `RED HAIR`) and in which cases a literal interpretation is more appropriate (e.g., `RED APPLE`). The approach by Dessalles [124] does not face this criticism, since it *always* makes use of contrast vectors and thus treats *all* adjectives as relative.

The adjective should be interpreted as relative in the current context

Concept	Δ_S	p^-	p^+	μ_0	c	W		
						$w_{\delta_{\text{COLOR}}}$	$w_{\delta_{\text{SHAPE}}}$	$w_{\delta_{\text{TASTE}}}$
BANANA	Δ	(0.50, 0.10, 0.35) (0.70, 0.10, 0.50) (0.75, 0.10, 0.50)	(0.75, 0.30, 0.55) (0.80, 0.30, 0.70) (0.85, 0.30, 1.00)	1.0	20.0	0.75	1.50	0.75
GREEN	$\{\delta_{\text{COLOR}}\}$	(0.45, $-\infty$, $-\infty$)	(0.55, $+\infty$, $+\infty$)	1.0	40.0	1.00	–	–
BLUE	$\{\delta_{\text{COLOR}}\}$	(0.20, $-\infty$, $-\infty$)	(0.30, $+\infty$, $+\infty$)	1.0	40.0	1.00	–	–

Table 3.7: Definitions of the BANANA concept and the properties GREEN and BLUE for the concept combination example.

More complex cases

At this point, we would also like to refer to Murphy [298, Chapter 12] who points out that there are many more cases of modifier-head constructions such as ATOMIC ENGINEER, CORPORATE LAWYER OR MOVIE PSYCHIATRIST, which cannot be easily modeled in an intersection-based way as discussed above (cf. Section 1.1.2). For the sake of simplicity, we do not consider these more complex cases for our current discussion and refer the interested reader to Gärdenfors' comments on such types of concept combinations in [181, Chapter 13] and [179, Chapter 4].

Fruit space example:
GREEN BANANA...

Instead, we illustrate the first two cases (compatible and incompatible adjective-noun pairs for non-relative adjectives) by considering the examples GREEN BANANA and BLUE BANANA in our example fruit space from Section 2.3.4. Table 3.7 shows the definitions of the BANANA concept and the properties GREEN and BLUE. As one can easily see, the intersection of the cores of \tilde{S}_{GREEN} and $\tilde{S}_{\text{BANANA}}$ is not empty, i.e., the property and the concept are compatible. We can thus derive $\tilde{S}_{\text{GREEN BANANA}}$ as follows:

```
>>> green_banana = banana.intersect_with(green)
>>> print(green_banana)
core: {[0.5, 0.1, 0.35]-[0.55, 0.3, 0.55]}
mu: 1.0
c: 20.0
weights: <{'color': 0.84, 'taste': 0.72, 'shape': 1.44},
          {'color': {0: 1.0}, 'taste': {2: 1.0}, 'shape': {1: 1.0}}>
```

... and BLUE BANANA

On the other hand, the intersection of the cores of \tilde{S}_{BLUE} and $\tilde{S}_{\text{BANANA}}$ is empty, i.e., the property and the concept are incompatible. We thus need to first project the BANANA concept onto $\Delta' = \{\delta_{\text{SHAPE}}, \delta_{\text{TASTE}}\}$ before intersecting it with the property BLUE:

```
>>> tmp = banana.project_onto({'shape':[1], 'taste':[2]})
>>> blue_banana = tmp.intersect_with(blue)
>>> print(blue_banana)
core: {[0.2, 0.1, 0.35]-[0.3, 0.3, 0.55],
       [0.2, 0.1, 0.5]-[0.3, 0.3, 1.0]}
mu: 1.0
c: 20.0
weights: <{'taste': 0.6666666666666666, 'color': 1.0,
          'shape': 1.3333333333333333},
          {'taste': {2: 1.0}, 'color': {0: 1.0}, 'shape': {1: 1.0}}>
```

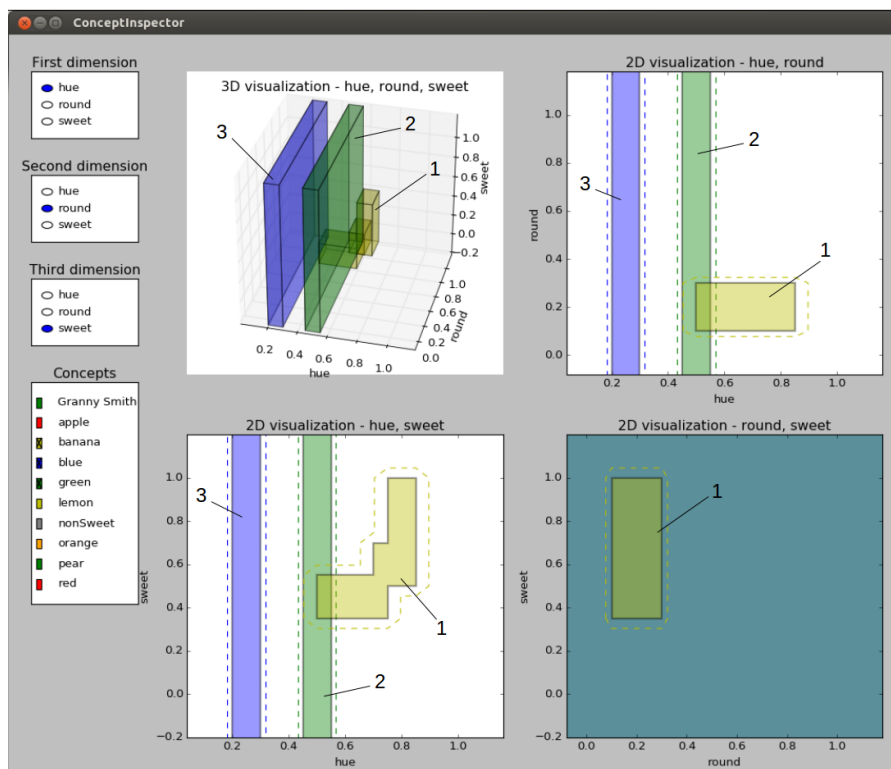


Figure 3.21: Screenshot of the ConceptInspector tool, illustrating the original concepts BANANA (1), GREEN (2), and BLUE (3) used for the concept combination example.

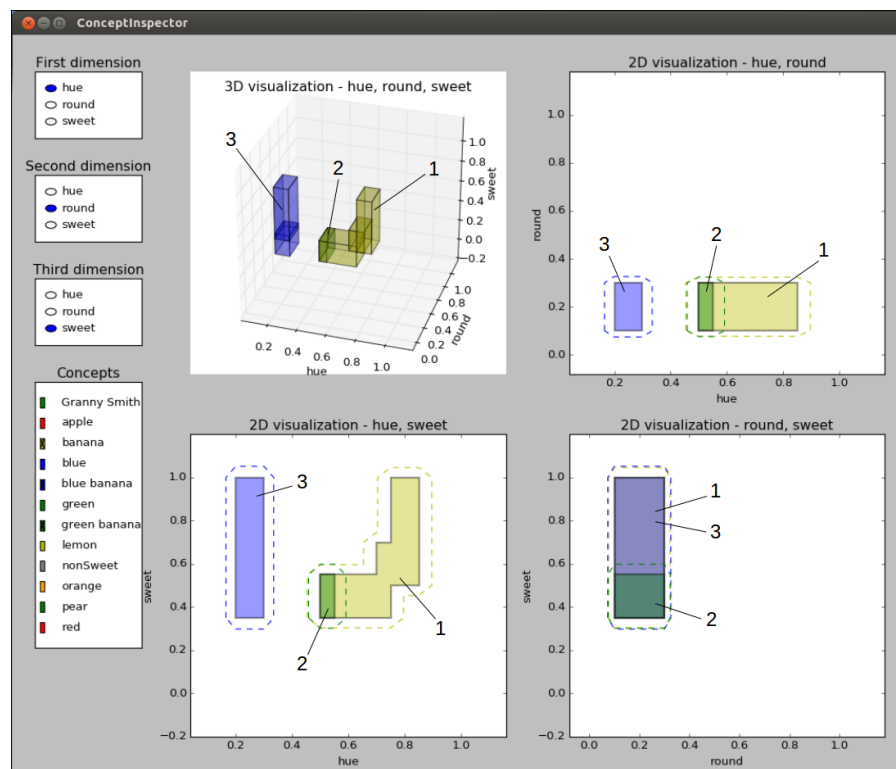


Figure 3.22: Screenshot of the Concept Inspector tool, illustrating the original BANANA cocnept (1) and the combined concepts GREEN BANANA (2) and BLUE BANANA (3).

Figures 3.21 and 3.22 illustrate the original and the resulting concepts, respectively. As one can see, in the case of GREEN BANANA, we get a much narrower area with respect to the SWEETNESS dimension than in the case of BLUE BANANA. This illustrates that the correlations between different domains can play an important role (cf. Sections 1.1.2 and 2.2.1: A GREEN BANANA comes with a clear expectation about its TASTE, while we cannot make such predictions for a BLUE BANANA.

Visualization and interpretation

Banaee et al. [26, 27] consider the somewhat inverse problem of generating descriptions for unknown observations (cf. Section 1.2.5): Instead of using a verbal description to narrow down the conceptual region according to the concept combination procedure from above, they start with a point in the conceptual space and try to find a verbal description using a combination of existing concepts.

Generating a verbal description as inverse concept combination

Banaee et al. propose to check for each domain individually whether the new observation is contained in any conceptual region. If this is not the case, they compute the distance to the closest conceptual regions. The membership and distance information is then used to generate phrases like "The observation is similar to X, but very Y and a bit Z." where X is a concept (i.e., a region in the conceptual space), and Y and Z are properties (which in their approach are defined as intervals on individual dimensions of the conceptual space) in which the given observation differs from the given concept X.⁷ Banaee et al. were able to show that the descriptions generated by their system are sufficient for humans to select the correct stimulus among a given set of candidates. Moreover, the descriptions of their system involve both concepts and properties, which was rated as more helpful by users than the descriptions of two other competitor systems which used only concepts and only properties, respectively.

The approach

In principle, this procedure is also supported by our formalization: By computing the membership of a given new observation to all the concepts, one can select the best matching concept (or none, if all membership values are small). If the match was not perfect (indicated by a membership value below a certain threshold), one can look at the domains that contribute most to the mismatch. This can for instance be done by projecting the concept onto each individual domain and computing the membership of the given observation to the projected concepts. Also a contrast vector approach (using the closest point in the concept's core) could be implemented to this end. In these domains, the properties to which the observation has the highest membership can be used for enriching the description. While we have not conducted any experiments in this direction, we think that in principle it is possible to devise a procedure similar to the one proposed by Banaee et al. using our formalization of conceptual spaces.

Description generation with our formalization

⁷ This difference-based description is somewhat reminiscent of the contrast vector approach by Dessalles [124] (see Section 2.4.1).

Formalization	Intersection	Union	Negation	Subspace Projection	Axis-Parallel Cut
Aisbett and Gibbon [11]	(-)	(-)	-	(-)	(-)
Raubal [327]	-	-	-	-	-
Ahlqvist [10]	(-)	(-)	-	(-)	(-)
Rickard [329]	-	-	-	(-)	-
Rickard et al. [330]	-	-	-	(-)	-
Adams and Raubal [3]	✓	(-)	-	(-)	(-)
Lewis and Lawry [253]	✓	-	-	(-)	(-)
Our Formalization	✓	✓	-	✓	✓

Table 3.8: Overview of different formalizations of the conceptual spaces framework based on their supported operations on concepts ("✓" means "available", "-" means "not available", and "(-)" means "could be easily added").

Outlook

Just as for the concept formation scenario, also our discussion of concept combination is necessarily brief and of limited depth. Nevertheless, we were able to sketch how some of the important mechanisms can be in principle mapped onto our formalization, using the intersection and projection operations from Sections 3.1.2 and 3.4.1, respectively. Needless to say, these sketched proposals need to be implemented and tested in the future to obtain empirical support.

3.7 COMPARISON TO OTHER FORMALIZATIONS

Overview

Table 3.8 summarizes the availability of different concept creation and concept combination operations in different formalizations of the conceptual spaces framework that we introduced in Section 2.4.2. From all operations we consider, only the intersection is explicitly implemented in some of the formalizations from the literature. Union, subspace projection, and axis-parallel cut are never considered explicitly, but can be quite easily added to many of the formalizations. The negation operation is inherently difficult for the general conceptual spaces approach and can thus not successfully be addressed in any of the formalizations. Let us now look at the individual formalizations in more detail.

The formalization by Aisbett and Gibbon [11] describes concepts as convex regions in pointed metric spaces. It is mostly concerned with the interplay of symbols and geometrically represented concepts in a dynamic system and does not explicitly provide any constructive operations for combining concepts. However, both the intersection of conceptual regions and the projection of conceptual regions onto individual domains are used at various points of their work and could therefore easily be formalized as explicit operations. As properties in their formalization can consist of either a single region ("natural property") or a set of regions ("complex property"), a union of multiple properties or concepts also does not seem to be problematic. Finally, also an axis-parallel cut which divides a given concept into two parts based on a threshold on a single dimension can in principle be applied to their representation, as the resulting regions are still guaranteed to remain convex. While no explicit operations are provided, they could thus be easily added.

*Aisbett and Gibbon:
operations could be
easily added*

Since Raubal's formalization [327] does not give a mathematical definition for properties and concepts, it also does not contain any combination operations on concepts. Also adding such operations is not possible unless concepts and properties are properly formalized.

*Raubal: no
operations at all*

Ahlqvist's formalization [10] (which is based on rough fuzzy sets) does not define any combination operations on concepts. One can, however, imagine relatively straightforward definitions for intersection and subspace projection based on the standard definitions for fuzzy sets (cf. Sections 3.1.2 and 3.4.1). The intersection of two concepts can be defined as the intersection of the respective rough fuzzy sets, and a subspace projection can be implemented by removing the respective domains from the given concept. Since Ahlqvist's formalization does not explicitly require concepts to be convex or even connected, the result of the union operation does not need to fulfill any particular constraints. Thus, also a union of concepts can be easily added based on the respective definition for fuzzy sets (cf. Section 3.2.1). Finally, also an axis-parallel cut could be incorporated by modifying the membership functions of the rough fuzzy set in an appropriate way. While the formalization itself does not come with any operations, it can thus be easily extended based on fuzzy set theory.

*Ahlqvist: adding
standard fuzzy set
operations*

The formalization by Rickard [329] represents concepts as co-occurrence matrices of properties. It does not explicitly define any generative operations on concepts. One could, however, imagine that a subspace projection can be achieved by simply removing the rows and columns of the co-occurrence matrix which represent properties from the domain one wishes to remove. Intersection and union of concepts would need to operate on the respective co-occurrence matrices. If the original observations used to estimate the co-occurrence statistics are still available, one could compute the co-occurrence values for the intersection and union of the observations, respectively. If, however, only the co-occurrence

*Rickard:
co-occurrence
matrices are too
inflexible*

matrices are given, a proper transformation reflecting the intersection or union of the concepts may be more difficult to obtain. Finally, an axis-parallel cut would correspond to splitting one property into two parts which in turn would modify the number of properties used in *all* co-occurrence matrices. Thus, an axis-parallel cut applied to one concept \tilde{S} would potentially also influence other concepts $\tilde{S}' \neq \tilde{S}$. It therefore seems that an axis-parallel cut cannot be easily added to Rickard's formalization.

Rickard et al.: same criticism as for Rickard

As the formalization by Rickard et al. [330] is quite similar to Rickard's proposal [329], the comments made above directly apply to this formalization as well. Although Rickard et al. define an intersection operation for properties, this operation cannot be easily generalized to concepts, because concepts and properties are represented in different ways. Both the formalizations by Rickard [329] and by Rickard et al. [330] are thus not capable of providing many of the operations under consideration due to the representation of concepts as co-occurrence matrices.

Adams and Raubal: intersection available, others can be easily added

Adams and Raubal [3] define concepts by using one convex polytope per domain. Their formalization also provides some constructive operations on concepts, namely the intersection of convex polytopes and the conjunctive combination of concepts discussed in Section 3.6.2. While not being formalized, one can easily imagine definitions for union, subspace projection, and axis-parallel cut: The union operation can for instance be implemented by computing the convex hull of the two original polytopes for each domain. The subspace projection can be formalized by removing the convex polytopes of the domains to be eliminated from the given concept. Finally, the axis-parallel cut can be realized by adding another hyperplane to two copies of the original polytope. The resulting regions are still guaranteed to be convex polytopes. Overall, this formalization provides an explicit definition of the intersection operation, and could be easily extended to include most of the other operations as well.

Lewis and Lawry: only union and negation difficult to add

Lewis and Lawry [253] use random sets for representing both properties and concepts. They focus their formalization on conjunctive concept combination. An intersection of concepts is just a special case of this conjunctive concept combination (namely, when both concepts are defined on the same set of underlying domains) and is thus available in their formalization. A union of concepts, however, cannot be easily added to their formalization as it might lead to non-convex results. Again, a subspace projection is not explicitly formalized, but could potentially be implemented by removing the respective domains from a concept's definition. An axis-parallel cut would mainly change one of the properties based on which a concept is defined. One would then need to replace this property in the respective combination space, which seems to be a feasible extension of their formalization.

Our own formalization

Finally, our own formalization includes almost all operations listed in Table 3.8. Like all other formalizations, we are, however, unable to give a

meaningful definition for conceptual negation. As argued in Section 3.3 this problem is caused by adopting a geometric approach of knowledge representation based on Minkowski metrics and by requiring concepts to be star-shaped. It is thus an inherent shortcoming of the overall conceptual spaces approach. Nevertheless, Table 3.8 illustrates that our proposed formalization covers a wider variety of generative conceptual operations than any of the previous formalizations.

3.8 SUMMARY

In this chapter, we have extended our formalization of the conceptual spaces framework with a comprehensive set of operations for creating new concepts based on existing ones. Each of these operations has been included in our implementation, and example usages have been illustrated. Moreover, we have argued that this set of operations is useful for carrying out learning tasks like concept formation and reasoning tasks like concept combination.

Lessons learned

The work presented in this chapter leaves open various strands of future research: With respect to the intersection and union operations, we have argued to use a simple midpoint heuristic for restoring star-shapedness. The exact implications of using this heuristic versus a numerical optimization for finding the optimal solution are unclear and should be investigated. From an implementational point of view, also the runtime of the intersection operation urges for further improvements. Furthermore, we observed that the union of LEMON and ORANGE yields unintuitive results and explained this by the very simplified structure of the conceptual fruit space. This of course calls for more complex conceptual spaces to be used as further test cases for all the operations defined in this chapter. Moreover, our formalization is so far lacking a meaningful negation operation. Given that this is an inherent weakness of the conceptual spaces framework itself, it might also be a worthwhile avenue of future research that would not only benefit our formalization, but the conceptual spaces community as a whole. Finally, although we have argued in Section 3.6 that the operations presented in this chapter are useful for both learning and reasoning processes, this has to be supported with empirical evidence from actual applications.

Open ends

Although the operations defined in this chapter can be applied to any concepts (e.g., by taking the union of the concepts CHAIR and JOCKEY), there need to be some constraints that indicate whether applying this operator is actually cognitively plausible and meaningful. These constraints could for instance be based on the number of shared domains and on the similarity of the original concepts. In Chapter 4, we will define several ways of measuring relations between concepts, which can potentially be used for making such decisions.

Outlook

4

MEASURING RELATIONS BETWEEN CONCEPTS

4.1	Concept Size	148
4.1.1	The Size of a Fuzzified Cuboid's α -Cut	149
4.1.2	The Size of a Concept	155
4.1.3	Implementation and Example	157
4.2	Subthood	161
4.2.1	Definition	162
4.2.2	Implementation and Example	166
4.3	Implication	168
4.3.1	Definition	168
4.3.2	Implementation and Example	169
4.4	Concept Similarity	171
4.4.1	Similarity as Inverse Distance	172
4.4.2	Definition	177
4.4.3	Implementation and Example	179
4.5	Betweenness	184
4.5.1	Betweenness in the Literature	185
4.5.2	Definition	187
4.5.3	Implementation and Example	190
4.6	Supported Applications	199
4.6.1	Concept Formation	199
4.6.2	Commonsense Reasoning	200
4.7	Comparison to Other Formalizations	203
4.8	Summary	206

So far, we have developed mathematical definitions for conceptual spaces and concepts (Chapter 2) as well as operations for creating new concepts based on existing ones (Chapter 3). In this chapter, we provide formal ways of measuring the relations between different concepts. These relations can be used for various types of commonsense reasoning, and for deciding whether or not one of the operations from Chapter 3 should be applied to a given concept.

Motivation

The size of a conceptual region gives an intuition about its position in the conceptual hierarchy: If the conceptual region is quite small, then only few observations belong to this concept, hence it is quite specific (such as GRANNY SMITH). On the other hand, if the conceptual region is very large, then it includes many possible observations (such as FRUIT). While the absolute size of a conceptual region may not be very informative, comparing the size of different conceptual regions to each other gives an impression of their relative level of generality. In Section 4.1, we therefore develop an analytical solution for computing the size of a fuzzy conceptual regions as defined in Chapter 2.

Size

Subsethood Also the geometric notion of subsethood can be related to conceptual hierarchies: Since the region representing GRANNY SMITH is a subset of the region representing APPLE, we know that all observations that belong to the GRANNY SMITH concept also belong to the APPLE concept. This means that GRANNY SMITH is a subordinate concept (i.e., a specialization) of APPLE. In Section 4.2, we therefore propose a way for measuring the degree of subsethood between two concepts.

Implication In many symbolic AI systems, rules in the form $APPLE \wedge RED \Rightarrow SWEET$ play an important role. We have already discussed in Chapter 3 how the logical conjunction and disjunction operators can be mapped onto intersection and union, respectively. In Section 4.3, we argue that in the context of conceptual spaces, the degree of implication between two concepts can be equated with their degree of subsethood.

Similarity As we have stated earlier, conceptual spaces are based on the notion of semantic similarity: Distance in the conceptual space is inversely related to similarity. So far, we have only considered the similarity between points in the conceptual space, i.e., between individual observations. However, we may not only be interested in the particular similarity between a specific apple and a specific pear, but in the general similarity of the underlying concepts APPLE and PEAR. In Section 4.4, we propose two different definitions for conceptual similarity which both fulfill a set of desirable properties.

Betweenness In Chapter 2, we have used the ternary relation of geometric betweenness to define both convex and star-shaped sets. In the conceptual spaces framework, geometric betweenness can be equated with conceptual betweenness which can help us to identify "intermediate" cases between two observations. In Section 4.5, we generalize the notion of betweenness from points in the conceptual space to concepts in order to quantify to which degree MASTER STUDENT is conceptually between BACHELOR STUDENT and PHD STUDENT. This can be used for commonsense reasoning, e.g., by concluding that all rules applying to both master and PhD students should also apply to master students.

*Examples,
application scenarios,
and related work*

For each of these relations, we provide a mathematical definition along with an illustrative example based on our fruit space from Section 2.3.4. We then illustrate their usefulness for practical applications in Section 4.6. In Section 4.7, we compare our work to other existing formalizations of the conceptual spaces framework before concluding this chapter in Section 4.8.

The research contributions presented in this chapter have previously been published in [41, 43, 46, 45].

4.1 CONCEPT SIZE

*Size as specificity of a
concept*

The size of a conceptual region gives an intuition about the specificity of this concept: Concepts represented by a large region are more general than concepts represented by a small region. This is one obvious aspect in which one can compare two concepts to each other. Moreover, the size

of a concept indicates the average (or maximal) dissimilarity of any two observations that belong to this region.¹ It therefore gives an indication about the "coherence" of the region and hence about the concept's overall position in the concept hierarchy: Superordinate concepts tend to be more heterogeneous than base level and subordinate concepts [298, Chapter 7] (cf. also Section 1.1.2).

We follow a bottom-up approach for defining the size of a conceptual region by first considering the size of an individual α -cut of a fuzzified cuboid in Section 4.1.1. In Section 4.1.2 we then integrate over all α -cuts to compute the size of a fuzzified cuboid and use this to obtain a closed formula for computing the size of a conceptual region. Finally, we present both a runtime analysis of our implementation and an illustrative example in Section 4.1.3.

Overview

4.1.1 The Size of a Fuzzified Cuboid's α -Cut

In this section, we consider the fuzzified version \tilde{C} of a cuboid $C \in S$, which can be interpreted as a single-cuboid concept $\tilde{C} = \langle C, \mu_0, c, W \rangle$ and which has already been defined in Section 3.1.3:

Fuzzified cuboids

Definition 3.4 (Fuzzified Cuboid)

Let $\tilde{S} = \langle S, \mu_0, c, W \rangle$ be a concept and $C \in S$ be a cuboid from its core. Then, the membership to the fuzzified cuboid \tilde{C} is defined as follows:

$$\mu_{\tilde{C}}(x) = \mu_0 \cdot \max_{y \in C} \left(e^{-c \cdot d_C^{\Delta}(x, y, W)} \right)$$

Figure 4.1 illustrates the α -cut of a two-dimensional fuzzified cuboid \tilde{C} both under the Euclidean metric (a) and under the Manhattan metric (b). As one can see, the area of these α -cuts can be divided into three different components (I–III). Let us first consider only the Euclidean case, i.e., Figure 4.1a, which is relevant if both dimensions belong to the same domain δ .

Visualization

The first component (I) corresponds to the crisp cuboid, which is defined based on its support points p^+ and p^- (cf. Definition 2.8). Its size can be computed as $(p_1^+ - p_1^-) \cdot (p_2^+ - p_2^-)$. If we introduce the shorthand notation $b_d = p_d^+ - p_d^-$, the size V_I of the first component can be written as follows:

Component I: crisp cuboid

$$V_I = b_1 \cdot b_2 \tag{4.1}$$

The second component (II) corresponds to points that violate the cuboid inequalities with respect to a single dimension. For instance, the right part of component II (i.e., the region labeled as II_a) corresponds to points that are in the interval $[p_1^+, p_1^+ + \phi] \times [p_2^-, p_2^+]$, i.e. all points that are at most ϕ to the right of the cuboid. Remember from Lemma 2.4 that each α -cut can be interpreted as an ϵ -neighborhood of the original cuboid C with $\epsilon = -\frac{1}{c} \cdot \ln \left(\frac{\alpha}{\mu_0} \right)$. We thus know that the distance

Component II: differences on a single dimension

¹ Thanks to Nina Poth for pointing this out to me.

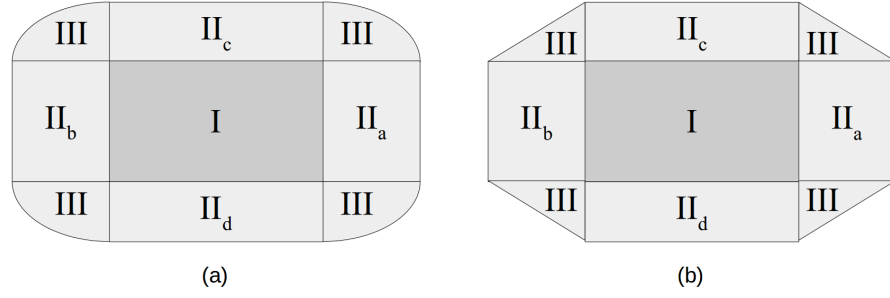


Figure 4.1: α -cut of a fuzzified cuboid under the Euclidean distance (a) and the Manhattan distance (b), respectively.

between any point x in the region II_a and its closest point $y \in C$ is bounded by ϵ :

$$\forall x \in [p_1^+, p_1^+ + \phi] \times [p_2^-, p_2^+] : \exists y \in C : d_C^{\Delta}(x, y) \leq -\frac{1}{c} \cdot \ln\left(\frac{\alpha}{\mu_0}\right)$$

Considering only d_1

Since we can always find a $y \in C$ such that $y_2 = x_2$, we only need to look at differences with respect to the dimension d_1 . Therefore, the distance can be written as follows:

$$d_C(x, y) = w_\delta \cdot \sqrt{w_1 \cdot (x_1 - y_1)^2}$$

Constraints for ϕ

Our ϕ corresponds to the largest distance with respect to the dimension d_1 that is allowed for an α -cut. That is, we look for $\phi = |x_1 - y_1|$ such that the following equation holds:

$$w_\delta \cdot \sqrt{w_1 \cdot \phi^2} = -\frac{1}{c} \cdot \ln\left(\frac{\alpha}{\mu_0}\right)$$

Solving for ϕ

Solving for ϕ yields the following result:

$$\phi = -\frac{1}{c \cdot w_\delta \cdot \sqrt{w_1}} \cdot \ln\left(\frac{\alpha}{\mu_0}\right)$$

The size of component II_a

The size of component II_a can now be computed by multiplying its width and its height:

$$\begin{aligned} V_{II_a} &= (p_1^+ + \phi - p_1^+) \cdot (p_2^+ - p_2^-) \\ &= -\frac{1}{c \cdot w_\delta \cdot \sqrt{w_1}} \cdot \ln\left(\frac{\alpha}{\mu_0}\right) \cdot (p_2^+ - p_2^-) \\ &= -\frac{1}{c \cdot w_\delta \cdot \sqrt{w_1}} \cdot \ln\left(\frac{\alpha}{\mu_0}\right) \cdot b_2 \end{aligned}$$

Putting together the pieces for component II

The size of component II_b is identical to V_{II_a} , and analogously we find that $V_{II_c} = V_{II_d} = -\frac{1}{c \cdot w_\delta \cdot \sqrt{w_2}} \cdot \ln\left(\frac{\alpha}{\mu_0}\right) \cdot b_1$. So the overall area that is covered by all parts of component II can be written as follows:

$$V_{II} = 2 \cdot V_{II_a} + 2 \cdot V_{II_c} = -\frac{1}{c} \cdot \ln\left(\frac{\alpha}{\mu_0}\right) \cdot \left(\frac{2 \cdot b_1}{w_\delta \cdot \sqrt{w_2}} + \frac{2 \cdot b_2}{w_\delta \cdot \sqrt{w_1}}\right) \quad (4.2)$$

The third component (III) consists of four parts that taken together form an ellipse. The major axis of the ellipse has a radius equal to the width of component II_a , and the minor axis of the ellipse has a radius equal to the height of component II_c :

$$r_1 = -\frac{1}{c} \cdot \ln\left(\frac{\alpha}{\mu_0}\right) \cdot \frac{1}{w_\delta \cdot \sqrt{w_1}}, \quad r_2 = -\frac{1}{c} \cdot \ln\left(\frac{\alpha}{\mu_0}\right) \cdot \frac{1}{w_\delta \cdot \sqrt{w_2}}$$

The overall size of this ellipse can be expressed as follows:

$$V_{III} = \pi \cdot r_1 \cdot r_2 = \pi \cdot \ln\left(\frac{\alpha}{\mu_0}\right)^2 \cdot \frac{1}{c^2 \cdot w_\delta^2 \cdot \sqrt{w_1} \cdot \sqrt{w_2}} \quad (4.3)$$

Having computed the size of the individual components in Equations 4.1, 4.2, and 4.3, we can now describe the size of the overall α -cut:

$$\begin{aligned} V(\tilde{C}^\alpha) &= V_I + V_{II} + V_{III} \\ &= b_1 \cdot b_2 - 2 \cdot \frac{1}{c} \cdot \ln\left(\frac{\alpha}{\mu_0}\right) \cdot \left(\frac{b_1}{w_\delta \cdot \sqrt{w_2}} + \frac{b_2}{w_\delta \cdot \sqrt{w_1}}\right) \\ &\quad + \pi \cdot \ln\left(\frac{\alpha}{\mu_0}\right)^2 \cdot \frac{1}{c^2 \cdot w_\delta^2 \cdot \sqrt{w_1} \cdot \sqrt{w_2}} \end{aligned} \quad (4.4)$$

If we now look at the Manhattan example (Figure 4.1b), where the two dimensions belong to different domains $\delta_1 \neq \delta_2$, we see that the components I and II can be computed analogously to the Euclidean case. However, under the Manhattan metric, the four parts of component III do not form an ellipse. Instead, we have four triangles. The width of each of these triangles corresponds to the radius of the major axis in the Euclidean case. Correspondingly, their height is analogous to the radius of the ellipse's minor axis. We thus get the following formula for the area enclosed by component III:

$$V_{III} = 4 \cdot \frac{1}{2} \cdot r_1 \cdot r_2 = 2 \cdot \ln\left(\frac{\alpha}{\mu_0}\right)^2 \cdot \frac{1}{c^2 \cdot w_{\delta_1} \cdot \sqrt{w_1} \cdot w_{\delta_2} \sqrt{w_2}}$$

In the Manhattan case, we thus get the overall formula:

$$\begin{aligned} V(\tilde{C}^\alpha) &= V_I + V_{II} + V_{III} \\ &= b_1 \cdot b_2 - 2 \cdot \frac{1}{c} \cdot \ln\left(\frac{\alpha}{\mu_0}\right) \cdot \left(\frac{b_1}{w_{\delta_2} \cdot \sqrt{w_2}} + \frac{b_2}{w_{\delta_1} \cdot \sqrt{w_1}}\right) \\ &\quad + 2 \cdot \ln\left(\frac{\alpha}{\mu_0}\right)^2 \cdot \frac{1}{c^2 \cdot w_{\delta_1} \cdot \sqrt{w_1} \cdot w_{\delta_2} \sqrt{w_2}} \end{aligned} \quad (4.5)$$

When comparing Equations 4.4 and 4.5, we see that the most prominent difference between the Manhattan and the Euclidean case concerns component III. This difference is caused by the different forms of hyperballs under both metrics (illustrated in Figure 4.2a). In general, a hyperball of radius r around a point p can be defined as the set of all points with a distance of at most r to p :

$$H = \{x \in CS \mid d(x, p) \leq r\}$$

*Component III:
differences on both
dimensions*

The size of the ellipse

*Overall size of the
 α -cut*

Manhattan example

Overall size

*Main difference:
component III*

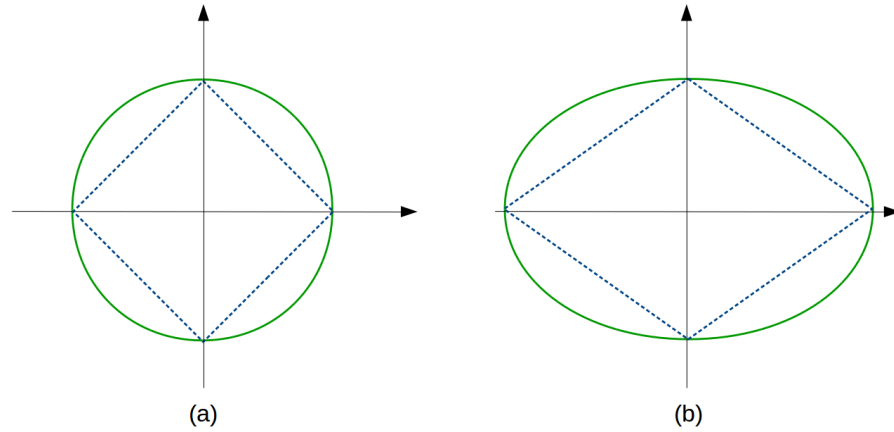


Figure 4.2: (a) A two-dimensional hyperball (i.e., a disk) under the Euclidean metric (green, solid) and under the Manhattan metric (blue, dotted). (b) Stretching the disks along a single dimension results in ellipses.

Hyperballs and hyperellipses

For different distance metrics the hyperballs resulting from this definition have a different shape: They are round for the Euclidean metric and diamond-shaped for the Manhattan metric. By stretching these hyperballs, one can obtain hyperellipses (see Figure 4.2b). It is easy to see that component III corresponds to such an ellipse in both the Euclidean and the Manhattan case. This stretching is based on the salience weights W . For instance, in Figure 4.1a, we assume that $w_{d_1} < w_{d_2}$ which means that we allow larger differences with respect to d_1 than with respect to d_2 . This causes the hyperball representing component III to be stretched along d_1 , thus obtaining the shape of a horizontal ellipse.

Components as extruded ellipses

Looking closer at Figure 4.1, we can observe that the other two components can also be described by ellipses: Component I is a zero-dimensional ellipse (i.e., a point) that was extruded in two dimensions with extrusion lengths of b_1 and b_2 , respectively. Component II consists of two one-dimensional ellipses (i.e., line segments) that were extruded in one dimension. Finally, component III consists of a single two-dimensional ellipse that was not extruded at all.

Notation for domains...

Let us denote by $\Delta_{\{d_1, \dots, d_i\}}$ the domain structure obtained by eliminating from Δ all dimensions $d \in D \setminus \{d_1, \dots, d_i\}$. The following example illustrates this notation:

$$\begin{aligned} \Delta &= \{\{d_1, d_2, d_3\}, \{d_4\}, \{d_5, d_6\}\} \\ \Delta_{\{d_1, d_2, d_3\}} &= \{\{d_1, d_2, d_3\}\} \\ \Delta_{\{d_1, d_2, d_5\}} &= \{\{d_1, d_2\}, \{d_5\}\} \\ \Delta_{\{d_2, d_4, d_6\}} &= \{\{d_2\}, \{d_4\}, \{d_6\}\} \end{aligned}$$

... and hyperballs

Moreover, let $V(r, \Delta, W)$ be the size of a hyperball with radius r in a space specified by the domain structure Δ , that was stretched according to the weights in W (thus potentially taking the form of an ellipse).

Based on our observations about the three components of \tilde{C}^α in the two-dimensional case, we can now write $V(\tilde{C}^\alpha)$ as follows:

α -cut as union of extruded ellipses

$$\begin{aligned} V(\tilde{C}^\alpha) &= V\left(-\frac{1}{c} \cdot \ln\left(\frac{\alpha}{\mu_0}\right), W, \Delta_\emptyset\right) \cdot b_1 \cdot b_2 \\ &\quad + V\left(-\frac{1}{c} \cdot \ln\left(\frac{\alpha}{\mu_0}\right), W, \Delta_{\{d_1\}}\right) \cdot b_2 \\ &\quad + V\left(-\frac{1}{c} \cdot \ln\left(\frac{\alpha}{\mu_0}\right), W, \Delta_{\{d_2\}}\right) \cdot b_1 \\ &\quad + V\left(-\frac{1}{c} \cdot \ln\left(\frac{\alpha}{\mu_0}\right), W, \Delta\right) \end{aligned} \quad (4.6)$$

One can generalize this formula to higher dimensions:

Arbitrary number of dimensions

$$\begin{aligned} V(\tilde{C}^\alpha) &= \sum_{i=0}^n \left(\sum_{\substack{\{d_1, \dots, d_i\} \\ \subseteq D}} \left(\prod_{d \in D \setminus \{d_1, \dots, d_i\}} b_d \right) \right. \\ &\quad \left. \cdot V\left(-\frac{1}{c} \cdot \ln\left(\frac{\alpha}{\mu_0}\right), \Delta_{\{d_1, \dots, d_i\}}, W\right) \right) \end{aligned} \quad (4.7)$$

The outer sum of Equation 4.7 runs over the number n of dimensions with respect to which a given point $x \in \tilde{C}^\alpha$ lies outside of C . We then sum over all combinations $\{d_1, \dots, d_i\}$ of dimensions for which this could be the case, compute the volume $V(\cdot, \cdot, \cdot)$ of the i -dimensional hyperball under these dimensions, and extrude this intermediate result in all remaining dimensions by multiplying with $\prod_{d \in D \setminus \{d_1, \dots, d_i\}} b_d$.

Interpretation

Let us illustrate Equation 4.7 for the α -cuts from Figure 4.1: For $i = 0$, we can only select the empty set for the inner sum, so we end up with $b_1 \cdot b_2$, which is the size of the original cuboid (i.e., component I). For $i = 1$, we can either pick $\{d_1\}$ or $\{d_2\}$ in the inner sum. For $\{d_1\}$, we compute the size of the left and right part of component II by multiplying $V\left(-\frac{1}{c} \cdot \ln\left(\frac{\alpha}{\mu_0}\right), \Delta_{\{d_1\}}, W\right)$ (i.e., their combined width) with b_2 (i.e., their height). For $\{d_2\}$, we analogously compute the size of the upper and the lower part of component II. Finally, for $i = 2$, we can only pick $\{d_1, d_2\}$ in the inner sum, leaving us with $V\left(-\frac{1}{c} \cdot \ln\left(\frac{\alpha}{\mu_0}\right), \Delta, W\right)$, which is the size of component III. As one can thus see, Equation 4.7 results in Equation 4.6 in the two-dimensional case, which covers both Equations 4.4 and 4.5 as manually derived earlier.

Two-dimensional example

If all dimensions belong to the same domain (i.e., $\Delta = \{\{d_1, \dots, d_n\}\}$) or if all dimensions belong to different domains (which corresponds to $\Delta = \{\{d_1\}, \dots, \{d_n\}\}$), it is quite straightforward to compute $V(r, \Delta, W)$: One can simply use the size of a stretched Euclidean or Manhattan hyperball (see, e.g., [426]). However, when we consider cuboids with a dimensionality of $n \geq 3$, we in general need to deal with the combined metric d_C from Definition 2.7.

Hyperballs under the combined metric

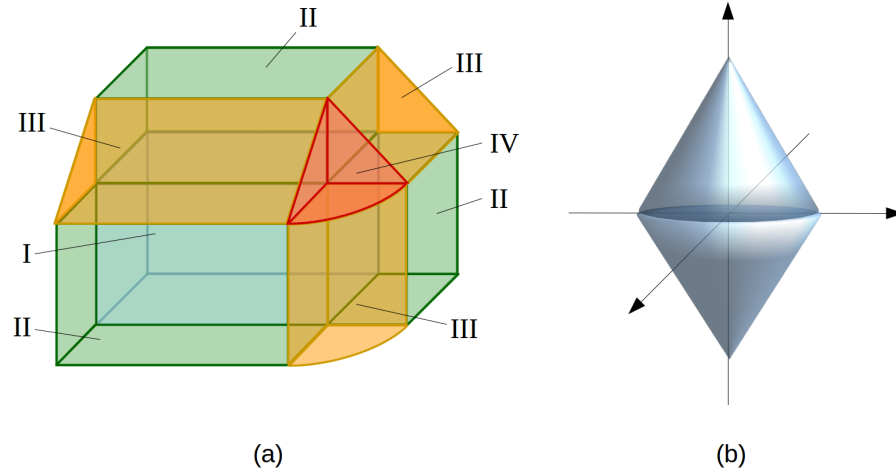


Figure 4.3: (a) Illustration of a three-dimensional fuzzified cuboid’s α -cut in a space consisting of two domains. (b) A hyperball in a three-dimensional space with two domains has the shape of a double cone.

A three-dimensional example

Figure 4.3a illustrates parts of a three-dimensional fuzzified cuboid’s α -cut in a space with a domain structure $\Delta = \{\{d_1, d_2\}, \{d_3\}\}$. As one can see, this α -cut can be again described as a set of extruded ellipses: The blue cuboid (labeled as I) is the original cuboid itself (i.e., a zero-dimensional ellipse extruded in three dimensions) and the green cuboids (II) are parts of one-dimensional ellipses extruded in two additional dimensions. The yellow shapes (III) are parts of two-dimensional ellipses extruded in one additional dimension², and the red shape (IV) is a part of a three-dimensional ellipse. It is interesting to observe that the three-dimensional ellipse under the combined metric in this case has the shape of a double cone (illustrated in Figure 4.3b).

Hyperball volume under d_C^Δ without salience weights

The following proposition gives us a general way of computing the size of a hyperball under the combined metric d_C^Δ , independent of the underlying domain structure. It makes use of Euler’s gamma function $\Gamma(z) = \int_0^\infty x^{z-1} e^{-x} dx$, which can be interpreted as a generalization of the factorial to real values, since $\forall n \in \mathbb{N} : \Gamma(n) = (n - 1)!$.

Proposition 4.1 (Volume of a Hyperball without Salience Weights)

The size of a hyperball with radius r in a space with the combined metric d_C^Δ , the domain structure Δ , and without domain and dimension weights can be computed in the following way, where n is the overall number of dimensions, n_δ is the number of dimensions in domain δ , and $\Gamma(\cdot)$ is Euler’s gamma function:

$$V(r, \Delta) = \frac{r^n}{n!} \prod_{\delta \in \Delta} \left(n_\delta! \frac{\pi^{\frac{n_\delta}{2}}}{\Gamma\left(\frac{n_\delta}{2} + 1\right)} \right)$$

Proof. See Appendix C.1. □

² Please note that there are two different types of these extruded 2D ellipses, two based on Manhattan ellipses and one based on a Euclidean ellipse.

Proposition 4.1 only considers an unweighted variant of d_C^Δ , where all domain and dimension weights are set to one. If we introduce salience weights as used in our formalization, we obtain the following result:

Adding salience weights

Proposition 4.2 (Volume of a Hyperball with Saliency Weights)

The size of a hyperball with radius r in a space with the weighted combined metric d_C , the domain structure Δ , and the set of weights W can be computed in the following way, where n is the overall number of dimensions, n_δ is the number of dimensions in domain δ , $\Gamma(\cdot)$ is Euler's gamma function, and $\delta(d)$ is the unique $\delta \in \Delta$ with $d \in \delta$:

$$V(r, \Delta, W) = \frac{1}{\prod_{d \in D} w_{\delta(d)} \sqrt{w_d}} \cdot \frac{r^n}{n!} \cdot \prod_{\delta \in \Delta} \left(n_\delta! \cdot \frac{\pi^{\frac{n_\delta}{2}}}{\Gamma(\frac{n_\delta}{2} + 1)} \right)$$

Proof. See Appendix C.1. □

We can now derive an equation for $V(\tilde{C}^\alpha)$, i.e., the size of a fuzzy cuboid's α -cut by combining Equation 4.7 and Proposition 4.2:

The size of an α -cut

Proposition 4.3 (Size of an α -Cut)

For a given fuzzified cuboid $\tilde{C} = \langle C, \mu_0, c, W \rangle$ and $\alpha \in [0, \mu_0]$, we can describe the size of its α -cut as follows, using $a_d = w_{\delta(d)} \cdot \sqrt{w_d} \cdot (p_d^+ - p_d^-) \cdot c$:

$$V(\tilde{C}^\alpha) = \frac{1}{c^n \prod_{d \in D} w_{\delta(d)} \sqrt{w_d}} \sum_{i=0}^n \left(\frac{(-1)^i \cdot \ln\left(\frac{\alpha}{\mu_0}\right)^i}{i!} \cdot \sum_{\substack{\{d_1, \dots, d_i\} \\ \subseteq D}} \left(\prod_{d \in D \setminus \{d_1, \dots, d_i\}} a_d \right) \cdot \prod_{\substack{\delta \in \\ \Delta_{\{d_1, \dots, d_i\}}} \left(n_\delta! \cdot \frac{\pi^{\frac{n_\delta}{2}}}{\Gamma(\frac{n_\delta}{2} + 1)} \right) \right)$$

Proof. See Appendix C.1. □

Proposition 4.3 gives us a closed (albeit quite complex) formula for computing the size of a fuzzified cuboid's α -cut. This will serve as our starting point for computing the size of a fuzzy conceptual region.

Outlook

4.1.2 The Size of a Concept

So far, we have only considered the size of a single α -cut of a single fuzzified cuboid. In the following, we will first derive the size of an overall fuzzified cuboid based on Proposition 4.3, before showing how to compute the size of an overall concept \tilde{S} .

Overview

When working with crisp sets, one can use the set cardinality to specify the size of a set. In the discrete case, set cardinality corresponds to the number of elements in the set. For a fuzzy set, one can use a so-called *measure* M to describe its size. This measure can in our context be defined as follows (cf. [74]):

Measure of a fuzzy set

Definition 4.1 (Measure)

A measure M on a conceptual space CS is a function $M : \mathcal{F}(CS) \rightarrow \mathbb{R}_0^+$ with $M(\emptyset) = 0$ and $(\tilde{A} \subseteq \tilde{B}) \Rightarrow (M(\tilde{A}) \leq M(\tilde{B}))$, where $\mathcal{F}(CS)$ is the fuzzy power set of CS .

Integral over the membership function

One common measure for fuzzy sets is the integral over the set's membership function, which simply computes the volume below the membership curve:

$$M(\tilde{A}) = \int_{CS} \mu_{\tilde{A}}(x) dx$$

Using the Lebesgue integral

This is equivalent to the Lebesgue integral over the size of the fuzzy set's α -cuts, which we denote by $V(\tilde{A}^\alpha)$:

$$M(\tilde{A}) = \int_0^1 V(\tilde{A}^\alpha) d\alpha \tag{4.8}$$

The size of a fuzzified cuboid

In order to derive the size of an overall fuzzified cuboid, we can therefore integrate over its α -cuts:

Proposition 4.4 (Size of a Fuzzified Cuboid)

The size of a fuzzified cuboid \tilde{C} can be computed as follows:

$$M(\tilde{C}) = \frac{\mu_0}{c^n \prod_{d \in D} w_{\delta(d)} \sqrt{w_d}} \sum_{i=0}^n \left(\sum_{\substack{\{d_1, \dots, d_i\} \\ \subseteq D}} \left(\prod_{\substack{d \in \\ D \setminus \{d_1, \dots, d_i\}}} a_d \right) \cdot \prod_{\substack{\delta \in \\ \Delta_{\{d_1, \dots, d_i\}}} \left(n_\delta! \cdot \frac{\pi^{\frac{n_\delta}{2}}}{\Gamma(\frac{n_\delta}{2} + 1)} \right) \right)$$

Proof. See Appendix C.1. □

The size of a concept

It is trivial to see that any concept \tilde{S} can be viewed as a union of fuzzified cuboids \tilde{C}_i . Moreover, the cuboids C_i of S intersect in P and use the same parameters μ_0 , c , and W . This means that any intersection of \tilde{C}_i and \tilde{C}_j can be computed by intersecting the respective crisp cuboids and keeping all other parameters fixed – the result of this intersection is then again a fuzzified cuboid. In order to compute the overall measure for \tilde{S} , one can therefore combine the measure of its fuzzified cuboids by using the inclusion-exclusion formula (cf. [68]), which generalizes the observation that $|A \cup B| = |A| + |B| - |A \cap B|$ from two to m sets:

Definition 4.2 (Size of a Concept)

Let $\tilde{S} = \langle S, \mu_0, c, W \rangle$ be a concept. Its size $M(\tilde{S})$ can be computed as follows:

$$M(\tilde{S}) = \sum_{l=1}^m \left((-1)^{l+1} \cdot \sum_{\substack{\{i_1, \dots, i_l\} \\ \subseteq \{1, \dots, m\}}} M \left(\bigcap_{i \in \{i_1, \dots, i_l\}} \tilde{C}_i \right) \right)$$

The outer sum in Definition 4.2 iterates over the number of cuboids under consideration (with m being the total number of cuboids in S) and the inner sum iterates over all sets of exactly l cuboids. When computing $M(\tilde{S})$ according to Definition 4.2, we thus rely on the formula from Proposition 4.4 for computing the size of various fuzzified cuboids.

Interpretation

Please note that $M(\tilde{S})$ is always computed only on Δ_S , i.e., the set of domains on which \tilde{S} is defined. This constraint is introduced in order to ensure that $M(\tilde{S})$ is always finite: Let us assume that we included an additional domain $\delta' \in \Delta \setminus \Delta_S$. Since all cuboids of S are not defined on the dimensions $d' \in \delta'$, the coordinates of their support points p^- and p^+ will be $-\infty$ and $+\infty$, respectively, on all of these dimensions d' . One can easily see that this causes $M(\tilde{S})$ to become infinitely large. In order to avoid this, we restrict the computation of $M(\tilde{S})$ to Δ_S .

Avoiding infinitely large results

4.1.3 Implementation and Example

Although the formula for $M(\tilde{C})$ derived in Proposition 4.4 is quite complex, it can be implemented via a set of nested loops. Instead of numerically approximating the integral over the membership function we can therefore compute it analytically, which is considerably faster. Also the inclusion-exclusion formula from Definition 4.2 can be easily implemented via a set of nested loops.

Implementation

The overall runtime of the size computation grows unfortunately exponentially with respect to both the number of cuboids and the number of dimensions in the conceptual space due to the respective inner sums from Proposition 4.4 and Definition 4.2. In order to obtain some concrete runtime estimates, we have measured the runtime of the size operation for 1,000 randomly created concepts. We have investigated the influence of both the number n of dimensions in the conceptual space and the number m of cuboids per concept separately. Moreover, we have also computed a numerical approximation of the integral over a given concept's membership function, this time using, however, only 100 examples in order to limit the overall computation time. All computations were run on a laptop with an Intel Core i5-6440HQ CPU (2.60 GHz quad core) processor and 8 GB main memory.

Runtime experiments

Let us first discuss how the number n of dimensions influences the runtime of the size operation. As we can see in Table 4.1 and Figure 4.4, there is a clear exponential trend: Already for a space with 16 dimensions, computing the size of a concept with a single cuboid takes on average around 1.6 seconds. This is clearly too slow for any large-scale computations. Future work should therefore focus on finding either more efficient exact algorithms or approximations for computing the size of a concept. Nevertheless, the numbers from Table 4.1 and Figure 4.4 clearly show that our closed formula for a concept's size is a significant improvement over a more naive approach of approximating the integral over the given concept's membership function numerically. We did not report runtime measurements of the

Runtime and the number of dimensions

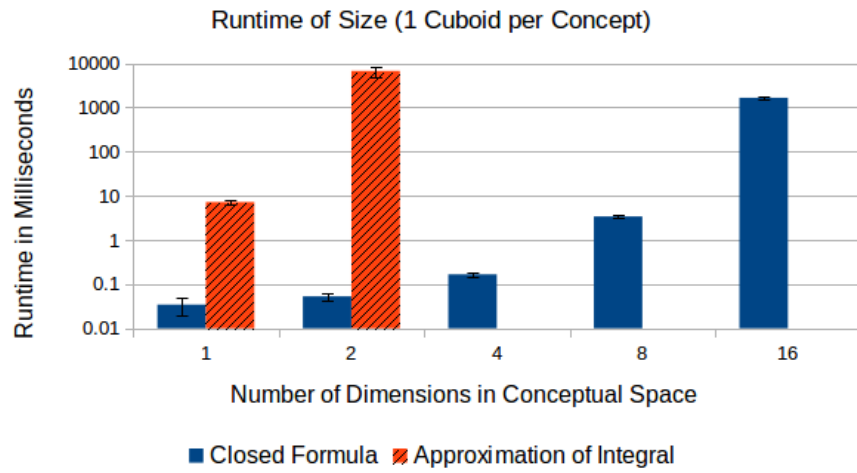


Figure 4.4: Average runtime of the size operation for single-cuboid concepts as a function of the number of dimensions in the conceptual space.

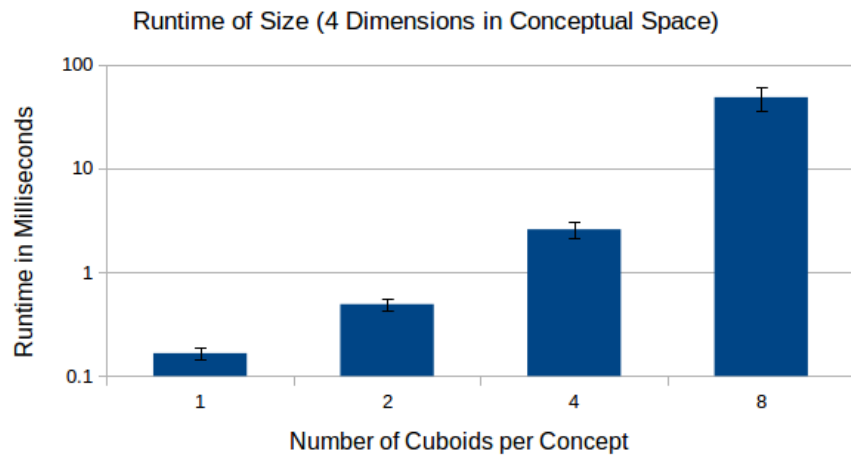


Figure 4.5: Average runtime of the size operation as a function of the number of cuboids per concept in a four-dimensional conceptual space.

Number n of Dimensions	Runtime in ms	Runtime of numerical approximation in ms
1	0.0343	7.2350
2	0.0525	6755.0964
4	0.1660	–
8	3.4105	–
16	1663.3628	–

Table 4.1: Average runtime of the size operation (averaged across 1,000 examples) in comparison to runtime of a numerical approximation (averaged across 100 examples) for randomly generated single-cuboid concepts.

Number m of Cuboids	Runtime in ms
1	0.1660
2	0.4920
4	2.5864
8	48.2449

Table 4.2: Average runtime of the size operation averaged across 1,000 randomly created concepts in a four-dimensional space.

numerical approximation for more than two dimensions in Table 4.1 as these computations quickly became computationally infeasible.

Let us now consider the influence of the number m of cuboids per concept in Table 4.2 (illustrated in Figure 4.5). Again, we can observe a clearly superlinear trend which seems, however, to be weaker than the one observed for the number of dimensions. If we compute the size of a concept consisting of eight cuboids in a four-dimensional space, we need on average 48 milliseconds, which is still reasonably fast. It thus seems to be more important to use a low-dimensional conceptual space than to approximate conceptual regions with a small number of cuboids. This is in line with the effects observed for the intersection operation in Section 3.1.3, where the number of cuboids also had a weaker impact on runtime than the number of dimensions.

Runtime and the number of cuboids

In order to illustrate the size computation, we will now compute the sizes of all fruit concepts from Chapter 2 (see Table 4.3). We illustrate the underlying computations for the examples of LEMON and APPLE.

Fruit space example

We first compute $M(\tilde{S}_{\text{LEMON}})$, because the core of this concept only involves a single cuboid. This allows us to directly use the formula from Proposition 4.4:

The size of the LEMON concept

$$M(\tilde{S}_{\text{LEMON}}) = M(\tilde{C}_{\text{LEMON}})$$

Concept	Δ_S	p^-	p^+	μ_0	c	W		
						$w_{\delta_{\text{COLOR}}}$	$w_{\delta_{\text{SHAPE}}}$	$w_{\delta_{\text{TASTE}}}$
PEAR	Δ	(0.50, 0.40, 0.35)	(0.70, 0.60, 0.45)	1.0	24.0	0.50	1.25	1.25
ORANGE	Δ	(0.80, 0.90, 0.60)	(0.90, 1.00, 0.70)	1.0	30.0	1.00	1.00	1.00
LEMON	Δ	(0.70, 0.45, 0.00)	(0.80, 0.55, 0.10)	1.0	40.0	0.50	0.50	2.00
GRANNY SMITH	Δ	(0.55, 0.70, 0.35)	(0.60, 0.80, 0.45)	1.0	50.0	1.00	1.00	1.00
APPLE	Δ	(0.50, 0.65, 0.35)	(0.80, 0.80, 0.50)	1.0	20.0	0.50	1.50	1.00
		(0.65, 0.65, 0.40)	(0.85, 0.80, 0.55)					
		(0.70, 0.65, 0.45)	(1.00, 0.80, 0.60)					
BANANA	Δ	(0.50, 0.10, 0.35)	(0.75, 0.30, 0.55)	1.0	20.0	0.75	1.50	0.75
		(0.70, 0.10, 0.50)	(0.80, 0.30, 0.70)					
		(0.75, 0.10, 0.50)	(0.85, 0.30, 1.00)					

Table 4.3: Definition of several fruit concepts for the size example.

$$\begin{aligned}
&= \frac{\mu_0^{(\text{LEMON})}}{(c^{(\text{LEMON})})^3 \prod_{d \in D} w_{\delta(d)} \sqrt{w_d}} \sum_{i=0}^3 \left(\sum_{\substack{\{d_1, \dots, d_i\} \\ \subseteq D}} \left(\prod_{\substack{d \in \\ D \setminus \{d_1, \dots, d_i\}}} a_d \right) \right) \\
&\quad \cdot \prod_{\substack{\delta \in \\ \Delta_{\{d_1, \dots, d_i\}}} \left(n_{\delta}! \cdot \frac{\pi^{\frac{n_{\delta}}{2}}}{\Gamma(\frac{n_{\delta}}{2} + 1)} \right) \\
&= \frac{1.00}{40.00^3 \cdot 0.50 \cdot 0.50 \cdot 2.00} \cdot (a_{\text{HUE}} a_{\text{ROUND}} a_{\text{SWEET}} \\
&\quad + 2 \cdot (a_{\text{HUE}} a_{\text{ROUND}} + a_{\text{HUE}} a_{\text{SWEET}} + a_{\text{ROUND}} a_{\text{SWEET}}) \\
&\quad + 4 \cdot (a_{\text{HUE}} + a_{\text{ROUND}} + a_{\text{SWEET}}) + 8) \\
&= \frac{1}{32,000} \cdot (2 \cdot 2 \cdot 8 + 2 \cdot (2 \cdot 2 + 2 \cdot 8 + 2 \cdot 8) \\
&\quad + 4 \cdot (2 + 2 + 8) + 8) \\
&= \frac{1}{32,000} \cdot (32 + 72 + 48 + 8) = \frac{160}{32,000} = \frac{1}{200} = 0.0050
\end{aligned}$$

The size of the APPLE
concept

Without comparing this result to another number, we cannot say whether this means that LEMON is a rather small or a rather large concept. Doing analogous computations for each of the APPLE concept's cuboids (following Proposition 4.4) and combining them with the inclusion-exclusion formula (according to Definition 4.2), we get the following results:

$$\begin{aligned}
M(\tilde{C}_{\text{APPLE},1}) &\approx 0.0271 \\
M(\tilde{C}_{\text{APPLE},2}) &\approx 0.0217 \\
M(\tilde{C}_{\text{APPLE},3}) &\approx 0.0271 \\
M(\tilde{C}_{\text{APPLE},1} \cap \tilde{C}_{\text{apple},2}) &\approx 0.0152 \\
M(\tilde{C}_{\text{APPLE},1} \cap \tilde{C}_{\text{apple},3}) &\approx 0.0098 \\
M(\tilde{C}_{\text{APPLE},2} \cap \tilde{C}_{\text{apple},3}) &\approx 0.0152
\end{aligned}$$

Concept	Size
PEAR	0.0163
ORANGE	0.0046
LEMON	0.0050
GRANNY SMITH	0.0018
APPLE	0.0455
BANANA	0.0690

Table 4.4: Sizes of all fruit concepts.

$$\begin{aligned}
M(\tilde{C}_{\text{APPLE},1} \cap \tilde{C}_{\text{apple},2} \cap \tilde{C}_{\text{apple},3}) &\approx 0.0098 \\
M(\tilde{S}_{\text{APPLE}}) &= M(\tilde{C}_{\text{APPLE},1}) + M(\tilde{C}_{\text{APPLE},2}) + M(\tilde{C}_{\text{APPLE},3}) \\
&\quad - M(\tilde{C}_{\text{APPLE},1} \cap \tilde{C}_{\text{APPLE},2}) \\
&\quad - M(\tilde{C}_{\text{APPLE},1} \cap \tilde{C}_{\text{APPLE},3}) \\
&\quad - M(\tilde{C}_{\text{APPLE},2} \cap \tilde{C}_{\text{APPLE},3}) \\
&\quad + M(\tilde{C}_{\text{APPLE},1} \cap \tilde{C}_{\text{APPLE},2} \cap \tilde{C}_{\text{APPLE},3}) \\
&\approx 0.0455
\end{aligned}$$

As one would have expected, the `APPLE` concept is much larger than the `LEMON` concept. This is caused by both the larger size of the crisp core and by the smaller value of c . We can interpret this as `LEMON` being more narrow (and therefore more specific) than the wider and more general `APPLE` concept. In our implementation, these sizes can be computed as follows:

```

>>> lemon.size()
0.0050000000000000002
>>> apple.size()
0.0455

```

Table 4.4 shows all fruit concepts from Section 2.3.4 and their corresponding sizes. One can also see that the size of `LEMON` and `ORANGE` differs although their cores have an identical size. This is caused by the different weights and the different values of c , which causes the α -cuts of `LEMON` to be tendentially larger than the α -cuts of `ORANGE`.

4.2 SUBSETHOOD

In order to represent knowledge about a hierarchy of concepts, one needs to be able to determine whether one concept is a subset of another concept. For instance, the fact that $\tilde{S}_{\text{GRANNY SMITH}} \subseteq \tilde{S}_{\text{APPLE}}$ indicates that `GRANNY SMITH` is a subordinate concept (i.e., a specialization) of `APPLE`.³

³ One could also say that the fuzzified cuboids \tilde{C}_i are sub-concepts of \tilde{S} , because $\tilde{C}_i \subseteq \tilde{S}$, cf. similar remarks in Section 2.3.2 for cuboids and cores.

Interpretation

Sizes of other fruit concepts

Subsethood for conceptual hierarchies

In Section 4.2.1, we provide both a binary and a soft definition of subsethood for concepts. Afterwards, we illustrate these definitions with our fruit space example in Section 4.2.2.

4.2.1 Definition

Subsethood for crisp sets

Let us first consider the definition of subsethood for crisp sets:

Definition 4.3 (Subsethood for Crisp Sets)

Let S_1 and S_2 be two crisp sets in a conceptual space CS . We say that S_1 is a subset of S_2 ($S_1 \subseteq S_2$) if and only if $\forall x \in CS : (x \in S_1 \Rightarrow x \in S_2)$.

As one can see, subsethood is a binary property: Either a set S_1 is a subset of another set S_2 or it is not. We can relatively easily check whether a given core S_1 is a subset of another core S_2 by comparing their underlying cuboids: Every cuboid $C_i^{(1)}$ of S_1 must be contained in S_2 , which implies that also its support points $p^{(i)+}$ and $p^{(i)-}$ lie inside S_2 .

Subsethood for fuzzy sets

Since we represent concepts as fuzzy sets, we now consider the classical definition of subsethood for fuzzy sets:⁴

Definition 4.4 (Subsethood for Fuzzy Sets)

Let \tilde{S}_1 and \tilde{S}_2 be two fuzzy sets in a conceptual space CS . We say that \tilde{S}_1 is a subset of \tilde{S}_2 ($\tilde{S}_1 \subseteq \tilde{S}_2$) if and only if $\forall x \in CS : \mu_{\tilde{S}_1}(x) \leq \mu_{\tilde{S}_2}(x)$. This is equivalent to requiring subsethood according to Definition 4.3 for all α -cuts \tilde{S}_1^α and \tilde{S}_2^α .

Conditions for subsethood

Explicitly evaluating the right hand side of this expression is, however, infeasible in practice, if the conceptual space uses real-valued dimensions, since then there are infinitely many $x \in CS$. We will now present a set of necessary and jointly sufficient conditions for subsethood:

Proposition 4.5 (Conditions for Crisp Subsethood)

Let $\tilde{S}_1 = \langle S_1, \mu_0^{(1)}, c^{(1)}, W^{(1)} \rangle$ and $\tilde{S}_2 = \langle S_2, \mu_0^{(2)}, c^{(2)}, W^{(2)} \rangle$ be two concepts. Then, $\tilde{S}_1 \subseteq \tilde{S}_2$ if and only if all of the following conditions are fulfilled:

1. $\mu_0^{(1)} \leq \mu_0^{(2)}$
2. $S_1 \subseteq \tilde{S}_2^{\mu_0^{(1)}}$
3. $\Delta_{S_2} \subseteq \Delta_{S_1}$
4. $\forall d \in D_{S_2} : c^{(1)} \cdot w_{\delta(d)}^{(1)} \cdot \sqrt{w_d^{(1)}} \geq c^{(2)} \cdot w_{\delta(d)}^{(2)} \cdot \sqrt{w_d^{(2)}}$

Proof. See Appendix C.2. □

Advantages and limitations

The conditions of Proposition 4.5 can be easily checked, allowing us to derive conclusions about conceptual hierarchies. However, this notion of subsethood is still binary: Either $\tilde{S}_1 \subseteq \tilde{S}_2$ or $\tilde{S}_1 \not\subseteq \tilde{S}_2$. For making more fine-granular distinctions, we need a soft notion of subsethood.

⁴ Please note that this definition has already been used for some of the previous lemmata and propositions.

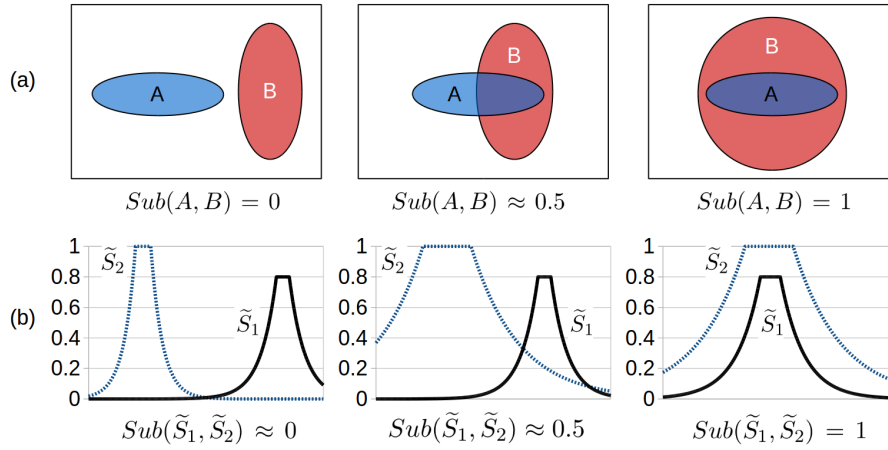


Figure 4.6: Illustrating the degree of subsethood with crisp sets in a two-dimensional space (a) and fuzzy concepts in a one-dimensional space (b).

There has already been some considerable amount of work on defining degrees of subsethood for fuzzy sets (see e.g., [74, 105, 150, 444]). However, many of the definitions made in the literature require that the underlying space is discrete. They are not applicable in our case, because we assume a continuous space. The following definition [226] works also in a continuous space and is conceptually quite straightforward:

Definition 4.5 (Degree of Subsethood for Fuzzy Sets)

Let \tilde{S}_1 and \tilde{S}_2 be two fuzzy sets in a conceptual space CS. Their degree of subsethood $Sub(\tilde{S}_1, \tilde{S}_2)$ can be computed as follows, using a measure M (cf. Definition 4.1) and the standard fuzzy intersection (cf. Definition 3.3):

$$Sub(\tilde{S}_1, \tilde{S}_2) = \frac{M(\tilde{S}_1 \cap \tilde{S}_2)}{M(\tilde{S}_1)}$$

One can interpret Definition 4.5 intuitively as the percentage of \tilde{S}_1 that is also in \tilde{S}_2 . This notion of subsethood is illustrated in Figure 4.6 for both crisp sets and concepts. It has the following properties:

1. $Sub(\tilde{S}_1, \tilde{S}_2) \in [0, 1]$
2. $Sub(\tilde{S}_1, \tilde{S}_2) = 0 \Leftrightarrow \tilde{S}_1 \cap \tilde{S}_2 = \emptyset$
3. $Sub(\tilde{S}_1, \tilde{S}_2) = 1 \Leftrightarrow \tilde{S}_1 \cap \tilde{S}_2 = \tilde{S}_1 \Leftrightarrow \tilde{S}_1 \subseteq \tilde{S}_2$

In Section 3.1.2, we have already defined an intersection operation (Definition 3.2), and in Section 4.1.2, we have defined a measure for concepts (Definition 4.2). We can use them to adapt Definition 4.5:

Definition 4.6 (Modified Degree of Subsethood for Concepts)

Let \tilde{S}_1 and \tilde{S}_2 be two concepts. Their degree of subsethood $Sub(\tilde{S}_1, \tilde{S}_2)$ is defined as follows:

$$Sub(\tilde{S}_1, \tilde{S}_2) = \frac{M(I(\tilde{S}_1, \tilde{S}_2))}{M(\tilde{S}_1)}$$

Degree of subsethood

Properties of the degree of subsethood

Degree of subsethood for concepts

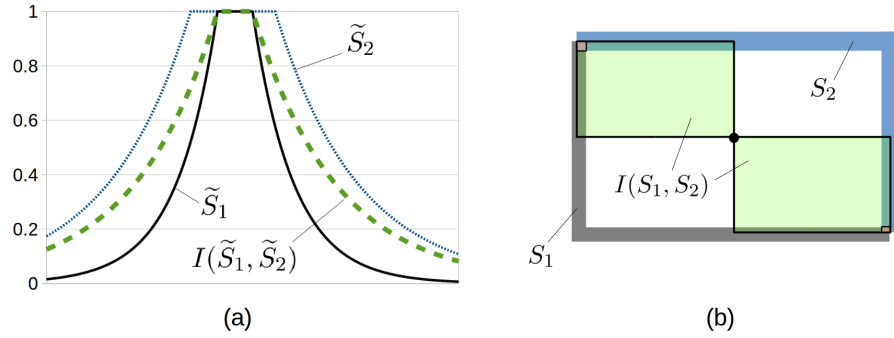


Figure 4.7: Two problematic cases for Definition 4.6. (a) Perfect subsethood with different sensitivity parameters c results in $Sub(\tilde{S}_1, \tilde{S}_2) > 1$. (b) Overextension in the modified intersection causes $Sub(\tilde{S}_1, \tilde{S}_2) > 1$.

Considering only common domains

Definition 4.6 gives us a more fine-grained way to talk about subsethood than Proposition 4.5. If \tilde{S}_1 and \tilde{S}_2 are not defined on the same domains, then we first project them onto their shared subset of domains before computing their degree of subsethood. This is done to ensure that the size of their intersection can be meaningfully compared to the size of the first concept.

Problem based on the modified intersection

When computing the intersection of two concepts with different sensitivity parameters $c^{(1)}, c^{(2)}$ and different weights $W^{(1)}, W^{(2)}$, one needs to define new parameters c' and W' for the resulting concept. Please recall from Definition 3.2 that the new sensitivity parameter c' is set to the minimum of $c^{(1)}$ and $c^{(2)}$ and that the new set of weights W' are obtained through a linear interpolation between $W^{(1)}$ and $W^{(2)}$. Now if $c^{(1)} > c^{(2)}$, then $c' = \min(c^{(1)}, c^{(2)}) = c^{(2)} < c^{(1)}$. If furthermore $W^{(1)} = W^{(2)}$, $\mu_0^{(1)} = \mu_0^{(2)}$, and $S_1 \subseteq S_2$, then it is easy to see that $\tilde{S}' = I(\tilde{S}_1, \tilde{S}_2) = \langle S_1, \mu_0^{(1)}, c^{(2)}, W^{(1)} \rangle$. Since $c' = c^{(2)} < c^{(1)}$ and all other parameters are identical, $M(\tilde{S}') > M(\tilde{S}_1)$. In this case, we get $Sub(\tilde{S}_1, \tilde{S}_2) > 1$. This is illustrated for two one-dimensional concepts in Figure 4.7a. However, $Sub(\tilde{S}_1, \tilde{S}_2) > 1$ violates the first property of a fuzzy degree of subsethood, namely, $Sub(\tilde{S}_1, \tilde{S}_2) \in [0, 1]$. However, this property is necessary, if we would like to interpret $Sub(\tilde{S}_1, \tilde{S}_2)$ as a fuzzy degree of truth.

Our proposed solution

We propose to prevent this undesired behavior by using the same values of c and W for computing both $M(I(\tilde{S}_1, \tilde{S}_2))$ and $M(\tilde{S}_1)$. More specifically, we propose to use $c^{(2)}$ and $W^{(2)}$. One can interpret this approach as follows: The salience weights W influence the distance metric by setting a certain context, indicating the relevance of the different domains and dimensions. Using the weights from $W^{(2)}$ means that when checking whether \tilde{S}_1 is a subset of \tilde{S}_2 , the context is set by \tilde{S}_2 . For instance, when judging whether $\tilde{S}_{\text{TOMATO}}$ is a subset of $\tilde{S}_{\text{VEGETABLE}}$, we focus our attention on the features that are crucial to the definition of the VEGETABLE concept. Using $c^{(2)}$ can be interpreted as another effect of this

context setting: The sensitivity parameter c determines the fuzziness of the concept and using $c^{(2)}$ translates to assuming that we treat all concepts in the current context to be as imprecise as \tilde{S}_2 .

A drawback of using the same c and W for the computation of $M(I(\tilde{S}_1, \tilde{S}_2))$ and $M(\tilde{S}_1)$, is that now the third property of the fuzzy degree of subsethood, namely that $Sub(\tilde{S}_1, \tilde{S}_2) = 1$ is equivalent to $\tilde{S}_1 \subseteq \tilde{S}_2$, does no longer hold. While $\tilde{S}_1 \subseteq \tilde{S}_2$ still implies $Sub(\tilde{S}_1, \tilde{S}_2) = 1$, we can easily find counterexamples for the other direction: Let $\tilde{S} = \langle S, \mu_0, c, W \rangle$ be a concept and define $\tilde{S}' = \langle S, \mu_0, c', W \rangle$ with $c' < c$. One can easily see that with our proposed solution $Sub(\tilde{S}', \tilde{S}) = 1$, but that $\tilde{S}' \not\subseteq \tilde{S}$. In practical applications, however, this is not necessarily a problem: Once we have established that $Sub(\tilde{S}_1, \tilde{S}_2) = 1$, we can simply check the conditions from Proposition 4.5 to decide whether $\tilde{S}_1 \subseteq \tilde{S}_2$.

Disadvantage of our proposed solution

Unfortunately, the fact that our intersection operation uses overextension as a repair mechanism prevents us from ensuring that $Sub(\tilde{S}_1, \tilde{S}_2) \in [0, 1]$ in all cases, even when using identical values for c and W . A pathological example for this can be seen in Figure 4.7b. There seems to be no general way of resolving this issue which does not come with serious drawbacks: One could ensure that $I(\tilde{S}_1, \tilde{S}_2) \subseteq \tilde{S}_1$ by using underextension instead of overextension in defining the intersection. However, as already mentioned in Section 3.1.1, this may force us to make an arbitrary choice about which cuboids to remove. Moreover, if the raw intersection result needs to be approximated by a cuboid (see Sections 3.1.2 and 3.1.3), this typically also results in an overextension. If the intersection operation is expected to return a cuboid-based result, this approximation step cannot be easily removed. Relaxing the requirement that the intersection operation must return a single valid concept would be another way of avoiding overextensions and problematic cases like in Figure 4.7b. However, this would be a strong modification of our work so far.

Another problem based on overextensions

The only way of completely ensuring that the degree of subsethood is always confined to the interval $[0, 1]$ is to use the standard definition for the intersection of fuzzy sets from Definition 3.3, namely $\mu_{\tilde{S}_1 \cap \tilde{S}_2}(x) = \min(\mu_{\tilde{S}_1}(x), \mu_{\tilde{S}_2}(x))$, instead of our modified intersection operator. One can argue that for computing the degree of subsethood, the intersection is only an intermediate result and does not need to be a valid concept. However, if the intersection result is not a valid concept, then our closed formula for computing M is not applicable. Therefore, one would need to compute the size of the intersection by using numerical optimization techniques. As we have seen in Section 4.1.3, the runtime of such a numerical optimization is, however, prohibitively large even for simple examples. For now, we assume that the example from Figure 4.7b is pathological and does not occur too often in practice. Nevertheless, more research is needed in order to find a way for overcoming such problematic cases.

Using the standard intersection is not an option

	2 Dimensions	4 Dimensions	8 Dimensions
2 Cuboids	13	0	0
4 Cuboids	28	0.	-5

Table 4.5: Absolute number of cases with $Sub(\tilde{S}_1, \tilde{S}_2) > 1$ for 10,000 randomly generated pairs of concepts for different numbers of cuboids per concept and different dimensionalities of the conceptual space.

4.2.2 Implementation and Example

Implementation

Both the crisp and the fuzzy definition of subsethood can be implemented easily: In order to decide whether $\tilde{S}_1 \subseteq \tilde{S}_2$, one simply needs to check the conditions from Proposition 4.5. In order to compute $Sub(\tilde{S}_1, \tilde{S}_2)$ according to Definition 4.6, one can make use of the implementations of $I(\tilde{S}_1, \tilde{S}_2)$ and $M(\tilde{S})$, which were described in Sections 3.1.3 4.1.3, respectively. It is obvious that the runtime for computing subsethood directly depends on the runtime of these two operations. We therefore did not conduct dedicated runtime experiments for the degree of subsethood.

Investigating the overextension-based problem

In the Section 4.2.1, we have seen that in some cases, we may get the unintuitive result of $Sub(\tilde{S}_1, \tilde{S}_2) > 1$ (cf. Figure 4.7b). In order to confirm our suspicion that this happens only in pathological cases, we have computed the degree of subsethood for 10,000 randomly generated pairs of concepts, varying both the number of cuboids per concept and the number of dimensions in the conceptual space. The absolute number of cases with $Sub(\tilde{S}_1, \tilde{S}_2) > 1$ among these 10,000 examples is reported in Table 4.5.

Interpreting the results

As we can see, these cases are indeed quite rare in practice with relative frequencies of less than 0.3%. Since the concepts we expect to use in practical applications are not randomly generated but based on actual observations, one may speculate that they are even unlikelier to have a structure as depicted in Figure 4.7b. Thus, while $Sub(\tilde{S}_1, \tilde{S}_2) > 1$ is not satisfactory from a theoretical point of view, it does not seem to be a serious limitation in practice.

Fruit space example

Let us now again look at our fruit space from Section 2.3.4 in order to illustrate our definition of subsethood. Table 4.6 lists selected fruit concepts (namely, APPLE, GRANNY SMITH, and APPLE-PEAR, which was defined as the intersection of APPLE and PEAR in Section 3.1.3).

Crisp subsethood of GRANNY SMITH and APPLE

One can easily see that $\tilde{S}_{\text{GRANNY SMITH}} \subseteq \tilde{S}_{\text{APPLE}}$, because all conditions from Proposition 4.5 are fulfilled. Note that $\tilde{S}_{\text{APPLE-PEAR}} \not\subseteq \tilde{S}_{\text{APPLE}}$, because for d_{ROUND} we find that:

$$\begin{aligned} c^{(\text{APPLE-PEAR})} \cdot w_{\delta_{\text{SHAPE}}}^{(\text{APPLE-PEAR})} \cdot \sqrt{w_{d_{\text{ROUND}}}^{(\text{APPLE-PEAR})}} \\ = 5 \cdot 1.375 \cdot 1.0 < 5 \cdot 1.5 \cdot 1.0 \end{aligned}$$

⁵ Please note that this combination was not included in our experiments due to the prohibitively large runtime of the subsethood operation in this scenario.

Concept	Δ_S	p^-	p^+	μ_0	c	W		
						$w_{\delta_{\text{COLOR}}}$	$w_{\delta_{\text{SHAPE}}}$	$w_{\delta_{\text{TASTE}}}$
GRANNY SMITH	Δ	(0.550, 0.700, 0.350)	(0.600, 0.800, 0.450)	1.0	50.0	1.00	1.00	1.00
APPLE	Δ	(0.50, 0.65, 0.35)	(0.80, 0.80, 0.50)	1.0	20.0	0.50	1.50	1.00
		(0.65, 0.65, 0.40)	(0.85, 0.80, 0.55)					
		(0.70, 0.65, 0.45)	(1.00, 0.80, 0.60)					
APPLE-PEAR	Δ	(0.500, 0.625, 0.350)	(0.700, 0.625, 0.450)	0.4724	20.0	0.500	1.375	1.125

Table 4.6: Three concepts used to illustrate our subsethood computations.

$$= c^{(\text{APPLE})} \cdot w_{\delta_{\text{SHAPE}}}^{(\text{APPLE})} \cdot \sqrt{w_{\delta_{\text{ROUND}}}^{(\text{APPLE})}}$$

It is also obvious that there is no crisp subsethood relation between GRANNY SMITH and APPLE-PEAR. Our implementation confirms this:

Crisp subsethood of GRANNY SMITH and APPLE-PEAR

```
apple.crisp_subset_of(granny_smith)
False
apple.crisp_subset_of(apple-pear)
False
granny_smith.crisp_subset_of(apple)
True
granny_smith.crisp_subset_of(apple-pear)
False
apple-pear.crisp_subset_of(apple)
False
apple-pear.crisp_subset_of(granny_smith)
False
```

Now let us look at the values returned when computing the fuzzy degree of subsethood:

Considering soft subsethood

```
apple.subset_of(granny_smith)
0.07635041551246535
apple.subset_of(apple-pear)
0.05762893130047096
granny_smith.subset_of(apple)
1.0
granny_smith.subset_of(apple-pear)
0.06859652328421048
apple-pear.subset_of(apple)
1.0
apple-pear.subset_of(granny_smith)
0.12933550211333572
```

As one can see, $Sub(\tilde{S}_{\text{GRANNY SMITH}}, \tilde{S}_{\text{APPLE}}) = 1$, as we would expect. Moreover, $Sub(\tilde{S}_{\text{APPLE-PEAR}}, \tilde{S}_{\text{APPLE}}) = 1$ because differences in c and W are ignored when computing the fuzzy degree of subsethood. All other degrees of subsethood are rather small, but still positive, since the intersection between any two concepts is never completely empty.

Interpretation

4.3 IMPLICATION

Implication for grounding rule-based systems

Implications play a fundamental role in rule-based systems and all approaches that use formal logics (i.e., the symbolic layer, cf. Section 1.2.3) for knowledge representation. It is therefore desirable to define an implication operation on concepts, such that one is able to express facts like $\text{APPLE} \Rightarrow \text{RED}$ within our formalization.

Relation to Rickard's co-occurrence statistics

Such an implication operation can also be used to obtain the co-occurrence statistics used in Rickard's formalization of conceptual spaces [329]: Recall from Section 2.5.2 that Rickard represents a concept like APPLE as a matrix containing the co-occurrence statistics of its properties (e.g., RED and SWEET). These co-occurrence statistics can be interpreted as conditional probabilities $\mathbb{P}(S_{\text{RED}}|S_{\text{SWEET}})$ and $\mathbb{P}(S_{\text{SWEET}}|S_{\text{RED}})$ and are in Rickard's account estimated based on frequencies in a given set of examples. If we assume that APPLE , RED , and SWEET are represented as fuzzy conceptual regions, then these conditional probabilities can be grounded in a fuzzy degree of implication: $\mathbb{P}(S_{\text{RED}}|S_{\text{SWEET}})$ can be computed by measuring to which degree the implication $\text{APPLE} \wedge \text{RED} \Rightarrow \text{SWEET}$ is considered to be true. Thus, the co-occurrence statistics used by Rickard can be extracted from our representation, making our approach at least as expressive as his.

In Section 4.3.1, we argue that a soft degree of implication can be obtained by reusing the degree of subethood. In Section 4.3.2, we then illustrate this intuition using our fruit space example.

4.3.1 Definition

Classical definitions of implication for fuzzy sets

In the fuzzy set literature [280], implications are defined by generalizing the truth table of the crisp implication to fuzzy truth values: A fuzzy implication is a function $\text{Impl} : [0, 1] \times [0, 1] \rightarrow [0, 1]$ such that $\text{Impl}(0, x) = 1$ (a false antecedent implies anything), $\text{Impl}(1, y) = y$ (a true antecedent can be ignored), and $\text{Impl}(x, 1) = 1$ (a true consequent is implied by anything). Depending on the additional requirements for this function (e.g., monotonicity or continuity), one can find different fuzzy implications, for instance the Gödel implication with $\text{Impl}(x, y) = 1$ for $x \leq y$ and $\text{Impl}(x, y) = y$ otherwise.

Shortcomings in our context

This notion of implication is, however, defined on pairs of truth values and not on pairs of fuzzy sets. It is thus similar to fuzzy conjunction and disjunction operators. When applying a fuzzy implication to a pair of fuzzy sets, the result is therefore again a fuzzy set, which describes the local validity of the implication for each point in the underlying space. However, if we want to check whether being an APPLE implies being RED , and if both APPLE and RED are represented as fuzzy sets, we intuitively expect a single number that indicates the degree to which the implication between the two sets can be considered to hold. This is a more coarse-grained view than the one provided by standard fuzzy logic, but in our opinion better suited for supporting abstract reasoning.

Concept	Δ_S	p^-	p^+	μ_0	c	W		
						$w_{\delta_{\text{COLOR}}}$	$w_{\delta_{\text{SHAPE}}}$	$w_{\delta_{\text{TASTE}}}$
LEMON	Δ	(0.70, 0.45, 0.00)	(0.80, 0.55, 0.10)	1.0	40.0	0.50	0.50	2.00
APPLE	Δ	(0.50, 0.65, 0.35) (0.65, 0.65, 0.40) (0.70, 0.65, 0.45)	(0.80, 0.80, 0.50) (0.85, 0.80, 0.55) (1.00, 0.80, 0.60)	1.0	20.0	0.50	1.50	1.00
RED	$\{\delta_{\text{COLOR}}\}$	(0.90, $-\infty$, $-\infty$)	(1.00, $+\infty$, $+\infty$)	1.0	40.0	1.00	–	–
GREEN	$\{\delta_{\text{COLOR}}\}$	(0.45, $-\infty$, $-\infty$)	(0.55, $+\infty$, $+\infty$)	1.0	40.0	1.00	–	–
NONSWEET	$\{\delta_{\text{TASTE}}\}$	($-\infty$, $-\infty$, 0.00)	($+\infty$, $+\infty$, 0.20)	1.0	14.0	–	–	1.00

Table 4.7: Definitions of several fruit concepts and properties used to illustrate the implication operation.

In our context, the degree of implication between two geometrically represented concepts corresponds to their degree of subsethood: If \tilde{S}_1 is a subset of \tilde{S}_2 , then for any point $x \in CS$, being contained in \tilde{S}_1 implies also being contained in \tilde{S}_2 . Whether belonging to the APPLE concept also implies belonging to the RED concept can be quantified by checking whether the APPLE region is a subset of the RED region within the COLOR domain. We therefore propose to simply re-use our soft notion of subsethood from Definition 4.6 in order to quantify the degree of implication between two concepts:

Definition 4.7 (Degree of Implications for Concepts)

Let \tilde{S}_1 and \tilde{S}_2 be two concepts. Their degree of implication $Impl(\tilde{S}_1, \tilde{S}_2)$ is defined as their degree of subsethood $Sub(\tilde{S}_1, \tilde{S}_2)$:

$$Impl(\tilde{S}_1, \tilde{S}_2) = Sub(\tilde{S}_1, \tilde{S}_2) = \frac{M(I(\tilde{S}_1, \tilde{S}_2))}{M(\tilde{S}_1)}$$

4.3.2 Implementation and Example

Implementing the implication operation is trivial as we simply need to call the method for computing the degree of subsethood. The runtime of the implication thus equals the runtime of the degree of subsethood, which is directly based on the runtimes of size and implication (cf. Section 4.2.2). Please note that of course also the problematic cases of $Sub(\tilde{S}_1, \tilde{S}_2) > 1$ observed in Section 4.2.1 carry over to our implementation of the implication operation. However, as we have seen in Section 4.2.2, such pathological cases appear quite infrequently in practice.

Let us now look at the implication relations between several concepts and properties from our fruit space (see Table 4.7 and Figure 4.8). Let us first consider the relation between LEMON and NONSWEET, which can be evaluated in our code as follows:

```
lemon.implies(non_sweet)
1.0
non_sweet.implies(lemon)
0.6000000000000001
```

Re-using the degree of subsethood

Implementation

*Fruit space example:
LEMON and
NONSWEET*

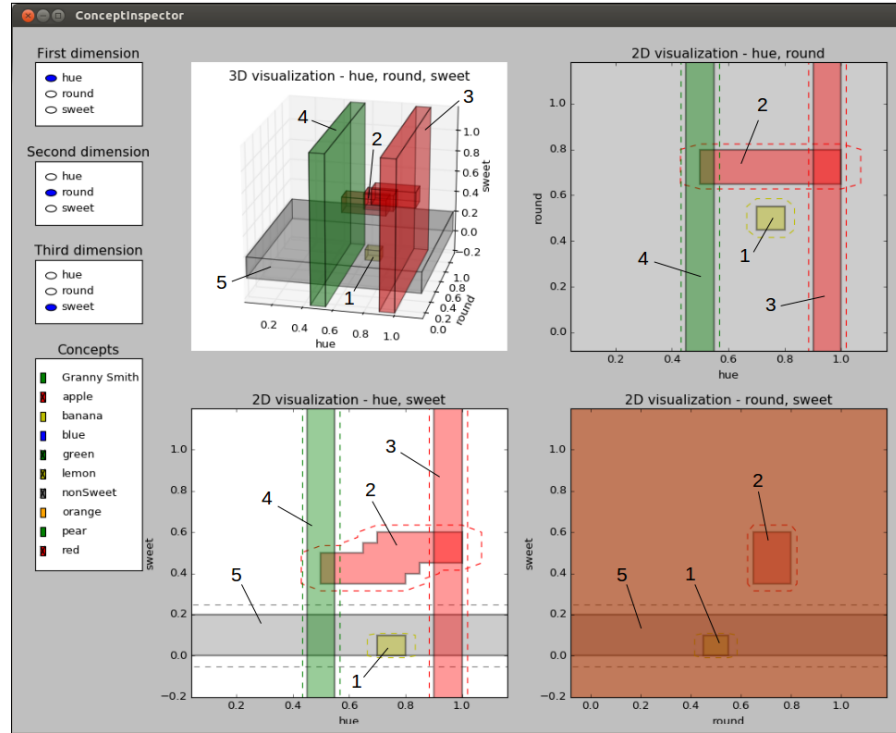


Figure 4.8: Screenshot of the ConceptInspector tool, illustrating of the concepts and properties for the implication example: LEMON (1), APPLE (2), RED (3), GREEN (4), NONSWEET (5).

Interpretation

As one can see, $Impl(\tilde{S}_{LEMON}, \tilde{S}_{NONSWEET}) = 1$ which means that all lemons are not sweet. However, $Impl(\tilde{S}_{NONSWEET}, \tilde{S}_{LEMON}) \approx 0.60$ which indicates that not all non-sweet things are lemons.

*Orthogonal domains:
RED and NONSWEET*

If the two concepts under consideration do not share any common domains, they also do not have any implication relation:

```
red.implies(non_sweet)
0.0
non_sweet.implies(red)
0.0
```

*Implication works in
a domain-wise way:
APPLE and RED*

Let us now highlight some general properties of our implication operation by considering APPLE and RED:

```
apple.implies(red)
0.2727272727272726
red.implies(apple)
1.0
```

Interpretation

The fact that $Impl(\tilde{S}_{APPLE}, \tilde{S}_{RED}) \approx 0.2727$ indicates that there is only a partial overlap between the regions describing APPLE and RED, respectively – in other words, there are some apples which are not red. Although $Impl(\tilde{S}_{RED}, \tilde{S}_{APPLE}) = 1$, this does not mean that all red things are apples. For example, also for a TOMATO concept, we might get $Impl(\tilde{S}_{RED}, \tilde{S}_{TOMATO}) = 1$. If APPLE and TOMATO are two different concepts (occupying for example different regions in the TASTE domain), then it clearly does not make any sense to say that all red things are both apples

and tomatoes. It is important to keep in mind that the implication is computed as the degree of subthood *only on the COLOR domain*. As stated in Section 4.2.1, we do not look at the full APPLE concept when computing the degree of implication/subthood, but only at its projection onto the COLOR domain. This projection can be interpreted as a property describing the typical COLOR of apples. Therefore, the correct interpretation of the implication is that all colors that are classified as RED are also classified as APPLE colors (and similarly, as TOMATO colors). An artificial agent could conclude that all RED things are apples only if APPLE is the only concept \tilde{S} for which $Impl(\tilde{S}_{RED}, \tilde{S})$ reaches a high value. Then, apples would be the only RED things known to the agent. However, if we also have another concept like TOMATO, then this conclusion cannot be drawn. Overall, a high value of $Impl(\tilde{S}_1, \tilde{S}_2)$ where $\Delta_{S_1} \subseteq \Delta_{S_2}$ does not necessarily tell us that the implication $x \in \tilde{S}_1 \Rightarrow x \in \tilde{S}_2$ must be true for all $x \in CS$, but only that it *can* be true for some $x \in CS$.

Overall, we can thus say that high values of $Impl(\tilde{S}_1, \tilde{S}_2)$ are by themselves not very diagnostic if $\Delta_{S_1} \subsetneq \Delta_{S_2}$ (e.g., $Impl(\tilde{S}_{RED}, \tilde{S}_{APPLE})$), because the implication is evaluated on Δ_{S_1} only, not taking account variations in other domains in $\Delta_{S_2} \setminus \Delta_{S_1}$. High values of $Impl(\tilde{S}_1, \tilde{S}_2)$ are, however, quite informative if $\Delta_{S_2} \subseteq \Delta_{S_1}$ (e.g., $Impl(\tilde{S}_{APPLE}, \tilde{S}_{RED})$), because the implication is evaluated on all domains of Δ_{S_2} .

On a related note, we would like to mention that while the fact that $Impl(\tilde{S}_{RED}, \tilde{S}_{APPLE}) = 1$ is by itself not very informative, a combination of multiple such implications on different domains can be interesting: If also $Impl(\tilde{S}_{ROUND}, \tilde{S}_{APPLE}) = 1$ and $Impl(\tilde{S}_{SWEET}, \tilde{S}_{APPLE}) = 1$, then we have information from different domains that are orthogonal to each other. If there is no other concept \tilde{S} other than APPLE which has also high values for all three implications, then we can be quite confident in saying that "a thing that is RED, ROUND, and SWEET at the same time is usually an APPLE".

Similarly, also $Impl(I(\tilde{S}_{RED}, \tilde{S}_{ROUND}, \tilde{S}_{SWEET}), \tilde{S}_{APPLE}) = 1$ is quite informative, because the intersection of the three properties is defined on the full fruit space. Also the value of $Impl(I(\tilde{S}_{APPLE}, \tilde{S}_{NONSWEET}), \tilde{S}_{GREEN})$ can give valuable information when being compared to $Impl(APPLE, \tilde{S}_{GREEN})$:

```
(apple.intersect_with(non_sweet)).implies(green)
0.2857142857142858
apple.implies(green)
0.18181818181818182
```

While an APPLE is in general not necessarily GREEN, an APPLE which is NONSWEET has a much higher expectation of being GREEN. Essentially, this captures the correlation between the TASTE and the COLOR domain in the APPLE concept and can be linked to our argument about co-occurrence statistics from above.

4.4 CONCEPT SIMILARITY

The similarity of concepts can be used as a basis for commonsense

General interpretation of implications

Orthogonal information from multiple domains

Implications of intersections

Relation to correlations

Similarity for commonsense reasoning

reasoning in various contexts. Firstly, similarity plays an important role in recommendation systems: If we know that Alice enjoyed the "Lord of the Rings" trilogy, then we can expect that she will also like the "The Hobbit" movies due to the similarity of the respective movie trilogies [123]. Moreover, the similarity of concepts can give information about their potential usage: From a perceptual point of view, pencils and crayons are quite similar to each other. If an autonomous agent now observes that pencils are used for writing or drawing, it can generalize this usage to crayons. Finally, conceptual similarity can also be used for finding appropriate substitutes. Let us assume that a household robot tries to make an apple pie. However, there are currently no apples in the kitchen, only pears and oranges. If the robot is able to extract from its conceptual space that APPLE is more similar to PEAR than to ORANGE, it can infer that it should use the pears as a substitute, not the oranges.

Similarity of points

In Section 2.1.1, we have introduced Gärdenfors' definition of similarity between points in a conceptual space:

$$Sim(x, y) = e^{-c \cdot d(x, y)}$$

Towards similarity of fuzzy concepts

This definition can, however, not easily be generalized from points to fuzzy conceptual regions and has been criticized in the literature. In Section 4.4.1, we discuss Tversky's criticism of distance-based accounts of similarity [412] and how this criticism can be addressed in the conceptual spaces framework. Afterwards, we propose two definitions of conceptual similarity in Section 4.4.2, which are then illustrated in Section 4.4.3 by using our fruit space example.

4.4.1 Similarity as Inverse Distance

Criticism of similarity spaces

Before we formalize the similarity of conceptual regions, we would like to discuss to what extent psychological similarity between individual observations can be represented in a spatial way. In his very influential article [412], Tversky has argued that despite being widely used, similarity spaces are not always adequate for representing psychological similarity. He especially criticizes that dissimilarity is identified with a distance metric in the similarity space. A distance metric needs to fulfill three properties (minimality, symmetry, and the triangle inequality, cf. the proof of Lemma 2.1) and Tversky argues that all three of them are not necessarily fulfilled for dissimilarities:

Violations of minimality in recognition experiments

The *minimality* criterion asserts that $\forall x, y \in CS : d(x, y) \geq d(x, x) = 0$. Tversky considers recognition experiments as a counter-example for this assertion: In recognition experiments, participants are shown pairs of stimuli for a very short amount of time and are then asked to report as fast as possible whether the two stimuli are identical or different. One can represent dissimilarity by the probability of a stimulus pair to be judged as different. Tversky argues, that in this setting, we can find two different stimuli $x \neq y$ for which empirical data suggests that $d(x, x) \neq d(y, y)$. However, minimality would require that

$d(x, x) = d(y, y) = 0$. Moreover, one can sometimes observe cases with $d(x, y) < d(x, x)$, which also violates minimality.

Symmetry requires that $\forall x, y \in CS : d(x, y) = d(y, x)$. Here, Tversky points out that a stimulus x being similar to a stimulus y essentially means that " x is like y ". This is a directional statement, which might not be reversible. He argues that in natural language, usually the more salient stimulus is chosen as y (such as in "an ELLIPSE is like a CIRCLE" rather than "a CIRCLE is like an ELLIPSE"). Moreover, statements of the type " x is like y " often involve metaphorical mappings. Tversky uses the following example to support this point of view: If we say "a MAN is like a TREE", we imply that a MAN has roots (e.g., in his home region). However, if we say "a TREE is like a MAN", we imply that a TREE has a life history. Tversky concludes that symmetry is therefore often violated for dissimilarity judgments.

Violations of symmetry due to directionality of similarity

The *triangle inequality* is fulfilled if $\forall x, y, z \in CS : d(x, y) + d(y, z) \geq d(x, z)$. Again, Tversky is able to give a counter-example: CUBA and SOVIET RUSSIA are judged to be similar because of their political system. Moreover, CUBA and JAMAICA are considered to be similar because of their geographical location. The triangle inequality would imply that SOVIET RUSSIA and CUBA are also relatively similar to each other. However, they are typically considered to be quite dissimilar, because they share no common features. Therefore, also the triangle inequality is easily violated.

Violations of the triangle inequality based on shared features

In addition to the three properties mentioned above, Tversky introduces the *diagnosticity effect* as another empirical observation which cannot be easily explained if dissimilarity is interpreted as spatial distance. This effect takes place if the similarity judgment between two stimuli is influenced by other stimuli in the current context. For instance, Tversky describes the results of a study where participants were asked to select which country is most similar to AUSTRIA. When presented with the options SWEDEN, POLAND, and HUNGARY, participants mostly selected SWEDEN. However, when presented with SWEDEN, NORWAY, and HUNGARY, participants picked HUNGARY. At the time of the experiment, POLAND and HUNGARY were both communist countries, while SWEDEN and AUSTRIA were capitalist democracies. In the first case, it thus seems that the political and economic system was the most crucial aspect for the similarity judgment. In the second case, however, geographical information became more pertinent with SWEDEN and NORWAY being Scandinavian countries, and AUSTRIA and HUNGARY being located in central Europe. If dissimilarity is represented by distance in a similarity space, the distances between AUSTRIA and SWEDEN and between AUSTRIA and HUNGARY should, however, not be affected by the context given through POLAND and NORWAY, respectively.

The diagnosticity effect: similarity is context-dependent

Overall, Tversky [412] concludes that spatial representations of similarity are often not adequate. In order to solve the problems outlined above, he proposes a *feature matching* technique as an alternative to spatial representations of similarity. In his proposal, each stimulus x is represented as a set of binary features X (which can be

Tversky's feature matching technique

present or absent). The similarity between two stimuli x and y is defined as a function of their common and different features, namely $Sim(x, y) = f(X \cap Y, X \setminus Y, Y \setminus X)$. Tversky showed that this account of similarity is capable of solving the aforementioned issues.

Feature matching
and conceptual
spaces

This proposed representation of a stimulus by a set of features is easily applicable to concepts. For instance, an APPLE can be described by being RED, ROUND, and SWEET. In our opinion, Tversky's feature matching approach can, however, not be used for properties such as RED, ROUND, and SWEET, since they cannot be easily expressed in lower-level terms. Moreover, the feature matching approach requires the identification of all relevant features for all stimuli which in itself is not a trivial process.⁶ One can thus argue that Tversky's approach suffers from the symbol grounding problem [190] (cf. Section 1.2.3). In contrast to this, the conceptual spaces framework explicitly focuses on the grounding of properties in perception by using a spatial representation of similarities. Nevertheless, Tversky's general criticism of similarity spaces needs to be addressed.

Accounting for
Tversky's criticism
with contrast vectors

Recently, Sileno et al. [369] have provided an account for Tversky's criticism within the theory of conceptual spaces. They do so by using the contrast vectors introduced by Dessalles [124] (cf. Sections 2.4.1 and 3.6.2). In this approach, metaphors such as "PETER is like a LION" are interpreted as double contrast: Sileno et al. compute the contrast vector between PETER and the PERSON prototype as well as the contrast vector between LION and the ANIMAL prototype. The metaphor is valid if both contrast vectors are similar to each other, i.e., point into similar directions. In the given example, one would assume that both contrast vectors have large entries with respect to the STRENGTH dimension, i.e., both individuals are identified as stronger than the average individual. Sileno et al. follow Tversky's argumentation by furthermore requiring that the difference to the prototype should be more distinctive for the second stimulus (i.e., LION). Please note that this alternative definition of conceptual similarity is no longer based on *distances* in the similarity space, but on the similarity of *directions* inside this space. Sileno et al. show how contrast vectors can solve the problems posed by Tversky for symmetry, the triangle inequality, and the diagnosticity effect:

Violations of
symmetry based on
distinctive
characteristics

The similarity judgments based on contrast vectors do not necessarily fulfill *symmetry*. For instance, "TEL AVIV is like NEW YORK" activates the distinctive characteristics of NEW YORK (i.e., the ways in which it differs from the average city), while "NEW YORK is like TEL AVIV" activates the distinctive characteristics of TEL AVIV, thus leading potentially to different results. This effect is based on the assumption that the difference of the second item to its prototype is always more pronounced than the respective difference of the first item to its prototype.

Violations of the
triangle inequality
based on distinctive
characteristics

The violation of the *triangle inequality* can be explained by the type of

⁶ This can be related to the *frame problem* [284] in AI, which describes the difficulty of limiting the set of beliefs about the environment that need to be changed after executing an action – in short, how to know which things can be expected to stay the same unless explicitly defined otherwise.

contrast vector being used in the similarity judgments. When comparing the contrast vectors between CUBA and the prototypical country on the one hand, and between SOVIET RUSSIA and the prototypical country on the other hand, we expect both of these vectors to have large entries with respect to the dimension representing the POLITICAL AFFILIATION. Moreover, the respective contrast vectors for CUBA and JAMAICA have large commonalities with respect to the GEOGRAPHICAL LOCATION. However, the contrast vectors for SOVIET RUSSIA and JAMAICA do not have considerable commonalities with respect to any of the dimensions.

The *diagnosticity effect* can be explained by constructing a prototype of the set of countries under consideration. In the first case (where AUSTRIA is deemed to be more similar to SWEDEN than to HUNGARY in the context of POLAND), AUSTRIA differs from this prototype mostly in the direction of POLITICAL AFFILIATION (as does SWEDEN). In the second case (where POLAND is replaced by NORWAY), the prototype changes, and the difference with respect to the GEOGRAPHICAL LOCATION becomes more prominent. Thus, by manipulating the set of alternatives, one implicitly also manipulates the prototype and thus the contrast vectors, leading potentially to different similarity judgments.

Finally, Sileno et al. argue that violations of *minimality* can be explained as follows: If a given stimulus x is close to the prototype p of the overall group, then also any other stimulus y which is also close to the group prototype p might be a satisfactory choice in recognition experiments – in both cases, the contrast vector between x and p , and between y and p , respectively, is very small and not very expressive. If stimulus identity is computed by comparing contrast vectors, then very small contrast vectors may simply lie below the detection threshold for differences, and may thus be regarded as identical.

Sileno et al. conclude that the criticism by Tversky can be circumvented if contrast is used as basis of similarity rather than geometrical distance. They furthermore point out that Tversky's feature matching approach requires to estimate additional model parameters which determine how the common features and the distinctive features are weighted against each other. Their contrast-based approach on the other hand is free of such parameter fitting issues.

Our formalization of conceptual spaces does not use contrast vectors to define the similarity of points. Since we define the similarity of points as an exponentially decaying function of their distance (in agreement with Gärdenfors' original proposal [179] and based on findings by Shepard [368]), we shall thus also comment on the issues pointed out by Tversky [412]. In our opinion, three of the four effects can be explained through the the salience weights which express the relative importance and prominence of the respective domain or dimension in a given context (cf. [179, Section 4.3.3]). Please recall that in our formalization we compute the distance within a conceptual space as follows:

Definition 2.7 (Combined Metric)

Let CS be a conceptual space based on dimensions $d \in D$, which are partitioned

Explaining the diagnosticity effect based on contrast to the group average

Violations of minimality due to small contrast vectors

Contrast vectors as a solution to Tversky's criticism

Accounting for Tversky's criticism without contrast vectors

into domains $D \supseteq \delta \in \Delta$. Let W_Δ be the set of positive domain weights w_δ for all $\delta \in \Delta$ with $\sum_{\delta \in \Delta} w_\delta = |\Delta|$. Let moreover W_δ be the set of positive domain weights w_d for all $d \in \delta$ with $\sum_{d \in \delta} w_d = 1$. Let furthermore $x, y \in CS$. Their distance according to the combined metric is defined as follows, where $W = \langle W_\Delta, \{W_\delta\}_{\delta \in \Delta} \rangle$:

$$d_C^\Delta(x, y, W) = \sum_{\delta \in \Delta} w_\delta \cdot d_E^\delta(x, y, W_\delta) = \sum_{\delta \in \Delta} \left(w_\delta \cdot \sqrt{\sum_{d \in \delta} w_d \cdot |x_d - y_d|^2} \right)$$

Saliency weights for representing context

By manipulating the saliency weights w_δ and w_d , we can control the influence of the dimension-wise distances $|x_d - y_d|$ on the overall distance between x and y . These saliency weights thus represent the current context in which the similarity judgment is made. Manipulating these saliency weights can reproduce the effects observed by Tversky:

Violations of symmetry by using different saliency weights

When discussing the violation of *symmetry*, both Tversky [412] and Sileno et al. [369] assume that the second stimulus in a comparison (such as "TEL AVIV is like NEW YORK") sets the context of the comparison. We can therefore use the saliency weights associated with the second stimulus when computing the similarity of two stimuli. One can easily see that different saliency weights lead to different semantic distances, and thus to a violation of symmetry. This can also be related to our arguments from Section 4.2.1 in the context of the subsethood operation, where we also advocated to use the saliency weights and sensitivity parameter of the second concept.

Violations of the triangle inequality based on domain weights

Also the example with respect to the *triangle inequality* can be solved through saliency weights. Let us for now assume that our conceptual space consists only of two domains reflecting the political system and the geographical location, respectively. CUBA and SOVIET RUSSIA have a small distance within the political domain, but a large distance with respect to geography. Their similarity becomes large if we put a large domain weight on the political domain and a relatively small weight on the geographic domain. CUBA and JAMAICA on the other hand have a large distance in the political domain, but a small distance in the geographic domain. By shifting the saliency weights towards the latter, we can again achieve a high similarity value. Finally, SOVIET RUSSIA and JAMAICA have relatively large distances with respect to both domains. No matter how we choose the domain weights (assuming that they have to be normalized), we cannot achieve a small distance and thus high similarity. Therefore, SOVIET RUSSIA and JAMAICA are judged to be not very similar to each other.

Explaining also the diagnosticity effect with domain weights

Finally, the account of the *diagnosticity effect* given by Sileno et al. [369] can be adapted to our proposal: If the context is set by AUSTRIA, SWEDEN, HUNGARY, and POLAND, then the political domain receives a large saliency weight as it is very discriminative in this context. This results in SWEDEN being the most similar country to AUSTRIA based on their shared political system. On the other hand, if the context includes AUSTRIA, SWEDEN, HUNGARY, and NORWAY, the geographic domain becomes more prominent and receives a higher saliency weight

than the political domain. In this case, HUNGARY will be considered most similar to AUSTRIA based on their geographic proximity.

While we can therefore model the three effects discussed above by using salience weights, it is still unclear how to exactly determine the salience weights of the current context and we do not offer any general way of doing so in our formalization. Here, the approach taken by Sileno et al. [369, 370] (cf. Section 2.3.3) seems to be a promising avenue of research: In principle, one could determine the salience weights by computing the entries of the contrast vector and normalizing them appropriately.

The violation of the *minimality* criterion cannot be addressed with salience weights. In addition to the argumentation by Sileno et al. [369] we can give another possible explanation: Perception corresponds in the case of the conceptual spaces framework to mapping objects from the real world onto points in the similarity space. In general, we can assume that perception is a noisy process, which means that the coordinates in the conceptual space are subject to small random translations. If an object is observed for a longer period of time, the multiple noisy observations can be aggregated into a stable estimate (e.g., by taking a moving average of the coordinates). However, in the speeded classification tasks discussed by Tversky, objects are only presented for a very brief amount of time. One could thus argue that the noise cannot be filtered out successfully and that the distances are now computed between noisy estimates rather than true coordinates, leading sometimes to erroneous results, such as a violation of the minimality property.

We think that the arguments made above show that Tversky's criticism can be addressed within the original conceptual spaces framework by manipulating the salience weights (see also [179, Section 4.3.3] for a similar argumentation). Albeit raising important issues, Tversky's criticism [412] does therefore not prevent our formalization from providing a meaningful way of measuring similarity. So far, we have confined our discussion to dissimilarity as a distance between *points* in the conceptual space. In the following, we attempt to generalize the notion of similarity from points to fuzzy conceptual regions.

4.4.2 Definition

In this section, we propose two different definitions for the similarity of fuzzy concepts. Whenever the two concepts are defined on two different sets of domains $\Delta_1 \neq \Delta_2$, we first project them onto their set of common domains $\Delta' = \Delta_1 \cap \Delta_2$ before computing their similarity value. For example, the conceptual similarity of BASEBALL and APPLE should not be zero, because both have a similar SHAPE and SIZE. However, APPLE is also defined on the TASTE domain, but BASEBALL presumably not. Thus, when judging the similarity of the concepts BASEBALL and APPLE, we consider only their set of common domains. Since properties from different domains such as SWEET and ROUND do not share any commonalities (i.e.,

How to obtain salience weights

Violations of minimality due to noisy observations

Summary

Similarity only considers common domains

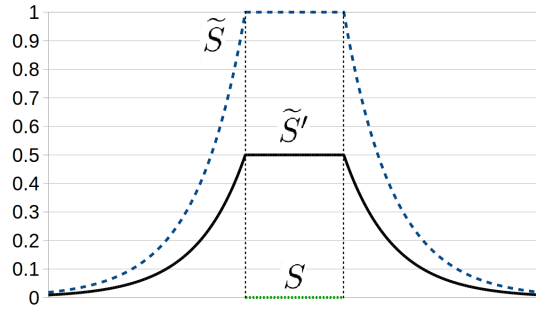


Figure 4.9: Two fuzzy sets \tilde{S} and \tilde{S}' with identical cores, weights, and sensitivity parameter, for which similarity functions based on the distance of cores violate the first criterion.

$\Delta' = \emptyset$), their similarity is defined to be zero.

Criteria for similarity functions

We expect that any similarity function $Sim(\tilde{S}_1, \tilde{S}_2) \in [0, 1]$ fulfills a number of mathematically formulated semantic constraints:

Definition 4.8 (Similarity Function)

A function $Sim(\tilde{S}_1, \tilde{S}_2) \in [0, 1]$ is called a similarity function, if it fulfills the following criteria for all concepts \tilde{S}_1, \tilde{S}_2 :

1. $(Sim(\tilde{S}_1, \tilde{S}_2) = 1) \Rightarrow (Sub(\tilde{S}_1, \tilde{S}_2) = 1)$
2. $(\tilde{S}_1 = \tilde{S}_2) \Rightarrow (Sim(\tilde{S}_1, \tilde{S}_2) = 1)$
3. $(\tilde{S}_1 \subseteq \tilde{S}_2) \Rightarrow (Sim(\tilde{S}_1, \tilde{S}_2) \geq Sim(\tilde{S}_2, \tilde{S}_1))$

Perfect similarity implies subethood

The first criterion states that if \tilde{S}_1 is perfectly similar to \tilde{S}_2 , then those two concepts should also stand in a subethood relation. This requires a strong semantic relationship to hold between the two concepts. One could also require that $\tilde{S}_1 = \tilde{S}_2$, but this seems to be too strong as it would prevent a perfect similarity of subordinate concepts (e.g., GRANNY SMITH) to their superordinate concept (e.g., APPLE).

Self-similarity is maximal

The second criterion requires that the similarity of a given concept to itself is always maximal and can be related to the minimality criterion discussed in Section 4.4.1.

Directionality of similarity for subsets

The third criterion finally prevents supersets from having a higher similarity to their subsets than the other way around. It ensures, for instance, that $Sim(\tilde{S}_{GRANNY\ SMITH}, \tilde{S}_{APPLE}) \geq Sim(\tilde{S}_{APPLE}, \tilde{S}_{GRANNY\ SMITH})$.

Similarity based on distances of cores violates the first criterion

If we base the similarity of two concepts \tilde{S}_1 and \tilde{S}_2 on the distance between their cores S_1 and S_2 (e.g., by computing their minimal distance $d_{\min}(S_1, S_2) = \min_{x \in S_1} \min_{y \in S_2} d(x, y)$, their Hausdorff distance $d_H(S_1, S_2) = \max(\sup_{x \in S_1} \inf_{y \in S_2} d(x, y), \sup_{y \in S_2} \inf_{x \in S_1} d(x, y))$, or the distance of their prototypical points), we always violate the first property from Definition 4.8: Consider $\tilde{S} = \langle S, \mu_0, c, W \rangle$ and $\tilde{S}' = \langle S, \mu'_0, c, W \rangle$ with $\mu'_0 < \mu_0$ as illustrated in Figure 4.9. Clearly, $\tilde{S}' \subseteq \tilde{S}$ and therefore $Sub(\tilde{S}', \tilde{S}) = 1$. On the other hand, $Sub(\tilde{S}, \tilde{S}') < 1$. As the cores are

identical, their distance is zero. If we use $Sim(\tilde{S}_1, \tilde{S}_2) = e^{-c \cdot d(S_1, S_2)}$, then $Sim(\tilde{S}, \tilde{S}') = 1$, but $Sub(\tilde{S}, \tilde{S}') < 1$. We therefore exclude these possible definitions from our consideration.

If we define $Sim(\tilde{S}_1, \tilde{S}_2) = \max_{x \in S_1} \mu_{\tilde{S}_2}(x)$, we always violate the second property from Definition 4.8 for $\tilde{S}_1 = \tilde{S}_2$ with $\mu_0 < 1$. We therefore do not consider this possible definition any further.

Similarity as highest membership violates the second criterion

We now propose two definitions that fulfill all of the requirements stated above. Firstly, we can again reuse our definition of subsethood from Section 4.2.1:

Similarity as subsethood

Proposition 4.6 (Sim_S is a Similarity Function)

$Sim_S(\tilde{S}_1, \tilde{S}_2) = Sub(\tilde{S}_1, \tilde{S}_2)$ is a similarity function according to Definition 4.8.

Proof. See Appendix C.3. □

Secondly, we can use the Jaccard index $\frac{|A \cap B|}{|A \cup B|}$, which is a common similarity measure between sets:

Similarity based on the Jaccard index

Proposition 4.7 (Sim_J is a Similarity Function)

$Sim_J(\tilde{S}_1, \tilde{S}_2) = \frac{M(I(\tilde{S}_1, \tilde{S}_2))}{M(U(\tilde{S}_1, \tilde{S}_2))}$ is a similarity function according to Definition 4.8.

Proof. See Appendix C.3. □

Both proposed definitions are similar to each other in the sense that they look at the overall fuzzy sets and not just at their cores. Moreover, they build upon our definitions of intersection, union, and size. The Jaccard index is symmetric, whereas the degree of subsethood is asymmetric. Depending on the application scenario, either one of them can be used. The symmetric nature of the Jaccard index Sim_J might be more convincing from a mathematical perspective. On the other hand, the asymmetric nature of Sim_S matches psychological evidence suggesting that similarity judgments by humans tend to be asymmetric [412] as discussed in Section 4.4.1. In principle, also the Jaccard index could be made asymmetric by always using the salience weights of the second concept instead of the interpolated weights returned by the intersection and union operators (cf. our definition of subsethood in Section 4.2.1). We do, however, not consider such a variant, since it seems somewhat counterintuitive to modify the symmetric Jaccard index in such a way.

Commonalities and differences between the two approaches

4.4.3 Implementation and Example

Both definitions of similarity can be easily implemented based on existing functionality (namely, concept size, intersection, union, and subsethood). It is clear that the runtime for computing similarity depends directly on the runtime of these operations. We therefore did not conduct separate runtime experiments but refer back to the respective

Implementation

Concept	Δ_S	p^-	p^+	μ_0	c	W		
						$w_{\delta_{\text{COLOR}}}$	$w_{\delta_{\text{SHAPE}}}$	$w_{\delta_{\text{TASTE}}}$
PEAR	Δ	(0.50, 0.40, 0.35)	(0.70, 0.60, 0.45)	1.0	24.0	0.50	1.25	1.25
ORANGE	Δ	(0.80, 0.90, 0.60)	(0.90, 1.00, 0.70)	1.0	30.0	1.00	1.00	1.00
LEMON	Δ	(0.70, 0.45, 0.00)	(0.80, 0.55, 0.10)	1.0	40.0	0.50	0.50	2.00
GRANNY SMITH	Δ	(0.55, 0.70, 0.35)	(0.60, 0.80, 0.45)	1.0	50.0	1.00	1.00	1.00
APPLE	Δ	(0.50, 0.65, 0.35)	(0.80, 0.80, 0.50)	1.0	20.0	0.50	1.50	1.00
		(0.65, 0.65, 0.40)	(0.85, 0.80, 0.55)					
		(0.70, 0.65, 0.45)	(1.00, 0.80, 0.60)					
BANANA	Δ	(0.50, 0.10, 0.35)	(0.75, 0.30, 0.55)	1.0	20.0	0.75	1.50	0.75
		(0.70, 0.10, 0.50)	(0.80, 0.30, 0.70)					
		(0.75, 0.10, 0.50)	(0.85, 0.30, 1.00)					

Table 4.8: Definitions of several fruit concepts for our exemplary similarity computations.

	PEAR	ORANGE	LEMON	GRANNY SMITH	APPLE	BANANA
PEAR	1.0000	0.0000	0.0002	0.0096	0.1181	0.0558
ORANGE	0.0001	1.0000	0.0000	0.0004	0.0446	0.0000
LEMON	0.0011	0.0000	1.0000	0.0000	0.0025	0.0048
GRANNY SMITH	0.0613	0.0000	0.0000	1.0000	1.0000	0.0002
APPLE	0.0504	0.0053	0.0002	0.0764	1.0000	0.0012
BANANA	0.0136	0.0000	0.0002	0.0000	0.0007	1.0000

Table 4.9: Similarity values based on the degree of subsethood Sim_S for all fruit concepts, rounded to four decimal places.

results from Sections 3.1.3, 4.1.3, and 4.2.2.

Fruit space example

Let us again use some examples from our fruit space to illustrate the results of applying the two proposed definitions. Table 4.8 and Figure 4.10 show the fruit concepts, and Tables 4.9 and 4.10 contain the respective similarity values for all pairs of concepts, using Sim_S and Sim_J , respectively. In the code, the similarity values can be computed like this:

```
apple.similarity_to(pear, method="subset")
0.05043467196991022
apple.similarity_to(pear, method="Jaccard")
0.0398322124027715
```

Symmetry and self-similarity

Table 4.9 shows the asymmetric nature the subsethood-based similarity Sim_S , while Table 4.10 illustrates that the Jaccard index Sim_J is a symmetric similarity function: For instance, consider the pair of concepts APPLE and GRANNY SMITH:

$$Sim_S(\tilde{S}_{\text{APPLE}}, \tilde{S}_{\text{GRANNY SMITH}}) \approx 0.0764$$

$$Sim_S(\tilde{S}_{\text{GRANNY SMITH}}, \tilde{S}_{\text{APPLE}}) = 1.0000$$

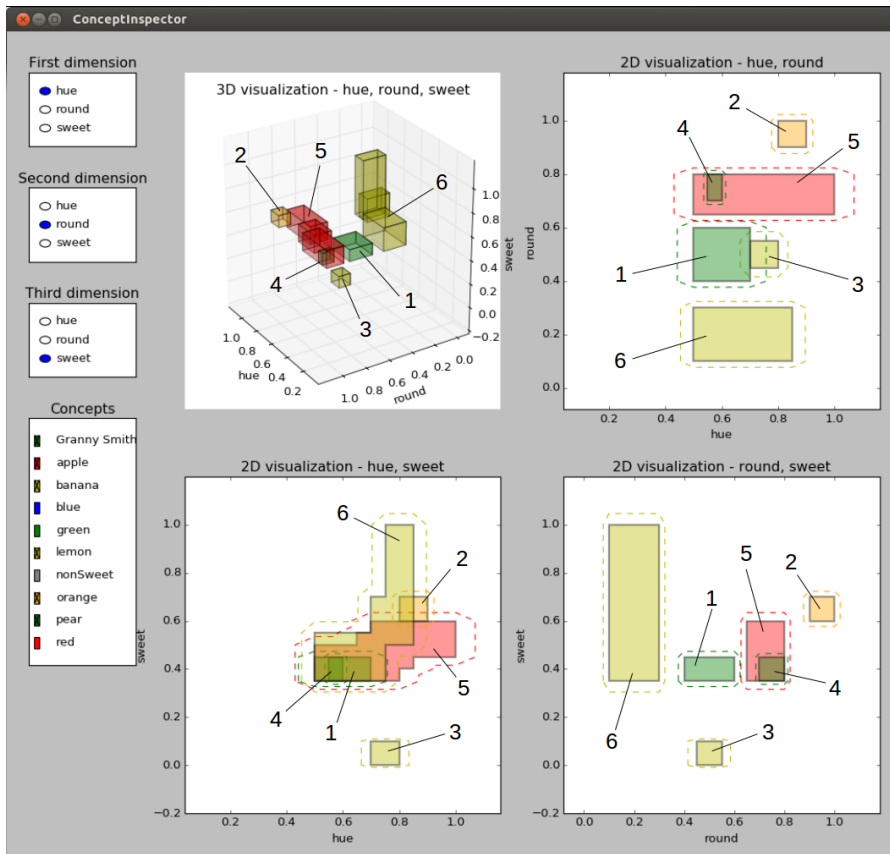


Figure 4.10: Screenshot of the ConceptInspector tool, showing all concepts used in our similarity computations. Concepts are labeled as follows: PEAR (1), ORANGE (2), LEMON (3), GRANNY SMITH (4), APPLE (5), and BANANA (6).

	PEAR	ORANGE	LEMON	GRANNY SMITH	APPLE	BANANA
PEAR	1.0000	0.0000	0.0002	0.0151	0.0398	0.0107
ORANGE	0.0000	1.0000	0.0000	0.0001	0.0069	0.0000
LEMON	0.0002	0.0000	1.0000	0.0000	0.0003	0.0003
GRANNY SMITH	0.0151	0.0001	0.0000	1.0000	0.1537	0.0000
APPLE	0.0398	0.0069	0.0003	0.1537	1.0000	0.0003
BANANA	0.0107	0.0000	0.0003	0.0000	0.0003	1.0000

Table 4.10: Similarity values based on the Jaccard index Sim_J for all fruit concepts, rounded to four decimal places.

$$\text{Sim}_J(\tilde{S}_{\text{APPLE}}, \tilde{S}_{\text{GRANNY SMITH}}) \approx 0.1537$$

$$\text{Sim}_J(\tilde{S}_{\text{GRANNY SMITH}}, \tilde{S}_{\text{APPLE}}) \approx 0.1537$$

Interpretation

Under Sim_S , GRANNY SMITH is perfectly similar to APPLE, but APPLE is not very similar to GRANNY SMITH. This can be explained by the simple fact that in our representation $\tilde{S}_{\text{GRANNY SMITH}} \subseteq \tilde{S}_{\text{APPLE}}$, but $\tilde{S}_{\text{APPLE}} \not\subseteq \tilde{S}_{\text{GRANNY SMITH}}$, which of course strongly affects Sim_S . For both variants of the similarity function, we observe perfect self-similarity, but rather low values for all other similarities. Many similarity values are below 0.00005 and are thus shown as zeros in Tables 4.10 and 4.9.

PEAR is more similar
to APPLE than to
GRANNY SMITH

Let us take a look at a few selected similarity values returned by Sim_S and Sim_J : For both functions, we observe that PEAR is more similar to APPLE than to GRANNY SMITH. This makes intuitive sense when looking at the locations of the respective cores in Figure 4.10 and at their definitions in Table 4.8 – the cores of PEAR and APPLE are simply closer to each other than the cores of PEAR and GRANNY SMITH. Moreover, the GRANNY SMITH concept is quite narrow due to its higher value for the sensitivity parameter c , which also contributes to the lower similarity value.

PEAR is more similar
to APPLE than to
BANANA

We can also observe that PEAR is more similar to APPLE than to BANANA for both Sim_S and Sim_J . Again, by looking at the visualization in Figure 4.10 one can confirm that this seems plausible, because the PEAR core is closer to the core of APPLE than to the core of BANANA, while both APPLE and BANANA have a similar degree of fuzziness.

Is PEAR more similar
to GRANNY SMITH or
to BANANA?

The two similarity functions differ, however, with respect to the ordering of the values for $\text{Sim}(\tilde{S}_{\text{PEAR}}, \tilde{S}_{\text{GRANNY SMITH}})$ and $\text{Sim}(\tilde{S}_{\text{PEAR}}, \tilde{S}_{\text{BANANA}})$: The subsethood-based similarity function Sim_S claims that PEAR is more similar to BANANA than to GRANNY SMITH, while the Jaccard index Sim_J results in the reverse ordering. One can verify with Table 4.8 that the distance between the cores of PEAR and GRANNY SMITH equals the distance between the cores of PEAR and BANANA. In both cases, only differences with respect to the SHAPE domain are relevant. The BANANA concept has a lower value of c than the GRANNY SMITH concept, making it more fuzzy overall. However, it puts a higher emphasis on the SHAPE domain than the GRANNY SMITH concept, making its membership function steeper with respect to δ_{SHAPE} . Based on the given conceptual space, there is no strong reason to prefer any of the ordering over the other, therefore both of them seem to be somewhat plausible.

Asymmetry of Sim_S
for PEAR and APPLE

Furthermore, we can observe a difference with respect to symmetry, namely, that $\text{Sim}_J(\tilde{S}_{\text{PEAR}}, \tilde{S}_{\text{APPLE}}) = \text{Sim}_J(\tilde{S}_{\text{APPLE}}, \tilde{S}_{\text{PEAR}})$, but that $\text{Sim}_S(\tilde{S}_{\text{PEAR}}, \tilde{S}_{\text{APPLE}}) > \text{Sim}_S(\tilde{S}_{\text{APPLE}}, \tilde{S}_{\text{PEAR}})$. This observation can be explained by the fact that PEAR is a subset of APPLE in both the COLOR and the TASTE domain (but not on the SHAPE domain), whereas APPLE is not a subset of PEAR on any of the domains. This naturally influences the subsethood based similarity function Sim_S .

Low similarities for
LEMON and ORANGE

We can furthermore note that for both similarity functions the simi-

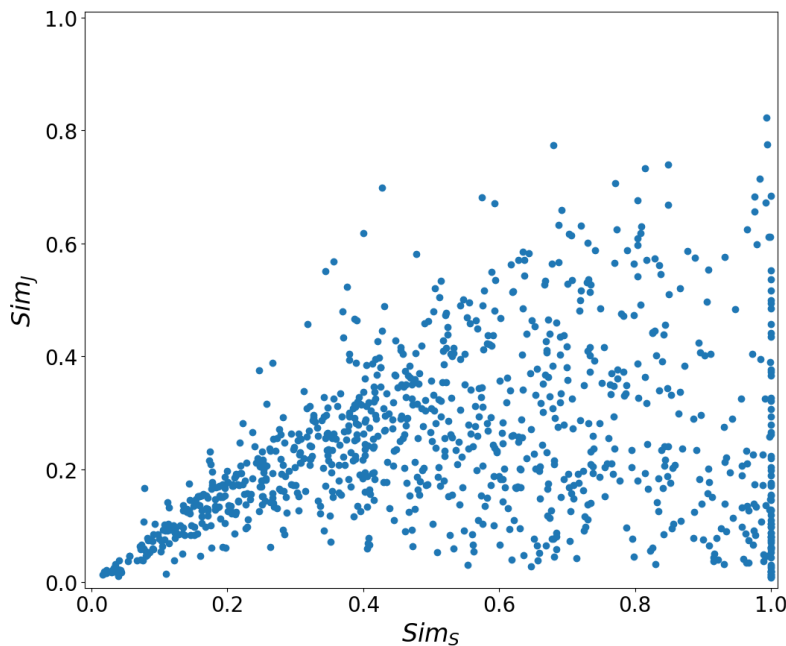


Figure 4.11: Scatter plot of similarity values obtained with Sim_S and Sim_J for 1,000 randomly generated pairs of concepts in a four-dimensional conceptual space.

larity between LEMON and ORANGE (and vice versa) is quite low, which is cognitively not very plausible. However, we would like to refer back to Section 3.2.2 where also the union of LEMON and ORANGE was deemed to be cognitively implausible. There, we argued that this problem is not caused by the union operation, but rather by the way our fruit space is constructed. The same line of argument applies here: By the simplistic construction of our fruit space, LEMON and ORANGE are quite far apart from each other which of course leads to low similarity values. If the fruit space was augmented with additional domains and dimensions that include, for instance, also the REGION where the given fruit is typically grown, its TEXTURE and CONSISTENCY, then LEMON and ORANGE would be closer to each other, leading to higher similarity values.

All other table entries can be interpreted in a similar way. We conclude our discussion here with the remark that we cannot make a general recommendation for any of the two similarity functions based on the results from this toy example, but that a decision for one of the two similarity functions needs to be made based on the concrete application and a more thorough evaluation based on real data.

Choice of similarity function depends on the application

In order to compare the two definitions of conceptual betweenness more thoroughly, we have generated 1,000 pairs of random concepts and have computed their similarity values with respect to both Sim_S

Additional simulations for comparing Sim_S and Sim_J

and Sim_J . We employed a four-dimensional conceptual space and two cuboids per concept. Figure 4.11 shows a scatter plot of our results.

Only limited agreement overall

As we can see, there tends to be only limited agreement among the two definitions. We can observe a tendency of Sim_S producing larger similarity values than Sim_J . This can be explained by the fact that the formulas for Sim_S and Sim_J share the same numerator (namely, the size of the intersection), while their denominator differs: Sim_J uses the size of the union, which can be expected to be larger than the size of the second concept, which is used by Sim_S . However, there are also exceptions to this overall tendency, which are likely to be based on the different types of salience weights used in the computations of the respective sizes (weights of the second concept for Sim_S and interpolated weights for Sim_J).

Distribution of perfect similarity scores

We can furthermore observe a relatively large number of cases with $Sim_S(\tilde{S}_1, \tilde{S}_2) = 1$, while $Sim_J(\tilde{S}_1, \tilde{S}_2)$ never occurs in our simulations. One can easily see that $Sim_J(\tilde{S}_1, \tilde{S}_2) = 1$ can only happen if $S_1 = S_2$ (i.e., the cores are identical) and $\mu_0^{(1)} = \mu_0^{(2)}$ (cf. the proof of Proposition 4.7), which is quite unlikely. $Sim_S(\tilde{S}_1, \tilde{S}_2) = 1$ on the other hand happens whenever $\tilde{S}_1 \subseteq \tilde{S}_2$, which occurs more frequently. Moreover, as already discussed in Section 4.2.1, our choice of always using the salience weights of the second concept when computing the degree of subthood causes even more cases with $Sim_S(\tilde{S}_1, \tilde{S}_2) = 1$.

Correlation analysis

When looking at the correlation between Sim_S and Sim_J , we observe values of Pearson's $r \approx 0.3220$ (which measures linear correlation) and Spearman's $\rho \approx 0.2932$ (which measures monotone correlation). So while the similarities produced by the two definitions seems to be correlated to some degree, they are definitely not interchangeable. Again, a more thorough analysis might reveal which definition is preferable for which practical use cases.

4.5 BETWEENNESS

Betweenness for commonsense reasoning

Conceptual betweenness can be a valuable source for commonsense reasoning [123]: If one concept (e.g., MASTER STUDENT) is conceptually between two other concepts (e.g., BACHELOR STUDENT and PHD STUDENT), then it is expected to share all properties and behaviors that the two other concepts have in common (e.g., typically being found on campus, or having to pay an enrollment fee).

Betweenness is context-independent

As Derrac and Schockaert [123] have argued, betweenness is a qualitative rather than a quantitative notion and is invariant under linear transformations – the actual scaling of the individual dimensions of the conceptual space has therefore no influence on betweenness-based reasoning. As we have seen in Section 4.4.1, the salience weights used for such a rescaling may, however, heavily affect similarity-based approaches. Thus, betweenness is robust to changes in context.

Betweenness for points

In Section 2.1.2, we have already introduced a definition of betweenness based on a given distance metric d :

Definition 2.2 (Betweenness)

Let $x, y, z \in CS$ and d be a metric on CS . The point y is said to lie between x and z (denoted as $B_d(x, y, z)$) if and only if $d(x, y) + d(y, z) = d(x, z)$.

Note that Definition 2.2 is only applicable to *points* in a conceptual space. Moreover, it is defined in a crisp way: A point y either is completely between two other points x and z or it is not. In general, we would, however, also like to talk about a fuzzy *degree* of betweenness for triples of *concepts*.

Generalizing
betweenness

In Section 4.5.1, we first review definitions for conceptual betweenness from the literature. Afterwards, we derive two definitions for the betweenness of fuzzy concepts in Section 4.5.2, which are then illustrated in Section 4.5.3 using again our example fruit space from Section 2.3.4.

4.5.1 Betweenness in the Literature

Schockaert and Prade [352, 353] and Derrac and Schockaert [122, 123] have thoroughly studied conceptual betweenness in conceptual spaces as a basis for commonsense reasoning, and have provided various generalizations of Definition 2.2. In [353], Schockaert and Prade propose to generalize the betweenness relation from points to regions in the following way:

Crisp betweenness of
crisp regions

$$\begin{aligned}\overline{bet}(X, Y, Z) &\Leftrightarrow (\exists y \in Y : \exists x \in X : \exists z \in Z : B(x, y, z)) \\ \underline{bet}(X, Y, Z) &\Leftrightarrow (\forall y \in Y : \exists x \in X : \exists z \in Z : B(x, y, z))\end{aligned}$$

Here, $\overline{bet}(X, Y, Z)$ requires only a single point of Y to lie between X and Z , whereas $\underline{bet}(X, Y, Z)$ requires this for *all* points in Y . Please note that these definitions of betweenness are still crisp.

Interpretation

In [122], Derrac and Schockaert propose different soft notions of betweenness for points:

Soft betweenness of
points

$$\begin{aligned}Btw_1(x, y, z) &= \|\vec{y}\vec{p}\| \\ Btw_2(x, y, z) &= \begin{cases} Btw_1(x, y, z) & \text{if } \cos(\vec{x}\vec{z}, \vec{x}\vec{y}) \geq 0 \\ & \wedge \cos(\vec{z}\vec{x}, \vec{z}\vec{y}) \geq 0 \\ +\infty & \text{otherwise} \end{cases} \\ Btw_3(x, y, z) &= \frac{d(x, z)}{d(x, y) + d(y, z)}\end{aligned}$$

Here, $Btw_1(x, y, z)$ measures the distance of point y to its projection p onto the line through x and z (see Figure 4.12a). Perfect betweenness corresponds to a value of 0 and greater numbers signify a weaker betweenness relation. This is somewhat unintuitive from the perspective of fuzzy logic as adopted in our work, where perfect betweenness can be mapped to a value of 1 and weaker betweenness relations should be represented by values from the interval $[0, 1)$. Moreover, $Btw_1(x, y, z)$

Btw_1 and Btw_2 are
based on a projection

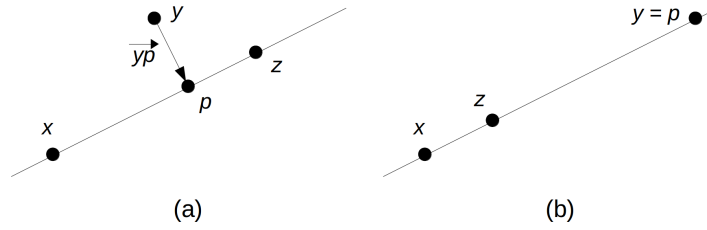


Figure 4.12: (a) Illustration of Btw_1 as proposed by Derrac and Schockaert [122]. (b) Problematic case for Btw_1 as motivation for Btw_2 .

has the disadvantage that a point y can obtain perfect betweenness without actually lying between x and z . This case is illustrated in Figure 4.12b. Here, $Btw_1(x, y, z) = 0$ (indicating perfect betweenness), because y already lies on the line through x and z which means that $y = p$ and $\|\vec{yp}\| = 0$. However, y is clearly not between x and z . The definition of $Btw_2(x, y, z)$ fixes this problem by restricting the betweenness to the line segment between x and z through checking the cosines between the respective vectors.

Btw₁ and Btw₂ assume the Euclidean distance

Please note that both Btw_1 and Btw_2 assume a Euclidean space and are thus only applicable within a single domain, but not across domains: The implicit assumption of a Euclidean space can be understood from the fact that only points y on the straight line segment between x and z are considered to be perfectly between x and z . In Section 2.1.2, we showed that this is the definition of betweenness under the Euclidean metric, whereas all points in the bounding box between x and z would be considered to be between x and z under the Manhattan metric.

Btw₃ as soft generalization of Definition 2.2

The third variant $Btw_3(x, y, z)$ turns the crisp notion of betweenness from Definition 2.2, which requires that $d(x, z) = d(x, y) + d(y, z)$ into a soft measure of betweenness by comparing the relative size of the two terms. Due to the triangle inequality, the denominator cannot be smaller than the numerator and $Btw_3(x, y, z)$ is thus limited to the interval $(0, 1]$ with a value of 1 indicating perfect betweenness. This definition thus corresponds to our expectations from fuzzy logic and fits in nicely with our prior definitions of subsethood, implication, and similarity. Moreover, $Btw_3(x, y, z)$ can be used with any distance metric (including d_M and d_C^Δ) and can thus be easily applied to the complete conceptual space.

Generalizing soft betweenness to crisp regions

All three of these definitions are again based on points. Derrac and Schockaert [122] propose to generalize them from points to crisp regions in the following way:

$$Btw_1^R(X, Y, Z) = \frac{1}{|Y|} \sum_{y \in Y} \min_{x \in X} \min_{z \in Z} Btw_1(x, y, z)$$

$$Btw_2^R(X, Y, Z) = \frac{1}{|Y|} \sum_{y \in Y} \min_{x \in X} \min_{z \in Z} Btw_2(x, y, z)$$

$$Btw_3^R(X, Y, Z) = \frac{1}{|Y|} \sum_{y \in Y} \max_{x \in X} \max_{z \in Z} Btw_3(x, y, z)$$

In all three cases, we compute for each point $y \in Y$ its optimal betweenness value across all possible choices over points $x \in X$ and $z \in Z$. We then aggregate this intermediate result across all $y \in Y$ by using an unweighted average. Please note that all three definitions assume that the conceptual region Y consists only of a finite set of points. If this is not the case, the summation would have to be replaced by an integral.

Interpretation as average of optimal values

In their experiments, Derrac and Schockaert [122] represent each concept as a set of exemplar points, which they obtained with a k-means clustering procedure (to be introduced in Chapter 7). They argue that one can easily construct a region out of these points by computing their convex hull. For efficiency reasons, they, however, only use these exemplars to compute the betweenness relations Btw_1^R , Btw_2^R , and Btw_3^R . Their experimental results indicate that Btw_1 and Btw_2 tend to perform slightly better than Btw_3 in commonsense reasoning tasks.

Experimental results

4.5.2 Definition

Because the concepts in our formalization cannot be described by a finite set of points, the definitions of Derrac and Schockaert [122] from Section 4.5.1 are not directly applicable. We will therefore derive our own definition of betweenness for fuzzy sets. Please note that a concept \tilde{S}_2 can only be located between two other concepts \tilde{S}_1 and \tilde{S}_3 if all of these concepts are defined on the same domains. One can, for instance, not say that `BASEBALL` is conceptually between `APPLE` and `ORANGE`, because it does not have a `TASTE`. We will start with the binary notion of betweenness for points, which will be subsequently generalized in order to derive a soft betweenness measure for fuzzy sets.

We need soft betweenness for fuzzy sets

As already stated above, we define in our formalization that a point y is between two other points x and z based on the combined metric d_C^Δ :

Crisp betweenness for points

Definition 2.2 (Betweenness)

Let $x, y, z \in CS$ and d be a metric on CS . The point y is said to lie between x and z (denoted as $B_d(x, y, z)$) if and only if $d(x, y) + d(y, z) = d(x, z)$.

We generalize $B_d(x, y, z)$ (henceforth referred to as $B(x, y, z)$ for better readability) from points to crisp regions by using *bet* from Schockaert and Prade [353] (cf. Section 4.5.1):

Crisp betweenness for crisp regions

$$B(S_1, S_2, S_3) \Leftrightarrow (\forall y \in S_2 : \exists x \in S_1 : \exists z \in S_3 : B(x, y, z))$$

In order to generalize from crisp to fuzzy sets, we can simply require that $B(\tilde{S}_1^\alpha, \tilde{S}_2^\alpha, \tilde{S}_3^\alpha)$ is true for all α -cuts (cf. Definitions 2.12 and 2.13 for convexity and star-shapedness):

Crisp betweenness for fuzzy regions

$$\begin{aligned} B(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3) &\Leftrightarrow \forall \alpha \in [0, 1] : B(\tilde{S}_1^\alpha, \tilde{S}_2^\alpha, \tilde{S}_3^\alpha) \\ &\Leftrightarrow \forall \alpha \in [0, 1] : \forall y \in \tilde{S}_2^\alpha : \exists x \in \tilde{S}_1^\alpha : \exists z \in \tilde{S}_3^\alpha : B(x, y, z) \end{aligned}$$

Considering empty
 α -cuts

If $\tilde{S}_2^\alpha = \emptyset$, then $B(\tilde{S}_1^\alpha, \tilde{S}_2^\alpha, \tilde{S}_3^\alpha)$ is true independent of \tilde{S}_1^α and \tilde{S}_3^α . If $\tilde{S}_2^\alpha \neq \emptyset$, but $\tilde{S}_1^\alpha = \emptyset$ or $\tilde{S}_3^\alpha = \emptyset$, then $B(\tilde{S}_1^\alpha, \tilde{S}_2^\alpha, \tilde{S}_3^\alpha)$ is false. Since we use a universal quantification over α , this means that also $B(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3)$ is false in this case.

Soft betweenness for
points

This definition is binary and thus allows only for relatively coarse-grained distinctions. In order to derive a *degree* of betweenness for fuzzy sets, we adapt the soft notion Btw_3 for points as provided by Derrac and Schockaert [122] (cf. Section 4.5.1 to our formalization:

$$B_{soft}(x, y, z) = \frac{d_C^\Delta(x, z, W)}{d_C^\Delta(x, y, W) + d_C^\Delta(y, z, W)}$$

Using the same
salience weights

Please note that we need to use the same set of salience weights W for all the three distance computations in order to ensure that $B_{soft}(x, y, z)$ gives meaningful results with $B_{soft}(x, y, z) \in [0, 1]$. One can easily see that $B(x, y, z)$ is true if and only if $B_{soft}(x, y, z) = 1$.

Soft betweenness for
fuzzy regions

We can use $B_{soft}(x, y, z)$ together with the extension principle [449] to generalize $B(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3)$ to a soft notion $B_{soft}^{inf}(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3)$:

Definition 4.9 (Infimum-Based Betweenness)

Let \tilde{S}_1, \tilde{S}_2 , and \tilde{S}_3 be three concepts defined on the same set of domains Δ_S . The soft degree to which \tilde{S}_2 lies between \tilde{S}_1 and \tilde{S}_3 can be computed as follows:

$$B_{soft}^{inf}(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3) = \inf_{\alpha \in [0, 1]} \inf_{y \in \tilde{S}_2^\alpha} \sup_{x \in \tilde{S}_1^\alpha} \sup_{z \in \tilde{S}_3^\alpha} \frac{d_C^{\Delta^s}(x, z, W^{(2)})}{d_C^{\Delta^s}(x, y, W^{(2)}) + d_C^{\Delta^s}(y, z, W^{(2)})}$$

Formal properties of
 B_{soft}^{inf}

We simply replaced $B(x, y, z)$ with $B_{soft}(x, y, z)$, each existential quantification with the supremum (i.e., the largest lower bound), and each universal quantification with the infimum (i.e., the smallest upper bound).⁷ Moreover, we decided to use the salience weights $W^{(2)}$ of the second concept for the distance computations. Potential alternatives include the usage of an unweighted distance or the creation of interpolated weights based on $W^{(1)}$, $W^{(2)}$, and $W^{(3)}$. For now, using $W^{(2)}$ seems, however, to be the most straightforward choice. One can easily see that $B(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3)$ if and only if $B_{soft}^{inf}(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3) = 1$. Moreover, if $\tilde{S}_2 \subseteq \tilde{S}_1$ (or $\tilde{S}_2 \subseteq \tilde{S}_3$), then $B_{soft}^{inf}(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3) = 1$, because we can pick for each $y \in \tilde{S}_2^\alpha$ always $x = y \in \tilde{S}_1^\alpha$, resulting in $B_{soft}(x, y, z) = 1$ for all possible α -cuts.

Context-dependence
of B_{soft}^{inf}

Since B_{soft}^{inf} considers all different α -cuts of the fuzzy concepts, and since the shapes of these α -cuts also depend on the salience weights, our definition of conceptual betweenness is no longer context-independent (in contrast to the arguments by Derrac and Schockaert [123]). The calculation of \tilde{S}_1^α , \tilde{S}_2^α , and \tilde{S}_3^α involves the parameters μ_0 , c , and W of the respective concepts – only the final distance computation of $B_{soft}(x, y, z)$ uses identical weights and is independent of μ_0 and c . Preserving the context-independence of the betweenness relation would

⁷ Since the α -cuts \tilde{S}_i^α always include their border, \inf and \sup over the points x, y, z are equivalent to \min and \max , respectively. However, as we will see in the proof of Proposition 4.8, $B_{soft}(x, y, z)$ is minimized for $\alpha \rightarrow 0$, but not for $\alpha = 0$.

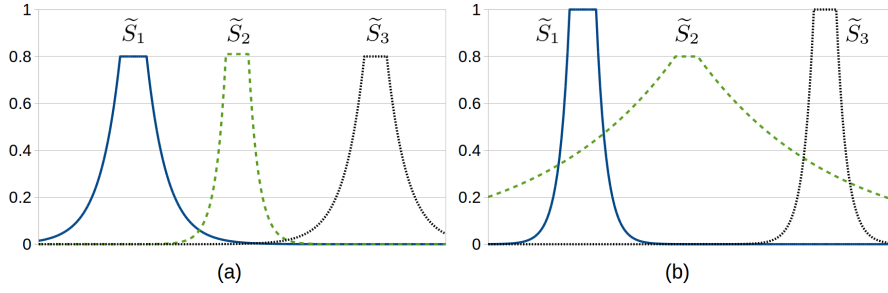


Figure 4.13: Two problematic cases for $B_{soft}^{inf}(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3)$ based on μ_0 (a), and c and W (b), respectively.

require us to use the same values of μ_0 , c , and W for all three concepts, which would then essentially reduce them to their crisp cores. Since this is not adequate in the context of our formalization, which emphasizes the fuzziness of conceptual regions, we have decided to accept the context-dependence of our betweenness relation.

Under certain circumstances, however, $B_{soft}^{inf}(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3)$ yields some rather unintuitive results: If we consider the three concepts in Figure 4.13a, where $\mu_0^{(1)} = \mu_0^{(3)} = 0.80$ and $\mu_0^{(2)} = 0.81$, then $\tilde{S}_1^{0.81} = \tilde{S}_3^{0.81} = \emptyset$ and $\tilde{S}_2^{0.81} \neq \emptyset$, and thus $B_{soft}^{inf}(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3) = 0$. This seems to indicate that \tilde{S}_2 is not at all between \tilde{S}_1 and \tilde{S}_3 even though from inspecting Figure 4.13a, one would expect $B_{soft}^{inf}(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3)$ to be rather large. One can easily see that this pathological case always occurs when $\mu_0^{(2)}$ is larger than both $\mu_0^{(1)}$ and $\mu_0^{(3)}$.

Problem based on maximal membership μ_0

Moreover, in Figure 4.13b we see a simple example where $\mu_0^{(2)} < \min(\mu_0^{(1)}, \mu_0^{(3)})$, and where also $B(S_1, S_2, S_3)$, but where will still get $B_{soft}^{inf}(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3) = 0$. The reason for this is that $\mu_{\tilde{S}_2}(x)$ decays slower than both $\mu_{\tilde{S}_1}(x)$ and $\mu_{\tilde{S}_3}(x)$. We can therefore pick a very small α for which we can find a point in \tilde{S}_2^α that is very far away from both \tilde{S}_1^α and \tilde{S}_3^α and thus has a low value for $B_{soft}(x, y, z)$. The following proposition specifies the exact condition under which this happens:

Problem based on weights W and sensitivity parameter c

Proposition 4.8 (Problematic Case for B_{soft}^{inf})

Let \tilde{S}_1, \tilde{S}_2 , and \tilde{S}_3 be three concepts. Assume we find a dimension $d^* \in \delta^* \in \Delta_S$ for which the following is true:

$$c^{(2)}w_{\delta^*}^{(2)}\sqrt{w_{d^*}^{(2)}} < \min\left(c^{(1)}w_{\delta^*}^{(1)}\sqrt{w_{d^*}^{(1)}}, c^{(3)}w_{\delta^*}^{(3)}\sqrt{w_{d^*}^{(3)}}\right)$$

Then, $B_{soft}^{inf}(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3) = 0$.

Proof. See Appendix C.4. □

The pathological cases from Figure 4.13 arise, because we compute the infimum over all α -cuts. In order to achieve a more generous degradation, one may thus want to aggregate over the α -cuts in a different way. One possible approach is to compute the *integral* over all

Aggregating over α -cuts with the integral

α -cuts, which is similar in spirit to the summation proposed by Derrac and Schockaert [122] (cf. Section 4.5.1). Since we integrate over α in the interval $[0, 1]$ and since the degree of betweenness computed for each α -cut also lies in the interval $[0, 1]$, the result of this integration will also be a number between zero and one.

Definition 4.10 (Integral-Based Betweenness)

Let \tilde{S}_1 , \tilde{S}_2 , and \tilde{S}_3 be three concepts defined on the same set of domains Δ_S . The integral-based soft degree to which \tilde{S}_2 lies between \tilde{S}_1 and \tilde{S}_3 can be computed as follows:

$$B_{soft}^{int}(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3) = \int_0^1 \min_{y \in \tilde{S}_2^\alpha} \max_{x \in \tilde{S}_1^\alpha} \max_{z \in \tilde{S}_3^\alpha} Btw_3(x, y, z) d\alpha$$

Properties of B_{soft}^{int}

When comparing Definition 4.10 to Definition 4.9, one can easily see that B_{soft}^{inf} is a lower bound of B_{soft}^{int} :

Lemma 4.1 (B_{soft}^{int} is Bounded by B_{soft}^{inf})

Let \tilde{S}_1 , \tilde{S}_2 , and \tilde{S}_3 be three concepts. Then, the following inequation holds:

$$B_{soft}^{inf}(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3) \leq B_{soft}^{int}(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3)$$

Proof. See Appendix C.4. □

Further desirable properties

Moreover, the following desirable properties of a betweenness function are also fulfilled by B_{soft}^{int} (based on Lemma 4.1), while avoiding the pathological cases from Figure 4.13:

Corollary 4.1 (Properties of B_{soft}^{int})

$B_{soft}^{int}(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3)$ preserves the following desirable properties for any three concepts \tilde{S}_1 , \tilde{S}_2 , and \tilde{S}_3 :

1. $B(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3) \Leftrightarrow (B_{soft}^{int}(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3) = 1)$
2. $(\tilde{S}_2 \subseteq \tilde{S}_1) \Rightarrow (B_{soft}^{int}(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3) = 1)$
3. $(B_{soft}^{int}(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3) = 1) \Rightarrow (\mu_0^{(2)} \leq \min(\mu_0^{(1)}, \mu_0^{(2)}))$

4.5.3 Implementation and Example

Implementation

Both proposed variants of betweenness need to minimize and maximize $B_{soft}(x, y, z)$ over a given α -cut of \tilde{S}_1 , \tilde{S}_2 , and \tilde{S}_3 . This has been implemented using numerical optimization algorithms from the `scipy.optimize` package. As stated in Section 4.5.2, B_{soft}^{inf} and B_{soft}^{int} differ only with respect to the aggregation over α : B_{soft}^{inf} computes the infimum over all α -cuts, while B_{soft}^{int} computes the integral. Both types of aggregation can again be implemented by using numerical optimization. For B_{soft}^{int} , the integral can, however, also be approximated by a Riemann sum (i.e., sampling a relatively small number of equidistant values for α , and then computing the average across all these α -cuts).

Number n of Dimensions	Runtime of B_{soft}^{inf} in ms	Runtime of B_{soft}^{int} in ms
1	129.6244	35.3474
2	1454.2154	230.6475
4	10681.6994	638.9526
8	–	2021.6039
16	–	4373.0850

Table 4.11: Average runtime of B_{soft}^{inf} and B_{soft}^{int} , averaged across 100 randomly created triples of single-cuboid concepts, using an average over 20 α -cuts to approximate the integral.

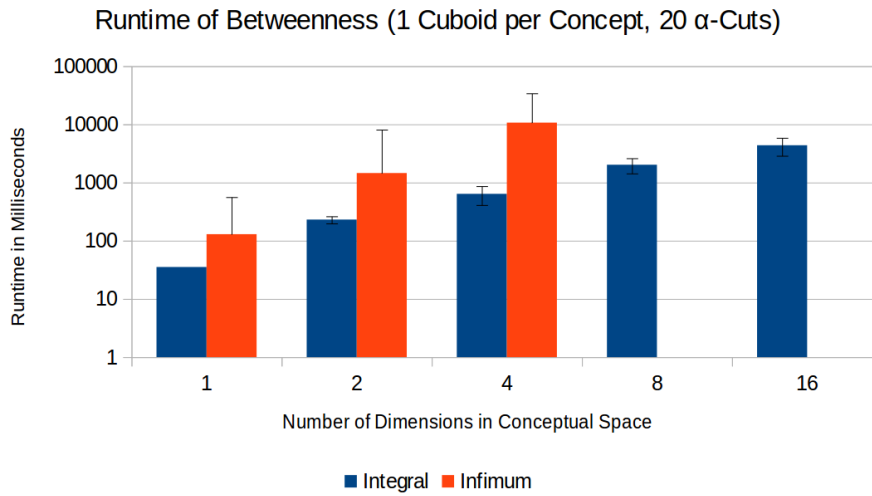


Figure 4.14: Average runtime of the betweenness operation for single-cuboid concepts as a function of the number of dimensions in the conceptual space.

We can assume that this approximation is considerably faster than using an additional layer of numerical optimization. Therefore, B_{soft}^{int} has been implemented as a Riemann sum, treating the number of α -cuts to use as a parameter that can be specified by the user.

Since neither B_{soft}^{inf} nor B_{soft}^{int} are based on any previously introduced operations, we have again conducted a small set of runtime experiments. We have measured the runtime of B_{soft}^{inf} and B_{soft}^{int} for 100 randomly generated triples of concepts, investigating the number n of dimensions in the conceptual space, the number m of cuboids per concept, and the number n_α of α -cuts used for approximating B_{soft}^{int} . All experiments were executed on a laptop with an Intel Core i5-6440HQ CPU (2.60 GHz quad-core) processor and 8 GB main memory.

Let us first analyze how the number n of dimensions influences the runtime of B_{soft}^{inf} and B_{soft}^{int} , using a single cuboid per concept and 20 α -cuts for approximating B_{soft}^{int} . As we can see from Table 4.11 and

Runtime experiments

Runtime and the number of dimensions

Number m of Cuboids	Runtime of B_{soft}^{inf} in ms	Runtime of B_{soft}^{int} in ms
1	1454.2154	230.6475
2	7595.1761	574.5356
4	14055.2697	1363.6137
8	27242.9803	2870.4694

Table 4.12: Average runtime of B_{soft}^{inf} and B_{soft}^{int} , averaged across 100 randomly created triples of concepts in a two-dimensional conceptual space, using an average over 20 α -cuts to approximate the integral.

Runtime of Betweenness (2 Dimensions in Conceptual Space, 20 α -Cuts)

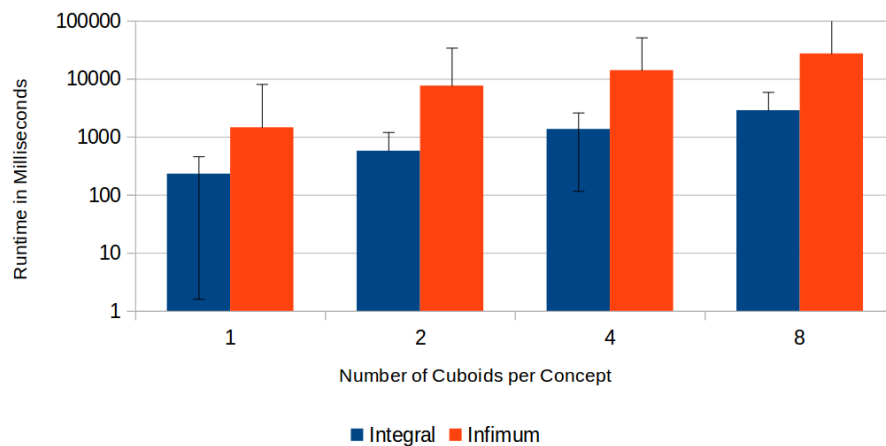


Figure 4.15: Average runtime of the betweenness operation as a function of the number of cuboids used to represent a concept in a two-dimensional space.

Figure 4.14, the runtime of both B_{soft}^{inf} and B_{soft}^{int} shows a superlinear growth with respect to the number of dimensions: In general, doubling the number of dimensions in the conceptual space leads to a runtime more than twice as large. We can also observe that B_{soft}^{int} is considerably faster than B_{soft}^{inf} and it seems to scale much better with an increasing number of dimensions. This illustrates that using an approximation instead of an additional numerical optimization over α comes with considerable performance improvements. Please note that we have not evaluated the runtime of B_{soft}^{inf} for more than four dimensions due to the prohibitively large computational effort.

Runtime and the number of cuboids

Let us now take a look at the effect of the number m of cuboids per concept, which is shown in Table 4.12 and Figure 4.15. Again, we can observe that B_{soft}^{int} is computed considerably faster than B_{soft}^{inf} . For B_{soft}^{int} , we again observe a superlinear growth of runtime with an increasing number of cuboids. For B_{soft}^{inf} , on the other hand, the growth in runtime seems to be slightly sublinear (with a factor of about 1.9 when going from two to four, and from four to eight cuboids, respectively). With a larger number of cuboids, also the standard deviation of the runtime

Number n_α of α -cuts	Runtime of B_{soft}^{inf} in ms	Runtime of B_{soft}^{int} in ms
20	1454.2154	230.6475
50	1461.7224	602.4602
100	1461.1522	1238.6765

Table 4.13: Average runtime of B_{soft}^{inf} and B_{soft}^{int} , averaged across 100 randomly created triples of single-cuboid concepts in a two-dimensional conceptual space.

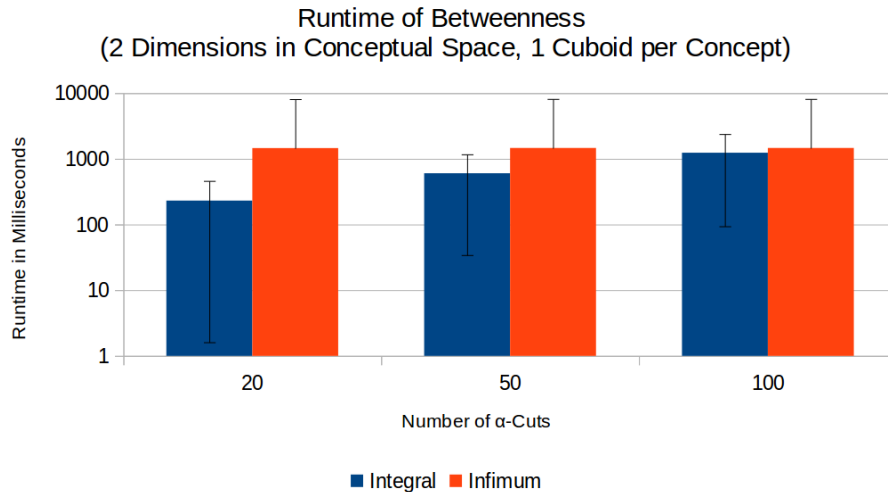


Figure 4.16: Average runtime of the betweenness operation as a function of the number of α -cuts used in approximating the integral, using single-cuboid concepts in a two-dimensional space.

tends to increase, especially for B_{soft}^{inf} , indicating that extreme cases with very large runtime have a larger impact on the average.

Finally, let us consider the effects varying the number n_α of α -cuts used to approximate B_{soft}^{int} . The corresponding results are shown in Table 4.13 and Figure 4.16. Using more α -cuts leads to an increased runtime with a slightly superlinear tendency. When using 100 α -cuts, the average runtime of B_{soft}^{inf} and B_{soft}^{int} becomes approximately identical. However, the runtime measurements for B_{soft}^{int} have a considerably smaller standard deviation than the ones for B_{soft}^{inf} (namely, 1194 milliseconds versus 6680 milliseconds), indicating a larger proportion of computationally costly cases for B_{soft}^{inf} than for B_{soft}^{int} . Moreover, we would like to point out that Table 4.13 contains three different estimates for the runtime of B_{soft}^{inf} , although B_{soft}^{inf} is not affected by the number of α -cuts varied across the three conditions. The fact that these three reported values only vary slightly highlights that our runtime estimates for the betweenness operation are reasonably stable for our analyses.

Overall, the runtime performance of both betweenness variants is quite high for any nontrivial conceptual space. This is based on the fact that we need to employ multiple nested numerical optimizations in order

Runtime and the number of α -cuts

Summary of the runtime experiments

to calculate the respective minima and maxima. The runtime of B_{soft}^{inf} is higher than the one of B_{soft}^{int} , because our implementation of B_{soft}^{inf} involves another optimization by finding the infimum over all α -cuts, whereas B_{soft}^{int} simply computes the average across a fixed number of α -cuts. Similar to our observations for the intersection and size operations (Sections 3.1.3 and 4.1.3, respectively), the number of dimensions in the similarity spaces tends to have a stronger impact on runtime than the number of cuboids per concept. Based on the overall quite high runtime for both B_{soft}^{inf} and B_{soft}^{int} , one might be tempted to use a simpler definition of betweenness. For instance, one could compute the degree of betweenness of the concepts' midpoints which would be computationally much more efficient. This would, however, ignore both the size of the concepts' cores and their respective degree of fuzziness, which is not satisfying from a theoretical perspective. Finding a good trade-off between theoretical soundness and a fast implementation should be a goal of future research, based on concrete applications of the proposed formalization and their respective requirements.

Additional simulations for comparing betweenness values

After having compared the runtime of B_{soft}^{inf} and B_{soft}^{int} , let us now compare the resulting degrees of betweenness. We have created 1,000 random triples of two-cuboid concepts in a four-dimensional space and have computed for each of them B_{soft}^{inf} as well as B_{soft}^{int} (using 20 and 100 α -cuts, respectively). By comparing the numerical results for these exemplary betweenness relations, we hope to get a better understanding of our proposed definitions.

Comparing B_{soft}^{inf} and B_{soft}^{int}

Figure 4.17 contains a scatter plot with the betweenness values of B_{soft}^{inf} and B_{soft}^{int} , using 100 α -cuts for the latter. The correlation between the two betweenness measures reaches only intermediate values with Pearson's $r \approx 0.5284$ (which measures linear correlation) and Spearman's $\rho \approx 0.6304$ (which measures monotone correlation), indicating that B_{soft}^{inf} and B_{soft}^{int} are not equivalent. We can clearly see in Figure 4.17 that $B_{soft}^{inf} = 0$ in most of the cases. It thus seems that the two failure cases illustrated in Figure 4.13 occur very frequently in our experiment. Please note that our code only executes the numerical optimization if neither of these cases applies (which can be detected based on the values of mu_0 , and the conditions from Proposition 4.8, respectively). Please recall from Tables 4.11 and 4.12 that B_{soft}^{inf} was on average quite slow in our runtime analysis. As the cases with $B_{soft}^{inf} = 0$ can be expected to have a very small runtime, this means that for all cases where $B_{soft}^{inf} > 0$, the actual runtime is much larger than the overall average. This also explains the large standard deviations observed for the runtime of B_{soft}^{inf} : It is either very fast (if one of the pathological cases applies) or very slow (if the numerical optimization is needed). It thus seems that B_{soft}^{inf} might not be very useful in practice: In addition to its large runtime, it equals zero for most triples of concepts, and this does not convey a very fine-grained sense of conceptual betweenness.

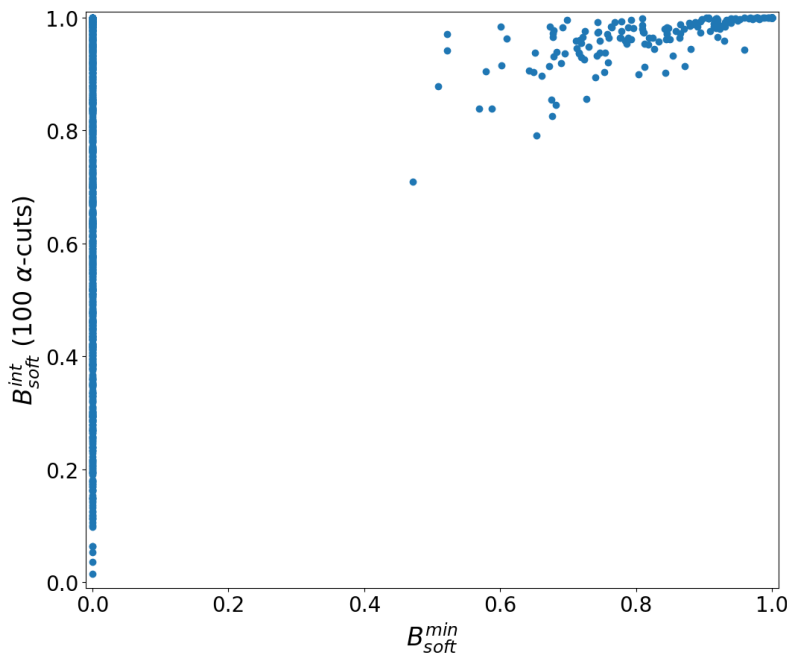


Figure 4.17: Scatter plot comparing the betweenness values obtained via B_{soft}^{inf} and B_{soft}^{int} (using 100 α -cuts) for 1,000 randomly generated triples of two-cuboid concepts in a four-dimensional conceptual space.

Let us now take a look at the influence of the number of α -cuts on the numerical values obtained by B_{soft}^{int} . Figure 4.18 compares the results for 20 and 100 α -cuts. We can immediately see that the numbers obtained through these two variants are strongly correlated (Pearson's $r \approx 0.9984$, Spearman's $\rho \approx 0.9976$). Although the exact numerical values differ slightly (which is to be expected), it seems like the number of α -cuts used for approximating B_{soft}^{int} is not very crucial in practice. Since more α -cuts considerably increase the runtime but do not seem to impact the resulting betweenness numbers that much, one might be able to find a good trade-off between runtime efficiency and quality of the results.

Betweenness values and the number of α -cuts

Let us now finally return to our fruit space example. Table 4.19 shows the definition of our fruit concepts and Figure 4.19 illustrates them. In Table 4.15, we show betweenness values for selected triples of concepts. In our implementation, one can compute these betweenness values as follows:

Fruit space example

```
apple.between(pear, orange, method="infimum")
0.0
apple.between(pear, orange, method="integral")
0.8994529846730694
apple.between(pear, orange, method="integral", num_alpha_cuts
= 100)
0.9002036026183976
```

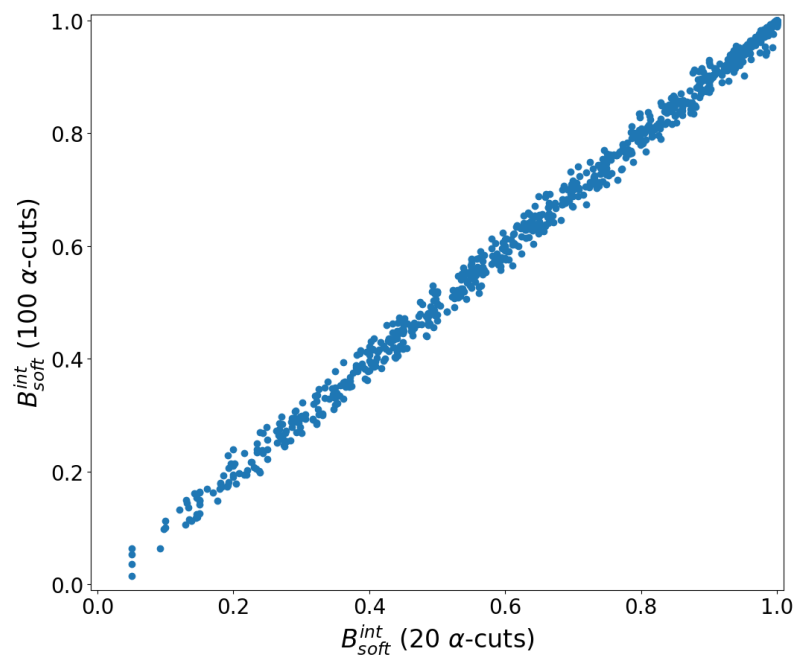


Figure 4.18: Scatter plot comparing the betweenness values obtained via B_{soft}^{int} for 1,000 randomly generated triples of two-cuboid concepts in a four-dimensional conceptual space, using 20 and 100 α -cuts, respectively.

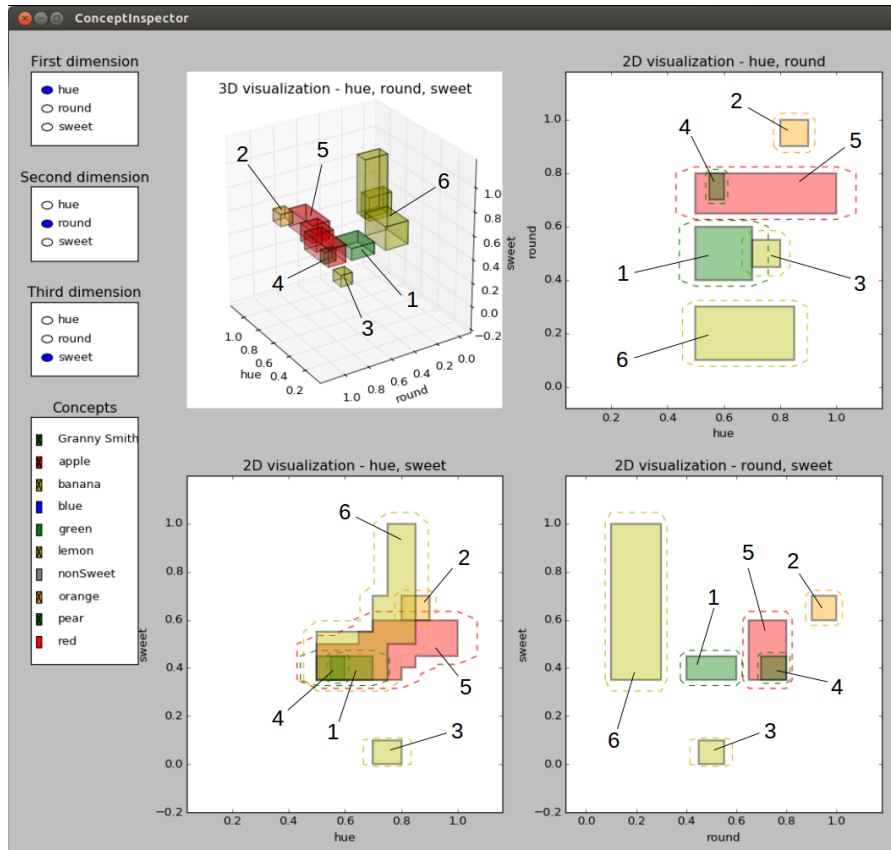


Figure 4.19: Screenshot of the ConceptInspector tool illustrating all fruit concepts used in our betweenness example. Concepts are labeled as follows: PEAR (1), ORANGE (2), LEMON (3), GRANNY SMITH (4), APPLE (5), and BANANA (6).

Concept	Δ_S	p^-	p^+	μ_0	c	W		
						$w_{\delta_{\text{COLOR}}}$	$w_{\delta_{\text{SHAPE}}}$	$w_{\delta_{\text{TASTE}}}$
PEAR	Δ	(0.50, 0.40, 0.35)	(0.70, 0.60, 0.45)	1.0	24.0	0.50	1.25	1.25
ORANGE	Δ	(0.80, 0.90, 0.60)	(0.90, 1.00, 0.70)	1.0	30.0	1.00	1.00	1.00
LEMON	Δ	(0.70, 0.45, 0.00)	(0.80, 0.55, 0.10)	1.0	40.0	0.50	0.50	2.00
GRANNY SMITH	Δ	(0.55, 0.70, 0.35)	(0.60, 0.80, 0.45)	1.0	50.0	1.00	1.00	1.00
APPLE	Δ	(0.50, 0.65, 0.35)	(0.80, 0.80, 0.50)	1.0	20.0	0.50	1.50	1.00
		(0.65, 0.65, 0.40)	(0.85, 0.80, 0.55)					
		(0.70, 0.65, 0.45)	(1.00, 0.80, 0.60)					
BANANA	Δ	(0.50, 0.10, 0.35)	(0.75, 0.30, 0.55)	1.0	20.0	0.75	1.50	0.75
		(0.70, 0.10, 0.50)	(0.80, 0.30, 0.70)					
		(0.75, 0.10, 0.50)	(0.85, 0.30, 1.00)					

Table 4.14: Definitions of several fruit concepts for our exemplary betweenness computations.

\tilde{S}_1	\tilde{S}_2	\tilde{S}_3	$B_{soft}^{inf}(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3)$	$B_{soft}^{int}(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3)$
APPLE	GRANNY SMITH	LEMON	1.0000	1.0000
APPLE	PEAR	BANANA	1.0000	1.0000
GRANNY SMITH	PEAR	BANANA	0.0000	0.9201
PEAR	APPLE	ORANGE	0.0000	0.8995
PEAR	GRANNY SMITH	ORANGE	1.0000	1.0000
APPLE	PEAR	GRANNY SMITH	0.4676	0.5275
ORANGE	BANANA	GRANNY SMITH	0.0000	0.3557

Table 4.15: Betweenness values for selected triples of fruit concepts, rounded to four decimal places. For B_{soft}^{int} , 20 α -cuts were used.

Examples with obvious betweenness relations

Both B_{soft}^{inf} and B_{soft}^{int} agree on the fact that GRANNY SMITH is completely between APPLE and LEMON, because $\tilde{S}_{\text{GRANNY SMITH}} \subseteq \tilde{S}_{\text{APPLE}}$. We furthermore observe that PEAR is considered to lie perfectly between APPLE and BANANA by both definitions of conceptual betweenness. As one can see in Figure 4.19, this is intuitively plausible based on the locations of the respective cores. A first difference between B_{soft}^{inf} and B_{soft}^{int} can be observed for the triple (GRANNY SMITH, PEAR, BANANA): Here, B_{soft}^{inf} returns a value of zero, while B_{soft}^{int} shows a more generous degradation with a value of approximately 0.92. Based on Figure 4.19, the latter value seems to be more meaningful, since a betweenness degree of zero does not reflect the geometric arrangement of the concepts' cores. Still, the fact that $B_{soft}^{int}(\tilde{S}_{\text{APPLE}}, \tilde{S}_{\text{PEAR}}, \tilde{S}_{\text{BANANA}}) > B_{soft}^{int}(\tilde{S}_{\text{GRANNY SMITH}}, \tilde{S}_{\text{PEAR}}, \tilde{S}_{\text{BANANA}})$ reflects that the betweenness relation is not perfect in this example. The triple (GRANNY SMITH, PEAR, BANANA) fulfills the condition of Proposi-

tion 4.8 (since $c^{(\text{GRANNY SMITH})} \gg c^{(\text{PEAR})}$) and thus corresponds to the pathological failure case of B_{soft}^{inf} from Figure 4.13b.

Based on Figure 4.19, one could also argue that both APPLE and GRANNY SMITH lie to some degree between PEAR and ORANGE. Here, B_{soft}^{int} assigns relatively high betweenness values in both cases, whereas B_{soft}^{inf} assigns a value of zero to the triple (PEAR, APPLE, ORANGE). This is again the failure case from Proposition 4.8 and Figure 4.13b. It seems plausible to assign a lower degree of betweenness to APPLE than to GRANNY SMITH, because the core of the APPLE concepts is larger and parts of it are further away from the straight line connecting the cores of PEAR and ORANGE than the core of GRANNY SMITH.

The final two lines of Table 4.15 illustrate cases where we would not expect a high degree of betweenness. Both betweenness functions assign lower, but still mostly non-zero degrees of betweenness to these combinations. One may argue that these betweenness values are too high, given the respective arrangement of cores. However, one needs to keep in mind that both B_{soft}^{inf} and B_{soft}^{int} take into account *all* α -cuts of the respective concepts, including for example also the 0.05-cuts, which may considerably overlap.

*Examples with
debatable
betweenness
relations*

*Examples without
betweenness
relations*

4.6 SUPPORTED APPLICATIONS

After having defined various operations for measuring relations between concepts, we would now like to sketch how these operations can be used in both learning and reasoning tasks. In Section 4.6.1, we again consider concept formation as an important type of learning process, before looking at commonsense reasoning strategies in Section 4.6.2.

Overview

4.6.1 Concept Formation

Please recall from Section 3.6.1 that concept formation is an incremental clustering process [164]. In our case, each cluster can be represented by a concept. A concept formation algorithm makes one unlabeled observation at a time, assigns this observation to the best-matching cluster (or creates a new cluster), and then makes small adjustments to its inventory of clusters, if necessary. We have already argued in Section 3.6.1 that our formalization provides all operations necessary for implementing these updates, such as merging and splitting clusters with the union and cut operations, respectively. What was still missing from our discussion in Section 3.6.1 were, however, ways of deciding when to apply which kind of modification. In the following, we argue that the additional operations defined in this chapter can be used as parts of such an overall control strategy.

*Concept formation as
incremental
clustering*

Both the splitting and the merging of existing clusters needs to be initiated based on some predefined condition. For instance, we might want to specify that the clusters created by the concept formation

*Controlling for the
size of different
clusters*

algorithm should have a roughly equal size. Then, a cluster that grows too big would need to be split into two parts, and a cluster that becomes too small would need to be merged with one of its neighbors. The size operation defined in Section 4.1.2 quantifies the size of a concept and can thus be used together with some lower and upper thresholds to initiate the merge and split operations.

Identifying candidate clusters for merging

Once we have decided that a given cluster needs to be merged with one of its neighbors, we still need to find the most appropriate merging partner. In a first step, we can use concept similarity as defined in Section 4.4.2 in order to identify the closest neighbors of a given concept. One may then use the respective similarity values, potentially in combination with the sizes of the respective concepts, to decide which clusters to merge.

Creating a hierarchy of clusters

Many concept formation algorithms maintain an explicit hierarchy of clusters [164] in order to provide a description of the observation at multiple levels of abstraction: We might thus also be interested in extracting the emergent conceptual hierarchy (e.g., SKY BLUE being a special shade of BLUE) from our current set of clusters. This can be easily done using both the crisp and the soft subthood relations between concepts as defined in Section 4.2.1.

Incorporating top-down knowledge

In Section 1.3.4, we have already argued that the incorporation of top-down constraints in the concept learning process is desirable from a cognitive point of view. Given an initial set of concepts (either manually created or obtained via concept formation), one may thus specify logical rules such as $\forall x : \text{APPLE}(x) \wedge \text{RED}(x) \Rightarrow \text{SWEET}(x)$, whose validity should be preserved in future modifications of the concept inventory. Using the implication operation from Section 4.3.1 together with the intersection and union operators from Sections 3.1.2 and 3.2.1, respectively, we can evaluate the degree of truth for such logical rules. This can then be used as an additional constraint in the control strategy of the envisioned concept formation algorithm. Also zero shot learning based on definitions like $\text{GRANNY SMITH} \Leftrightarrow \text{GREEN} \wedge \text{SOUR} \wedge \text{ROUND}$ can be implemented in a similar way.

Outlook

Our discussion has been necessarily brief, but has hopefully still illustrated that the operations defined in this chapter can play an important role in the control strategy of a concept formation algorithm. We will return to the topic of concept formation in Chapter 7, where we will introduce three concrete concept formation algorithms from the literature and relate them to conceptual spaces. In Chapter 7, we will also the aspect of learning under top-down constraints in more detail.

4.6.2 Commonsense Reasoning

Symbolic reasoning

In classical symbolic AI, reasoning is mostly based on logical rules such as $\forall x : \text{APPLE}(x) \wedge \text{RED}(x) \Rightarrow \text{SWEET}(x)$. However, such symbolic reasoning systems are inherently limited to the rules and symbols they are provided with. Since it is generally infeasible to express all necessary

pieces of everyday knowledge in a symbolic knowledge base, these systems often fail to make deductions based on insufficient knowledge. Humans on the other hand are still able to draw plausible (yet not deductively provable) conclusions in such situations.

As already briefly introduced in Section 1.2.5, the area of commonsense reasoning therefore aims to provide appropriate techniques for drawing plausible conclusions from incomplete knowledge. In contrast to black box machine learning models, these commonsense reasoning techniques are usually also able to give a justification for their conclusions [123]. As already mentioned in Section 1.2.5, Schockaert and Prade [352, 353] as well as Derrac and Schockaert [122, 123] have shown how to implement such commonsense reasoning strategies with the conceptual spaces framework. In the following, we argue that also our formalization is capable of supporting these strategies.

Commonsense reasoning

Similarity-based reasoning assumes that similar concepts have similar properties, usages, and behaviors, and that they can be therefore treated alike. For instance, if we know that Alice liked the "Lord of the Rings" movies and if the concept LORD OF THE RINGS is relatively similar to the concept THE HOBBIT, then we can make the educated guess that Alice might also like the "The Hobbit" movies [123].

Similarity-based reasoning

As already described in Section 2.1.1, cognitive similarity of observations is defined as an exponentially decaying function of their distance in the similarity space. In Section 4.4.2, we have generalized this to a notion of similarity for fuzzy conceptual regions. Our formalization thus provides ways of quantifying the similarity of two observations (based on their weighted combined distance, see Sections 2.1.1 and 2.3.1), the similarity of an observation to a concept (based on its degree of membership, see Section 2.3.3), and the similarity of two concepts (based on subethood or the Jaccard index, see Section 4.4.2). It is therefore quite straightforward to implement similarity-based reasoning on top of our formalization.

Similarity-based reasoning and our formalization

Interpolative Reasoning assumes that any property, usage, or behavior that is shared by two known examples also applies to any intermediate case. It can thus be interpreted as implicitly using Gärdenfors' convexity constraint for conceptual regions (cf. Sections 1.2.1 and 2.1.2). If WINE SHOP is conceptually between GOURMET SHOP and LIQUOR STORE and if we know that both gourmet shops and liquor stores are subject to some sort of regulation, then we can assume that also wine shops are affected by this regulation [123].

Interpolative reasoning

In their work on interpolative reasoning in conceptual spaces, Schockaert and Prade [352, 353] as well as Derrac and Schockaert [122, 123] base interpolative reasoning on a geometric notion of betweenness for points and crisp regions (cf. Section 4.5.1). In Section 4.5.2, we have generalized their definitions of geometric betweenness to a soft notion of betweenness for fuzzy conceptual regions. Our formalization can thus also be used in the context of interpolative reasoning.

Interpolative reasoning and our formalization

Other commonsense reasoning strategies

Please note that there are also other types of commonsense reasoning strategies [123], which are currently not supported by our formalization. *A fortiori reasoning* is based on a meaningful ordering of observations and concepts along a single direction. For example, if one is not allowed to drink BEER at a certain age, one is probably also not allowed to drink WHISKEY at that age, because WHISKEY has a higher concentration of alcohol. *Analogy-based reasoning* on the other hand uses analogical relations between two pairs of observations or concept: If PUPPY relates to DOG like KITTEN relates to CAT, then knowledge about the difference or relation between KITTEN and CAT can be used to plausibly derive additional knowledge about the difference or relation between PUPPY and DOG. Derrac and Schockaert [123] have modeled these two strategies based on interpretable directions and based on the parallelism of difference vectors, respectively. These notions have not been explicitly included in our formalization, but would be natural future extensions of our work.

Extracting symbolic knowledge from conceptual spaces

All of the above mentioned commonsense reasoning strategies are implemented as part of the conceptual layer. At this point, we would also like to highlight how our formalization supports the extraction of commonsense knowledge into the symbolic layer. For instance, intra-domain correlations within a given concept are encoded in a geometric way in our formalization (cf. Section 2.2) and can be viewed as one specific type of commonsense knowledge. We have already sketched in Section 4.3 in the context of Rickard’s co-occurrence values [329] how the implication operation can be used to provide a symbolic description of the correlation. For instance, the observation that red apples tend to be sweet can be formally expressed as $\forall x : \text{APPLE}(x) \wedge \text{RED}(x) \Rightarrow \text{SWEET}(x)$. The degree of truth of this expression can be obtained by considering the intersection of the conceptual regions APPLE and RED and checking the subthood relation between the resulting region and the property SWEET. By systematically testing all relevant properties, one can thus identify rules with a high degree of truth, i.e., commonsense reasoning patterns that are expected to hold in most (but not necessarily all) cases. These rules can then be used in a symbolic system to draw appropriate conclusions based on cross-domain correlations. In Chapter 7, we will revisit this knowledge extraction aspect in the context of logic tensor networks [21, 358].

Outlook

As argued above, our formalization supports both similarity-based and interpolative reasoning with the definitions for concept similarity and conceptual betweenness from Sections 4.4.2 and 4.5.2, respectively. Other strategies such as a fortiori reasoning or analogy-based reasoning are currently not supported, but would be interesting future extensions. The exact reasoning strategy (i.e., a specific algorithm that uses similarity and betweenness values to arrive at a conclusion) has been left unspecified and would of course be needed for any practical applications. Nevertheless, our brief discussion has hopefully illustrated that the definitions from this chapter can be useful in the context of reasoning processes.

Formalization	Size	Subsethood	Implication	Similarity	Betweenness
Aisbett and Gibbon [11]	(-)	(-)	(-)	✓	(-)
Raubal [327]	-	-	-	-	-
Ahlqvist [10]	(-)	✓	(-)	✓	(-)
Rickard [329]	(-)	(-)	(-)	✓	-
Rickard et al. [330]	(-)	(-)	(-)	✓	-
Adams Raubal [3]	(-)	(-)	(-)	✓	(-)
Lewis Lawry [253]	(-)	(-)	(-)	(-)	(-)
Our Formalization	✓	✓	✓	✓	✓

Table 4.16: Overview of different formalizations of the conceptual spaces framework based on the relations between concepts they define ("✓" means "available", "-" means "not available", and "(-)" means "could be easily added").

4.7 COMPARISON TO OTHER FORMALIZATIONS

Table 4.16 summarizes the availability of different relations between concepts in the different formalizations of the conceptual spaces framework that we introduced in Section 2.4.2. Conceptual similarity is covered by almost all formalizations, since it is a key operation of the conceptual spaces framework. Moreover, size, subsethood, and implication can often be easily added by computing the volume of a conceptual region and by checking for set inclusion, respectively. Also conceptual betweenness can be added to most formalizations based on the definitions by Derrac and Schockaert [122] for crisp sets and our generalization to fuzzy sets. Let us now take a closer look at the individual formalizations.

The formalization by Aisbett and Gibbon [11] describes concepts as crisp convex sets in pointed metric spaces. The only measure on concepts provided by this formalization is conceptual similarity, which is based on the Hausdorff distance of their respective regions A and B : $d_H(A, B) = \max(\sup_{a \in A} \inf_{b \in B} d(a, b), \sup_{b \in B} \inf_{a \in A} d(a, b))$. One can, however, argue that size and subsethood can be easily added to their formalization by resorting to set theory. Also an implication operation could be added by equating it with subsethood as in our proposal. Since their formalization starts from a generalized betweenness relation for points, it would also be conceivable to extend this notion to concepts.

Since Raubal [327] does not mathematically formalize properties and concepts, he is unable to give definitions for measuring relations between them. Without such a mathematical definition of conceptual regions, it is impossible to add any of the operations to his approach.

Overview

Aisbett and Gibbon: similarity based on Hausdorff distance

Raubal: no relations at all

Ahlqvist: similarity and subsethood based on fuzzy set theory

Ahlqvist’s formalization [10] uses rough fuzzy sets (a generalization of "regular" fuzzy sets, which uses an upper and a lower membership function) to represent concepts. It provides an explicit definition for concept similarity based on the so-called fuzzy dissemblance index $\delta(\tilde{A}, \tilde{B}) = \frac{1}{2}M(\tilde{A}^0 \cup \tilde{B}^0) \int_{\alpha=0}^1 (M(\tilde{A}^\alpha \cup \tilde{B}^\alpha) - M(\tilde{A}^\alpha \cap \tilde{B}^\alpha))d\alpha$, which aims to quantify the distance between two fuzzy sets. Conceptual overlap is defined in a similar way to our definition of subsethood, namely as $Sub(\tilde{A}, \tilde{B}) = \frac{M(\tilde{A} \cap \tilde{B})}{M(\tilde{B})}$. Please note that in this case, $M(\tilde{B})$ is used as denominator while we use $M(\tilde{A})$. Ahlqvist’s definition of conceptual overlap should therefore be interpreted as measuring whether \tilde{B} is a subset of \tilde{A} . Ahlqvist computes both similarity and concept overlap on individual properties and aggregates them through a weighted sum to obtain results for overall concepts. Our formalization on the other hand does not make such a distinction between properties and concepts. If one equates implication with subsethood, an implication measure can easily be added to his formalization based on his definition of conceptual overlap. Moreover, computing the integral over the membership functions of a given rough fuzzy set should give a reasonable measure of concept size. Finally, also our proposed ways of measuring betweenness can potentially be generalized from fuzzy sets to rough fuzzy sets and thus added to Ahlqvist’s formalization.

Rickard: similarity as fuzzy mutual subsethood

In the formalization by Rickard [329], concepts are represented as matrices containing co-occurrence statistics of properties. These matrices are then interpreted as fuzzy sets on the universe of ordered property pairs. This means that each matrix entry A_{ij} is interpreted as the membership $\mu_{\tilde{A}}(i, j)$ of the property pair (i, j) in the fuzzy set \tilde{A} . Concept similarity is then explicitly formalized by using fuzzy mutual subsethood which is defined as follows:

$$E(\tilde{A}, \tilde{B}) = \frac{\sum_{(i,j)} \min(A_{ij}, B_{ij})}{\sum_{(i,j)} \max(A_{ij}, B_{ij})}$$

Relation to the Jaccard index

Since the number of property pairs (i, j) is finite, one can interpret $\sum_{(i,j)} A_{ij}$ as the size of the fuzzy set \tilde{A} . Moreover, using $\mu_{\tilde{A} \cap \tilde{B}}(x) = \min(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x))$ and $\mu_{\tilde{A} \cup \tilde{B}}(x) = \max(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x))$, one can interpret the formula for $E(\tilde{A}, \tilde{B})$ as $\frac{M(\tilde{A} \cap \tilde{B})}{M(\tilde{A} \cup \tilde{B})}$, which is the definition of the Jaccard index. Rickard’s definition of similarity for concepts can thus be related to our own definition of Sim_J from Section 4.4.2.

Subsethood

When judging the similarity of an observation to a concept, Rickard makes use of fuzzy subsethood (i.e., $Sub(\tilde{A}, \tilde{B}) = \frac{M(\tilde{A} \cap \tilde{B})}{M(\tilde{A})}$). Although not explicitly formalized by Rickard, one could imagine to extend this usage also to measure subsethood between concepts. Again, if we equate implication with subsethood, also an implication measure can be easily added. Both fuzzy mutual subsethood and fuzzy subsethood as employed in Rickard’s formalization implicitly compute the size of a concept by summing the co-occurrence values of all property pairs.

One could easily use this procedure to also explicitly provide a way to measure the size of a concept. However, it is debatable whether conceptual betweenness could be formalized in the context of Rickard's work by simply applying our definitions to his fuzzy sets. One would need to investigate whether the resulting definition results in intuitively understandable results.

Rickard et al. [330] build upon the formalization by Rickard [329], hence essentially the same remarks apply. Rickard et al. define size and overlap (which is a similar notion as subthood) for properties. However, as concepts are not defined as regions in a similarity space, but based on co-occurrence statistics, these definitions do not directly carry over to concepts. Rickard et al. furthermore define a subtype relationship between concepts, which corresponds to specialization, generalization, or a combination of both, depending on the concepts being compared: For instance, if a concept *A* contains the (mutually exclusive) properties *RED* and *BLUE* on the *COLOR* domain, then a subtype *B* that is only associated to *RED* on the *COLOR* domain is a specialization of *A*. On the other hand, if a concept *C* is associated with the properties *RED* and *ARIAL* on the *COLOR* and *FONT* domains, then a subtype *D* that only involves *RED* (but not *ARIAL*) is a generalization of *C*. This subtype relationship is thus not equivalent to a subthood relation.

Rickard et al.: adding size and overlap for properties, as well as subtype relationship for concepts

Adams and Raubal [3] represent concepts by one convex polytope per domain. Their formalization contains two proposals for computing concept similarity, either based on the distance of the concepts' prototypes, or based on the distance of the convex hulls. Based on set theory, we can again imagine that size, subthood, implication, and betweenness could be added to their formalization, being, however, somewhat limited by the inherent crispness of their approach.

Adams and Raubal: similarity based on prototypes or convex hulls

The formalization by Lewis and Lawry [253] uses random sets to represent properties and formalizes concepts as random sets in binary combination spaces, where each dimension codes for the presence or absence of one property. Their formalization does not include any explicit measures for conceptual relations, as its focus lies entirely on conjunctive conceptual combination. One can, however, imagine to measure the size of a concept by integrating over the membership functions of its associated properties as well as its membership function in the combination space. Also a notion of conceptual similarity can be added by measuring the number of shared properties and the overlap of the membership functions. Subthood and implication can again be based on fuzzy subthood (cf. the formalizations by Ahlqvist [10] and Rickard [329]). Finally, Lewis and Lawry define a betweenness relation for the prototypes of composite concepts that use the same combination space. This can potentially be generalized to concepts in a similar way as in our own formalization.

Lewis and Lawry: no relations specified, but many could be added

Our own formalization

In this chapter, we have extended our formalization by explicitly adding five measures for conceptual relations. As one can see, our formalization is thus more comprehensive than all other competitors in this regard. We think that especially our fuzzy definitions of subthood and betweenness are an important contribution that can potentially be transferred to other formalizations.

4.8 SUMMARY

Lessons learned

In this chapter, we have further extended our formalization of the conceptual spaces framework by providing mathematical ways of measuring relations between concepts. We have also argued how these extensions of our framework can be used in both learning and reasoning tasks. The now overall quite large set of operations, both for creating new concepts and for measuring relations between concepts, makes our formalization (to the best of our knowledge) the most thorough and comprehensive formalization of conceptual spaces developed so far. Moreover, in contrast to all other formalizations we are aware of, our work comes with a concrete implementation for all of these operations.

Open ends: size

There are still some open issues that warrant further research: The computation of a concept's size is quite complex. Even though computing the closed formula is significantly faster than numerically approximating the integral over the concept's membership function, it might be desirable for high-dimensional spaces to use a faster approximation of the concept size. This is an especially crucial problem, since we use the concept size also as a building block for subthood, implication, and similarity. Runtime improvements of the concept size will therefore improve also the runtime of various other operators.

Subthood and implication

As we have discussed in Section 4.2.1, our proposed degree of subthood is not always confined to the interval $[0,1]$. Future research should investigate whether there is a straightforward and computationally feasible way of overcoming this limitation. Moreover, we have argued in Section 4.3.1 that the implication relation between two concepts can be equated with their subthood relation. This argument could use some further empirical support by using the subthood relation as implication in a reasoning application.

Similarity

In Section 4.4.2, we have provided two different definitions for conceptual similarity – it remains an open question which one of them is preferable in which concrete application contexts. Again, an empirical study in an application scenario can help to clarify this issue. Moreover, additional candidate definitions for conceptual similarity (e.g., based on the maximal Hausdorff distance over all α -cuts, inspired by [11]) could be investigated in future work.

Analogy-based reasoning

Finally, Derrac and Schockaert [123] have argued that the degree of parallelism for pairs of difference vectors can be a useful measure for supporting analogy-based reasoning: If DOG relates to PUPPY like CAT relates to KITTEN, then the geometrical direction from the DOG concept

to the PUPPY concept should be parallel to the corresponding direction from CAT to KITTEN. Obviously, a formalization of this intuition in our framework would be an interesting additional measure that could support yet another commonsense reasoning strategy.

The formalization provided in this and the previous chapters is the first significant contribution of this dissertation. Although we were able to identify several open ends, the current state of both the formalization and its open source implementation proves a solid foundation for both theoretical and practical research in the context of conceptual spaces. In Chapter 7, we will also consider the question to which extent the machine learning approaches discussed there can be used to learn conceptual regions as defined in our formalization.

Outlook

Part II

MACHINE LEARNING AND OPTIMIZATION BACKGROUND

In the second part of this dissertation, we introduce the mathematical background for our subsequent practical studies in Part III. This mathematical background includes on the one hand various machine learning algorithms, and on the other hand an optimization technique for obtaining similarity spaces from psychological data.

In Chapter 5, we introduce general notions in machine learning along with various algorithms and dimensionality reduction techniques. We then consider artificial neural networks (ANNs) for representation learning in more detail in Chapter 6. In Chapter 7, we then discuss various machine learning approaches for learning conceptual regions in a cognitively plausible way.

Finally, in Chapter 8, we introduce the technique of multi-dimensional scaling (MDS), which allows us to transform dissimilarity ratings from psychological experiments into a spatial representation. We supplement the discussion of several MDS algorithms with a hybrid proposal for grounding the dimensions of a conceptual space, which combines MDS with ANNs.

5

GENERAL MACHINE LEARNING BACKGROUND

5.1	General Notions in Machine Learning	212
5.1.1	Task and Model	213
5.1.2	Experience and Evaluation	215
5.1.3	Fitting the Model's Parameters	222
5.1.4	Practical Considerations	225
5.2	Machine Learning Algorithms	228
5.2.1	Linear Regression	229
5.2.2	Logistic Regression	233
5.2.3	Support Vector Machines	238
5.2.4	Decision Trees	250
5.2.5	k Nearest Neighbors	255
5.3	Dimensionality Reduction	258
5.3.1	Feature Selection	259
5.3.2	Feature Extraction	265
5.3.3	Metric Learning	271
5.4	Summary	279

Machine learning in general aims to approximate the function mapping inputs to outputs based on a given a collection of examples. In other words, given a set of example input-output pairs (x, y) , we try to find the function f that relates inputs to outputs, i.e., $f(x) = y$. Although this description is still relatively abstract, one can already see that this is a very restricted form of learning. It can be linked to behaviorism [377] in psychology, which follows a similar simplifying assumption of input-output mappings, and which has been largely superseded by more cognitive approaches. Nevertheless, machine learning is widely applicable in artificial systems, whenever the designers of the system are not able to hard-code a solution to a given problem and cannot anticipate all possible situations or changes over time [341, Chapter 18]. This is also the case when applying conceptual spaces in artificial agents: We may not be able to manually define all the dimensions of the similarity space, and we may want our agent to autonomously acquire and update conceptual regions over time.

Machine learning in a nutshell

In Section 5.1, we define machine learning problems more formally and introduce some general notions relevant to most machine learning contexts. In Section 5.2, we then give a brief introduction into several selected machine learning algorithms that will play a role in the remainder of this dissertation. In Section 5.3, we summarize different dimensionality reduction algorithms, which can be used to find a low-dimensional, but faithful representation of the input data. This typical

preprocessing step in a machine learning pipeline can be linked to the task of defining the domains and dimensions of a conceptual space. Finally, we conclude this chapter in Section 5.4 with a brief summary.

5.1 GENERAL NOTIONS IN MACHINE LEARNING

Formal definition of machine learning

According to Mitchell [295, Chapter 1], machine learning can be in broad terms defined as follows:

Definition 5.1 (Machine Learning)

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E . [295, Chapter 1]

Mitchell illustrates this definition by using the example of a program learning to play checkers. Here, the task T corresponds to playing checkers, the number of games that have been won by the program can be used as a performance measure P , and the experience E can consist of games played by the program against itself.

Important factors: component to be improved

Russell and Norvig [341, Chapter 18], on the other hand, use four factors to define a machine learning setup. Firstly, they consider the *component* of the artificial agent which shall be improved through machine learning, distinguishing for example perception from action selection. This can be related to the task T in Definition 5.1.

Prior knowledge and two types of learning

Their second factor consists of the *prior knowledge* of the agent, which allows for a distinction between *inductive learning* (where a general rule is learned from specific examples) and *deductive learning* (where a specialized rule is logically deduced from a more general one in order to speed up processing). This distinction is noted in Definition 5.1, but can be incorporated by assuming that the experience E contains both a set of input-output examples (for inductive learning) and background knowledge (for deductive learning) [295, Chapter 12]. We will ignore the role of background knowledge for now and put our focus exclusively on inductive learning. We will reconsider background knowledge and deductive learning in Chapter 7 in the context of learning under knowledge-based constraints.

Representation of the problem and its solution

Thirdly, Russell and Norvig consider the type of *representation* being used for solving the problem (e.g., first order logic) as an important factor. Also Mitchell does consider the task representation, however, not as part of the machine learning problem itself, but rather as a part of the strategy for solving such a problem. We will consider the task representation in more detail in Section 5.1.1.

Type of feedback

Finally, the type of *feedback* available to the agent is the fourth factor proposed by Russell and Norvig, and corresponds roughly to Mitchell's experience E , which will be covered in more detail in Section 5.1.2.

In the following, we will take a closer look at the individual components of a machine learning problem, referring to the definitions by

Mitchell [295, Chapter 1] and by Russell and Norvig [341, Chapter 18] as introduced above. In Section 5.1.1, we take a closer look at the task itself and at the representation being used in modeling this task. In Section 5.1.2, we then consider the different types of experiences and ways to evaluate the performance of a given model. After having covered the main framework conditions, we then take a general look at how to estimate a model's free parameters in Section 5.1.3, before commenting on practical issues (such as the so-called curse of dimensionality) in Section 5.1.4.

5.1.1 Task and Model

In its most general form, the *task* T is usually formulated in a relatively abstract way, e.g., "playing checkers" or "classifying objects in images". In order to operationalize such a formulation, one usually defines a particular *target function* f which needs to be learned by the system [295, Chapter 1]. In general, this target function specifies how the system should map inputs to outputs.

Operationalizing the task with a target function

The input to the system is usually assumed to be represented by a so-called *feature vector* $\vec{x} \in \mathbb{R}^n$, where each entry x_i represents the value on one feature describing the example [172, Chapter 5]. A feature is in this context any measurable property of the object to be processed. For example, in the context of playing checkers, one feature may measure the number of black pieces on the board, while for classifying objects in images, features may relate to the width or color of a detected object. Feature values can be binary, categorical, natural numbers, or real numbers. In the following, we assume that all features can be represented as real values, unless explicitly stated otherwise. If we equate the individual features with individual dimensions of a conceptual space, one can see that each feature vector $\vec{x} \in \mathbb{R}^n$ corresponds to a point $x \in CS$ in an n -dimensional conceptual space (cf. Section 1.2.5).¹

Feature vectors and feature values

Different types of target functions f give rise to different types of machine learning tasks, including the following [172, Chapter 5]:

Different types of machine learning tasks

- *Classification*: The input \vec{x} needs to be assigned to one of c classes, i.e., $f : \mathbb{R}^n \rightarrow \{1, \dots, c\}$. In an alternative formulation, f can be also defined as outputting a probability distribution over the different classes. If only two classes are considered, this is called *binary* classification. In the context of conceptual spaces, classification corresponds to a partitioning of the space into conceptual regions.

¹ In Part I of this dissertation, we considered an observation as a *point* x , while in the machine learning literature, an observation is considered to be a vector \vec{x} pointing from the origin to the point x . We will adopt the latter notation in the machine learning context. Moreover, we will try to ensure that other notational conventions from Part I (such as n denoting the number of dimensions) are preserved in order to remain as consistent as possible.

- *Regression*: A numerical value needs to be predicted for the input \vec{x} , i.e., $f : \mathbb{R}^n \rightarrow \mathbb{R}$. If multiple numerical values need to be predicted, this can be broken down into multiple simple regression problems.
- *Density Estimation*: The probability of observing a given input \vec{x} must be predicted, i.e., $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is a probability density function over the feature space.

Model, parameters, and hypothesis space

In order to learn a good approximation of the target function f , the system needs some way of representing it, i.e., a *model*. For instance, in a regression task, one can decide to model f as a polynomial function of \vec{x} with degree q , i.e., $\hat{f}(x) = b + \sum_{p=1}^q \vec{w}^{(p)} \cdot \vec{x}^p$, where $\vec{w}^{(p)} = (w_1^{(p)}, \dots, w_n^{(p)})$ is a vector of weights, $\vec{x}^p = (x_1^p, \dots, x_n^p)$ is the element-wise application of the p th power to \vec{x} , and $\vec{w}^{(p)} \cdot \vec{x}^p = \sum_{i=1}^n w_i^{(p)} x_i^p$ is the inner product of the vectors $\vec{w}^{(p)}$ and \vec{x}^p . Learning the task T then reduces to estimating the *free parameters* $\theta = \{b, w_1^{(1)}, \dots, w_n^{(q)}\}$ of this polynomial function (i.e., the intercept b and all entries of the different weight vectors $\vec{w}^{(1)}, \dots, \vec{w}^{(q)}$). One fixed set of parameters is also called a *hypothesis* h . If a model contains $|\theta|$ parameters, this can be visualized as a $|\theta|$ -dimensional parameter space (or *hypothesis space* H). Each hypothesis then corresponds to a single point in this hypothesis space. Optimizing the free parameters θ can then be interpreted as a search in this hypothesis space for the optimal hypothesis (as measured by the performance measure P , see Section 5.1.2).

Choosing an appropriate model: expressiveness vs. simplicity

Choosing an appropriate model \hat{f} for representing f involves optimizing a trade-off between expressiveness (which is needed in order to find a good approximation) and simplicity (which usually makes the learning process easier, since it corresponds to a smaller amount of parameters) [295, Chapter 1]. Moreover, simpler representations tend to make the evaluation of the model on novel data faster [341, Chapter 18]. Finally, if the representation is computationally too expressive (e.g., using all possible Turing machines), then finding a good solution within this hypothesis space can be prohibitively complex [341, Chapter 18].

Model capacity and hyperparameters

A more formal way of capturing expressiveness vs. simplicity is given by the *capacity* of a model, which can be defined as its ability to fit a wide variety of functions [172, Chapter 5]. Essentially, a model's capacity corresponds to the size of its hypothesis space, i.e., the number of free parameters. The capacity of a model is on the one hand influenced by the size of the feature space \mathbb{R}^n , i.e., by the number n of features used to describe an example \vec{x} . For instance, when considering our polynomial model from above, each weight vector $\vec{w}^{(p)}$ needs to have exactly as many entries as the feature vector \vec{x} . Adding one additional feature to the representation of \vec{x} thus adds another entry to each $\vec{w}^{(p)}$, thus increasing the number of free parameters and hence the capacity of the model. On the other hand, the capacity of a model can be partially controlled through its so-called *hyperparameters*. In our example from above, the degree q of the polynomial is the main hyperparameter of our model. It is not being optimized by the system itself, but needs

to be specified by the experimenter. This hyperparameter has a direct influence on the number of weight vectors $\vec{w}^{(p)}$. Hence, increasing q results in a larger number of free parameters and therefore in an increased capacity of the model.

Every model comes with an *inductive bias*, which is the set of (potentially tacit) assumptions that together with the training data deductively justifies the classifications of novel instances [295, Chapter 3]. One can distinguish a *restriction bias* (which puts a hard limit on the hypothesis space of the model) from a *preference bias* (which induces a preference relation over different hypotheses in a given hypothesis space). In the example of the polynomial function, a restriction bias would put a hard limit on the degree of the polynomial, e.g., by setting $q = 1$ and thus only allowing for linear models. A preference bias on the other hand would in principle allow arbitrary values for q in the model, while ensuring during learning that low values of q are preferred. In general, a preference bias is often the better choice since using a restriction bias comes with the risk of excluding the correct hypothesis from the hypothesis space [295, Chapter 3]. We will return to this topic in Section 5.1.3 in the context of regularization.

*Inductive bias:
restriction bias and
preference bias*

5.1.2 Experience and Evaluation

Let us now consider the second component of Definition 5.1, namely, the *experience* E . Mitchell [295, Chapter 1] distinguishes among different degrees of control which the learning system has about this experience: There can be no control (if a fixed set of examples is used), full control (e.g., when the system plays games against itself), or an intermediate form (where the system can ask for feedback on self-selected examples). In the following, we will only consider the first case, namely a fixed *dataset* of examples. We will consider a case of intermediate control in Chapter 7 in the context of language games. A dataset can in general be either unlabeled or labeled.

*Experience and
different degrees of
control*

Unlabeled datasets consist of a set of examples $N \vec{x}^{(j)}$ (each represented as an n -dimensional feature vector). The overall dataset can be described by a so-called *design matrix* $X \in \mathbb{R}^{N \times n}$, i.e., a table where each of the N rows represents one example $\vec{x}^{(j)}$ and where each of the n columns represents one feature. Unlabeled datasets are usually used for *unsupervised* learning tasks such as density estimation (where a probability distribution over the feature space needs to be learned, cf. Section 5.1.1) or clustering (where the overall dataset needs to be divided into groups of similar examples). A common characteristic of such unsupervised machine learning task is that they do not require any explicit output examples for the target function f .

*Unlabeled datasets
for unsupervised
learning*

In a *labeled dataset*, each example $\vec{x}^{(j)}$ is also annotated with a label $y^{(j)}$, i.e., the expected value of the target function f . The design matrix X is thus accompanied by a vector of labels \vec{y} . Labeled datasets are typically used for *supervised* learning tasks, where complete input-output pairs

*Labeled datasets for
supervised learning*

		Actual Class	
		TRUE	FALSE
Predicted Class	TRUE	True Positive (TP)	False Positive (FP)
	FALSE	False Negative (FN)	True Negative (TN)

Table 5.1: Confusion matrix for a binary classification task

$(\vec{x}^{(j)}, y^{(j)})$ for the target function f are expected. Supervised learning includes classification and regression tasks, where feature vectors need to be mapped onto pre-defined classes or pre-defined numerical values, respectively (cf. Section 5.1.1). In the intermediate case where only some, but not all data points are labeled, one talks about *semi-supervised learning* [341, Chapter 18]. In the following, we will put our focus on supervised learning with labeled datasets in the form of regression and binary classification. We will reconsider machine learning with unlabeled datasets again in Chapters 6 (in the form of autoencoders) and 7 (in the form of clustering algorithms).

Representativeness

For any type of dataset, it is important to ensure that the examples contained in the design matrix accurately reflect the underlying data distribution of the domain of interest [295, Chapter 1]. This means that the selected examples should be representative of the actual examples one can expect to encounter in practice. This may for instance concern the frequency of different classes in a classification setting or more broadly the distribution of individual feature values and their combinations. If a given dataset is not representative of the actual task or domain, then the function \hat{f} learned by the model (through optimizing its parameters θ) may end up being a relatively poor approximation of the true target function f .

Performance measure: comparing \hat{f} to f

Let us now turn to the *performance measure* P , which can also be called an *evaluation metric*. It measures how well a given model performs a given task T . Since the task T is usually formulated mathematically as a target function f , the performance measure typically (i.e., in supervised setting with labeled datasets) compares the output of the function \hat{f} learned by the model to the output of the actual target function f . This is usually done based on examples $(\vec{x}^{(j)}, y^{(j)})$ from the dataset, comparing $\hat{f}(\vec{x}^{(j)})$ to the ground truth label $y^{(j)} = f(\vec{x}^{(j)})$. One should note that perfect performance is often not feasible in practice, and that one should therefore determine the desired performance level based on the underlying application scenario [172, Chapter 11].

Binary classification: confusion matrix and accuracy

Let us first consider *binary classification* tasks. Overall, there are four different cases that one needs to distinguish (visualized in Table 5.1 as a so-called *confusion matrix*): If the prediction and the actual class are identical, we speak of a "true positive" (TP ; both the prediction and the the ground truth label are TRUE) or a "true negative" (TN ; both the prediction and the ground truth label are FALSE). In case of a mismatch, one can distinguish a "false positive" (FP ; prediction is TRUE, but label is FALSE) from a "false negative" (FN ; prediction is FALSE, but label is TRUE).

In order to derive a single performance metric, one can then compute the *accuracy* Acc of the model as the proportion of examples for which \hat{f} produces the same output as f :

$$Acc = \frac{TP + TN}{TP + FP + FN + TN}$$

Equivalently, one can measure the *error rate* Err , which measures the proportion of examples on which \hat{f} and f disagree:

$$Err = \frac{FP + FN}{TP + FP + FN + TN}$$

Error rate as counterpart of accuracy

Based on their definitions, one can easily see that $Acc, Err \in [0, 1]$ and that $Acc + Err = 1$. While accuracy can give a first intuition about the model's usefulness ("how often is the prediction correct?"), the error rate is mainly used to analyze improvements over previous models ("how many remaining misclassifications were we able to remove?").

Unfortunately, both accuracy and error rate can be misleading on imbalanced datasets, where one of the classes occurs much more frequently than the other: If we assume that 90% of the examples are labeled as `TRUE`, then also a model that completely disregards its input and always predicts `TRUE` obtains an accuracy of 90%. Despite this relatively high value, the model is likely to be useless in practice. Therefore, several other evaluation metrics have been defined (again based on the confusion matrix), which aim to be informative also on imbalanced datasets. Here, we will consider *Cohen's κ* [108], which can be interpreted as a normalized version of accuracy that takes into account the *probability p_e of random agreement* between the prediction and the ground truth:

Cohen's κ for imbalanced data

$$p_e = \mathbb{P}(\text{predicted} = \text{TRUE}) \cdot \mathbb{P}(\text{actual} = \text{TRUE}) \\ + \mathbb{P}(\text{predicted} = \text{FALSE}) \cdot \mathbb{P}(\text{actual} = \text{FALSE})$$

Here, the probabilities $\mathbb{P}(\text{actual} = \text{TRUE})$ and $\mathbb{P}(\text{actual} = \text{FALSE})$ can be derived by simply counting the distribution of the labels in the dataset. Analogously, $\mathbb{P}(\text{predicted} = \text{TRUE})$ and $\mathbb{P}(\text{predicted} = \text{FALSE})$ can be calculated for the outputs of the model on the given dataset. In the computation of p_e , we thus pretend that both the model \hat{f} and the target function f completely ignore their inputs and that they make their predictions solely based on class probabilities. Thus, p_e measures the level of agreement one would expect even in this hypothetical case. Cohen's κ now adjusts the accuracy Acc based on p_e :

Computing the probability of random agreement

$$\kappa = \frac{Acc - p_e}{1 - p_e}$$

One can thus think of Cohen's κ as measuring to what extent the accuracy of the model is higher than what we would expect by chance. Cohen's κ is confined to the interval $[-1, 1]$, with 1 representing perfect agreement between model and ground truth, 0 only random agreement,

Interpretation of Cohen's κ

and -1 perfect disagreement. In the example with an imbalanced dataset containing 90% examples from the positive class, the probability p_e of random agreement of an "always TRUE" classifier equals its accuracy (since it is indeed based only on class probabilities). Hence, Cohen's κ is zero in this case. We will return to Cohen's κ in Chapters 8 and 10, where we use it to evaluate the quality of interpretable directions in a conceptual space.

Baseline classifiers

As we have seen in the example with the imbalanced dataset, the choice of the evaluation metric can be quite important. In our discussion from above, we have considered a model that always returns TRUE. This is an example of a so-called *baseline* classifier – a very simplistic model, that may even disregard its input altogether. The point of using such baselines is to establish a lower performance bound, to which the model under consideration can be compared. Simple classification baselines include models with a fixed output (such as "always TRUE" and "always FALSE"), models with a probabilistic output (such as predicting both classes with equal probability, or based on their frequency in the dataset), and simplified versions of the actual model (i.e., with a very small capacity, either due to limited access to features, or due to respective hyperparameter settings).

*Regression task:
mean squared error*

In a *regression* setting, the desired output is usually a real number. It is thus quite unlikely that \hat{f} and f perfectly agree on the output for any example \vec{x} . Therefore, classification-based evaluation metrics are not directly applicable. In order to quantify the error of a prediction, one usually measures the numeric difference between $\hat{f}(\vec{x})$ and $f(\vec{x})$. This can be done by using the *mean squared error* (MSE), i.e., the average value of the squared differences between prediction and ground truth for all examples from the dataset:

$$MSE = \frac{1}{N} \sum_{j=1}^N \left(y^{(j)} - \hat{f}(\vec{x}^{(j)}) \right)^2$$

Optimal regression performance corresponds to a minimal value of the MSE, which is zero for a perfect prediction. Since the prediction can be arbitrarily far away from the ground truth, the MSE is not bounded from above, hence, $MSE \in [0, \infty)$. We will use the MSE in Chapters 9 and 12 to measure the quality of a regression from images to points in a psychological similarity space.

*Coefficient of
determination*

Another popular performance measure in regression tasks is the *coefficient of determination* R^2 [439]. It can be defined as follows [155, Section 7.2.3], where \bar{y} denotes the average across all $y^{(j)}$:

$$R^2 = 1 - \frac{S_{residual}}{S_{total}} = \frac{S_{total} - S_{residual}}{S_{total}}$$

with $S_{residual} = \sum_{j=1}^N (y^{(j)} - \hat{f}(\vec{x}^{(j)}))^2$

$$\text{and } S_{total} = \sum_{j=1}^N (y^{(j)} - \bar{y})^2$$

Here, S_{total} is the *total sum of squares* and measures the overall variance in the ground truth data by quantifying how much the target varies around its mean. $S_{residual}$ is called the *residual sum of squares* and measures how far the model's predictions depart from the ground truth, essentially using the MSE. The coefficient of determination can be interpreted as the amount of variance explained by the model ($S_{total} - S_{residual}$, i.e., the overall variance minus the remaining errors of the model) by the overall amount of variance found in the data (S_{total}). Similarly to Cohen's κ (which can be viewed as a normalized version of classification accuracy), we can interpret the coefficient of determination R^2 as a normalized version of the MSE.

How R^2 is computed

The highest possible value of R^2 is one and can only be achieved if $S_{residual}$ (i.e., the MSE) becomes zero. For a regression error larger than the variance of the ground truth (i.e., $S_{residual} > S_{total}$), R^2 becomes negative. Since the regression error can become arbitrarily large, the possible values of R^2 are not bounded from below, hence $R^2 \in (-\infty, 1]$. Please note that $R^2 = 0$ only happens if $S_{residual} = S_{total}$, i.e., if the model's prediction error is identical to the overall variance in the targets. We will use the coefficient of determination in Chapter 9 for evaluating the correlation between psychological dissimilarities and distances in a similarity space, and in Chapters 9 and 12 for quantifying the quality of a regression from images to points in this similarity space.

Values of R^2

Also for regression tasks, it makes sense to compare the model's performance to some simple baselines. These may again follow a fixed strategy (e.g., always predicting the mean of the targets, or another fixed value such as zero), a probabilistic strategy (for instance based on a normal distribution), or may use a simplified model with low capacity.

Baseline regressors

In machine learning, one is mainly interested in a model which performs well on previously unseen inputs, i.e., which is able to generalize well. The *generalization error* (or *test error*) of a model can be defined as the expected error on novel, previously unseen inputs. It needs to be distinguished from the *training error*, which is measured on the *training set*, i.e., the dataset which was used to estimate its free parameters [172, Chapter 5].² In practice, the generalization error can not be measured, but only be approximated by evaluating the model's performance on a *test set*, i.e., a separate set of examples which were not used during training. In order for this approximation to be meaningful, one has to assume that the examples in the two datasets are independent from each other, and that both datasets follow an identical probability distribution. These assumptions are often summarized as *i.i.d. assumptions*, which is an abbreviation for "independent and identically distributed" [172, Chapter 5]. Moreover, most machine learning problems tacitly include

Generalization error vs. training error

² Alternatively, one can also use the terms *generalization performance* and *training performance*.

the *stationarity assumption*, which states that the probability distribution of the data does not change over time [341, Chapter 18].

Generalization as
difference to
optimization

Goodfellow et al. [172, Chapter 8] note that this distinction into generalization error and training error is what separates machine learning from pure optimization: In an optimization task, minimizing a given cost function on a given dataset is the main task, and generalization to novel inputs is not considered at all. In machine learning, on the other hand, good generalization capabilities are the main goal, while optimizing performance on the training set is rather a means to an end. We will touch upon this distinction again in Chapter 8 in the context of the optimization technique of multidimensional scaling.

Underfitting and
overfitting

Two factors are very important for performing well on previously unseen examples [172, Chapter 5]: On the one hand, the model needs to achieve good performance on the training set. If this is not the case (e.g., because the model's capacity is too low for the given task), one speaks of *underfitting*. On the other hand, the difference between training performance and generalization performance should be small. If this is not the case, one speaks of *overfitting*, which corresponds to the model memorizing the examples from the training set, but being unable to generalize successfully. One can interpret this as the model picking up random patterns from the training set, which are not predictive on new examples from the test set. In general, the training error decreases with an increase in model capacity, until it asymptotes to the minimal possible error. The generalization error on the other hand tends to be a U-shaped function of model capacity [172, Chapter 5], based on underfitting issues for low model capacity and overfitting issues for high model capacity. As a rule of thumb, overfitting becomes more likely as the hypothesis space grows, and becomes less likely as the number of training examples grows [341, Chapter 18]. We will return to this trade-off in Section 5.1.4 when discussing the curse of dimensionality.

Model selection for
optimizing
hyperparameter
settings

The hyperparameters of a model (as in our example from Section 5.1.1 the degree q of the polynomial function) have a direct influence on its capacity and hence on its generalization error. One therefore often follows a so-called *wrapper method for model selection* [341, Chapter 18] by training the model with various hyperparameter settings and choosing the variant with the best performance. It is important to note that the performance of different hyperparameter settings should be evaluated neither on the training set nor on the test set [172, Chapter 5]: Optimizing hyperparameters on the training set (i.e., by measuring training performance) usually leads to settings with maximal capacity, since a model with a higher capacity can fit the training examples better. Hence, optimizing hyperparameters on the training set increases the risk of overfitting. On the other hand, if hyperparameters are optimized on the test set (i.e., by measuring test performance), then the model can "peek" at the test set [341, Chapter 18]: Examples from the test set are used to set the model's hyperparameters, which in turn indirectly influence the parameters of the model. Performance on the test set is then no longer be an accurate estimate of the model's generalization

performance, since the examples from the test set have influenced the estimation of the model's parameters and can thus not be considered to be novel to the overall system.

Therefore, a third dataset called the *validation set* is used to compare different hyperparameter settings. It should again be disjoint from both the training set and the test set and follow the same underlying distribution. The typical machine learning workflow is then to train multiple variants of the same model (by using different hyperparameter settings) on the training set, and to estimate their generalization error on the validation set. This validation set performance is then used to select the optimal hyperparameter configuration. Only when the overall training process is done, the test set is used to estimate the generalization error of the resulting model. In the example of the polynomial model from Section 5.1.1, we would use the training set to find optimal parameters b and $\vec{w}^{(p)}$. This can be done for different values of q , i.e., different variants of the polynomial model. We can then compare their performance on the validation set in order to decide which value of q we should use. The expected performance on previously unseen examples can then be computed on the test set. One should note that the validation set error is typically lower than the test set error since it is explicitly being minimized during hyperparameter optimization [172, Chapter 5].

The search for optimal hyperparameters should be conducted in a structured way in order to make the results reproducible. There are several approaches that can be taken [57, 154]: When conducting a *grid search*, one defines several possible values for each hyperparameter and evaluates the system for each possible combination of these values. This can be used to create an equally spaced grid of hyperparameter configurations which is easy to evaluate and to interpret. When choosing candidate values for real-valued hyperparameters, it is a common practice to pick them on an approximately logarithmic scale with the scale ends being chosen conservatively [172, Chapter 11]. A *random search* on the other hand randomly samples hyperparameter configurations based on predefined marginal distributions for the individual hyperparameters. A random search can give better results if some hyperparameters are much more important than others, but since the hyperparameter configurations are sampled in a less systematic way, it can be harder to interpret. Both grid search and random search can be used in an iterative way in order to focus on regions of the hyperparameter space, which have yielded promising results in prior iterations [172, Chapter 11]. Finally, there are more advanced methods such as genetic algorithms or Bayesian optimization, which are more powerful optimizers, but come at increased computational cost.

As argued above, in a typical machine learning setting, we need to split the overall dataset into two or three parts in order to estimate the generalization error and (if necessary) to optimize hyperparameters. There exist different evaluation schemata for making such a split:

Validation set for model selection

Model selection approaches: grid search and random search

How to split the dataset

Holdout cross-validation: a single fixed split

The most straightforward procedure is called *holdout cross-validation* [341, Chapter 18], where the overall dataset is randomly split into two or three disjoint parts, before any model is trained. This partitioning is then kept fixed for the whole machine learning process. A so-called *stratified split* ensures that the distribution of class labels or regression targets is approximately the same in all parts. Typical splits are 80-20 or 70-30 (if only considering training and test set) and 60-20-20, 70-15-15, or 80-10-10 (if considering training, validation, and test set). As one can see, there is no ideal split that has been agreed upon in the literature. This is based on the trade off between a good estimation of validation and test error (requiring relatively large validation and test sets, but leading to a relatively small training set), and reducing the risk of overfitting (requiring a large training set for models with higher capacity, and leading thus to smaller validation and test sets) [341, Chapter 18].

k-fold cross-validation: rotating through the data

Using a holdout cross-validation to split the overall dataset into three parts may, however, not be feasible if the overall number of training examples is very small. In such cases, one can use a technique called *k-fold cross-validation* [172, Chapter 5]. In this scheme, the overall dataset is partitioned into k non-overlapping subsets. In practice, often five or ten of these so-called *folds* are used [341, Chapter 18]. In the i th trial, the i th subset is used for testing while all other subsets are used for training the model. The overall generalization error is then estimated by averaging over the test set error of all trials. While allowing to make a reasonably confident estimate of the generalization error on small datasets, this cross-validation scheme comes at increased computational cost, since the model parameters need to be estimated k times instead of only once.

Leave-one-out: only a single data point for testing

In the extreme case, if the number of folds equals the number of data points, this is called a *leave-one-out evaluation*, which is applicable even to very small datasets, but computationally very costly. The leave-one-out error is furthermore known to be a high variance estimator of the true generalization error, which also tends to give overly optimistic results [178]. The results of a leave-one-out evaluation should thus be interpreted with caution.

5.1.3 Fitting the Model's Parameters

*Finding optimal parameters θ^**

So far, we have only described how the different ingredients for describing a machine learning problem can be defined, and how one can model the target function. Let us now discuss how the optimal parameters θ^* of the model can be found based on the training examples and the performance measure.

Loss: performance measure and regularizers

In general, the best fit of the model to the data is found by minimizing an appropriate *error function* (also called *cost function* or *loss*) [295, Chapter 1]. This loss J is closely related to the performance measure P , but often not identical to it. Typically, the loss consists of different additive terms where at least one term refers to the performance measure P (e.g., the MSE in a regression task), while other terms

introduce additional constraints. These constraints often aim to improve generalization and are called *regularizers*, since they incite the selection of a "more regular" or less complex model [341, Chapter 18]. Using regularization can thus be seen as a preference bias (cf. Section 5.1.1), which tries to choose the simplest hypothesis consistent with the data [341, Chapter 18]. This can be linked to the philosophical principle of *Occam's razor*, which postulates that the solution or explanation with the fewest underlying assumptions is preferable [295, Chapter 3]. In general, regularization can be defined as constraining the learning algorithm in such a way that the difference between training and test error is reduced, while the training error is basically left unchanged [172, Chapter 5]. Regularization thus tries to constrain the model's capacity in a soft way in order to avoid overfitting problems (cf. Section 5.1.2). Since one needs to find an appropriate trade-off between the different terms in the loss function, they are typically weighted against each other. These weights can be interpreted as additional hyperparameters of the model.

When considering our polynomial regression model from Section 5.1.1, we can for example add a so-called *weight decay* regularizer, which puts a penalty on large weights. When optimizing the weights of the model, we thus do not only maximize the performance measure P (i.e., the mean squared error), but we also try to minimize the magnitude of the model's weights. In other words, the regularizer we introduced gives a preference to some solutions from the hypothesis space (namely, the ones with small weights) over others (namely, the ones with large weights). The strength of this preference can be controlled with the aforementioned weights of the two terms in the loss function.

Instead of using a regularizer, one can also limit the model's capacity by performing dimensionality reduction, i.e., compressing the original feature vectors into a more compact format [341, Chapter 18]. We will introduce different dimensionality reduction techniques in Section 5.3. If the model's capacity can be directly controlled through a hyperparameter (as in the example of the polynomial regression with the degree q of the polynomial), constraining the possible values of this hyperparameter can offer another way of preferring simpler hypotheses. Capacity restrictions enforced through dimensionality reduction or by constraining hyperparameter values are often restriction biases, while regularization is a preference bias (cf. Section 5.1.1).

In general, the optimal parameters θ^* of a model can be found by minimizing the loss function J . For some models (such as a linear regression), this minimization can be solved analytically. For more complex models (such as artificial neural networks), finding an analytical solution is, however, not easily possible. In many cases, one thus resorts to iterative optimization methods, which improve their solution step by step. For neural networks, typically a variant of gradient descent is used (which will be introduced in Section 5.2.2). Other machine learning algorithms (such as for example decision trees), however, come with special-case optimizers, since general optimization strategies are not easily applicable [172, Chapter 5]. Please recall that the model param-

Example: polynomial regression

Reducing model capacity through dimensionality reduction and hyperparameters

Minimizing the loss function

eters are always optimized on the training set, i.e., they are chosen in such a way that they minimize the loss function on the examples from the training set. The validation and the test set are then used to estimate the generalization performance of the model according to the performance measure P (cf. Section 5.1.2).

Maximum likelihood for selecting model parameters

When using a probabilistic approach for modeling the machine learning problem, one is typically interested in estimating the underlying probability distribution $\mathbb{P}(\vec{x})$ (or $\mathbb{P}(y|\vec{x})$ in the supervised case). The *maximum likelihood estimator* for the model parameters θ can then be defined as follows, based on the design matrix X [172, Chapter 5]:

$$\theta_{ML} = \arg \max_{\theta} \mathbb{P}_{model}(X|\theta) = \arg \max_{\theta} \prod_{j=1}^N \mathbb{P}_{model}(\vec{x}^{(j)}|\theta) \quad (5.1)$$

Expected value of empirical distribution

As one can see in Equation 5.1, we try to find the set of model parameters θ^* , which maximize the likelihood of the observed data from the training set, given the model – hence the term "maximum likelihood"). If we consider the empirical distribution \mathbb{P}_{data} of the training set instead of explicitly iterating through the examples from the design matrix, we can replace the product over the individual examples by the expected value under the empirical distribution. We then arrive at the following notation, where we also take the logarithm³ for mathematical convenience:

$$\begin{aligned} \theta_{ML} &= \arg \max_{\theta} \mathbb{E}_{\vec{x} \sim \mathbb{P}_{data}} \log \mathbb{P}_{model}(\vec{x}|\theta) \\ &= \arg \min_{\theta} -\mathbb{E}_{\vec{x} \sim \mathbb{P}_{data}} \log \mathbb{P}_{model}(\vec{x}|\theta) \end{aligned} \quad (5.2)$$

Cross-entropy and KL divergence

One can think of Equation 5.1 as an approximation of Equation 5.2 – optimizing Equation 5.1 thus is a practical proxy for optimizing 5.2. The term being minimized in Equation 5.2 is often referred to as the *cross-entropy* $H(\mathbb{P}_{data}, \mathbb{P}_{model})$ [172, Section 3.13]. One can easily see that minimizing the cross-entropy is equivalent to minimizing the so-called *KL divergence* $D_{KL}(\mathbb{P}_{data}||\mathbb{P}_{model}) = \mathbb{E}_{\vec{x} \sim \mathbb{P}_{data}} [\log \mathbb{P}_{data}(\vec{x}) - \log \mathbb{P}_{model}(\vec{x}|\theta)]$, which measures the difference between the two probability distributions [172, Chapter 5]. This means that by optimizing the model parameters based on Equation 5.1, we try to minimize the difference between \mathbb{P}_{model} and \mathbb{P}_{data} , i.e., adapt the model's predictions as closely as possible to the actual data distribution.

Maximizing the posterior probability

Instead of choosing θ by maximizing the likelihood $\mathbb{P}(\vec{x}|\theta)$ of the data given the parameters, one can maximize the posterior probability $\mathbb{P}(\theta|\vec{x})$ of the parameters given the data, using Bayes rule $\mathbb{P}(\theta|\vec{x}) = \frac{\mathbb{P}(\vec{x}|\theta) \cdot \mathbb{P}(\theta)}{\mathbb{P}(\vec{x})}$ [172, Chapter 5]:

$$\theta_{MAP} = \arg \max_{\theta} \mathbb{P}(\theta|X) = \arg \max_{\theta} (\mathbb{P}(X|\theta) \cdot \mathbb{P}(\theta))$$

The resulting *maximum a posteriori estimator* can be seen as the hypothesis which is most probable given the data, while also incorporating

³ Please note that we use log to refer to the binary logarithm unless stated otherwise.

prior knowledge about the parameters θ in the form of a prior distribution $\mathbb{P}(\theta)$. This prior distribution is usually quite broad, but may incorporate a preference for "simpler" solutions [172, Chapter 5]. This can be seen as a form of regularization similar to the weight decay introduced above.

One can use the maximum a posteriori estimator to derive the *minimum description length principle*, which can be interpreted as a Bayesian version of Occam's razor [295, Chapter 6]. In the following we use $h \in H$ to denote the hypothesis corresponding to the parameter values θ (cf. Section 5.1.1):

*Minimum
description length*

$$\begin{aligned} h_{MAP} &= \arg \max_h (\mathbb{P}(X|h) \cdot \mathbb{P}(h)) \\ &= \arg \max_h (\log(\mathbb{P}(X|h)) + \log(\mathbb{P}(h))) \\ &= \arg \min_h (-\log(\mathbb{P}(X|h)) - \log(\mathbb{P}(h))) \end{aligned}$$

Here, $-\log(\mathbb{P}(h))$ can be interpreted as the description length of the hypothesis h if an optimal encoding for the hypothesis space H is used. Similarly, $-\log(\mathbb{P}(X|h))$ can be viewed as the description length of the data X under an optimal encoding, assuming that the hypothesis h is known. Given an optimal encoding, we can thus find the maximum a posteriori hypothesis by minimizing both the description length of the hypothesis (preferring short hypotheses over longer ones) and the description length of the data given the hypotheses (preferring hypotheses with few misclassifications over inaccurate ones). This can be seen as a Bayesian version of Occam's razor as discussed above, which aims to select the shortest hypothesis that is able to explain the data. However, since the optimal encodings for both hypotheses and data are often not known in practice, following the minimum description length principle for an arbitrary encoding does not necessarily lead to a maximum a posteriori hypothesis [295, Chapter 6].

This section has introduced some general notions and concepts for optimizing model parameters. They will partially resurface in Section 5.2 in the context of various machine learning models. Especially the probabilistic view will play a role again in Chapter 6 for the mathematical background of artificial neural networks.

5.1.4 Practical Considerations

After having covered the main ingredients to specifying a machine learning problem (namely, task T , experience E , and performance measure P) as well as a combination of model and parameter fitting for solving this problem, we would now like to discuss the relation between model capacity, the number of training examples, and the generalization error.

*Relation between
capacity, training
data, and
generalization*

*Smoothness prior
and curse of
dimensionality*

Most machine learning algorithms generalize to unseen examples by making use of the so-called *smoothness prior*, which assumes that $f(\vec{x} + \vec{\epsilon}) \approx f(\vec{x})$, i.e., that the target function does not change much within a small region [172, Chapter 5]. Generalizing to unseen examples then essentially corresponds to interpolation and extrapolation based on the examples from the training set. However, this requires that for each newly observed instance \vec{x} in the feature space, we have observed a sufficient amount of training examples that are located in the same region. If the feature space has n dimensions and we need to distinguish v different values along each of these dimensions, one can easily see that overall v^n regions need to be distinguished from each other (cf. Figure 5.1). If we would like to cover each of these regions with at least one data point, we need a training set with $O(v^n)$ examples.⁴ If the dataset contains many features (i.e., if n is very large), then the amount of training examples needed to adequately cover this feature space grows unfeasibly large very quickly. This effect is the so-called *curse of dimensionality* [172, Chapter 5].

*Distance of closest
neighbors*

Another way of viewing the curse of dimensionality is provided by Russell and Norvig [341, Section 18.8]. They assume that a set of N points is uniformly distributed in an n -dimensional unit hypercube. They furthermore define l to be the average side length of the smallest hypercube containing the k nearest neighbors for a given query point. The size of a neighborhood (containing k points) is l^n , while the size of the unit hypercube (containing all N points) is 1. On average, one can thus expect that $l^n = \frac{k}{N}$. Solving for l yields $l = \left(\frac{k}{N}\right)^{\frac{1}{n}}$. Considering $N = 1,000,000$ data points and $k = 10$ neighbors, Russell and Norvig point out that for $n = 3$ the average side length equals $l \approx 0.02$, but for $n = 200$, it already reaches a value of $l \approx 0.94$. This means that in high-dimensional spaces, the nearest neighbors of a given query point tend to be quite far away, making a successful generalization based on interpolation or extrapolation incredibly difficult.

*Fighting the curse of
dimensionality with
more data and fewer
features*

The most straightforward ways of fighting the curse of dimensionality consist in increasing the size of the dataset and reducing the number of features. Since collecting more data is not always feasible in practice (especially in an exponential way), one typically focuses on the latter approach. We will discuss the two main variants of dimensionality reduction in Section 5.3, namely feature selection (which selects a subset of the given features) and feature extraction (which combines the existing features to obtain a smaller number of novel features).

*The curse of
dimensionality and
artificial neural
networks*

In our experimental studies in Part III of this dissertation, we will mainly make use of artificial neural networks, which will be introduced in Chapter 6. We would nevertheless already now like to comment on the ways in which deep learning approaches aim to overcome the curse of dimensionality. Essentially, deep learning assumes that the

⁴ We use the so-called "big O" notation from computer science here to indicate that the number N of training examples needs to grow exponentially with v^n , i.e., $N \approx c_1 \cdot v^n + c_2$ with constant values c_1 and c_2 .

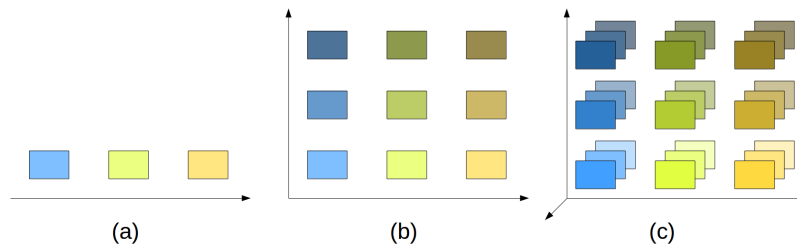


Figure 5.1: Illustration of the curse of dimensionality for three values per feature and one (a), two (b), and three (c) dimensions in the feature space, respectively.

data was generated by a composition of multiple factors, potentially at multiple levels in the hierarchy [172, Chapter 5]. For example, convolutional neural networks (which are commonly used in computer vision) are based on the assumption that many important features are invariant with respect to their exact location in the image: A face can be located in the center of the image or in one of the corners, but the underlying features discriminating it from non-face objects remain unchanged. Moreover, deep learning approaches typically assume that even in very large feature spaces, relevant inputs are found only in a hand full of small connected regions which are called *manifolds*. For example, if images are represented by individual pixel values, most randomly generated points in this feature space appear to humans as meaningless noise when visualized as image. Well-structured images such as photographs of faces seem to occupy only a small part of the overall feature space. By correctly identifying these manifolds, the model can focus on the relevant regions in the feature space while discarding irrelevant inputs. By harnessing underlying regularities such as location invariance and manifolds, deep learning approaches augment the smoothness prior with further priors in order to improve generalization [172, Chapter 5]. We will discuss this in more depth in Chapter 6.

Finally, we would like to focus on additional practical considerations when training machine learning systems. We start with two recommendations by Russell and Norvig [341, Chapter 18], who note that both the size of the dataset and the features being used to describe instances are of utmost importance for successful applications of machine learning: All machine learning algorithms tend to improve their performance with an increased amount of training data, so collecting more data may be more efficient than extensive hyperparameter tuning. Goodfellow et al. [172, Chapter 11] propose to collect more data if the performance on the training set is acceptable, but the generalization to the validation set is poor, since this is a sign of overfitting. Moreover, Russell and Norvig [341, Chapter 18] point out that the hypothesis space of any machine learning model is inherently limited by the feature space it uses – if the given set of features is not capable of making certain distinctions, then no model can be expected to successfully learn these distinctions. The process of *feature engineering* is concerned with obtaining a good

*Practical importance
of data and features*

*Performance measure
as starting point and
incremental changes*

set of expressive and discriminatory features and is often informed by domain knowledge about the specific task at hand.

Goodfellow et al. [172, Chapter 11] recommend to start the overall machine learning process by defining a suitable performance measure P and the desired performance level of the system with respect to this performance measure. They furthermore suggest to establish a working end-to-end pipeline as soon as possible and to repeatedly make incremental changes to this pipeline, while observing their effects on the evaluation results. Based on whether one observes overfitting or underfitting, the next adjustments to the system can be planned and implemented. This can be linked to agile approaches to software engineering such as scrum [405], which also focus on short iterations with regular feedback.

*How to debug
machine learning
systems*

Goodfellow et al. [172, Chapter 11] furthermore discuss different debugging strategies in the context of machine learning. They note that machine learning systems are inherently difficult to debug, since their intended behavior can hardly be specified a priori – otherwise one would not need to apply machine learning and could simply hard-code the respective functionality instead. In order to nevertheless get a sense of the model's behavior, they propose different indirect debugging strategies: Firstly, one can *visualize the model in action*, i.e., look at some example inputs and the corresponding model outputs. This can serve as a first sanity check and also provide some intuition about the overall degree of performance. Moreover, they recommend to *visualize the worst mistakes and most difficult cases*, i.e., to explicitly consider confident misclassifications and examples, for which the model's confidence (as for instance expressed by the class probability) is very low. Goodfellow et al. argue that analyzing these failure cases can give important insights about necessary changes in preprocessing steps such as feature extraction. A third debugging strategy focuses on *fitting a tiny dataset* which consists of only one or a hand full of examples. Since memorizing a simple example should be trivial for any machine learning algorithm, a failure can in this case indicate bugs in the implementation. If only a single training example is used, these bugs are moreover more likely to be found, since it is feasible to manually double-check the intermediate results of the algorithm.

5.2 MACHINE LEARNING ALGORITHMS

After having considered some general notions in the field of machine learning, we are now ready to introduce several popular machine learning algorithms, which will play a role in later chapters of this dissertation. Unless stated otherwise, these algorithms consider the task of binary classification. This section is not intended as a comprehensive overview machine learning algorithms and does for instance not explicitly cover naive Bayes classifiers [295, Chapter 6], because they will not be used in our practical work in Part III of this dissertation.

We start our overview of different machine learning algorithms by considering a *linear least-squares regression* in Section 5.2.1, one of the most straightforward regression models. We also consider two forms of regularization for this simple model, yielding so-called *ridge regression* and *lasso regression* models, respectively. In Section 5.2.2, we show a relatively straightforward way of using a linear model for classification tasks by introducing *logistic regression*. This also involves a first discussion of gradient descent, which will play again a role for training artificial neural networks in Chapter 6. Afterwards, we present *support vector machines* in Section 5.2.3, which also use a linear model, but aim to find a decision boundary that maximizes the distance to the data points from the training set in order to allow for good generalization. We leave the realm of linear models in Section 5.2.4, where we introduce *decision tree* learning, which results in relatively interpretable, rule-like models. In this context, we also briefly introduce *random forests* as an ensemble of decision trees, which are generally considered to be a powerful machine learning algorithm. Our overview of standard machine learning algorithms is concluded in Section 5.2.5 with the *k nearest neighbor* algorithm, which classifies a novel data point based on its closest neighbors in the feature space, and which can therefore be linked to exemplar models from psychology (cf. Section 1.1.1).

Overview

When considering the choice for a specific machine learning algorithm, Russell and Norvig [341, Chapter 18] propose to use decision trees for discrete features, especially if one suspects that many of them may be irrelevant. They argue that a *k* nearest neighbor approach may be superior if large amounts of data but little to no prior knowledge is available and if the number of features is relatively low. In general, they consider support vector machines to be one of the best methods to try first if the dataset is not too large. We would like to add, that in practice, it is often helpful to initially test various algorithms and then to focus on one or two approaches with promising initial results. Thanks to modern machine learning libraries like `scikit-learn`⁵ or `Weka`⁶, this is easily possible without too much implementational overhead.

How to choose a classifier

5.2.1 Linear Regression

Linear regression is one of the most straightforward and most widely applied regression algorithms. It assumes that the real-valued target y can be described as a linear function of the real-valued feature values x_i [341, Section 18.6]. The resulting *linear model* looks as follows:

The linear model

$$\hat{f}(\vec{x}) = w_0 + \sum_{i=1}^n w_i x_i = w_0 + \vec{w} \cdot \vec{x}$$

If we define $x_0 = 1$, we can obtain a more compact representation, which is sometimes used for notational convenience:

⁵ See <https://scikit-learn.org>.

⁶ See <https://www.cs.waikato.ac.nz/ml/weka/>.

$$\hat{f}(\vec{x}) = \sum_{i=0}^n w_i x_i = \vec{w} \cdot \vec{x}$$

One can show that if the scalar targets $y^{(j)}$ are indeed based on a linear combination of the features in $\vec{x}^{(j)}$, but subject to normally distributed noise, then the most likely values for the weights in \vec{w} can be obtained by minimizing the mean squared error [341, Section 18.6], which can in this case be written as follows:

$$MSE = \frac{1}{N} \sum_{j=1}^N \left(y^{(j)} - \hat{f}(\vec{x}^{(j)}) \right)^2 = \frac{1}{N} \sum_{j=1}^N \left(y^{(j)} - \vec{w} \cdot \vec{x}^{(j)} \right)^2$$

Minimizing the MSE

Due to the minimization of the mean squared error, this approach is often also referred to as *least squares* regression. The minimum of the MSE can be obtained by setting its partial derivatives with respect to the individual weights w_i to zero. Since the mean squared error is a convex loss function, it has only a single minimum, which can be found analytically with the following formula [341, Section 18.6]:

$$\vec{w}^* = (X^T X)^{-1} X^T \vec{y}$$

Here, X^T denotes the matrix transpose of the design matrix X and \vec{y} contains all target values for the individual examples (cf. Section 5.1.2).

L_1 and L_2
regularization

If the number n of features is very large, or if the feature space includes irrelevant features, a linear regression might overfit to the training data. In order to counteract such overfitting tendencies, one can add a regularization term to the loss function being optimized (cf. Section 5.1.3). Typical types of regularization for a linear regression are L_1 regularization $\sum_{i=0}^n |w_i|$ and L_2 regularization $\sum_{i=1}^n |w_i|^2$, resulting in so-called *lasso* and *ridge* regressors, respectively. Since these regularizers penalize large weights, they are also referred to as *weight decay* regularizers (cf. Section 5.1.3). While one can still obtain an analytical solution for ridge regression, this is in general not possible for lasso regression, where numerical optimization methods are needed. Gradient descent as one example of such a technique will be introduced in Section 5.2.2.

Lasso regression
leads to sparse
models

Lasso regression tends to result in sparse models, where many weights equal zero, while no such effect can be observed for ridge regression [341, Section 18.6]. A visual explanation for this observation is given in Figure 5.2: Optimizing the overall loss, which is a linear combination of the mean squared error and the regularization term, involves a trade-off between these two terms. Figure 5.2a illustrates the hypothesis space for a linear regression with only a single feature, which consists of the intercept w_0 and the weight w_1 associated with this single feature. Regions of identical MSE values are illustrated as red solid "contour lines", and regions of identical values for the L_1 regularization term as blue dashed contour lines. The regularization term is obviously minimal for $w_1 = w_2 = 0$. Since L_1 regularization

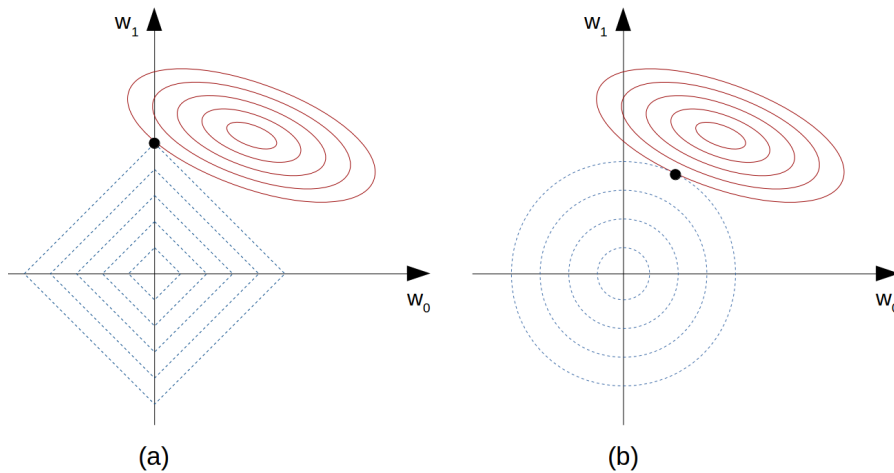


Figure 5.2: Comparison of lasso regression (a) and ridge regression (b) with MSE loss illustrated by solid red lines and regularization term illustrated by dashed blue lines.

is just a sum over the individual weights, the contour lines have the shapes of diamonds.⁷ An optimum of the overall lasso loss function (i.e., the linear combination of MSE and L_1 regularization) is obtained at the point where the contour lines of the MSE term and the L_1 regularization term first intersect. As one can see in Figure 5.2a, this is most likely to happen at the corners of the diamond. In Figure 5.2a, this happens on the w_1 axis, where $w_0 = 0$. A lasso regression is thus likely to find solutions where some weights are set to zero. This essentially means that the corresponding features are discarded. A lasso regression can thus be used as an embedded method for feature selection (which will be introduced more thoroughly in Section 5.3.1).

For the ridge regression, on the other hand, the contour lines have the form of circles (cf. Figure 5.2b). This is based on the fact that the L_2 regularization considers the sum over the *squared* weights.⁸ As one can see in Figure 5.2b, the meeting point of MSE term and regularization term has no particular preference, which generally leads to solutions not lying on any of the axes.

The core difference between L_1 and L_2 regularization is thus that the contour lines of the L_2 regularization are rotationally invariant, while the ones of the L_1 regularization heavily depend on the axes. One can thus say that using L_1 regularization assumes that the individual features themselves are (semantically) important, while the usage of L_2 regularization interprets them as a relatively arbitrary basis for the overall feature space [341, Section 18.6].

Ridge regression does not prefer sparse models

Rotational invariance as key difference

⁷ One could say that L_1 regularization computes the Manhattan distance between the weight vector and the origin, and that the contour lines in Figure 5.2a are thus hyperballs under the Manhattan metric. This has already played a role in Section 4.1.1 in the context of computing the size of a fuzzy conceptual region.

⁸ Again, this can be related our considerations from Section 4.1.1 with respect to hyperballs under the Euclidean distance.

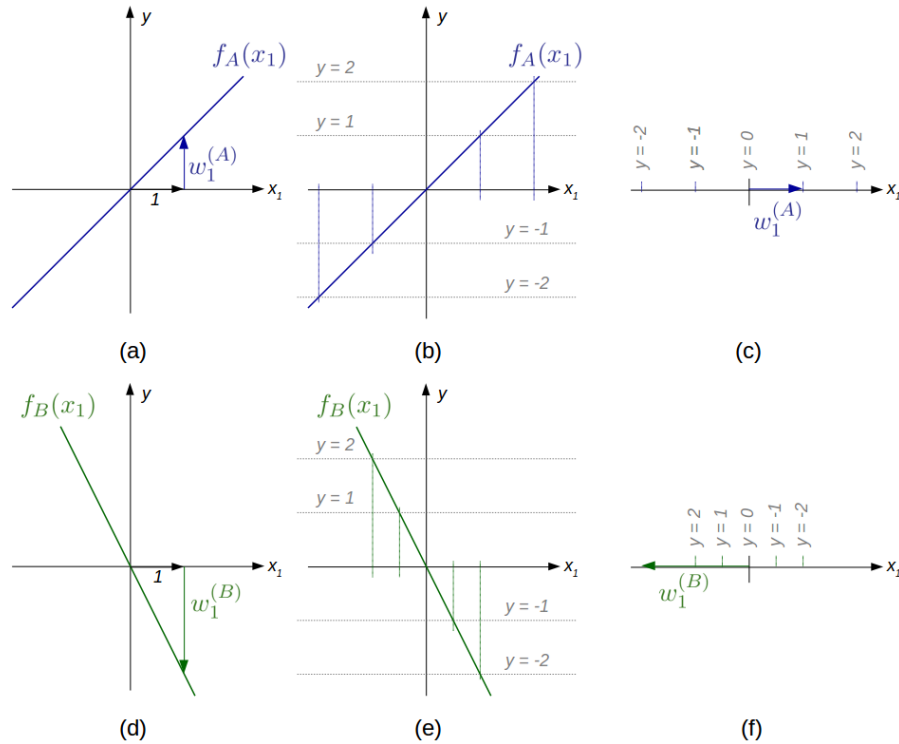


Figure 5.3: Interpreting the weight vector of a linear model as direction in a one-dimensional feature space.

Interpreting the weight vector as a direction

Visualization for one-dimensional feature space

Visualization for two-dimensional feature space

We would like to point out that we can interpret the weights \vec{w} of a linear model as a direction in the feature space: If we ignore the intercept w_0 , there is exactly one weight w_i for each of the features. The direction given by the weight vector \vec{w} can then be interpreted as showing the direction from data points with small target values $y^{(j)}$ to data points with large target values $y^{(j)}$.

Let us provide a visualization of this interpretation. Consider the two linear functions $f_A(x_1) = w_1^{(A)} \cdot x_1$ and $f_B(x_1) = w_1^{(B)} \cdot x_1$ in Figure 5.3a and d. For the sake of simplicity, we assume that in both cases $w_0 = 0$, i.e., there is no intercept. As $w_1^{(A)} > 0$ and $w_1^{(B)} < 0$, $f_A(x_1)$ is monotonically increasing while $f_B(x_1)$ is monotonically decreasing. As we can see in Figure 5.3a and d, the magnitude of w_1 determines the steepness of the function: Since $|w_1^{(B)}| > |w_1^{(A)}|$, $f_B(x_1)$ is steeper than $f_A(x_1)$. Figure 5.3b and e illustrate this by intersecting several constant functions with $f_A(x_1)$ and $f_B(x_1)$, respectively. Figure 5.3c and f show compressed one-dimensional visualizations where the weight vectors $w_1^{(A)}$ and $w_1^{(B)}$ are shown on the x_1 -axis along with the points where $f_A(x_1)$ and $f_B(x_1)$ reach different given values. As we can see in this illustration, w_1 points into the direction of increasing function values. Moreover, for larger absolute values of w_1 , the "contour lines" move closer together, indicating a steeper function.

The latter visualization can also be generalized to two-dimensional spaces as illustrated in Figure 5.4. In both examples, the weight vectors

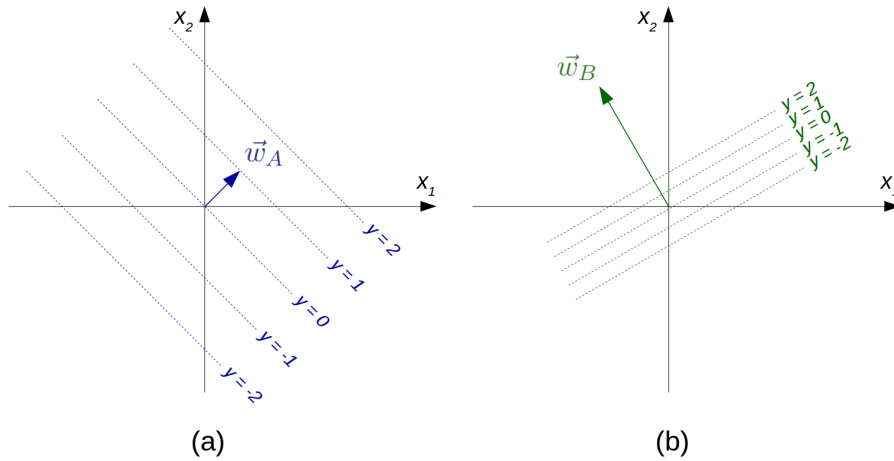


Figure 5.4: Interpreting the weight vector of a linear model as direction in a two-dimensional feature space.

point into the direction of increasing function values. A smaller weight vector (Figure 5.4a) again corresponds to a slower growth of the function value than a larger weight vector (Figure 5.4b) as illustrated by the contour lines.

We will use both a linear regression and a lasso regression in Chapters 9 and 12 as two simple off-the-shelf regressors for learning the mapping from images to points in a given similarity space. In Chapter 8, we will furthermore argue that one can use a linear regression for finding interpretable directions in a psychological similarity space. This is based on the interpretation of the weight vector as pointing from regions with small function values to regions with large function values. We will then apply this approach in practice in Chapter 11 on a psychological similarity space for the SHAPE domain.

Outlook

5.2.2 Logistic Regression

The linear regression presented in the previous section is an example for the class of linear models, i.e., machine learning methods which represent their hypothesis by using a linear function of the input. Linear models can not only be used for regression tasks, but also for classification. In the simplest case (which is called the *perceptron model*), this can be done by computing $\vec{w} \cdot \vec{x}$ and assigning class membership based on the sign of the result: A given data point \vec{x} is classified as belonging to the positive class if $\vec{w} \cdot \vec{x} \geq 0$ and as belonging to the negative class if $\vec{w} \cdot \vec{x} < 0$ [341, Section 18.6]. If we look again at Figure 5.4, we can see that this corresponds to using the contour line with $y = 0$ (which is defined by its normal vector \vec{w}) as a decision boundary: All examples lying on the one side of this linear separator are classified as positive and all all examples on the other side as negative.

*Linear models for classification:
perceptron model*

A *logistic regression* can be thought of as a soft generalization of this perceptron model [341, Section 18.6]: Instead of using the sign function

Logistic regression as soft generalization

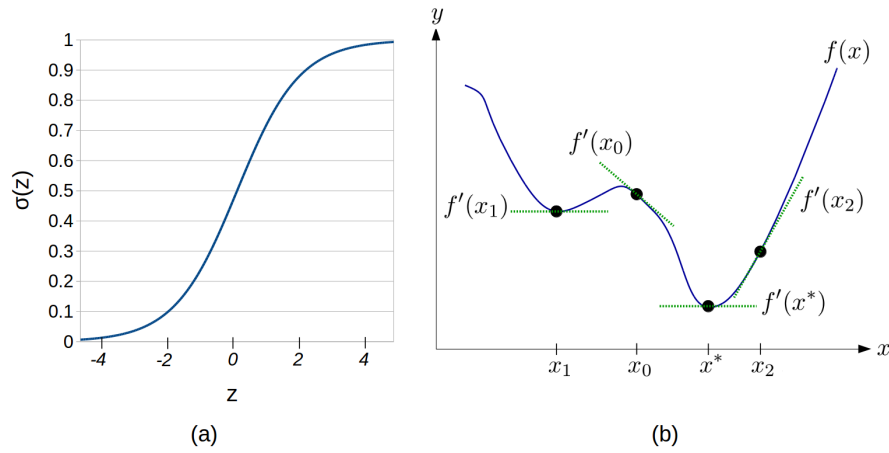


Figure 5.5: (a) The sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$. (b) Illustration of gradient descent for a function \hat{f} .

for converting the output of the linear model into a class assignment, a logistic regression uses the so-called *logistic function* or *sigmoid function* $\sigma(z) = \frac{1}{1+e^{-z}}$, which is illustrated in Figure 5.5a. This sigmoid function provides a smooth transition between values of zero and one, indicating membership to the negative and the positive class, respectively. The overall model can then be written as follows:

$$\hat{f}(\vec{x}) = \frac{1}{1 + e^{-\vec{w} \cdot \vec{x}}} = \sigma(\vec{w} \cdot \vec{x})$$

Since $\hat{f}(\vec{x}) \in (0, 1)$, one can interpret the resulting number also as a probability of data point \vec{x} belonging to the positive class [172, Chapter 5]. As the component $\vec{w} \cdot \vec{x}$ remains unchanged in comparison to a linear regression, we can still interpret \vec{w} as the normal vector of a separating hyperplane, which is now, however, located at $\hat{f}(\vec{x}) = 0.5$.

Gradient descent as optimization technique

Unfortunately, there is no closed-form solution for finding the optimal weight vector \vec{w}^* of a logistic regression [341, Section 18.6]. One therefore optimizes \vec{w} with an iterative optimization method called *gradient descent*, which can in general be applied if analytical solutions of an optimization problem are either not possible or too costly to compute [71, Chapter 8]. The general procedure of minimizing an arbitrary function $f(\vec{x})$ with gradient descent is shown Algorithm 5.1 and illustrated with a one-dimensional example in Figure 5.5b. Even though gradient descent can only find a *local* optimum and cannot be guaranteed to succeed in a reasonable amount of time, in practice it often finds near-optimal solutions quite quickly [172, Chapter 5].

Minimization by following the gradient

In the one-dimensional case (i.e., where the vector \vec{x} reduces to a single scalar value x), the gradient of the function $f(\vec{x}) = f(x)$ is equivalent to its derivative $f'(x) = \frac{\partial f(x)}{\partial x}$, which gives the slope of $f(x)$ at a given point x . It is positive, if increasing x leads to an increase of $f(x)$, and it is negative, if increasing x causes a decrease of $f(x)$. We can thus iteratively minimize f by changing x in small steps with the opposite

Algorithm 5.1: General gradient descent procedure.**Input:** Function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, learning rate α , threshold ϵ **Output:** Point $\vec{x} \in \mathbb{R}^n$ which approximately minimizes f

```

1 Set  $\vec{x}$  to an arbitrary starting value
2 repeat
3   Compute gradient  $\nabla_{\vec{x}} f(\vec{x}) = \left( \frac{\partial f(\vec{x})}{\partial x_1}, \dots, \frac{\partial f(\vec{x})}{\partial x_n} \right)$ 
4   Update  $\vec{x} = \vec{x} - \alpha \cdot \nabla_{\vec{x}} f(\vec{x})$ 
5 until  $|\nabla_{\vec{x}} f(\vec{x})| < \epsilon$ 
6 return  $\vec{x}$ 

```

sign of the derivative until we find a point \hat{x} for which $f'(\hat{x}) = 0$ [172, Section 4.3] (cf. line 5 in Algorithm 5.1). Such stationary points include local minima (where $f(\hat{x})$ is smaller than for all neighboring points), local maxima (where $f(\hat{x})$ is larger than for all neighboring points), and saddle points (all other cases). When trying to minimize a given function $f(x)$, we are in general interested in finding a *global minimum*, i.e., the point x^* with the lowest value $\hat{f}(x^*)$ for all possible points x . Please note that each global minimum is also a local minimum, but that not all local minima are global minima. For instance, the function $f(x)$ in Figure 5.5b has two local minima (x_1 and the global minimum x^*). In both of these points, the derivative of f equals zero, as indicated by the horizontal tangents.

Gradient descent aims at finding the global minimum by starting with a randomly chosen point (cf. line 1 in Algorithm 5.1) and iteratively improving its current estimate. Let us consider the example from Figure 5.5b and assume that our current estimate of the global minimum is x_0 . The derivative of f with respect to x in x_0 corresponds to the slope of the tangent at x_0 , and is negative in our case. This means that increasing x results in a decrease of $f(x)$. Gradient descent uses this information to improve its estimate of the global minimum by taking a small step "downhill", i.e., by moving to the right to a point $x_0 + \Delta x$, because the derivative is negative. In order to define the size of the update Δx , one typically multiplies the derivative $f'(x)$ with a constant α , which is called the *step size* or *learning rate* [341, Section 18.6]. Thus, our update rule is $x_{t+1} = x_t - \alpha \cdot f'(x_t)$.

Choosing a good value for the learning rate α is difficult. If α is set too small, then convergence is slow, because the individual updates are very small. On the other hand, if α is set too large, then the steps might be too large to allow for convergence. In the example from Figure 5.5b, we might never reach x^* with a large α , because we may always step over the "valley" and end up oscillating between x_0 and x_2 . One solution to overcome this problem is to start with a relatively large learning rate and to decrease it over time [295, Chapter 4]. In Chapter 6, we will introduce more sophisticated variants of gradient descent for artificial neural networks, which automatically adapt their learning rate based on the recent update history.

A simple update step in the one-dimensional example

Choosing an appropriate learning rate

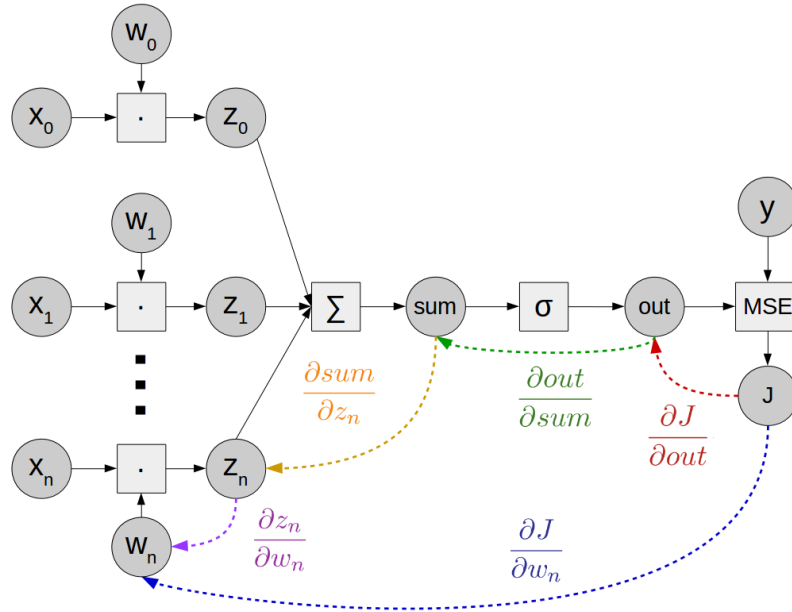


Figure 5.6: Illustration of the computation graph for a logistic regression and the partial derivatives involved in computing the gradient of the loss with respect to an individual weight.

Generalization to higher-dimensional input spaces

Gradient descent can of course also be applied to functions with vectors as input, i.e., $f : \mathbb{R}^n \rightarrow \mathbb{R}$. Here, we consider the partial derivatives $\frac{\partial f(\vec{x})}{\partial x_i}$ which measure how $f(\vec{x})$ changes at the point \vec{x} if only the coordinate x_i is slightly increased. The gradient of f is then defined as the vector of all partial derivatives of f and is denoted by $\nabla_{\vec{x}} f(\vec{x}) = \left(\frac{\partial f(\vec{x})}{\partial x_1}, \dots, \frac{\partial f(\vec{x})}{\partial x_n} \right)$ (cf. line 3 in Algorithm 5.1). Since this gradient has the same dimensionality as \vec{x} , it can be interpreted as a vector in the input space which specifies the direction of steepest increase in f [295, Chapter 4]. We can thus minimize $f(\vec{x})$ by iteratively moving \vec{x} in the opposite direction of the gradient $\nabla_{\vec{x}} f(\vec{x})$ [172, Section 4.3]. Thus, our update rule for the general case is $\vec{x}_{t+1} = \vec{x}_t - \alpha \cdot \nabla_{\vec{x}} f(\vec{x})$ (cf. line 4 in Algorithm 5.1).

Logistic regression: minimizing loss with respect to weights

In the case of a logistic regression, we apply gradient descent to the weight vector \vec{w} in order to minimize the loss function $J(\vec{w})$. This requires, however, a way to compute the gradient $\nabla_{\vec{w}} J(\vec{w})$, i.e., the partial derivatives $\frac{\partial J(\vec{w})}{\partial w_i}$ for the individual weights w_i . These partial derivatives can be obtained by applying the *chain rule* [341, Section 18.6], i.e., the observation that for $h(x) = g(f(x))$, we get the derivative $h'(x) = g'(f(x)) \cdot f'(x)$. Figure 5.6 visualizes the computations taking place in a logistic regression in the form of a computation graph, assuming that the mean squared error is used to compute the loss. Here, individual nodes correspond to variables (represented as circles) or operations on variables (represented as rectangles). As one can see from Figure 5.6, the overall partial derivative $\frac{\partial J(\vec{w})}{\partial w_i}$ with respect to

any weight w_i can be factored into different components based on the individual computation steps by repeatedly applying the chain rule:

$$\frac{\partial J(\vec{w})}{\partial w_i} = \frac{\partial J}{\partial out} \frac{\partial out}{\partial sum} \frac{\partial sum}{\partial z_i} \frac{\partial z_i}{\partial w_i}(\vec{w})$$

We can derive a closed form expression for $\frac{\partial J(\vec{w})}{\partial w_i}$ by considering the individual components one after another and replacing them with their concrete form. For the sake of simplicity, we now consider only a single training example (\vec{x}, y) .

Computing the partial derivative step by step

Since we assume the usage of the mean squared error, we know that the loss for a single training example can be computed as $J = (y - out)^2$. Here, we can apply the chain rule again by defining $g(a) = a^2$ and $f(out) = y - out$, leading us to $\frac{\partial g(f(out))}{\partial out} = g'(f(out))f'(out)$. Since $f'(out) = -1$ and $g'(a) = 2a$, we thus get that $\frac{\partial J}{\partial out} = -2(y - out)$.

$$\frac{\partial J}{\partial out}$$

The next term, $\frac{\partial out}{\partial sum}$ is simply based on the derivative of the sigmoid function, which can conveniently be written as $\sigma'(a) = \sigma(a)(1 - \sigma(a))$. Hence, $\frac{\partial out}{\partial sum} = out(1 - out)$.

$$\frac{\partial out}{\partial sum}$$

The derivative $\frac{\partial sum}{\partial z_i}$ is based on $sum = \sum_{i=0}^n z_i$, which is a linear function in z_i . One can thus easily see that $\frac{\partial sum}{\partial z_i} = 1$

$$\frac{\partial sum}{\partial z_i}$$

Finally, we observe that $\frac{\partial z_i}{\partial w_i} = x_i$, since $z_i = x_i w_i$ is linear in w_i with a factor of x_i (which is treated as a constant in this context).

$$\frac{\partial z_i}{\partial w_i}$$

By combining all of these insights, we thus get the following closed form for the partial derivative $\frac{\partial J(\vec{w})}{\partial w_j}$:

Putting the pieces together

$$\frac{\partial J(\vec{w})}{\partial w_j} = \frac{\partial J}{\partial out} \frac{\partial out}{\partial sum} \frac{\partial sum}{\partial z_j} \frac{\partial z_j}{\partial w_j}(\vec{w}) = -2(y - out) \cdot out(1 - out) \cdot 1 \cdot x_j$$

The overall update rule for w_i can thus be written as follows (using the fact that $out = \hat{f}(\vec{x})$ and ignoring the constant value 2):

$$w_i = w_i + \alpha \cdot (y - \hat{f}(\vec{x})) \cdot \hat{f}(\vec{x})(1 - \hat{f}(\vec{x})) \cdot x_i$$

So far, we have only considered a single data point. However, the loss function should be minimized over *all* data points in the training set. In *batch gradient descent*, one computes the overall loss over all examples and hence the gradient over the whole training set [172, Chapter 5]. This can, however, be computationally quite expensive and thus very slow [341, Section 18.6].

Generalizing to multiple examples: batch gradient descent

Typical loss functions such as the MSE decompose into a sum over per-example losses for all examples from the training set. In *stochastic gradient descent*, one thus computes the loss and hence the gradient only for a single training example [341, Section 18.6] or for a small mini-batch of examples [172, Chapter 5]. This is typically considerably faster than batch gradient descent [341, Section 18.6]. The key insight behind stochastic gradient descent is that the gradient itself is an expectation, which can be approximated using only a small subset of examples [172, Chapter 5]. Since only a subset of training examples is considered for each update, stochastic gradient descent minimizes a different error function in each step. This can help the algorithm to avoid getting stuck

Frequent small updates: stochastic gradient descent

in local minima since the different error functions have different local minima in different locations of the weight space [295, Chapter 4]. We will consider different variants of gradient descent in more detail in Chapter 6 in the context of artificial neural networks.

Outlook

We will not directly use a logistic regression in any of our experiments in Part III of this dissertation. Nevertheless, it will re-appear in different contexts throughout this dissertation: As we will see in Chapter 6, a logistic regression can be seen as fundamental building block of artificial neural networks, where individual neurons use a linear model followed by a nonlinear activation function for determining their output. Also the optimization of weights based on gradient descent will resurface in this context, as already noted above. Moreover, a generalization of the logistic regression to multiple classes (involving the so-called *softmax* activation function as generalization of the sigmoid function) will be used both for artificial neural networks in Chapter 6 and for a soft partitioning of a given conceptual space into conceptual regions in Chapter 7.

5.2.3 Support Vector Machines

Support vector machines (SVMs) make use of a constrained linear model and go back to the work of Vapnik and colleagues (see for example [73, 110]). We will base our overview mainly on the brilliant lecture by Winston [436].

Support vector machine as linear model

A support vector machine is a supervised machine learning model for classification which tries to separate two classes using a hyperplane, i.e., a generalization of lines and planes to higher dimensions. This hyperplane can be expressed by the equation $\vec{w} \cdot \vec{x} + b = 0$ where \vec{x} corresponds to the feature vector, \vec{w} is a weight vector, and b is an intercept. This is the linear model already discussed in Sections 5.2.1 and 5.2.2 for the linear and logistic regression, respectively. Unseen data points are classified by evaluating $\text{sign}(\vec{w} \cdot \vec{x} + b)$, i.e., by determining on which side of the hyperplane they lie. This is an important difference to the logistic regression, where the sigmoid function is used to obtain classification probabilities (cf. Section 5.2.2). As argued in Section 5.2.1, the weight vector \vec{w} can be interpreted as normal vector of the decision hyperplane, pointing from negative to positive examples.

Not all separating hyperplanes are equally good

In Figure 5.7a, we can see a two-dimensional feature space with two hyperplanes separating positive and negative examples. One can argue that while both hyperplanes perfectly separate the two classes, they are probably not an optimal choice: Both lines lie very close to some of the training examples, even though this is not necessary for separating the classes. If we assume that unseen instances lie in general close to the examples of their respective class, this can lead to errors in generalization [341, Section 18.6]. For instance, consider the unclassified data point illustrated as a gray diamond in Figure 5.7a: It lies closer

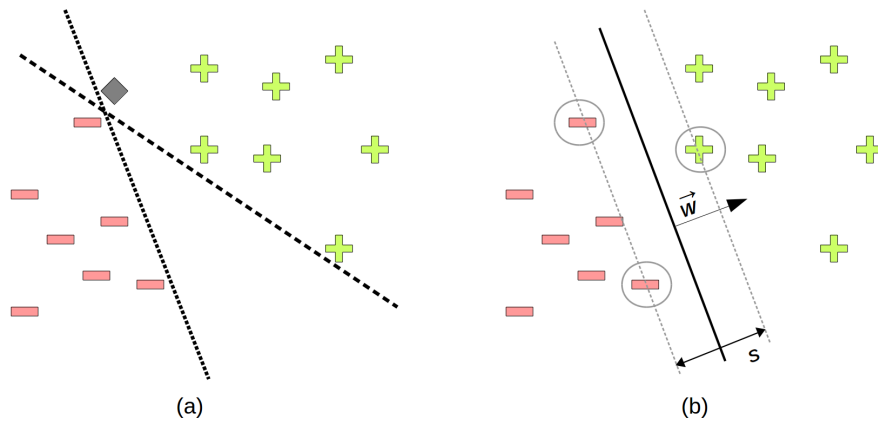


Figure 5.7: (a) Non-optimal decision boundaries in a two-dimensional feature space. (b) Maximum margin separating hyperplane.

to the negative examples than to the positive examples. Therefore, we intuitively expect it to be classified as negative. However, it lies on the positive side of both decision boundaries and is hence treated as positive example by both of them.

In order to minimize the expected generalization loss, a support vector machine tries to find not any separating hyperplane (as for instance a logistic regression would do), but a so-called *maximum margin* hyperplane, i.e., a decision boundary which has the largest possible distance to all example data points. This is illustrated in Figure 5.7b, where the size of this margin is denoted by s . Maximizing s can be seen as a safety measure against future misclassifications. The name support vector machine stems from the observation that only certain data points (the so-called *support vectors*) are crucial for defining the decision boundary, namely the ones closest to the hyperplane (highlighted by circles in Figure 5.7b). The crucial role of these support vectors is reflected in the name "support vector machine". In the following, we will derive a way of finding such a maximum margin separating hyperplane.

Maximum margin hyperplanes

In order to constrain the linear model to find a hyperplane that separates the training examples, we enforce the following constraint for all examples $(\vec{x}^{(j)}, y^{(j)})$:

$$y^{(j)} (\vec{w} \cdot \vec{x}^{(j)} + b) - 1 \geq 0 \quad (5.3)$$

As one can easily see, this means that we require $\vec{w} \cdot \vec{x}^{(j)} + b \geq 1$ for positive examples (where $y^{(j)} = 1$) and $\vec{w} \cdot \vec{x}^{(j)} + b \leq -1$ for negative examples (where $y^{(j)} = -1$).⁹ Moreover, we require for data points, which lie exactly on the margin (i.e., for the support vectors \vec{x}^+ and \vec{x}^-), that they take the exact values $+1$ and -1 , respectively:

$$y^{(j)} (\vec{w} \cdot \vec{x}^{(j)} + b) - 1 = 0 \quad (5.4)$$

Classification constraint for the hyperplane

Stronger constraint for support vectors

⁹ Please note that while most other machine learning algorithms represent the two classes with the labels 0 and 1, support vector machines use the values -1 and 1. This is mainly for notational convenience, but does not entail any deeper conceptual differences.

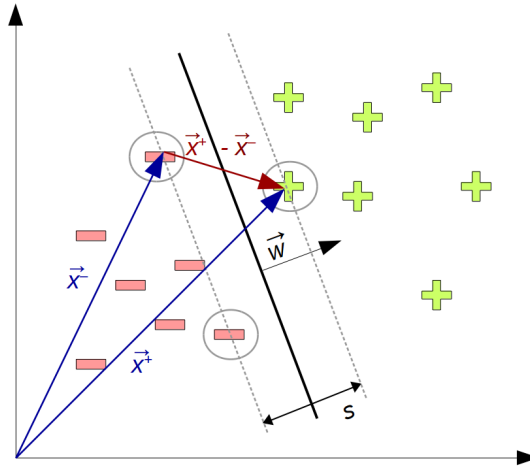


Figure 5.8: Computing the margin size s based on two support vectors \vec{x}^+ and \vec{x}^- and the weight vector \vec{w} .

Computing the size of the margin

In order to find a maximum margin hyperplane, we need to quantify the size s of the margin. This width can be computed as follows (cf. Figure 5.8): We first obtain the difference vector $(\vec{x}^+ - \vec{x}^-)$ between two support vectors \vec{x}^+ and \vec{x}^- belonging to the positive and the negative class, respectively. Then, we project this difference vector onto the weight vector \vec{w} , which is a normal vector of our hyperplane. The length of the projected vector can be obtained based on their dot product $(\vec{x}^+ - \vec{x}^-) \cdot \vec{w}$: In general, the dot product of two vectors $\vec{a}, \vec{b} \in \mathbb{R}^n$ is defined as $\vec{a} \cdot \vec{b} = \sum_{i=1}^n a_i b_i$ (cf. Section 5.2.1) and returns a scalar value, which is equivalent to $\|\vec{a}\| \cdot \|\vec{b}\| \cdot \cos \alpha$, where α denotes the angle between \vec{a} and \vec{b} . The length of the vector \vec{a} when projected onto the direction \vec{b} equals $\|\vec{a}\| \cdot \cos \alpha$, i.e., $\frac{\vec{a} \cdot \vec{b}}{\|\vec{b}\|}$. We can thus obtain the size s of the margin by dividing $(\vec{x}^+ - \vec{x}^-) \cdot \vec{w}$ through the length $\|\vec{w}\|$ of our weight vector:

$$s = (\vec{x}^+ - \vec{x}^-) \cdot \frac{\vec{w}}{\|\vec{w}\|} = \frac{1}{\|\vec{w}\|} (\vec{x}^+ \cdot \vec{w} - \vec{x}^- \cdot \vec{w}) \tag{5.5}$$

Using the classification constraints for support vectors

From the classification constraints for support vectors in Equation 5.4, we know that $1 \cdot (\vec{w} \cdot \vec{x}^+ + b) - 1 = 0$, and thus $\vec{w} \cdot \vec{x}^+ = \vec{x}^+ \cdot \vec{w} = 1 - b$. Analogously, we can derive that $\vec{x}^- \cdot \vec{w} = -1 - b$. Inserting this into Equation 5.5 gives us the following result:

$$s = \frac{1}{\|\vec{w}\|} (1 - b + 1 + b) = \frac{2}{\|\vec{w}\|} \tag{5.6}$$

Maximizing the margin means minimizing $\|\vec{w}\|$

Essentially, Equation 5.6 tells us that we can maximize the size of the margin by minimizing the norm of \vec{w} . In order to find a maximum margin hyperplane, we thus need to minimize $\|\vec{w}\|$ under the classification constraints from Equation 5.3. One now could apply gradient descent to solve this problem [341, Section 18.9]. However, the approach described in the following has shown to be much more elegant and efficient.

Our optimization problem for finding the maximum margin hyperplane concerns minimizing $\|\vec{w}\|$ (cf. Equation 5.6) under the constraints of correct classification from Equation 5.3. This can be described as follows, where \vec{w} and b are the free parameters to be optimized:

A constrained optimization problem

$$\text{minimize } \frac{1}{2}\|\vec{w}\|^2 \quad (5.7)$$

under the constraints $y^{(j)}(\vec{w} \cdot \vec{x}^{(j)} + b) - 1 \geq 0 \quad \forall j \in \{1, \dots, N\}$

Please note that minimizing $\|\vec{w}\|$ is equivalent to minimizing $\frac{1}{2}\|\vec{w}\|^2$. We use the latter formulation, since it will turn out to be mathematically more convenient. Continuous optimization problems under a given set of constraints can be transformed into their so-called *dual form* by using *Lagrange multipliers* α_j [60]. The equivalent Lagrangian of this optimization problem can be written as follows, with $\vec{\alpha} \in \mathbb{R}^N$ being a vector of Lagrange multipliers with one entry per constraint:

The dual form of the problem

$$L(\vec{w}, b, \vec{\alpha}) = \frac{1}{2}\|\vec{w}\|^2 - \sum_{j=1}^N \alpha_j [y^{(j)}(\vec{w} \cdot \vec{x}^{(j)} + b) - 1] \quad (5.8)$$

under the constraints $\alpha_j \geq 0 \quad \forall j \in \{1, \dots, N\}$

Solving the optimization problem from Equation 5.7 in the original space is equivalent to finding a saddle point of $L(\vec{w}, b, \vec{\alpha})$, where $L(\vec{w}, b, \vec{\alpha})$ reaches a minimum with respect to both \vec{w} and b and a maximum with respect to $\vec{\alpha}$ [73]. The Kuhn-Tucker theorem states that the following condition must be fulfilled by a solution $(\vec{w}^*, b^*, \vec{\alpha}^*)$ of Equation 5.8 [110]:

Conditions for a solution in the dual form

$$\alpha_j^* [y^{(j)}(\vec{x}^{(j)} \cdot \vec{w}^* + b^*) - 1] = 0 \quad \forall j \in \{1, \dots, N\} \quad (5.9)$$

This means that α_j^* can be non-zero only for the cases where $y^{(j)}(\vec{x}^{(j)} \cdot \vec{w}^* + b^*) = 1$, i.e., for the support vectors. For all other vectors from the training set, $y^{(j)}(\vec{x}^{(j)} \cdot \vec{w}^* + b^*) > 1$, hence $\alpha_j^* = 0$.

Constraints for $\vec{\alpha}^$*

For every minimum of $L(\vec{w}, b, \vec{\alpha})$ with respect to \vec{w} , all derivatives $\frac{\partial L(\vec{w}^*, b^*, \vec{\alpha}^*)}{\partial w_i}$ must be zero. Based on Equation 5.8, we can thus derive the following constraints:

*Constraints for \vec{w}^**

$$\frac{\partial L(\vec{w}^*, b^*, \vec{\alpha}^*)}{\partial w_i} = w_i^* - \sum_{j=1}^N \alpha_j^* y^{(j)} x_i^{(j)} \stackrel{!}{=} 0$$

As one can easily see, the optimal weight w_i^* can be written as a linear sum over the examples $x_i^{(j)}$, weighted by their class label $y^{(j)}$ and the Lagrange multipliers α_j^* . Since this holds for every individual weight w_i^* , we can write the overall optimal weight vector \vec{w}^* as follows:

$$\vec{w}^* = \sum_{j=1}^N \alpha_j^* y^{(j)} \vec{x}^{(j)} \quad (5.10)$$

We know based on Equation 5.9 that the α_j are non-zero only for the relatively small number of support vectors. This means that the normal

vector \vec{w}^* of the resulting maximum margin hyperplane can be written as a linear combination of a small number of support vectors.

*Constraints for b^**

Also for finding b^* , we can require the partial derivative of $L(\vec{w}^*, b^*, \vec{\alpha}^*)$ with respect to b to be zero, which yields based on Equation 5.8 the following result:

$$\frac{\partial L(\vec{w}^*, b^*, \vec{\alpha}^*)}{\partial b} = - \sum_{j=1}^N \alpha_j y^{(j)} = 0 \quad (5.11)$$

Putting everything together

Let us now use the optimal weight vector \vec{w}^* from Equation 5.10 as well as the constraint on b^* from Equation 5.11 to rewrite the Lagrangian from Equation 5.8. We start by rewriting the squared norm of \vec{w}^* as its dot product with itself. Moreover, we split up the sum over j into its individual components and use the variable name j_1 instead:

$$\begin{aligned} L(\vec{w}^*, b, \vec{\alpha}) &= \frac{1}{2} \|\vec{w}^*\|^2 - \sum_{j_1=1}^N \alpha_{j_1} [y^{(j_1)} (\vec{w}^* \cdot \vec{x}^{(j_1)} + b) - 1] \\ &= \frac{1}{2} \vec{w}^* \cdot \vec{w}^* - \sum_{j_1=1}^N \alpha_{j_1} y^{(j_1)} \vec{x}^{(j_1)} \vec{w}^* - \sum_{j_1=1}^n \alpha_{j_1} y^{(j_1)} b \\ &\quad + \sum_{j_1=1}^N \alpha_{j_1} \end{aligned}$$

*Applying the definition of \vec{w}^**

Let us now insert the definition of \vec{w}^* from Equation 5.10. We can furthermore pull b in front of the sum, since it is independent of j_1 . This gives us the following intermediate result:

$$\begin{aligned} L(\vec{w}^*, b, \vec{\alpha}) &= \frac{1}{2} \vec{w}^* \cdot \vec{w}^* - \sum_{j_1=1}^N \alpha_{j_1} y^{(j_1)} \vec{x}^{(j_1)} \vec{w}^* - \sum_{j_1=1}^n \alpha_{j_1} y^{(j_1)} b \\ &\quad + \sum_{j_1=1}^N \alpha_{j_1} \\ &= \frac{1}{2} \left(\sum_{j_1=1}^N \alpha_{j_1} y^{(j_1)} \vec{x}^{(j_1)} \right) \left(\sum_{j_2=1}^N \alpha_{j_2} y^{(j_2)} \vec{x}^{(j_2)} \right) \\ &\quad - \sum_{j_1=1}^N \alpha_{j_1} y^{(j_1)} \vec{x}^{(j_1)} \cdot \left(\sum_{j_2=1}^N \alpha_{j_2} y^{(j_2)} \vec{x}^{(j_2)} \right) \\ &\quad - b \sum_{j_1=1}^N \alpha_{j_1} y^{(j_1)} + \sum_{j_1=1}^N \alpha_{j_1} \end{aligned}$$

Applying Equation 5.11

We can now use Equation 5.11 to eliminate the term involving b , and we put additional parentheses around $\sum_{j_1=1}^N \alpha_{j_1} y^{(j_1)} \vec{x}^{(j_1)}$:

$$\begin{aligned}
L(\vec{w}^*, b, \vec{\alpha}) &= \frac{1}{2} \left(\sum_{j_1=1}^N \alpha_{j_1} y^{(j_1)} \vec{x}^{(j_1)} \right) \left(\sum_{j_2=1}^N \alpha_{j_2} y^{(j_2)} \vec{x}^{(j_2)} \right) \\
&\quad - \sum_{j_1=1}^N \alpha_{j_1} y^{(j_1)} \vec{x}^{(j_1)} \cdot \left(\sum_{j_2=1}^N \alpha_{j_2} y^{(j_2)} \vec{x}^{(j_2)} \right) \\
&\quad - b \sum_{j_1=1}^N \alpha_{j_1} y^{(j_1)} + \sum_{j_1=1}^N \alpha_{j_1} \\
&= \frac{1}{2} \left(\sum_{j_1=1}^N \alpha_{j_1} y^{(j_1)} \vec{x}^{(j_1)} \right) \left(\sum_{j_2=1}^N \alpha_{j_2} y^{(j_2)} \vec{x}^{(j_2)} \right) \\
&\quad - \left(\sum_{j_1=1}^N \alpha_{j_1} y^{(j_1)} \vec{x}^{(j_1)} \right) \left(\sum_{j_2=1}^N \alpha_{j_2} y^{(j_2)} \vec{x}^{(j_2)} \right) \\
&\quad + \sum_{j_1=1}^N \alpha_{j_1}
\end{aligned}$$

As one can easily see, we can now merge the first and the third term:

Merging terms

$$\begin{aligned}
L(\vec{w}^*, b, \vec{\alpha}) &= \frac{1}{2} \left(\sum_{j_1=1}^N \alpha_{j_1} y^{(j_1)} \vec{x}^{(j_1)} \right) \left(\sum_{j_2=1}^N \alpha_{j_2} y^{(j_2)} \vec{x}^{(j_2)} \right) \\
&\quad - \left(\sum_{j_1=1}^N \alpha_{j_1} y^{(j_1)} \vec{x}^{(j_1)} \right) \left(\sum_{j_2=1}^N \alpha_{j_2} y^{(j_2)} \vec{x}^{(j_2)} \right) \\
&\quad + \sum_{j_1=1}^N \alpha_{j_1} \\
&= -\frac{1}{2} \left(\sum_{j_1=1}^N \alpha_{j_1} y^{(j_1)} \vec{x}^{(j_1)} \right) \left(\sum_{j_2=1}^N \alpha_{j_2} y^{(j_2)} \vec{x}^{(j_2)} \right) \\
&\quad + \sum_{j_1=1}^N \alpha_{j_1}
\end{aligned}$$

Finally, we convert the two sums into a double sum:

Merging sums

$$\begin{aligned}
L(\vec{w}^*, b, \vec{\alpha}) &= -\frac{1}{2} \left(\sum_{j_1=1}^N \alpha_{j_1} y^{(j_1)} \vec{x}^{(j_1)} \right) \left(\sum_{j_2=1}^N \alpha_{j_2} y^{(j_2)} \vec{x}^{(j_2)} \right) \\
&\quad + \sum_{j_1=1}^N \alpha_{j_1} \\
&= \sum_{j_1=1}^N \alpha_{j_1} - \frac{1}{2} \sum_{j_1=1}^N \sum_{j_2=1}^N \alpha_{j_1} \alpha_{j_2} y^{(j_1)} y^{(j_2)} \vec{x}^{(j_1)} \cdot \vec{x}^{(j_2)}
\end{aligned} \tag{5.12}$$

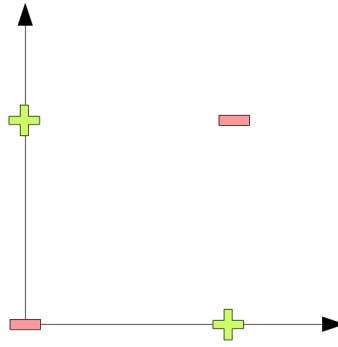


Figure 5.9: The XOR problem is not linearly separable.

*Finding the
maximum margin
hyperplane*

As we can see in Equation 5.12, the optimization problem in its dual form depends now only on α – all terms involving \vec{w}^* or b have disappeared. Finding the optimal hyperplane thus corresponds to maximizing Equation 5.12 over $\vec{\alpha}$. Since Equation 5.12 describes a convex optimization problem [436], it can be efficiently solved by using for instance quadratic programming techniques [341, Section 18.9]. Based on the resulting optimal Lagrange multipliers $\vec{\alpha}^*$, we can then obtain the optimal weight vector \vec{w}^* through Equation 5.10. The optimal value b^* for the intercept can then be obtained based on \vec{w}^* and the classification constraint for support vectors from Equation 5.4. Finally, we can note that in Equation 5.12, the training data only appears in the form of dot products $\vec{x}^{(j_1)} \cdot \vec{x}^{(j_2)}$ of pairs of examples, which will become important for generalizations of the SVM model discussed so far.

*Classification rule
with \vec{w}^* and b^**

We can now insert the optimal \vec{w}^* from Equation 5.10 and the optimal value of b^* into the decision rule used for classifying new examples, leading to the following result:

$$\text{sign}(\vec{w}^* \cdot \vec{x} + b^*) = \text{sign}\left(\sum_{j=1}^N \alpha_j y^{(j)} \vec{x}^{(j)} \cdot \vec{x} + b^*\right) \quad (5.13)$$

As we can see from Equation 5.13, also the classification rule can be expressed based on dot products of the novel example \vec{x} with the training examples $\vec{x}^{(j)}$. It is important to keep in mind that all α_j but the ones corresponding to the support vectors are zero. This means that we only need to compute the dot product of the novel example \vec{x} with the relatively small number of support vectors, not with the whole dataset. Again, this will become relevant for the upcoming generalization of the SVM model.

*Linear SVMs are
limited to linearly
separable classes*

So far, we have only considered *linear* support vector machines, i.e., SVMs with a completely linear model. Moreover, we assumed that the training data is *linearly separable* in the given feature space, i.e., it is possible to separate the two classes with a hyperplane. However, this assumption is often violated in practice. A simple toy example of linear non-separability is the *XOR problem* illustrated in Figure 5.9. In such cases, a linear support vector machine as discussed so far is not

capable of finding a maximum margin hyperplane that separates the two classes – the optimization may thus not converge.

Instead of modifying SVMs to allow for nonlinear decision boundaries, we can use a nonlinear *mapping function* $\phi(\vec{x})$ to project our data points from the original feature space \mathcal{X} into a higher-dimensional space \mathcal{Z} , where the two classes are linearly separable. We can then use a linear SVM to find a maximum margin hyperplane in this higher-dimensional space \mathcal{Z} . This hyperplane in \mathcal{Z} then corresponds to a nonlinear decision boundary in the original feature space \mathcal{X} .

Projecting the data into a higher-dimensional feature space

As we have seen in Equations 5.12 and 5.13 for learning and classification, respectively, we only need the training data in form of dot products $\vec{x}^{(j_1)} \cdot \vec{x}^{(j_2)}$. This means, that the projection function $\phi(\vec{x})$ is only used in terms of the form $\phi(\vec{x}^{(j_1)}) \cdot \phi(\vec{x}^{(j_2)})$. That is, we assume that the co-domain \mathcal{Z} of the mapping function ϕ is a space equipped with an inner product. Instead of explicitly computing $\phi(\vec{x}^{(j_1)}) \cdot \phi(\vec{x}^{(j_2)})$ by first applying ϕ and then computing the dot product, we can often directly compute the inner product of the projected data points by using a so-called *kernel function* $\kappa(\vec{x}^{(j_1)}, \vec{x}^{(j_2)}) = \phi(\vec{x}^{(j_1)}) \cdot \phi(\vec{x}^{(j_2)})$. Typical kernel functions can be formulated based on the original vectors $\vec{x}^{(j_1)}$ and $\vec{x}^{(j_2)}$ without needing explicit access to the mapping function ϕ . This allows for more efficient computations [341, Section 18.9]. Using the more efficient kernel $\kappa(\vec{x}^{(j_1)}, \vec{x}^{(j_2)})$ instead of the often more complex mapping function $\phi(\vec{x})$ by replacing all dot products $\vec{x}^{(j_1)} \cdot \vec{x}^{(j_2)}$ with $\kappa(\vec{x}^{(j_1)}, \vec{x}^{(j_2)})$ is often referred to as the *kernel trick*. It allows us to find a nonlinear decision boundary with a support vector machine without needing to explicitly define the projection space \mathcal{Z} . Popular kernel functions include the following [73]:

The kernel trick

- The *linear kernel* $\kappa(\vec{x}^{(j_1)}, \vec{x}^{(j_2)}) = \vec{x}^{(j_1)} \cdot \vec{x}^{(j_2)}$ does not transform the feature space at all and is equivalent to the linear SVM as described above.
- The *polynomial kernel* $\kappa(\vec{x}^{(j_1)}, \vec{x}^{(j_2)}) = (\vec{x}^{(j_1)} \cdot \vec{x}^{(j_2)} + 1)^q$ corresponds to the inner product in a 2^q -dimensional space [341, Section 18.9]. One can easily see, that computing $\kappa(\vec{x}^{(j_1)}, \vec{x}^{(j_2)})$ is in this case considerably more efficient than first applying the mapping function $\phi(\vec{x})$ and then computing the inner product in the resulting 2^q -dimensional space. The polynomial kernel comes with an additional hyperparameter q and has been used in the first successful applications of SVMs on handwritten digit recognition [73, 110].
- The *radial basis kernel* $\kappa(\vec{x}^{(j_1)}, \vec{x}^{(j_2)}) = e^{-\frac{\|\vec{x}^{(j_1)} - \vec{x}^{(j_2)}\|^2}{\sigma^2}}$ results in decision boundaries resembling Gaussians, and can be interpreted as a form of template matching [172, Chapter 5]. Again, σ is an additional hyperparameter, which needs to be specified by the experimenter. If σ is chosen too small, this leads to an increased risk of overfitting, since the decision boundary will then correspond to small circles around the individual data points [436].

Type of kernel as important hyperparameter

The selection of an appropriate kernel function depends on the concrete application scenario and can be seen as a high-level form of hyperparameter optimization.

Soft margin SVMs

In some cases, linear separation may, however, also not be obtainable when using standard kernels. Cortes and Vapnik [110] have therefore proposed a generalization of regular SVMs by using *soft margin* hyperplanes. They introduce slack variables $\xi_j \geq 0$ for each of the training examples $(\vec{x}^{(j)}, y^{(j)})$ in order to represent misclassifications. They start their considerations with the following optimization problem for some small $\epsilon > 0$:

$$\text{minimize } \Phi(\vec{\xi}) = \sum_{j=0}^N \xi_j^\epsilon \quad (5.14)$$

$$\text{under the constraints } y^{(j)} (\vec{w} \cdot \vec{x}^{(j)} + b) \geq 1 - \xi_j \quad \forall j \in \{1, \dots, N\}$$

$$\text{and } \xi_j \geq 0 \quad \forall j \in \{1, \dots, N\}$$

Minimize misclassifications while maximizing the margin

This corresponds to minimizing the number of misclassifications, because minimizing $\Phi(\vec{\xi})$ means minimizing all ξ_j , which obtain their minimal value of zero for a correct classification. Solving the optimization problem from Equation 5.14 thus leads to a minimal subset of remaining errors on the training set $(\vec{x}^{(j_1)}, y^{(j_1)}), \dots, (\vec{x}^{(j_k)}, y^{(j_k)})$. One could then remove these data points from the training set, and find a maximum margin hyperplane separating the remaining examples. This can be mathematically formalized in a single optimization problem, where λ is a constant and $F(u)$ is a monotonic convex function:

$$\text{minimize } \frac{1}{2} \|\vec{w}\|^2 + \lambda \cdot F\left(\sum_{j=1}^N \xi_j^\epsilon\right) \quad (5.15)$$

$$\text{under the constraints } y^{(j)} (\vec{w} \cdot \vec{x}^{(j)} + b) \geq 1 - \xi_j \quad \forall j \in \{1, \dots, N\}$$

$$\text{and } \xi_j \geq 0 \quad \forall j \in \{1, \dots, N\}$$

Properties of the soft margin approach

The constant λ is a hyperparameter that controls the trade-off between maximizing the margin (first term) and minimizing the number of misclassifications (second term). Cortes and Vapnik [110] showed, that for sufficiently large λ and sufficiently small ϵ , the optimal values for \vec{w}^* and b^* describe the hyperplane with the smallest number of errors on the training set and the largest possible margin on the remaining examples. Moreover, for linearly separable problems, the solution of the soft margin formulation is identical to the solution obtained by ordinary SVMs. Cortes and Vapnik develop a soft variant of Equation 5.12 and show, that the properties observed above (i.e., the weight vector being a linear combination of support vectors, and all computations being based on dot products) also hold for this soft margin formulation. Hence, the kernel trick is also applicable to soft margin SVMs.

Let us now discuss some general properties of support vector machines. Both the classification function and the construction of the hyperplane only depend on a relatively small number of support vectors. In contrast to that, most other machine learning algorithms use *all* data points from the training set for estimating the decision boundary [73, 110]. For instance, in a logistic regression, all data points are considered in gradient descent when minimizing the loss function (cf. Section 5.2.2). While a logistic regression may thus be sensitive to the addition or deletion of training examples, which lie far away from the class boundaries, support vector machines are quite robust to such changes in the data. Moreover, as already stated in the discussion of Equation 5.12, the optimization problem being solved by SVMs can be formulated as maximizing a quadratic form (since the Lagrangian multipliers to be optimized only appear as α_{j_1} and $\alpha_{j_1} \cdot \alpha_{j_2}$) in the positive quadrant (because all α_j are non-negative). This optimization problem has no local optima and a unique solution under some mild constraints [73]. In contrast to that, the decision boundary learned by a logistic regression is not uniquely determined, since the loss function optimized by gradient descent may have multiple local minima. Finding the solution to the SVM optimization problem (and hence obtaining the maximum margin hyperplane) is thus insensitive to the initialization of the optimization method. Moreover, since a maximum margin approach is used, the performance of the solution is relatively insensitive to small changes with respect to the parameters \vec{w} or \vec{a} [73].

General properties of SVMs: unique solution that can be easily found

Support vector machines are also a very flexible method due to the usage of the kernel trick – if the data is not linearly separable, we can use a wide variety of kernel functions to project it into higher-dimensional spaces, where a linear separation can be achieved more easily [341, Section 18.9]. By using kernel functions instead of actual projections, this can often be done in a computationally very efficient way.

Flexibility due to the kernel trick

Outliers in the dataset can also be identified relatively easily by using an SVM: In the soft margin approach, they tend to have large values for the slack variables ξ_j [110] (since they are more likely to be misclassified), and in general, they also tend to have large values for the Lagrange multipliers α_j [73].

Identification of outliers

Furthermore, support vector machines tend to generalize well due to their use of a maximum margin separator, which leads to an automatic tuning of the model's capacity [73]. An illustrative justification for this claim is given by Cortes and Vapnik [110]: For any machine learning algorithm, the probability of making an error on the test set depends both on the frequency of errors on the training set and on a confidence interval. This confidence interval is influenced, among others, by the capacity of the classifier and the size of the training set. A smaller capacity typically leads to a smaller confidence interval (since overfitting becomes less likely), but also to a larger training error (since fitting the data becomes harder). Occam's razor states that one should choose the most simple solution for the given problem (cf. Section 5.1.3). This corresponds to keeping the frequency of training errors at zero while minimizing the confidence interval. Support vector machines

Good generalization through automatic capacity tuning

follow exactly this principle by aiming for perfect classification while minimizing $\|\vec{w}\|^2$, i.e., the model's capacity. This minimization of $\|\vec{w}\|^2$ can thus also be seen as a built-in variant of weight decay regularization (cf. Sections 5.1.3 and 5.2.1).

Upper bound of the generalization error

A related argument has been made by Boser et al. [73] who estimate an upper bound of the generalization error through a leave-one-out procedure (cf. Section 5.1.2) on the training examples, assuming perfect classification on the training set. If the data point \vec{x} being left out is not a support vector, then removing \vec{x} will not change the resulting hyperplane, and \vec{x} will still be classified correctly. If, however, a support vector is removed from the training set, then the decision boundary will change. This former support vector then may or may not be classified correctly, depending on how much the hyperplane has changed. Overall, the expected number of errors in this leave-one-out procedure is thus limited by the number of support vectors. Since the number of support vectors is in general relatively low and does not depend on the number $|\theta|$ of free parameters of the model, the generalization error can be expected to be low even in very high-dimensional feature spaces.

Training an SVM is costly

Since training a support vector machine corresponds to finding optimal values for the Lagrange multipliers α_j , the number of free parameters in the model is not fixed – it grows linearly with the number N of training examples, because each data point adds another classification constraint and thus another entry to $\vec{\alpha}$. Moreover, the optimization product involves *all* pairwise dot products $\vec{x}^{(j_1)} \cdot \vec{x}^{(j_2)}$ of *all* training examples, which grows quadratically with the number N of training examples. Therefore, support vector machines often come with a very high computational cost for large datasets [172, Chapter 5]. Once the parameters have, however, been estimated, an evaluation of the model on new data points is quite efficient due to the dependence on only a relatively small number of support vectors.

Training SVMs on large datasets

In order to mitigate the scaling problem for learning, Cortes and Vapnik [110] have proposed to partition the training data into small subsets. For each of these subsets, an individual SVM can be trained, resulting in an individual hyperplane and an individual set of support vectors for each subset. Then, in order to find the optimal solution for the overall dataset, only the support vectors from the individual subsets need to be considered. While this two-stage process can certainly speed up computations, it still does not scale well to very large datasets.

Linear SVMs and conceptual spaces

Let us now discuss the relation between support vector machines and conceptual spaces. In general, we assume that conceptual regions are convex or star-shaped. Linear support vector machines use a hyperplane to divide the overall feature space into two convex half-spaces, and are thus in line with the convexity requirement. However, half-spaces may be too coarse-grained for describing many concepts, such as the ones used in our fruit space example in Part I of this dissertation. When generalizing SVMs to multiple classes, one usually follows a *one-vs-all* scheme [332], where for each class, an individual SVM is trained in order to separate this class from all other classes. Unfortunately, this

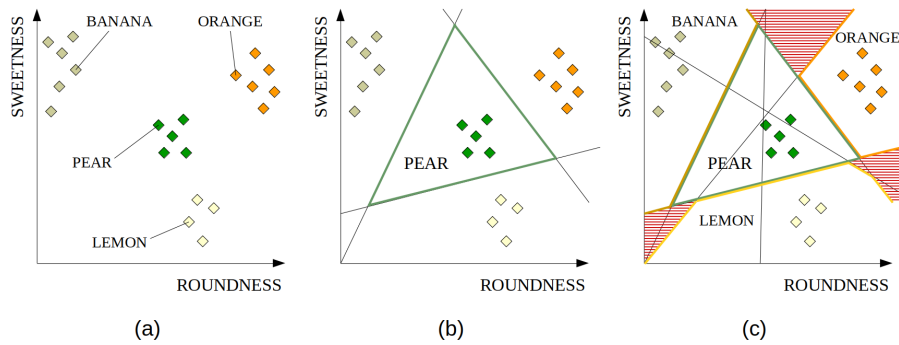


Figure 5.10: (a) Example conceptual space, where PEAR is not linearly separable from the union of all other concepts. (b) Convex polytope of the PEAR concept based on all-vs-all SVMs. (c) Voronoi-like partitioning of the space is prohibited by unassigned regions.

does not work very well for classes that are "surrounded" by other classes in the given feature space, since this is not a linearly separable problem (see Figure 5.10a for an example). If one instead followed an *all-vs-all* scheme [332] by training an individual SVM to discriminate between each pair of classes, one could combine the hyperplanes of all SVMs involving a given class into a convex polytope (see Figure 5.10b). This idea is reminiscent of the formalization of the conceptual spaces framework by Adams and Raubal [3] (see Sections 2.4.2 and 2.5.1). One could also attempt to construct a partitioning of the overall space, similar to the Voronoi tessellations considered by Gärdenfors [179] (cf. Section 1.2.2), by aggregating *all* hyperplanes from all pairs of classes. However, we can see in Figure 5.10c, that this is not easily possible – the highlighted regions do not belong to *any* convex polytope and are thus not assigned to any conceptual region. This effect is caused by the fact that each individual SVM uses its own set of support vectors – unlike the Voronoi tessellation, where the same set of prototypical points is used to generate *all* conceptual boundaries (cf. Section 1.2.2). This causes the observed inconsistencies when aggregating all hyperplanes. One could potentially relate these unassigned regions to the borderline cases considered by Douven et al. [136] in their collated Voronoi diagrams (cf. Section 2.4.1). In general, all-vs-all schemes are computationally quite costly, since one needs to train $\frac{c \cdot (c-1)}{2}$ classifiers for c different classes. In combination with the generally costly training of SVMs, learning conceptual boundaries can thus quickly become infeasible in practice.

As mentioned above, using an SVM with a radial basis kernel gives rise to convex Gaussian-like regions, which may be a better reflection of conceptual regions. Using such RBF-SVMs also allows us to use the one-vs-all scheme, i.e., to train one SVM per class. By using an appropriate kernel function, support vector machines can thus be restricted to learning conceptual regions of a desired form. However, the usage of a kernel function comes with an implicit transformation of the underlying feature space – conceptual boundaries would then strictly speaking not be determined in the original conceptual space: We would train the classifier in a nonlinear transformation of the original conceptual

The kernel trick and conceptual spaces

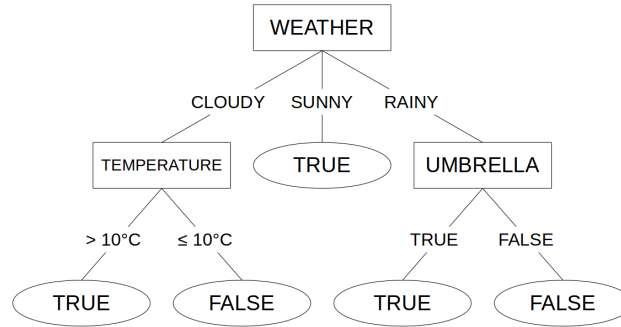


Figure 5.11: Illustration of an example decision tree for deciding whether to take a walk.

space. While the kernel trick is quite powerful from a machine learning perspective, it is unfortunately not quite in line with the assumptions of the original framework, where locations and distances in the conceptual space play a fundamental role. This criticism can also be related to our discussion of Gaussian membership functions in Section 2.2.2.

Outlook

Although we are skeptical about using SVMs for learning conceptual regions, we will see in Chapter 8 that linear support vector machines can be used for identifying meaningful directions in a psychological similarity space [123]. We will apply this approach in Chapter 11 to identify directions for psychologically motivated candidate features in a similarity space for the SHAPE domain.

5.2.4 Decision Trees

Decision trees in a nutshell

All machine learning algorithms considered so far are based on an underlying linear model, where the feature vector $\vec{x}^{(j)}$ is multiplied with a weight vector \vec{w} , and where the resulting scalar value is then used to make a prediction. *Decision trees* follow a completely different approach: They classify new data points by conducting a sequence of tests on their features, which can be visualized as a tree [341, Section 18.3]. Take for instance the example from Figure 5.11, which considers a decision for or against taking a walk in the afternoon. Each *internal node* of this tree (illustrated as rectangles, e.g. WEATHER) selects a single feature, and the branches to its children represent the different possible values of this feature (e.g., CLOUDY, SUNNY, and RAINY). Each *leaf node* (illustrated as ellipses) corresponds to the classification decision (in the given example TRUE or FALSE). Please note that in principle, both the features and the target function can be discrete or continuous. For continuous features (such as TEMPERATURE in Figure 5.11), a simple threshold (such as 10°C in our example) is typically used to branch into the respective sub-trees.

Interpretation as set of rules

One can interpret a decision tree also as a disjunction over a set of conjunctive rules on the feature values – each path to one of the leaf nodes can be represented as one rule [295, Chapter 3]. This means that for discrete features and targets, any function from propositional logic can be expressed as a decision tree [341, Section 18.3]. The decision tree

from Figure 5.11 also contains a continuous feature `TEMPERATURE`, which can, however, be discretized by using a threshold (in our example 10°C). The overall decision tree can then be expressed by a disjunction over the following set of rules:

$$\begin{aligned} &(\text{WEATHER} = \text{CLOUDY}) \wedge (\text{TEMPERATURE} > 10^{\circ}\text{C}) \Rightarrow (\text{DECISION} = \text{TRUE}) \\ &(\text{WEATHER} = \text{CLOUDY}) \wedge (\text{TEMPERATURE} \leq 10^{\circ}\text{C}) \Rightarrow (\text{DECISION} = \text{FALSE}) \\ &\qquad\qquad\qquad (\text{WEATHER} = \text{SUNNY}) \Rightarrow (\text{DECISION} = \text{TRUE}) \\ &(\text{WEATHER} = \text{RAINY}) \wedge (\text{UMBRELLA} = \text{TRUE}) \Rightarrow (\text{DECISION} = \text{TRUE}) \\ &(\text{WEATHER} = \text{RAINY}) \wedge (\text{UMBRELLA} = \text{FALSE}) \Rightarrow (\text{DECISION} = \text{FALSE}) \end{aligned}$$

Decision trees are easy to interpret by humans, if they do not contain too many nodes [341, Section 18.3]. If we consider only continuous attributes, then each internal node can be interpreted as splitting the given region of the feature space into two subregions according to one of the dimensions. This means that each internal node corresponds to an axis-parallel hyperplane in the feature space [172, Chapter 5]. Since each leaf node can only be reached through a sequence of internal nodes, the region in the feature space that it represents is confined by the conjunction of the corresponding hyperplanes. Since each of these hyperplanes is parallel to one of the dimensions, the leaf nodes can be described as axis-parallel cuboids (cf. Definition 2.8 from Section 2.3.2). Goodfellow et al. [172, Chapter 5] pointed out that this can become problematic, if the optimal decision boundary is diagonal – in this case, the decision tree has to approximate it with a large number of small regions mimicking a step function, which can result in very large trees.

Interpretation of decision trees

How can such a decision tree be extracted from a given dataset? We will for now only consider discrete features and a classification task. Essentially, the learning algorithm can be characterized as a simple-to-complex, greedy, hill-climbing approach [295, Chapter 3]. Starting with the overall dataset, the algorithm looks for the feature which provides the best classification when considered on its own. Intuitively, this means that the distribution of class labels becomes more imbalanced (i.e., favoring one of the classes) in the different subsets of the data, which are based on the possible values of the selected feature. The selected feature becomes the root node of the tree, each possible feature value creates a branch, and in a recursive manner, each of the subsets is again split up based on the most useful feature. The algorithm creates a leaf node whenever all examples in the respective subset of data have the same class label or when no further features are left to explore. In the latter case, the value of the leaf node is determined by taking the most frequent label [341, Section 18.3].

Learning a decision tree

The inductive bias (cf. Section 5.1.1) of this learning algorithm is a preference bias, which favors shorter trees over larger trees, and which prefers trees that place informative features closer to the root [295, Chapter 3]. As Russell and Norvig [341, Section 18.3] note, one should not over-interpret the resulting tree: If two features are of similar importance, choosing among them is somewhat arbitrary. Different

Inductive bias of decision tree learning

choices may then result in differently looking trees, which, however, yield a very similar performance.

Feature importance based on entropy

The canonical approach to decision tree learning quantifies feature usefulness based on the information-theoretic notion of *entropy*, which quantifies the degree of uncertainty associated with a random variable [341, Section 18.3]. The entropy $H(A)$ of a random variable A is defined based on its probability distribution $\mathbb{P}(A)$:

$$H(A) = \mathbb{E}_{A \sim \mathbb{P}(A)} [-\log(\mathbb{P}(A))] \quad (5.16)$$

For a binary variable, Equation 5.16 is equivalent to the following formula for the entropy:

$$\begin{aligned} H(A) &= \mathbb{P}(A = \text{TRUE}) \cdot [-\log(\mathbb{P}(A = \text{TRUE}))] \\ &\quad + \mathbb{P}(A = \text{FALSE}) \cdot [-\log(\mathbb{P}(A = \text{FALSE}))] \\ &= \mathbb{P}(A = \text{TRUE}) \cdot [-\log(\mathbb{P}(A = \text{TRUE}))] \\ &\quad + (1 - \mathbb{P}(A = \text{TRUE})) \cdot [-\log(1 - \mathbb{P}(A = \text{TRUE}))] \end{aligned}$$

In the extreme cases of $\mathbb{P}(A = \text{TRUE}) = 1$ and $\mathbb{P}(A = \text{TRUE}) = 0$, the entropy $H(A)$ becomes zero, indicating minimal uncertainty. The entropy of a binary variable A becomes maximal with $H(A) = 1$ for $\mathbb{P}(A = \text{TRUE}) = 0.5$, indicating maximal uncertainty. In the case of decision trees, we use entropy as a measure for the impurity of a collection of training examples [295, Chapter 3].

Information gain as expected reduction in entropy

In the following, we consider the random variable Y , representing the labels $y^{(j)}$ from the dataset. We denote by X_i the i th feature of the dataset and by $Y(X_i = v)$ the random variable describing the labels in the data subset, where feature X_i takes the value v . In a set-based notation, this corresponds to $Y(X_i = v) = \{y^{(j)} \in \vec{y} \mid \vec{x}_i^{(j)} = v\}$. The *information gain* of the i th feature with respect to a given set of examples (X, \vec{y}) can now be defined as the expected reduction in entropy of Y by splitting X according to the possible values of X_i :

$$\text{Gain}(X, Y, i) = H(Y) - \sum_v \frac{|Y(X_i = v)|}{|Y|} H(Y(X_i = v)) \quad (5.17)$$

This is achieved by computing the entropy of every data subset ($H(Y(X_i = v))$), weighted by the respective relative size of this subset (dividing $|Y(X_i = v)|$ by $|Y|$). The sum over these weighted entropies is then subtracted from the entropy $H(Y)$ of the overall dataset.

Example for information gain

In order to illustrate the inner workings of Equation 5.17 for the information gain, let us consider a dataset, which contains an equal amount of positive and negative examples, i.e., $\mathbb{P}(Y = \text{TRUE}) = 0.5$. As stated above, this means that we get an entropy of $H(Y) = 1$. Let us now consider two features, X_1 and X_2 . We assume that both features are binary and that they split the dataset into two parts of identical size.

Information gain of a perfect feature

Let us furthermore assume that X_1 is sufficient for perfect classification, i.e., all data points with $X_1 = \text{TRUE}$ belong to the positive class, while all data points with $X_1 = \text{FALSE}$ belong to the negative class. This means that $H(Y(X_1 = \text{TRUE})) = 0$ and $H(Y(X_1 = \text{FALSE})) = 0$. We

thus get an information gain for the first feature of $Gain(X, Y, 1) = 1 - (\frac{1}{2} \cdot 0 + \frac{1}{2} \cdot 0) = 1$.

On the other hand, assume that X_2 is an entirely useless feature: Both for $X_2 = \text{TRUE}$ and $X_2 = \text{FALSE}$, the class labels are still equally frequent. This means that $H(Y(X_2 = \text{TRUE})) = H(Y(X_2 = \text{FALSE})) = H(Y) = 1$, and hence that $Gain(X, Y, 2) = 1 - (\frac{1}{2} \cdot 1 + \frac{1}{2} \cdot 1) = 0$. Thus, X_2 does not lead to any reduction in entropy.

In decision tree learning, we always select the feature with the largest information gain, i.e., which leads to a partition of the given set of example into more imbalanced (i.e., more homogeneous) subsets. In our example, we would thus select X_1 rather than X_2 . Please note that we have only considered simplified extreme cases – in reality, $Gain(X, Y, i)$ will be a real-valued number. Also $Gain(X, Y, i) < 0$ is possible, but only if the feature is counter-productive, i.e., if it *increases* the entropy rather than decreasing it.

Information gain of a useless feature

Selecting the feature with largest information gain

Decision tree learning is in general prone to overfitting since the tree grows until all examples are perfectly classified. If the dataset is very small or if it contains noise, this can cause the paths in the tree to become overly specialized and thus unable to generalize well to unseen examples [295, Chapter 3]. In principle, there are two ways for avoiding such overfitting issues in decision trees:

Decision trees may overfit

When using *early stopping*, the tree is not grown until perfect classification is achieved, but the growth is stopped at an earlier point. In a simple approach, the tree can only grow until a maximal depth is reached [172, Chapter 5] (enforcing an restriction bias, cf. Section 5.1.1), but defining this maximal depth can be difficult in practice [295, Chapter 3]. Moreover, early stopping may prevent the algorithm from discovering subsets of features which are uninformative on their own, but which may become highly informative when being combined with each other [341, Section 18.3].

Early stopping: limit the depth of the tree

When applying *pruning* techniques, one first grows a decision tree without any restrictions, and afterwards iteratively removes internal nodes and replaces them by leaf nodes. In order to decide whether removing an internal node is helpful or harmful, one typically considers performance on the validation set [295, Chapter 3], although other approaches (such as a threshold on the information gain or statistical tests [341, Section 18.3]) are also possible. The order of removal can be either based on the performance improvement (conducting the most effective pruning steps first) [295, Chapter 3], or based on the position in the tree (starting to prune at the leaves and working upwards towards the root). Pruned trees often work better than their unpruned predecessors, and tend to be easier to interpret, because they contain fewer nodes [341, Section 18.3].

Pruning: grow a deep tree, then shorten it

Another variant of pruning converts the tree into a set of rules by representing the path to each leaf node as a conjunction of feature values [295, Chapter 3] as already mentioned above. Then, each rule is generalized by deleting preconditions whose removal improves classification performance on the validation set. One then sorts the pruned rules

Rule-based pruning

by their estimated accuracy on the overall dataset, and uses them for classification in this order. This approach allows for more fine-grained modifications than removing complete sub-trees and may thus lead to better results. However, the resulting set of rules can in general no longer be represented as a decision tree.

Extensions to decision trees

There are several extensions of the vanilla decision tree learning as sketched above. For instance, *continuous-valued features* can be incorporated into the learning algorithm by dynamically selecting one or multiple thresholds on this feature, which maximize information gain for the given set of examples [295, Chapter 3]. Another extension considers *regression* instead of classification tasks – in this case, each leaf node can either represent the average target value of all its examples or it can perform a linear regression on a subset of features for these examples [341, Section 18.3]. Finally, there exist other variants of determining the best feature, e.g., by using a variant of information gain called *gain ratio*, which also takes into account the number of different feature values v and reduces the preference for many-valued features [295, Chapter 3].

Random forests as ensembles of decision trees

Multiple decision trees can also be grouped together into an ensemble of predictors in order to improve prediction performance and reduce overfitting. One of the most popular ensemble-based approaches is called *random forest* [78]: It grows an ensemble of unpruned decision trees, using a different subset of training examples and/or a different subset of features for each tree. During training, each tree is allowed to overfit its subset of data. Since the different trees have access to different features and different training examples, they extract different classification rules. By aggregating the predictions of the individual trees (for example by taking a majority vote), one can derive an overall prediction for the random forest. As Breiman [78] showed, overfitting is typically not an issue for random forests, since the individual trees overfit on different aspects of the data, largely canceling out each other's mispredictions. In general, a random forest is successful if it contains individual trees that make accurate classifications, but that are largely uncorrelated to each other in their behavior. Random forests can be used not only for classification tasks, but have also successfully been applied to regression, e.g., in predicting numeric relevance scores for documents in an information retrieval setting [296].

Decision trees and conceptual spaces

As already stated above, the internal nodes of a decision tree use a threshold for continuous-valued features and thus represent an axis-parallel hyperplane. If we consider a continuous conceptual space like in Part I of this dissertation, this means that every leaf node with its classification decision corresponds to a cuboid-based region in the conceptual space (cf. Definition 2.8). This means that every single leaf node corresponds to a convex region. However, since the individual sub-trees in decision tree learning are optimized independently from each other, the resulting decision tree may have multiple leaf nodes for a given class. The regions represented by these leaf nodes are not necessarily connected – we can thus in general not enforce convexity or

star-shapedness for conceptual regions learned by a decision tree. This is quite a serious drawback from the conceptual spaces perspective. Enforcing such a constraint would require a strong modification of the learning algorithm, which could then no longer follow a greedy divide-and-conquer approach. Despite their interpretability and their relation to our cuboid-based formalization of conceptual spaces, decision trees are thus not directly applicable to conceptual spaces. The same arguments also apply to random forests, whose ensemble-based approach makes a straightforward definition and interpretation of conceptual regions even more difficult.

We will thus not consider any of these two techniques in Chapter 7, when we discuss learning conceptual regions in cognitively plausible ways. However, we will use random forest regressors as an easy-to-use, off-the-shelf baseline for our ANN-based approach to mapping images to points in psychological similarity spaces in Chapter 9.

Outlook

5.2.5 k Nearest Neighbors

The k nearest neighbor (k NN) algorithm is an example for an *instance-based learning* algorithm. These types of algorithms are sometimes also called *lazy learning* methods, because they do not extract an explicit model from the training data, but postpone the generalization step to the time of classification [295, Chapter 8]. This classification step in instance-based approaches typically involves comparing the novel input to all examples from the training set based on some form of similarity measure. In contrast to *parametric models* (which use a fixed number of parameters to represent the target function), instance-based learning approaches assume that the dataset cannot be summarized in such a compact form [341, Section 18.8]. Instance-based learning algorithms have the advantage that they do not have to form a fixed global model, but that they can form individual local models on the fly during classification [295, Chapter 8]. However, since the whole training set needs to be consulted during classification, this may come with an increased computational cost [295, Chapter 8].

Instance-based learning

Classification in the k nearest neighbor algorithm works as follows [341, Section 18.8]: Given a new example \vec{x} , find the k examples $\{\vec{x}^{j_1}, \dots, \vec{x}^{j_k}\}$ from the training set that are nearest to \vec{x} . Usually, the Euclidean distance is used for finding these k nearest neighbors [295, Chapter 8]. In a classification problem, the most common value among these neighbors is chosen as output, while in a regression setting, the mean value of these neighbors can be used. Alternatively, one can run a linear regression based only on the k nearest neighbors in order to interpolate a suitable output value for \vec{x} in a regression setting [341, Section 18.8]. The single hyperparameter of this algorithm in its basic form is the number k of neighbors to consider. In general, small values of k tend to lead to overfitting, while very large values of k lead to underfitting [341, Section 18.8].

k nearest neighbors in a nutshell

Possible distance functions

Although the Euclidean distance is usually employed for finding the the k nearest neighbors, it is also possible to use other distance metrics. Russell and Norvig [341, Section 18.8] list the following distance metrics and their corresponding use cases:

- The *Euclidean distance* $d_E(\vec{x}, \vec{x}') = \sqrt{\sum_{i=1}^n (x_i - x'_i)^2}$ should be used if the features measure similar properties such as the WIDTH, HEIGHT, and LENGTH of physical objects. This relates to the usage of the Euclidean distance within the domains of the conceptual spaces frameworks (cf. Section 2.1.1).
- The *Manhattan distance* $d_M(\vec{x}, \vec{x}') = \sum_{i=1}^n |x_i - x'_i|$ should be preferred if the features correspond to separate aspects (e.g., AGE, WEIGHT, and GENDER of a person). This can be related to the usage of the Manhattan distance for combining different domains of the conceptual spaces framework (cf. Section 2.1.1).
- The *Hamming distance* $d_H(\vec{x}, \vec{x}') = |\{1 \leq i \leq n \mid x_i \neq x'_i\}|$ can be used if features are binary or nominal (i.e., if feature values can only be compared based on their identity, but no ordering, differences, or ratios can be meaningfully defined).
- The *Mahalanobis distance* $d_S(\vec{x}, \vec{x}') = \sqrt{(\vec{x} - \vec{x}')^T \Sigma^{-1} (\vec{x} - \vec{x}')}$ takes into account also the covariance between the features in the form of the covariance matrix Σ , and can be used if such a covariance is expected to play a role in the given dataset. As already discussed in Section 2.2.2 in the context of multivariate Gaussians, the Mahalanobis distance corresponds to the Euclidean distance in a space that has been transformed based on the covariance matrix.

Feature normalization

Russell and Norvig [341, Section 18.8] furthermore point out that the individual features should be normalized before applying the k nearest neighbor algorithm, in order to ensure that all features contribute equally to the overall distance.

Distance-weighted k NN

In addition to varying the distance metric, one can furthermore introduce a distinction between the different neighbors based on their distance to the novel example. In the resulting *distance-weighted* k nearest neighbor algorithm, closer neighbors receive a larger weight than neighbors that are further away, for instance by weighting them with the inverse square of their distance to the query point [295, Chapter 8]. Closer neighbors then receive a larger weight in the majority vote (for classification) or in the weighted average (for regression).

Properties of the k nearest neighbor algorithm

Especially the distance-weighted version of the k nearest neighbors algorithm is relatively robust to noise if k is chosen large enough [295, Chapter 8]. It has a relatively high capacity, but comes with a high computational cost and the requirement for a large training set [172, Chapter 5]. This high computational cost during classification is based on the requirement for finding the k nearest neighbors among all training examples, which makes it necessary to use efficient indexing structures such as trees or hashing [341, Section 18.8]. The k nearest

neighbors algorithm suffers heavily from the curse of dimensionality since proximity information becomes less reliable in high-dimensional spaces [7], and the nearest neighbors in such high-dimensional spaces are not very near [341, Section 18.8] (cf. Section 5.1.4).

Moreover, the k nearest neighbor algorithm is not capable of detecting the importance of the individual features, since all features are treated equally when computing the distance in the overall feature space [172, Chapter 5]. This becomes especially problematic, if many features are irrelevant to the task at hand. One can potentially solve this problem by introducing weights for the individual features, which need to be estimated by minimizing the classification error in a cross-validation setting [295, Chapter 8]. These feature weights would then resemble the salience weights used in the conceptual spaces framework (cf. Sections 2.1.2 and 2.3.1). Alternatively, one can try to remove irrelevant features by using feature selection techniques [295, Chapter 8], which will be introduced in Section 5.3.1.

Feature importance

The k nearest neighbor algorithm can be linked to the exemplar theory of concepts from cognitive psychology (cf. Section 1.1.1): The exemplar theory of concepts assumes that humans represent each concept by a set of representative examples [298, Chapter 4]. For instance, the DOG concept is represented by a certain number of specific dogs one recalls from one's personal experiences. The exemplar theory of concepts furthermore suggests that new observations are classified by computing their semantic similarity to all stored exemplars from all known concepts. The new observation is then classified based on the concept to which the most similar exemplar(s) belong.

Relation to the exemplar theory of concepts

One can easily see how the k NN algorithm implements such an exemplar-like classification model: In both cases, we simply store previous examples without abstracting from them. When classifying a new observation, we look at previously seen examples that are similar to this new observation. In the k NN algorithm, this is done by using the examples with smallest distance, making an interpretation of exemplar theory in the context of conceptual spaces quite straightforward: Observations are stored as points in the conceptual space, the combined metric (cf. Section 2.3.1) is used to find the k nearest neighbors of a query point (potentially influenced by salience weights), and classifications are made based on the labels of these k nearest neighbors. One should, however, note, that the decision boundaries of such an exemplar-based approach are not necessarily convex or star-shaped – which is, however, assumed by the conceptual spaces framework. Nevertheless, if we assume that conceptual regions are convex or star-shaped, then also the distribution of observed instances should be roughly convex or star-shaped, leading to approximately convex or star-shaped decision boundaries by the k NN classifier, respectively.

k NN and conceptual spaces

Furthermore, one can identify the prototype theory of concepts (cf. Section 1.1.1) with the special case of $k = 1$: One interpretation of the prototype theory states that each concept is represented by a "best example" [298, Chapter 3] which may be abstracted from multiple

Relation to the prototype theory of concepts

exemplars. If we assume that each concept is represented by such a prototypical instance and if the training set consists exactly of these prototypes, then a k nearest neighbor classification with $k = 1$ gives rise to a Voronoi tessellation of the space [341, Section 18.8] – which is the exact mechanism used by Gärdenfors to illustrate classification in conceptual spaces [179, Section 3.9] (cf. Section 1.2.2).

Outlook As one can see, a k nearest neighbor approach can be interpreted as one particular way of implementing the link between conceptual spaces and both the prototype and the exemplar theory of concepts. We will reconsider this link in Chapter 7, when discussing concept learning in conceptual spaces. Moreover, due to its simplicity, the k nearest neighbors algorithm is often considered as a target classifier in the area of metric learning (to be introduced in Section 5.3.3), where the main goal is to learn a useful distance metric on a given dataset, such that the accuracy of a simple downstream classifier is maximized. It may thus serve as a useful proxy when trying to optimize or analyze the structure of a similarity space. This aspect will resurface in Chapter 8, when discussing similarity spaces extracted from psychological data.

5.3 DIMENSIONALITY REDUCTION

*Feature engineering
and dimensionality
reduction*

In practical machine learning applications, it is often not possible to know a priori which features are necessary for good performance. In the process of *feature engineering*, one therefore tries to create many different features, which based on domain knowledge may be useful for the problem at hand. This can result in a relatively large number of candidate features, some of which may be different variants of the same underlying idea. As argued in Section 5.1.4, the curse of dimensionality states that successful generalization becomes increasingly more difficult in high-dimensional feature spaces. Therefore, one needs a general way of reducing the size of the feature vector – this procedure is called *dimensionality reduction*.

*The optimal number
of features*

It is in general unclear a priori, how many dimensions are necessary for an adequate representation of the data. One can treat the desired number of features as another hyperparameter of the overall system, and optimize it on the validation set. Another possibility is to use measures related to the specific dimensionality reduction technique in order to find a suitable cutoff value. Such a measure can, for instance, score the faithfulness of the compressed representation with respect to the original feature space.

*Dimensionality
reduction and
conceptual spaces*

One can relate the overall task of dimensionality reduction to the interplay of the subsymbolic layer and the conceptual layer (cf. Section 1.2.3). In his book [179], Gärdenfors describes the computational problem of constructing a conceptual space as follows:

The prime problem is that the information received by the receptors is too rich and too unstructured. What is needed

is some way of transforming and organizing the input into a mode that can be handled on the conceptual or symbolic level. This basically involves finding a more *economic* form of representation: going from the subconceptual to the conceptual level usually involves a *reduction of the number of dimensions* that are represented [...]. [179, Chapter 6.5] (emphasis in original)

If we assume that the initial features represent the raw perceptual input, then deriving a conceptual space can be framed as a dimensionality reduction process. One may, however, speculate that, for instance, in the case of visual input (represented as RGB values of individual pixels), a highly nonlinear and relatively complex transformation is needed in order to obtain a conceptual space. We will revisit such nonlinear approaches also in Chapter 6 in the context of artificial neural networks.

Nonlinear methods for perceptual input

In general, one can distinguish two approaches for dimensionality reduction: In *feature selection* (Section 5.3.1), one aims to identify an optimal subset of the given candidate features. This selected feature subset should be predictive for the target task while minimizing redundancy among the features [213]. Since the resulting description of the instances is directly based on the original candidate features, their interpretation is usually quite straightforward.

Feature selection finds the optimal feature subset

The second approach to dimensionality reduction is called *feature extraction* (Section 5.3.2) and is based on a projection of the original feature space \mathcal{X} into a lower-dimensional space \mathcal{Z} .¹⁰ Here, one is interested in defining novel features based on the existing set of candidate features. A popular example, which will be discussed in Section 5.3.2, is the *principal component analysis (PCA)*, which obtains novel features as linear combinations of the original candidate features. It creates those novel features in such a way, that they capture the directions of largest variance in the data. Since the features obtained by feature extraction techniques are based on a more or less complex transformation of the original features, their interpretation may be difficult.

Feature extraction projects the data into a lower-dimensional space

In this section, we will furthermore consider the area of *metric learning* (Section 5.3.3), which can be considered a special case of feature extraction: One aims to learn the parameters of a distance function, such that the resulting pair-wise distances between examples have desirable properties (e.g., being highly predictive of class membership).

Metric learning finds a distance function with desirable properties

5.3.1 Feature Selection

Feature selection is the process of selecting a small subset of features without transforming them, thus preserving their interpretation [213]. The objectives of feature selection include improved prediction performance by defying the curse of dimensionality, reduced storage need

Goals of feature selection

¹⁰ Please note that this is a somewhat inverse approach to the kernel trick discussed in the context of support vector machines (Section 5.2.3), where we looked for a higher-dimensional space in order to obtain linear separability

and training times, as well as a better understanding of the data [178]. Typically, one aims to keep the most relevant features while eliminating redundant ones [213]. As we will see below, most feature selection techniques judge feature relevance by accessing the classification or regression targets $y^{(j)}$, which means that they operate in a supervised manner. Guyon and Elisseeff [178] therefore emphasize, that just like hyperparameter optimization, also feature selection should take place on the validation set.

*Feature ranking:
score the features
individually*

Feature ranking can be considered as the most basic variant of feature selection. Here, a ranking criterion is chosen, which estimates the usefulness of an individual feature for the given task. In a regression task, for example, the linear correlation between the feature and the target can be used as a ranking criterion. This ranking criterion is then applied to each of the features independently to obtain a relevance score. The list of features is then sorted according to their respective scores and only the highest scoring features are kept [178]. Instead of defining an arbitrary threshold on the ranking criterion or a fixed number of features to keep, one can also introduce an artificial feature to the dataset, whose values are generated randomly based on a specified probability distribution. All features with a score lower than this artificial feature can then be safely discarded [178].

Correlation criteria

Guyon and Elisseeff [178] distinguish three broad types of ranking criteria: *Correlation criteria* such as Pearson's correlation coefficient r and the coefficient of determination R^2 (introduced in Section 5.1.2) measure the linear correlation between a given feature and the desired output. They can be naturally applied to regression tasks, but can also be employed for binary classification tasks, if the classes are encoded as $y^{(j)} = \pm 1$.

*Mutual information
as information-
theoretic criterion*

Information-theoretic criteria, on the other hand, treat both the feature and the target as random variables, and evaluate the dependence of their respective probability distributions. For example, the *mutual information* $MI(A, B)$ between two random variables A and B can be defined as follows based on the entropy $H(A)$:

$$MI(A, B) = H(A) - H(A|B) = H(B) - H(B|A) \quad (5.18)$$

Please recall from Section 5.2.4, that the entropy $H(A)$ measures the amount of uncertainty associated with a random variable A and is defined as $H(A) = \mathbb{E}_{A \sim \mathbb{P}(A)}[-\log(\mathbb{P}(A))]$ (cf. Equation 5.16). Mutual information therefore quantifies how much of this uncertainty about A is removed if we are allowed to observe B (or conversely, how much uncertainty about B is removed when observing A). While providing a more fine-grained way of analysis than correlation criteria, information theoretic criteria typically need to be approximated based on the examples from the given dataset, since the underlying true probability distributions are unknown in practice.

*Single feature
classifiers*

Finally, the performance of a *single feature classifier* can be used as a ranking criterion for classification tasks: This classifier has only access

to a single given feature and uses a simple threshold on that feature for classification. The accuracy of such a simple classifier can be used to rank the features according to their predictive power.

Feature ranking is very straightforward and can be easily scaled to very large feature sets. However, a combination of two or more features, which are useless by themselves, may still be highly predictive for the given task. Such highly predictive subsets of features may not be selected through feature ranking, which focuses exclusively on each feature's individual relevance [178]. Moreover, feature ranking approaches may select subsets of features, which are highly redundant.

Properties of feature ranking

These shortcomings of feature ranking approaches motivate the need for a more complex approach called *feature subset selection*, where the predictive power of complete feature subsets is considered. An exhaustive search through all possible feature subsets is, however, often computationally infeasible due to a combinatorial explosion [213]. Therefore, heuristic search strategies are often applied in practice, including greedy search, beam search, or more complex approaches such as simulated annealing and genetic algorithms [178].

Feature subset selection: score subsets of features

One can furthermore distinguish two search directions: In *forward selection*, one starts with an empty set of features and iteratively adds features, while *backward elimination* starts from the full feature set and iteratively eliminates irrelevant features [178, 213]. Guyon and Elisseeff [178] note that small perturbations of the experimental conditions can lead to different results for feature subset selection approaches. They therefore propose to use a *bootstrapping* approach, which applies the same feature selection methodology on multiple subsets of the training examples and uses the union of the selected feature subsets as a final feature set for training the classifier. Moreover, depending on the way in which the feature subsets under consideration are being scored, one can distinguish three types of feature subset selection approaches, namely, filter methods, wrapper methods, and embedded methods.

Search direction and bootstrapping

Filter methods use a performance measure, which is independent from the machine learning algorithm, that will be applied later on. They are thus related to the feature ranking approach, and in principle, similar performance measures (such as correlation or mutual information) can be used [213]. However, unlike feature ranking, filter methods always consider the whole subset of features in their evaluation. Since filter methods are based on heuristic scoring functions, they are generally quite fast and provide a selection of features, that is independent of the machine learning algorithm to be used later on [178]. This reduces the risk of overfitting and allows one to easily replace the model.

Filter methods use heuristics for scoring

Wrapper methods on the other hand use the performance of a trained model on the validation set for assessing the usefulness of a feature subset [178, 213]. Since they need to train the actual model for each feature subset under consideration, they are often considerably slower than filter methods and tend to be biased towards the machine learning algorithm being used [213]. Both of these effects can, however, be counteracted to some degree by using efficient search strategies such as

Wrapper methods are based on model predictions

greedy search, and by employing fast machine learning algorithms (e.g., using decision trees rather than SVMs) [178, 213]. In practice, wrapper methods often tend to yield a better performance of the final model than filter methods [213].

*Embedded methods
look inside the model*

Finally, *embedded methods* perform feature selection as part of the training process, and are typically specific to the given machine learning algorithm, such as decision trees, random forests, and logistic regression [178, 213]. Generally, embedded methods tend to be more efficient than wrapper methods, because the model only needs to be trained once [178]. Since the implicit objective function of feature selection is to maximize the prediction performance while minimizing the number of features [178], embedded methods can also be related to regularization schemes such as the L_1 penalty of a lasso regressor which incites many weights to become zero [213] (cf. Section 5.2.1).

*Hybrid methods
combine multiple
approaches*

Of course, multiple approaches to feature selection can be combined with each other, leading to so-called *hybrid methods*. Jović et al. [213] for example propose to first apply feature ranking or a filter method to reduce the number of candidate features, and then to use a wrapper method to select the optimal feature subset. They argue that this approach is both efficient and effective since it combines the advantages of filter methods and wrapper methods. Also Guyon and Elisseeff [178] propose a similar approach, namely to use a simple linear predictor as a filter method, followed by a non-linear predictor as a wrapper method on the remaining features.

*Creating a
conceptual space
with feature selection*

Feature selection methods can in principle be used to construct a conceptual space from a given machine learning dataset as demonstrated by Banaee et al. [26, 27] (cf. Section 1.2.5). They assume, that the given dataset M (e.g., about leaves of different plant species) consists of the following three components:

- The set of possible class labels $Y = \{Y_1, \dots, Y_c\}$ (e.g., NERIUM OLEANDER, TILIA TOMENTOSA, etc.).
- The set of features $X = \{X_1, \dots, X_n\}$, where each feature comes with a linguistic name and a range of possible values (which can be a list of categories or an interval of real numbers). For example, features could include the ASPECT RATIO of the leaf and the ECCENTRICITY of its bounding ellipse.
- The set of observations $D = \{o^{(j)} : (\vec{x}^{(j)}, y^{(j)}) \mid 1 \leq j \leq N\}$, where $\vec{x}^{(j)}$ is a feature vector (with one entry for each of the features from X) and $y^{(j)}$ is the class label assigned to this observation.

Banaee et al. assume that the concepts in the resulting conceptual space correspond to the different classes from Y . Moreover, the dimensions spanning the conceptual space are a subset of X . In order to identify this subset of X , they use a filter method which considers the usefulness of the features X for classifying the observations D

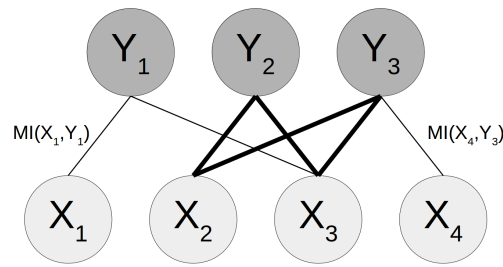


Figure 5.12: Identifying domains as bicliques in the feature selection approach by Banaee et al. [26, 27].

with respect to the class labels Y . When picking the optimal subset of features based on the results of this filter method, they also group them into domains in order to derive a valid structure for a conceptual space.

In order to measure the usefulness of a feature X_i (e.g., ECCENTRICITY) to a class label Y_k (e.g., NERIUM OLEANDER), they use mutual information (MI) as defined in Equation 5.18: $MI(X_i, Y_k)$ should be high, while the mutual information $MI(X_i, X_{i'})$ between the feature X_i and any previously selected feature $X_{i'}$ (e.g., ASPECT RATIO) should be low. For each of the classes in Y , Banaee et al. thus derive a ranking of all features from X based on their MI score. They only consider the top t features as being relevant for the respective class. Banaee et al. also note that if properly normalized, the MI scores can be used as salience weights representing the importance of the respective dimension X_i for the respective concept Y_k .

After having selected a set of candidate features for each class, Banaee et al. group them into domains by using the following procedure, illustrated in Figure 5.12: They create a graph, where each candidate feature X_i (e.g., ECCENTRICITY) and each class Y_k (e.g., NERIUM OLEANDER) is represented as a node. An edge is introduced between X_i and Y_k if X_i is among the t highest-ranking candidate features for Y_k as determined in the previous step. This edge is then weighted with the respective MI score $MI(X_i, Y_k)$. Please note that this graph is *bipartite*, i.e., there are two groups of nodes (features and classes), and all edges involve two nodes from different groups.

Banaee et al. now identify domains by looking for bicliques in this graph. A *biclique* in a bipartite graph is a subset of nodes where each node from one group is connected to all nodes from the other group. In the example from Figure 5.12, a trivial biclique is for instance given by $\{X_1, Y_1\}$, and a maximal biclique consists of $\{X_2, X_3, Y_2, Y_3\}$. For each biclique, Banaee et al. define a score based on the biclique's size and the weights on its edges. They then iteratively select the biclique with the largest score, define a domain based on its features, and remove the corresponding feature nodes and edges from their overall graph. For instance, if the biclique $\{\text{ASPECT RATIO, ECCENTRICITY, NERIUM OLEANDER, TILIA TOMENTOSA}\}$ is the maximum biclique with the highest score, a new domain will be defined based on the dimensions ASPECT RATIO and ECCENTRICITY and these dimensions will not be considered for any other domains. Please note that the class nodes (in this example NERIUM OLEANDER and TILIA

Mutual information as scoring criterion

Creating a bipartite feature-class graph

Domains as maximal bicliques

TOMENTOSA) are not removed, and are thus allowed to be part of multiple bicliques, i.e., to be defined on multiple domains. The algorithm by Banaee et al. selects bicliques until each class has been part of at least one biclique.

Intuition behind the biclique approach

Banaee et al. motivate their biclique approach as follows: A domain in the conceptual spaces framework is defined as a set of dimensions, which inherently belong together from a semantic point of view (cf. Sections 1.2.1 and 2.1.1). Banaee et al. assume that a pair of dimensions is semantically related if they are relevant for the same concepts. By selecting a biclique, one therefore selects a set of features which are highly relevant for the same set of concepts, and can hence be expected to contain related pieces of information.

Constructing conceptual regions

A concept for a given class Y_k (e.g., NERIUM OLEANDER) can now be constructed by using all observations from D , which are labeled with Y_k , projecting them into the conceptual space (i.e., taking their feature values with respect to the selected dimensions), and constructing their convex hull. Please note that Banaee et al. project each observation $o^{(j)} = (\vec{x}^{(j)}, y^{(j)})$ only onto the domains relevant for the concept $y^{(j)}$. This means, that some concepts may be defined only on a single domain, while others may be defined on multiple domains. The salience weights associated with the concept are based on the association strengths obtained in the filtering step.

Case studies

Banaee et al. provide two case studies illustrating the application of their proposed procedure. Both for a dataset of leaves and for a dataset of time series, they found that the domains constructed by their approach grouped together similar features and could thus be given a meaningful overall interpretation. For instance, on the leaves dataset, one domain contained the features ASPECT RATIO, ELONGATION, and ECCENTRICITY, all of which can be used to describe whether a given leaf has an elongated or blob-like shape. On the time series data, the two domains which emerged could be described as capturing the trend direction of the pattern (rising or falling) and the overall shape of the pattern (smooth vs. fluctuating), respectively.

Limitations

Overall, the approach by Banaee et al. mainly differs from standard feature selection approaches by introducing a domain structure on the selected features. Although their results indicate that the resulting domains seem to be interpretable, it is unclear whether this can be generally expected. Moreover, this approach assumes that all features needed to construct a conceptual space are already given a priori and that one simply needs to select the right subset of features. For many perceptual domains like shape, this might not be true. Banaee et al. furthermore acknowledge that based on the dataset being used, the quality dimensions of the resulting conceptual space do not necessarily reflect human perception. One can furthermore criticize that concepts are only defined on the domains, in whose construction they were involved. While this may be an efficient way of storing relevant information, it blurs the distinction between properties and concepts from the conceptual spaces framework (cf. Sections 1.2.1 and 2.1.2). Finally, when defining concepts, cross-domain correlations are not taken into account.

Nevertheless, the work by Banaee et al. establishes an important link between dimensionality reduction and conceptual spaces.

5.3.2 Feature Extraction

In contrast to feature selection, *feature extraction* attempts to reduce the dimensionality of the feature space by constructing a small set of novel features, which are typically a combination of the original ones [178, 213]. In general, feature extraction can be interpreted as an unsupervised learning task, where one tries to find a better representation of the given data, which preserves the main information while being less complex [172, Chapter 5]. The more specific goal of feature extraction depends on the application scenario, which might for example put more emphasis on a faithful reconstruction of the original features, or on the usefulness of the novel representation for making accurate predictions in a downstream task [178].

Goals of feature extraction

Goodfellow et al. [172, Chapter 5] distinguish three generally desirable properties of a learned representation: In general, a *lower-dimensional representation* of the data is desired in order to fight the curse of dimensionality (cf. Section 5.1.4). Moreover, one may also be interested in a *sparse representation*, where most entries of the feature vector are zero for most inputs. This can, for example, be achieved by clustering the data points in the original feature space and replacing them with their respective cluster ID [178]. We will introduce such clustering techniques in Chapter 7. Also an *independent representation* may be of interest, where the sources of variation in the data distribution are disentangled in such a way that the constructed features are statistically independent. The topic of disentangled representations will be discussed in more depth in Chapter 6. As Goodfellow et al. [172, Chapter 5] note, these different desiderata may be partially conflicting, such as a sparse representation being often quite high-dimensional in order to prevent dramatic information loss. In some contexts, they can, however, also be addressed simultaneously – for example, removing redundancy yields both a lower-dimensional and a more independent representation.

Desirable properties for learned representations

While there exists a wide variety of feature extraction methods, we will limit our discussion in this section to one of the most well-known algorithms, namely principal components analysis (PCA). Other variants of feature extraction will be discussed later, including metric learning (Section 5.3.3), autoencoders and other neural networks (Chapter 6), and multidimensional scaling (Chapter 8).

Principal components analysis (PCA) provides a lower-dimensional representation of a given dataset by constructing novel features as a linear combination of the original features [71, Chapter 24]. These novel features are chosen in such a way, that they have no correlation with each other [172, Chapter 5], and that they capture most of the variance in the data [438], assuming that the directions of largest variance in the data correspond to the directions of largest information [438]. The goals

Goals of principal components analysis

of PCA include simplifying the description of the dataset, compressing the size of the original design matrix, and analyzing the underlying structure of the data [2, 438].

Mathematical properties

Mathematically speaking, PCA provides an orthogonal, linear transformation of the data which minimizes the reconstruction error as measured by the mean squared error [172, Chapter 5]. This transformation can be expressed by a simple matrix multiplication [172, Chapter 2], which corresponds to a projection from the original n -dimensional feature space \mathcal{X} onto a t -dimensional hyperplane (i.e., subspace) \mathcal{Z} [438].

Standardizing as preprocessing step

Before applying PCA to a design matrix $X \in \mathbb{R}^{N \times n}$, this matrix is usually standardized by subtracting the column averages and dividing the columns by their standard deviation [2, 79, 438]. While the centering step is mainly for mathematical convenience, the normalization of the columns to standard variance ensures that all features have the same influence in the subsequent processing, even if they are measured with different units [2, 438].

Searching for the direction of largest variance

In our derivation of PCA, we follow Bro and Smilde [79] by first considering the special case of $t = 1$. This means that we try to find a single latent variable Z which expresses most of the variance in the design matrix X . Let us denote by X_i the random variable corresponding to the i th feature (cf. Section 5.2.4) and by \vec{x}_i the values of this feature in the design matrix, i.e., the i th column of the design matrix X . In a similar way, the latent variable Z can be expressed as an N -dimensional vector \vec{z} containing one feature value for each example. As stated above, in the context of PCA, we are interested in expressing \vec{z} as a linear combination of the different \vec{x}_i , i.e., $\vec{z} = w_1 \vec{x}_1 + \dots + w_n \vec{x}_n = X \vec{w}$ for a given weight vector $\vec{w} \in \mathbb{R}^n$. Since the length of the vector \vec{w} does not play a role for the transformation, we simply constrain it to be $\|\vec{w}\| = 1$. Moreover, we would like to choose \vec{z} in such a way that it captures as much of the variance in X as possible. Hence, we need to solve the following optimization problem:

$$\vec{w}^* = \arg \max_{\vec{w} \in \mathbb{R}^n, \|\vec{w}\|=1} \text{Var}[\vec{z}] = \arg \max_{\vec{w} \in \mathbb{R}^n, \|\vec{w}\|=1} \text{Var}[X \vec{w}]$$

Variance and centering

The *variance* $\text{Var}[Z]$ of a random variable Z is defined as $\mathbb{E}_{Z \sim \mathbb{P}(Z)}[(Z - \mathbb{E}_{Z \sim \mathbb{P}(Z)}[Z])^2]$, i.e., the expected squared difference from its expected value. Since we centered our design matrix X in the preprocessing step, the mean value of each original feature X_i is zero, i.e., $\mathbb{E}_{X_i \sim \mathbb{P}(X_i)}[X_i] = 0$. One can easily see, that therefore $\mathbb{E}_{Z \sim \mathbb{P}(Z)}[Z] = \mathbb{E}_{X_i \sim \mathbb{P}(X_i)}[\sum_{i=1}^n w_i X_i] = \sum_{i=1}^n w_i \cdot \mathbb{E}_{X_i \sim \mathbb{P}(X_i)}[X_i] = 0$. Therefore, $\text{Var}[Z] = \mathbb{E}_{Z \sim \mathbb{P}(Z)}[Z^2]$. Since $\mathbb{E}_{Z \sim \mathbb{P}(Z)}[Z^2]$ can be approximated by taking the average of all squared values of Z over the dataset, we can maximize $\mathbb{E}_{Z \sim \mathbb{P}(Z)}[Z^2]$ by maximizing $\sum_{j=1}^N z_j \cdot z_j = \vec{z}^T \vec{z}$. We can thus rewrite our optimization problem as follows:

$$\vec{w}^* = \arg \max_{\vec{w} \in \mathbb{R}^n, \|\vec{w}\|=1} \text{Var}[Z] = \arg \max_{\vec{w} \in \mathbb{R}^n, \|\vec{w}\|=1} \vec{z}^T \vec{z} = \arg \max_{\vec{w} \in \mathbb{R}^n, \|\vec{w}\|=1} \vec{w}^T X^T X \vec{w}$$

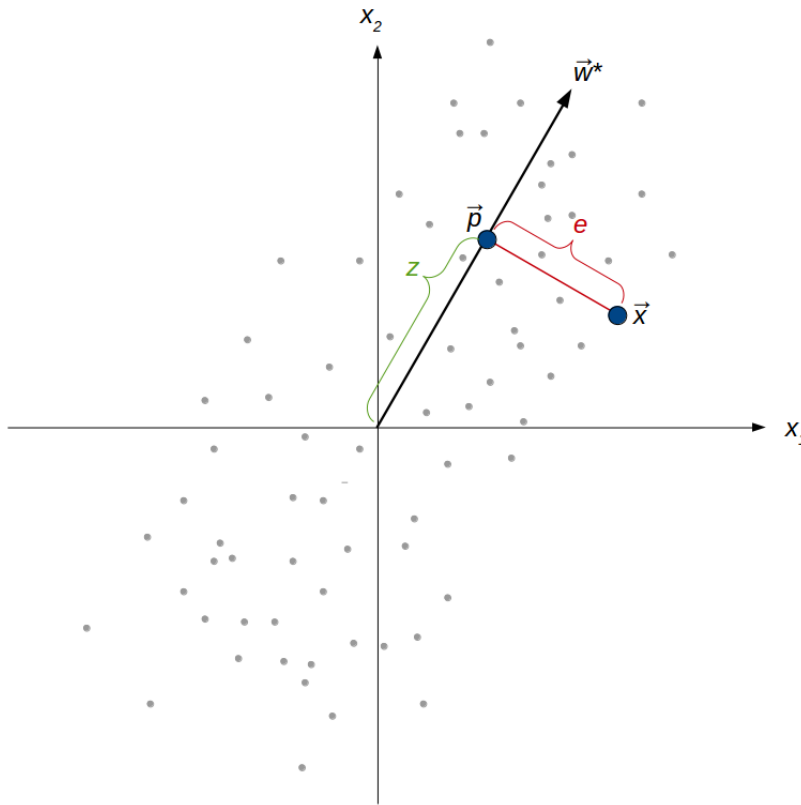


Figure 5.13: Illustration for finding the vector \vec{w}^* by maximizing the variance of $X\vec{w}$ for a two-dimensional feature space.

In order to see how \vec{w}^* can be found, we need to introduce the concepts of eigenvalue and eigenvector. An *eigenvector* \vec{q} of a given matrix A is a vector that gets stretched by a factor of λ (its so-called *eigenvalue*) if multiplied with A , i.e., $A\vec{q} = \lambda\vec{q}$. In our case, we consider $A = X^T X$, and note that $\vec{w}^T X^T X \vec{w}$ can be maximized by choosing the eigenvector $\vec{w}^* = \vec{q}$ of $X^T X$ with the largest associated eigenvalue λ : Since our solution is constrained to having unit length, we know that $(\vec{w}^*)^T \vec{w}^* = 1$. Overall, we thus get that $(\vec{w}^*)^T X^T X \vec{w}^* = (\vec{w}^*)^T \lambda \vec{w}^* = \lambda (\vec{w}^*)^T \vec{w}^* = \lambda$. Luckily, finding the largest eigenvalue of a given matrix along with its associated eigenvector is a standard problem in linear algebra, for which already various efficient algorithms have been developed.

Hence, in order to find a new feature Z which is a linear combination of the existing features X_i , we simply need to identify the eigenvector \vec{w}^* associated with the largest eigenvalue of $X^T X$. We can then translate X into \vec{z} by right-multiplying it with \vec{w}^* , i.e., $\vec{z} = X\vec{w}^*$. One can furthermore show, that the optimal reconstruction of X based on \vec{z} can be computed as $X \approx \vec{z}(\vec{w}^*)^T$ [79]. It is also possible to derive the optimal weight vector \vec{w}^* by minimizing the squared Euclidean distance between X and $\vec{z}(\vec{w}^*)^T$ (i.e., the reconstruction error) [172, Chapter 2].

Figure 5.13 illustrates the geometric interpretation of our discussion so far using a two-dimensional feature space. As one can see, the given data points are clearly correlated. Through the procedure described

Eigenvectors and eigenvalues

Transformations between old and new features

Geometric interpretation

above, we can discover the weight vector \vec{w}^* , which points in the direction of greatest variance. For a given data point \vec{x} , the value of the new feature z can be computed as $z = \vec{x} \cdot \vec{w}^* = x_1w_1^* + x_2w_2^*$. This corresponds to the distance of its projection \vec{p} onto the weight vector \vec{w}^* from the origin. The best reconstruction of the original data point \vec{x} based only on the value z is given by $z\vec{w}^* = \vec{p}$. It has an absolute reconstruction error of e , which denotes the distance between \vec{x} and \vec{p} .

Searching for multiple novel features

In most practical cases, we are, however, not interested in only finding a single feature Z corresponding to the direction of largest variance in the data. Rather, we aim to find multiple features Z_1, \dots, Z_t (represented as a matrix Z of transformed coordinates), which can all be expressed as linear combinations of the original features X_1, \dots, X_n (represented by the original design matrix X). When applying a PCA, we furthermore require that the directions $\vec{w}^{(k)}$ corresponding to the individual features Z_k shall be orthogonal to each other. This means, that we require $\forall k \neq l : (\vec{w}^{(k)})^T \cdot \vec{w}^{(l)} = 0$.

Eigenvalue decomposition

When determining the optimal weight vector \vec{w}^* for the direction with the most variance in the data, we used the largest eigenvalue of $X^T X$. One can generalize this approach to multiple target features by using the t largest eigenvalues of $X^T X$. They (and their corresponding eigenvectors) can be found by computing the *eigenvalue decomposition* of $X^T X$. In general, the eigenvalue decomposition of any symmetric matrix $A \in \mathbb{R}^{n \times n}$ is defined as follows [71, Chapter 7]:

$$A = Q \Lambda Q^T \tag{5.19}$$

Properties of the matrices

Here, Λ is a diagonal matrix of eigenvalues, and Q is a matrix of eigenvectors, both being of size $n \times n$. Each column of Q corresponds to one eigenvector $\vec{q}^{(k)}$, and the corresponding diagonal entry λ_k of Λ to its associated eigenvalue. Moreover, Q is orthonormal, which means that $QQ^T = Q^T Q = I$. This especially implies that $\forall k \in \{1, \dots, n\} : \|\vec{q}^{(k)}\| = (\vec{q}^{(k)})^T \cdot \vec{q}^{(k)} = 1$. If A is symmetric, then its eigenvalues and eigenvectors are guaranteed to be real-valued, and the eigenvectors are orthogonal to each other (i.e., $\forall k \neq l : (\vec{q}^{(k)})^T \cdot \vec{q}^{(l)} = 0$).

Approximation of the original matrix

By rewriting Equation 5.19, we can show that the matrix A can be written as a sum of individual matrices, which are based on the different eigenvectors and eigenvalues:

$$\begin{aligned} A &= Q \Lambda Q^T = \begin{pmatrix} \lambda_1 \vec{q}^{(1)} & \lambda_2 \vec{q}^{(2)} & \dots & \lambda_n \vec{q}^{(n)} \end{pmatrix} \begin{pmatrix} (\vec{q}^{(1)})^T \\ (\vec{q}^{(2)})^T \\ \vdots \\ (\vec{q}^{(n)})^T \end{pmatrix} \\ &= \lambda_1 \vec{q}^{(1)} (\vec{q}^{(1)})^T + \lambda_2 \vec{q}^{(2)} (\vec{q}^{(2)})^T + \dots + \lambda_n \vec{q}^{(n)} (\vec{q}^{(n)})^T \end{aligned} \tag{5.20}$$

Typically, one sorts the eigenvalues in descending order, such that $\lambda_1 > \lambda_2 > \dots$. The individual $\lambda_k \vec{q}^{(k)} (\vec{q}^{(k)})^T$ in Equation 5.20 are then ordered in such a way, that the first matrices have a larger impact on the

overall result. We can thus use Equation 5.20 to approximate A based on the first t eigenvalues and their associated eigenvectors.

Thus, in order to compute a PCA of a given design matrix X , we can simply compute the eigenvalue decomposition of $X^T X$ and use the eigenvectors associated with the t largest eigenvalues as weight vectors $\bar{w}^{(k)}$ [71, Chapter 24]. As noted above, using the t largest eigenvalues of $X^T X$ and their associated eigenvalues gives rise to an approximation of $X^T X$, whose approximation error E (in analogy to the discussion of Figure 5.13) is based on the terms $\lambda_k \bar{q}^{(k)} (\bar{q}^{(k)})^T$ for $k > t$, i.e., the eigenvectors not considered by the PCA.

PCA based on eigenvalue decomposition

Instead of using an eigenvalue decomposition of $X^T X$, one can also compute a PCA by using a *singular value decomposition* (SVD) of the design matrix $X \in \mathbb{R}^{N \times n}$. A singular value decomposition is a generalization of the eigenvalue decomposition to non-square matrices. The SVD of X can be written as follows (cf. [71, Chapter 7] and [2]):

Singular value decomposition

$$X = U \Phi V^T \quad (5.21)$$

Here, $U \in \mathbb{R}^{N \times t}$ is the matrix of left singular values, $V \in \mathbb{R}^{n \times t}$ is the matrix of right singular values, and $\Phi \in \mathbb{R}^{t \times t}$ is a diagonal matrix of singular values ϕ_k . Moreover, both U and V are orthonormal. By inserting the singular value decomposition of X (Equation 5.21) into the eigenvalue decomposition of $X^T X$ (Equation 5.19), one can see how the two are related (using that $U^T U = I$):

Properties of the matrices

$$\begin{aligned} Q \Lambda Q^T = X^T X &= (U \Phi V^T)^T (U \Phi V^T) \\ &= (V \Phi U^T) (U \Phi V^T) = V \Phi^2 V^T \end{aligned} \quad (5.22)$$

Thus, V contains the eigenvectors of $X^T X$ and the squared singular values ϕ_k correspond to the eigenvalues λ_k of $X^T X$. Understanding the PCA as a singular value decomposition of X can help to see how the transformed coordinates can be computed in an alternative way. In order to do so, let us multiply both sides of Equation 5.21 with V :

Relation between SVD and eigenvalue decomposition

$$Z = X V = U \Phi V^T V = U \Phi \quad (5.23)$$

Since Φ is a diagonal matrix and U is orthonormal, U contains a normalized version of the column vectors of the matrix of transformed coordinates $Z = X V$, while Φ represents the respective lengths of the column vectors of Z [438]. Thus, the individual singular values ϕ_k quantify to which extent the data points vary along the direction $\bar{v}^{(k)}$. More specifically, $\lambda_k = \phi_k^2$ describes how much of the overall variance is captured by the principal component $\bar{v}^{(k)}$ [71, Chapter 24].

Interpreting the singular values

One can furthermore show that the individual principal components are uncorrelated [172, Chapter 5]: The unbiased sample covariance matrix associated with the design matrix X is defined as $\text{Var}[X] = \frac{1}{N-1} X^T X$. Using Equations 5.22 and 5.23, we can thus derive

Principal components are uncorrelated

the following formula for $\text{Var}[Z]$, i.e., the covariance matrix of the transformed feature space:

$$\begin{aligned}\text{Var}[Z] &= \frac{1}{N-1} Z^T Z = \frac{1}{N-1} (X V)^T (X V) \\ &= \frac{1}{N-1} V^T X^T X V = \frac{1}{N-1} V^T (V \Phi^2 V) V^T \\ &= \frac{1}{N-1} \Phi^2\end{aligned}$$

Since Φ is a diagonal matrix, also $\text{Var}[Z]$ must be a diagonal matrix. This means that all off-diagonal entries and hence all correlations between principal components must be zero. While the PCA is able to eliminate correlations between features, it fails to address more complex, nonlinear dependencies [172, Chapter 5].

PCA as rotation of the feature space

For $t = n$ (meaning that no dimensionality reduction takes place), applying the transformation matrix V to X can be interpreted as rotating the basis of the feature space [172, Chapter 5]. To see why this is true, remember that V is orthonormal, i.e., $VV^T = V^T V = I$. If we apply this transformation matrix V to two points $\vec{x}^{(j_1)}$, $\vec{x}^{(j_2)}$, the scalar product of their transformations $V\vec{x}^{(j_1)}$, $V\vec{x}^{(j_2)}$ is identical to the scalar product of the original points:

$$\begin{aligned}\langle V\vec{x}^{(j_1)}, V\vec{x}^{(j_2)} \rangle &= (V\vec{x}^{(j_1)})^T (V\vec{x}^{(j_2)}) = (\vec{x}^{(j_1)})^T V^T V \vec{x}^{(j_2)} \\ &= (\vec{x}^{(j_1)})^T \vec{x}^{(j_2)} = \langle \vec{x}^{(j_1)}, \vec{x}^{(j_2)} \rangle\end{aligned}$$

This means that angles between points are preserved by the transformation V . In general, matrix multiplication acts as an affine transformation, which can include translation, scaling, reflection, rotation and shear mappings. However, only rotations and reflections preserve angles between vectors. Since we can always invert the direction of the individual principal components, we can thus represent the transformation always as a rotation of the original coordinate system.

Heuristics for choosing the number of dimensions

The most important hyperparameter of the PCA is the number t of dimensions in the resulting feature space \mathcal{Z} . Clearly, one faces a trade-off between compactness and expressiveness. There are, however, some general heuristics for choosing t : If the error of measurement of the data in X is known, one can choose t in such a way that the variance of the residual $E = X - ZV^T$ is of the same size as this error of measurement [438]. Another heuristic proposes to keep only components, whose eigenvalue is larger than the average eigenvalue [2]. Moreover, one can sort the eigenvalues from Λ in descending order and look for a large "gap" between neighboring values in this sequence – if the first t eigenvalues are relatively large, but all eigenvalues from $t + 1$ to n are quite small, then the first t components should explain most of the meaningful variance, while the remaining components can be attributed to noise [2]. Finally, if the output of the PCA is used as a feature space for a classifier, the number t of principal components can

be treated as a hyperparameter of the overall system and be optimized based on classification performance on the validation set [438].

Overall, a PCA does not only allow to find a more compact representation of a given dataset, but it can be used for additional tasks as well. Most importantly, also novel data points can be projected into the transformed space by applying the transformation matrix V [2]. Furthermore, the reconstruction error for such novel data points can be compared to the average reconstruction error with respect to the design matrix X to determine, whether the new data point belongs to the same distribution as the training set [438]. A similar approach can also be used to detect outliers, which are expected to have a large reconstruction error [438].

Additional usages of PCA

PCA has two crucial drawbacks, that limit its usefulness in the context of conceptual spaces: Firstly, the extracted dimensions correspond to the directions of greatest variance in the original feature space, but are not necessarily interpretable [179, Section 7.1.4]. Interpretable dimensions, however, are a corner stone of the conceptual spaces framework. Secondly, the transformation of the original feature space into the compressed space is only a *linear* one. PCA thus implicitly assumes that the original feature space is almost perfect and that we only need to combine features and remove some of them in order to find an optimal representation. If the given input space requires a nonlinear transformation, a PCA is not expected to provide useful results. When dealing with raw pixel information from images, which we would like to transform into a compact representation for the domain of SHAPES, such a linear transformation is probably not sufficient.

PCA and conceptual spaces

While a PCA is thus not directly applicable to obtaining the dimensions of a conceptual space, the fact, that it extracts the directions of largest variance, can be useful for estimating correlations between domains and dimensions. This aspect will be discussed in more detail in Chapter 7 in the context of learning conceptual regions. Moreover, the eigenvalue decomposition used by the PCA will also play an important role for multidimensional scaling in Chapter 8.

Outlook

5.3.3 Metric Learning

Metric learning is the task of learning a useful distance function, which represents the dissimilarity of two complex inputs [236]. This learned distance function is often used for nearest neighbor classifiers (like the k NN algorithm introduced in Section 5.2.5), which explicitly measure the distance between a new observation and examples from the training set in order to make a classification. Especially in the context of few shot learning, where one assumes that only a very limited number of training examples is available, having a good distance metric can considerably improve classification performance [427]. In many cases, the distance function is derived by transforming the feature space \mathcal{X} and using the unweighted Euclidean metric in the transformed space

Goals of metric learning

\mathcal{Z} .¹¹ If this transformation results in a lower-dimensional space, then dimensionality reduction takes place. The resulting low-dimensional embedding space \mathcal{Z} can then not only be useful for nearest-neighbor classifiers, but can also be used by other machine learning models in order to counteract the curse of dimensionality (cf. Section 5.1.4). Moreover, this low-dimensional space can also give some insights about the organization of the originally high-dimensional data [345].

Semi-formal definition

In general, metric learning can be viewed as a special case of feature extraction. The metric learning problem can be formalized as follows [236]: Given an input distance function $d(\vec{x}^{(j_1)}, \vec{x}^{(j_2)})$ and supervision information regarding the ideal distance between data points, construct a new distance function $\hat{d}(\vec{x}^{(j_1)}, \vec{x}^{(j_2)})$, which reflects the supervision information better than the original distance function $d(\vec{x}^{(j_1)}, \vec{x}^{(j_2)})$.

Types of constraints

The supervision information typically gives constraints on the distances between pairs of data points based on their class labels. Kulis [236] distinguishes two main types of supervision constraints:

Absolute similarity and dissimilarity constraints

When using *absolute constraints*, we define a set S consisting of pairs of data points, which should be similar, and a set D of pairs of data points, which should be dissimilar. For instance, we can require that all data points with the same class label should be similar, while data points with different class labels should be dissimilar. This can be expressed as follows, using two threshold constants $u < l$ for the respective distances:

$$\forall (\vec{x}^{(j_1)}, \vec{x}^{(j_2)}) \in S : \hat{d}(\vec{x}^{(j_1)}, \vec{x}^{(j_2)}) \leq u \quad (5.24)$$

$$\forall (\vec{x}^{(j_1)}, \vec{x}^{(j_2)}) \in D : \hat{d}(\vec{x}^{(j_1)}, \vec{x}^{(j_2)}) \geq l \quad (5.25)$$

Relative distance constraints

For *relative constraints*, we define a set R consisting of triples $(\vec{x}^{(j_1)}, \vec{x}^{(j_2)}, \vec{x}^{(j_3)})$ of data points, such that the data point $\vec{x}^{(j_1)}$ is more similar to $\vec{x}^{(j_2)}$ than to $\vec{x}^{(j_3)}$. This can again be based on class labels by requiring that $\vec{x}^{(j_1)}$ and $\vec{x}^{(j_2)}$ belong to the same class, while $\vec{x}^{(j_3)}$ belongs to a different class. It is, however, also possible to collect such relative distance constraints from humans. Elicitation methods for such ratings will be discussed in Chapter 8 in the context of multidimensional scaling. The relative distance constraints can be expressed as follows, using a nonnegative margin s (which is often set to one):

$$\forall (\vec{x}^{(j_1)}, \vec{x}^{(j_2)}, \vec{x}^{(j_3)}) \in R : \hat{d}(\vec{x}^{(j_1)}, \vec{x}^{(j_2)}) < \hat{d}(\vec{x}^{(j_1)}, \vec{x}^{(j_3)}) + s \quad (5.26)$$

Classification of approaches

One can furthermore distinguish different variants of metric learning based on the type of transformation being used. Kulis [236] distinguishes different approaches based on two dimensions: Global versus local methods, and linear versus nonlinear methods. Wang et al. [427] consider an additional dimension, which distinguishes task-specific from task-general models.

¹¹ This is a somewhat inverse approach to the kernel functions used for support vector machines (cf. Section 5.2.3): When applying the kernel trick, we use a kernel function as a shortcut to computing the inner product in a high-dimensional transformed space, while in metric learning, we use a standard distance metric in a low-dimensional transformed space in order to define a new metric in the original feature space.

Global methods transform the overall feature space by using a single transformation function ϕ [236]. The learned distance is then defined as the input distance on the transformed data points, i.e., $\hat{d}(\vec{x}^{(j_1)}, \vec{x}^{(j_2)}) = d(\phi(\vec{x}^{(j_1)}), \phi(\vec{x}^{(j_2)}))$. *Local methods* on the other hand learn several distance functions (or several transformation functions ϕ), e.g., one per class or cluster in the dataset.

Global vs. local methods

Linear methods apply a linear transformation to the feature space, which can be represented by a simple matrix transformation $\phi(\vec{x}) = G\vec{x}$ with some matrix G , whose parameters need to be optimized. This corresponds to learning weights for scaling and rotating the data (cf. the geometric interpretation of a PCA in Section 5.3.2). *Nonlinear methods* on the other hand make use of nonlinear transformations, which are in general more powerful, but may also be more prone to overfitting.

Linear vs. nonlinear methods

Wang et al. [427] consider the application of metric learning in few shot classification, i.e., where only very small datasets are available. They propose a distinction of the resulting transformation models into task-specific and task-invariant models. *Task-specific models* are trained exclusively on the specific few shot classification task at hand. On the other hand, *task-invariant models* are trained on related classification tasks with large-scale datasets and are applied to the few shot learning task without further modifications. Wang et al. also mention *hybrid models*, which combine task-invariant with task-specific information by first being optimized on large-scale datasets of related tasks, and then being fine-tuned on the specific few shot learning task at hand.

Task-specific vs. task-invariant models

Let us first consider *linear methods*. Here, the transformation function is based on a simple matrix multiplication $\phi(\vec{x}) = G\vec{x}$ with a matrix G and the Euclidean metric is typically used to measure distances in the transformed space. One can show that the resulting metric $\hat{d}(\vec{x}^{(j_1)}, \vec{x}^{(j_2)}) = d_E(G\vec{x}^{(j_1)}, G\vec{x}^{(j_2)})$ is equivalent to a generalized form of the Mahalanobis distance, where $\hat{d}(\vec{x}^{(j_1)}, \vec{x}^{(j_2)}) = \sqrt{(\vec{x}^{(j_1)} - \vec{x}^{(j_2)})^T A (\vec{x}^{(j_1)} - \vec{x}^{(j_2)})}$ with a positive semi-definite matrix $A = G^T G$ [236]. This is why linear metric learning is sometimes also referred to as *Mahalanobis metric learning*. Please note that $\hat{d}(\vec{x}^{(j_1)}, \vec{x}^{(j_2)})$ is equivalent to the Mahalanobis distance $d_M(\vec{x}^{(j_1)}, \vec{x}^{(j_2)})$ if and only if $A = \Sigma^{-1}$, where Σ is the covariance matrix of the data in the original feature space \mathcal{X} (cf. Section 2.2.2).

Linear methods learn a Mahalanobis-like distance

Kulis [236] provides a unified framework called *regularized transformation learning*, which covers most existing linear methods. He assumes that the overall loss being minimized is based on a regularizer $r(A)$, which only depends on the matrix A , and a set of individual loss functions J_q , which operate on the so-called *mapped inner product matrix* $X^T A X$ (where X is the design matrix):

General model for linear methods

$$J(A) = r(A) + \lambda \cdot \sum_{q=1}^Q J_q(X^T A X)$$

Here, λ is a hyperparameter controlling the trade-off between the

Loss functions can encode distance constraints

regularizer and the supervision information. With respect to the mapped inner product matrix, one can easily see that $\hat{d}(\vec{x}^{(j_1)}, \vec{x}^{(j_2)}) = (\vec{x}^{(j_1)} - \vec{x}^{(j_2)})^T A (\vec{x}^{(j_1)} - \vec{x}^{(j_2)}) = (\vec{e}^{(j_1)} - \vec{e}^{(j_2)})^T X^T A X (\vec{e}^{(j_1)} - \vec{e}^{(j_2)})$ with $\vec{x}^{(j)}$ being the j th standard basis vector, i.e., an N -dimensional vector with $e_j^{(j)} = 1$ and $\forall j' \neq j : e_{j'}^{(j)} = 0$. This demonstrates that constraints placed on $X^T A X$ are equivalent to constraints placed on $\hat{d}(\vec{x}^{(j_1)}, \vec{x}^{(j_2)})$.

Encoding distance constraints as loss functions

Kulis argues, that both types of supervision constraints introduced earlier can be transformed into a loss function by using the hinge loss $[z]_+ = \max(0, z)$. For absolute similarity and dissimilarity constraints, we can define the following loss functions based on Equations 5.24 and 5.25, respectively:

$$J_{(\vec{x}^{(j_1)}, \vec{x}^{(j_2)})}^S(X^T A X) = [\hat{d}(\vec{x}^{(j_1)}, \vec{x}^{(j_2)}) - u]_+ \quad (5.27)$$

$$J_{(\vec{x}^{(j_1)}, \vec{x}^{(j_2)})}^D(X^T A X) = [l - \hat{d}(\vec{x}^{(j_1)}, \vec{x}^{(j_2)})]_+ \quad (5.28)$$

For relative distance constraints, on the other hand, we can transform Equation 5.26 as follows:

$$J_{(\vec{x}^{(j_1)}, \vec{x}^{(j_2)}, \vec{x}^{(j_3)})}^R(X^T A X) = [\hat{d}(\vec{x}^{(j_1)}, \vec{x}^{(j_2)}) - \hat{d}(\vec{x}^{(j_1)}, \vec{x}^{(j_3)}) + s]_+ \quad (5.29)$$

Regularizers for linear models

As typical regularizers, Kulis identifies the squared Frobenius norm $\|A\|_F^2 = \sum_{i_1=1}^n \sum_{i_2=1}^n |a_{i_1 i_2}^2|$ of A , the matrix trace $\text{tr}(A) = \sum_{i=1}^n a_i i$, and $\text{tr}(A) - \log \det(A)$, where $\det(A)$ is the determinant of A , which corresponds to the product $\prod_{i=1}^n \lambda_i$ of its eigenvalues λ_i . These regularizers can be interpreted as special variants of weight decay (cf. Section 5.1.3). In the following, we consider some concrete examples of linear methods.

γ -separation for one shot learning

Fink [156] proposed a linear metric learning approach targeted at a one shot learning scenario: He considered the problem of discriminating between two classes c and c' , for which only a single training example is available, respectively, while using information from other classes c_k with many training examples. His approach uses a variant of relative distance constraints by defining a set of classes to be γ -separated, if the distance for any pair of points belonging to the same class is at least γ smaller than the distance for any pair of points belonging to different classes. Thus, γ corresponds to the margin size s , and the triples $(\vec{x}^{(j_1)}, \vec{x}^{(j_2)}, \vec{x}^{(j_3)})$ are chosen based on class membership for the high resource classes c_k . Fink then optimized the transformation matrix G in such a way, that all classes c_k are γ -separated, and that the value of γ is maximized. This maximization of γ is not directly considered by the regularized transformation learning framework. In Fink's approach, the transformation matrix G is thus optimized in such a way, that the classes c_k form well-separated clusters in the transformed feature space. In the context of one shot learning, the hope is that also the classes c and c' are then well-separated from all other classes, making a nearest neighbor classification feasible even in this one shot scenario.

Weinberger et al. [429] aimed to find a linear transformation of the feature space in order to improve the performance of a k nearest neighbors classifier. The goal of their *large margin nearest neighbor classification* (LMNN) approach is to ensure that the k nearest neighbors of any data point from the training set always belong to the same class, while examples from different classes are separated by a large margin. In order to ensure this, they define for each data point $\vec{x}^{(j)}$ its k target neighbors as the k data points from the training set with the same class label as $\vec{x}^{(j)}$, which have the smallest distance to $\vec{x}^{(j)}$ in the original feature space. Using the binary variables $\eta_{j_1 j_2}$ to denote whether $\vec{x}^{(j_2)}$ is a target neighbor of $\vec{x}^{(j_1)}$, and $y_{j_1 j_2}$ to denote whether $\vec{x}^{(j_1)}$ and $\vec{x}^{(j_2)}$ have the same class label (i.e., whether $y^{(j_1)} = y^{(j_2)}$), their loss function (also called *triplet loss* in the literature) is defined as follows:

$$J(A) = \sum_{j_1=1}^N \sum_{j_2=1}^N \eta_{j_1 j_2} \hat{d}(\vec{x}^{(j_1)}, \vec{x}^{(j_2)}) + \lambda \cdot \sum_{j_1=1}^N \sum_{j_2=1}^N \sum_{j_3=1}^N \eta_{j_1 j_2} \cdot (1 - y_{j_1 j_3}) \cdot [\hat{d}(\vec{x}^{(j_1)}, \vec{x}^{(j_2)}) - \hat{d}(\vec{x}^{(j_1)}, \vec{x}^{(j_3)}) + 1]_+$$

Here, the first term minimizes the distance between a given data point $\vec{x}^{(j_1)}$ and its target neighbors, while the second term maximizes the distance between the given data point $\vec{x}^{(j_1)}$ and all data points from other classes. The hyperparameter λ is used to specify the trade-off between these two competing terms. One can thus say that the triplet loss combines the two types of supervision constraints mentioned earlier: The first term corresponds to an absolute similarity constraint with $u = 0$ (cf. Equation 5.27) while the second term is a relative distance constraint with a margin size of $s = 1$ (cf. Equation 5.29). Please note that Weinberger et al. do not consider *all* data points from the same class, but only the k closest ones. This means that their LMNN approach does not assume that each class consists only of a single compact cluster, but allows classes to be represented by multiple individual regions, which may not be connected to each other.

Goldberger et al. [167] proposed *neighborhood component analysis* (NCA) as another way of optimizing a linear transformation of the feature space for a nearest neighbor classification. However, instead of treating transformation and classification as two different processing steps, they propose a single holistic system. Goldberger et al. use a soft neighbor assignment by defining the probability $\mathbb{P}(N_{j_1 j_2})$ of $\vec{x}^{(j_1)}$ selecting $\vec{x}^{(j_2)}$ as a neighbor (denoted as binary random variable $N_{j_1 j_2}$) based on a softmax function (a generalization of the sigmoid function from Section 5.2.2, to be discussed in more detail in Chapter 6) on the Euclidean distances in the transformed space:

$$\mathbb{P}(N_{j_1 j_2}) = \frac{e^{-\hat{d}(\vec{x}^{(j_1)}, \vec{x}^{(j_2)})}}{\sum_{j_3 \neq j_1} e^{-\hat{d}(\vec{x}^{(j_1)}, \vec{x}^{(j_3)})}}$$

Large margin nearest neighbor classifier

Interpretation of the triplet loss

Neighborhood component analysis

Interpretation of the neighborhood probabilities

Using an exponentially decaying function of distances relates to Gärdenfors' notion of semantic similarity [179] (cf. Sections 1.2.1 and 2.1.1). The probability of point $\vec{x}^{(j_1)}$ being correctly classified is then just the sum over all $P(N_{j_1 j_2})$ where $\vec{x}^{(j_2)}$ has the same class label as $\vec{x}^{(j_1)}$. Using an approach based on gradient descent (cf. Section 5.2.2), Goldberger et al. optimize the transformation matrix G such that the expected number of correctly classified points is being maximized. The supervision information used by Goldberger et al. does not directly fit into the two classes described above, because they do not explicitly define constraints on the distances. However, by maximizing the classifier's accuracy, they implicitly target a mapping, which moves data points from the same class closer together, while moving data points from different classes further apart.

Nonlinear methods

While linear models are conceptually straightforward and usually easy to train, a linear transformation of the feature space may sometimes not be sufficient in order to obtain a useful distance function $\hat{d}(\vec{x}^{(j_1)}, \vec{x}^{(j_2)})$. This is the motivation for *nonlinear metric learning methods*, which do not require the mapping function ϕ to be linear. However, in practice ϕ still belongs to a parametric class of function, whose parameters can be estimated in a reasonably straightforward way.

Using the kernel trick

According to Kulis [236], the easiest way to obtain a nonlinear method is by *kernelizing* a linear model, i.e., applying the kernel trick: One can usually reformulate the overall loss function of the linear model in such a way, that it only makes use of inner products $(\vec{x}^{(j_1)})^T \vec{x}^{(j_2)}$. One can then, analogously to support vector machines (Section 5.2.3, replace these inner products with a kernel function $\kappa(\vec{x}^{(j_1)}, \vec{x}^{(j_2)}) = \phi(\vec{x}^{(j_1)})^T \cdot \phi(\vec{x}^{(j_2)})$, which computes the inner product in a feature space induced by the (nonlinear) mapping function ϕ . If the kernel function κ as well as the regularizer $R(A)$ used in the model fulfill some mild constraints, the algorithms for estimating such a nonlinear model for metric learning can be generalized from the linear to the kernelized case.

Limitations of kernelization

However, as Kulis [236] notes, at least for the metric learning problem, using the kernel trick increases the number of parameters to be learned by the model from $O(n^2)$ to $O(N^2)$. Learning a kernelized linear model can thus become challenging for large datasets where $N \gg n$, an effect similar to the one already discussed for support vector machines in Section 5.2.3. We would furthermore like to add, that by using the kernel function κ instead of the underlying mapping function ϕ , we do no longer have an explicit access to the transformed space, which can therefore not be directly analyzed.

Nonlinear methods based on neural networks

Please note that kernelization is only one possible way to construct a nonlinear model. In the following, we will focus on another approach, which makes use of artificial neural networks to learn a nonlinear mapping function ϕ . In all of the examples made below, these artificial neural networks receive images (i.e., raw pixel information) as input and map them into a considerably lower-dimensional embedding space.

Salakhutdinov and Hinton [345] were among the first to use artificial neural networks for nonlinear metric learning. They argue that linear transformations only have a small number of free parameters and that they are therefore not able to capture higher-order correlations in the data. Thus, Salakhutdinov and Hinton propose to use a multilayer network to learn a nonlinear mapping function ϕ , which comes with more free parameters and may therefore be able to discover a better low-dimensional representation of the data. Salakhutdinov and Hinton extend the neighborhood component analysis (NCA) model by Goldberger et al. [167] by replacing the linear transformation with a neural network and by using gradient descent on the NCA loss function in order to tune the network's weights. More specifically, they use the structure of an autoencoder (briefly introduced in Section 1.3.3 and to be discussed in more detail in Chapter 6), which is trained both to accurately reconstruct the data points and to fulfill the NCA objective with respect to its bottleneck layer. Salakhutdinov and Hinton furthermore show that one can split the bottleneck layer (i.e., the lower-dimensional representation) into two parts and only apply the NCA objective to one of these parts. After training, this part of the bottleneck layer then captures mainly classification information, while the remaining part of the bottleneck layer mostly represents information relevant for a correct reconstruction of the data points.

Snell et al. [381] proposed *ProtoNet*, which combines a neural network as nonlinear mapping function ϕ with a prototype-based classification mechanism: In the transformed space \mathcal{Z} , they construct a category prototype \vec{p}_c for each of the classes by simply taking the average over all data points. The probability of a data point \vec{x} being labeled with class c is based on a (properly normalized) exponential decay of the distance between the projected data point $\phi(\vec{x})$ and the respective class prototype \vec{p}_c , formally $p(c|\vec{x}) \propto e^{-d(\phi(\vec{x}), \vec{p}_c)}$. This usage of exponentially decayed distances corresponds to the softmax function used by Goldberger et al. [167] for their NCA model. We can thus establish a strong link to the conceptual spaces framework, because also Gärdenfors [179] has proposed to represent each concept with a single prototype, to measure similarity between points by an exponentially decaying function of distance, and to classify data points based on their distance to the given prototypes (cf. Sections 1.2.1, 1.2.2, and 2.1.1).

Like many of the contributions mentioned above, also Triantafillou et al. [410] apply metric learning in order to facilitate few shot learning. Like both Salakhutdinov and Hinton [345] and Snell et al. [381], they make use of an artificial neural network for learning a nonlinear mapping function ϕ . In the resulting embedding space, they measure the similarity of two data points $\vec{x}^{(j_1)}$ and $\vec{x}^{(j_2)}$ with the Cosine similarity $Sim(\vec{x}^{(j_1)}, \vec{x}^{(j_2)}) = \cos \alpha = \frac{\vec{x}^{(j_1)} \cdot \vec{x}^{(j_2)}}{\|\vec{x}^{(j_1)}\| \cdot \|\vec{x}^{(j_2)}\|}$, which is based on the angle α between the two vectors. Their approach to enforcing constraints is inspired by information retrieval: For a given data point \vec{x} , they compute a ranking of all other points $\vec{x}^{(j)}$ from the dataset based on the Cosine similarity in the embedding space \mathcal{Z} . They define that a point

Nonlinear neighborhood component analysis

ProtoNet: metric learning and prototypes

Neural approach with ranking-based constraints

$\vec{x}^{(j)}$ is relevant to \vec{x} , if both points have the same class label. *Precision at k* for \vec{x} is then defined as the proportion of points $\vec{x}^{(j)}$ relevant to \vec{x} within the k highest-ranked points. The *mean average precision* is then computed by averaging over all positions k and all data points \vec{x} . By trying to maximize the mean average precision, Triantafillou et al. implicitly constrain the mapping function in such a way, that the points with the highest ranking for a given query point \vec{x} (i.e., the ones judged most similar to \vec{x}) are the ones with the same class label as \vec{x} .

Similarities between metric learning and conceptual spaces

The overall motivation of metric learning is indeed quite in line with the general idea of conceptual spaces: We are interested in a relatively low-dimensional space, where distances measured by the Euclidean metric are meaningful in the sense, that they can be used for classification. Also the formulation of similarity as an exponentially decaying function of distance used by Goldberger et al. [167] and Snell et al. [381] is in line with conceptual spaces. In general, the assumption of a nearest neighbor classifier can be related to an exemplar theory of concepts (cf. Sections 1.1.1 and 5.2.5), while the class prototypes used by ProtoNet [381] clearly relate to prototype theory (cf. Sections 1.1.1 and 1.2.2). While the conceptual spaces framework itself is in principle open to both variants, Gärdenfors [179] leans towards a prototype-based representation, since it is computationally more efficient. In theory, through additional constraints used in learning the transformation function ϕ , one could also incite convexity of conceptual regions, which is a key assumption of the original conceptual spaces framework.

Differences between metric learning and conceptual spaces

While we can therefore see certain parallels between metric learning and conceptual spaces, there are, however, also some considerable differences. In metric learning, both the application scenario and the constraints employed are mostly limited to classification scenarios. While a correct class assignment of data points to concepts is of course also important to the conceptual spaces framework, it also incorporates many other aspects such as conceptual betweenness (cf. Section 4.5, subethood of regions as an indication of a conceptual hierarchy (cf. Section 4.2), and concept combination (cf. Section 3.6.2), which are irrelevant to the metric learning problem. Also the domain structure enforced by conceptual spaces is not considered in metric learning. Furthermore, conceptual spaces aim to give a psychological grounding to the similarity spaces by relating them to human behavior in psychological studies. We are not aware of metric learning approaches explicitly incorporating psychological similarity ratings in their models. Finally, the conceptual spaces framework assumes that the dimensions spanning the similarity space are cognitively meaningful and thus easy to interpret. Again, metric learning is usually not focused on the interpretability of the resulting embedding space, but mostly interested in the final classification performance.

Outlook

Despite these obvious and (at least partially) important differences, metric learning is clearly related to conceptual spaces and will play a role in Chapter 7 in the context of incorporating background knowledge when learning conceptual regions. Moreover, our hybrid approach for

grounding conceptual similarity spaces in both psychological data and artificial neural networks, which will be presented in Chapter 8, has a strong relation to metric learning. Finally, we can envision a research program dedicated to bridging the aforementioned differences between metric learning and psychological theories such as conceptual spaces, which can potentially benefit both research areas.

5.4 SUMMARY

In this chapter, we have provided a general overview of machine learning. In the beginning, we laid out important general considerations about machine learning tasks. Moreover, we have presented several standard machine learning algorithms which can be used in an "off-the-shelf" manner, and which will be used in our experimental studies. Some of these approaches are directly applicable to learning conceptual regions, while others can be used for identifying interpretable directions. We considered linear models (namely, linear regression, logistic regression, and support vector machines), decision trees and random forest as rule-based approaches, and k nearest neighbors as an instantiation of the exemplar theory of concepts. We also discussed different approaches for dimensionality reduction, including feature selection, feature extraction through a principal component analysis, and metric learning. All of these approaches can be related to the translation of subsymbolic information to the conceptual layer.

The content of this chapter serves as a basis for our experiments in Chapters 9, 10, and 12. Before we describe these experimental studies, we, however, first need to introduce artificial neural networks as a powerful machine learning method in Chapter 6, and multidimensional scaling as another variant of feature extraction in Chapter 8. Moreover, the background provided in this chapter provides the basis for learning conceptual regions in a cognitively plausible way, which will be discussed in Chapter 7.

Lessons learned

Outlook

6

REPRESENTATION LEARNING WITH ARTIFICIAL NEURAL NETWORKS

6.1	Representation Learning	282
6.1.1	Desiderata for Good Representations	283
6.1.2	Evaluating Representations	285
6.2	Artificial Neural Networks	291
6.2.1	Fully Connected Feedforward Networks	293
6.2.2	Convolutional Neural Networks	298
6.2.3	Regularization	305
6.2.4	Backpropagation	309
6.2.5	Training Algorithms	311
6.2.6	Other Considerations	315
6.3	Network Architectures for Representation Learning	318
6.3.1	Autoencoders	319
6.3.2	Word Embeddings	323
6.3.3	Generative Adversarial Networks	331
6.3.4	Variational Autoencoders	337
6.4	Representation Learning for Conceptual Spaces	346
6.4.1	General Considerations	347
6.4.2	The Rectangle Domain	353
6.5	Summary	356

The first crucial step towards applying the conceptual spaces framework in artificial intelligence consists in specifying its underlying dimensions. While this can be manually done for a small amount of well-understood domains (e.g., COLOR), this step is much more problematic for more complex domains, which are less well understood (e.g., SHAPE). In this chapter, we introduce *representation learning*, which is a research area that focuses on learning useful representations from raw perceptual data (e.g., the pixel values of images) using artificial neural networks. Recent developments in this area focus on learning *disentangled representations*, where the individual dimensions of the extracted representation have a meaningful interpretation. We argue that especially these approaches are suitable for extracting the dimensions of a conceptual space from perceptual input.

*Grounding
conceptual spaces
with representation
learning*

In Section 6.1, we introduce the task of representation learning in more detail, focusing on general desiderata for good representations and how to quantitatively evaluate learned representations. We then introduce artificial neural networks (ANNs) as a powerful machine learning method in Section 6.2, covering both architectural considerations and training algorithms. In Section 6.3, we then put our focus

on various ANN-based approaches to representation learning, including autoencoders, word embeddings, and recent approaches such as InfoGAN [101] and β -VAE [196], which aim to extract disentangled representations. In Section 6.4, we then relate this to conceptual spaces, before concluding this chapter in Section 6.5.

Our discussion of representation learning for conceptual spaces in Section 6.4 is partially based on ideas presented in [44].

6.1 REPRESENTATION LEARNING

*Feature engineering
and representations*

The process of *feature engineering* is of crucial importance in any machine learning pipeline (cf. Sections 5.1.4 and 5.3). Having the "right" set of features (i.e., a good representation of the data) greatly benefits the performance of all machine learning algorithms. This also holds for many other information processing tasks, as illustrated by Goodfellow et al. [172, Chapter 15]: Dividing two integers is considerably easier in the Arabic numeral system than in the Roman numeral system, since the former makes use of a place value system. Although the task may be identical, choosing the right representation can greatly influence its difficulty and hence also the chances of success.

*Representation
learning*

The features of a good representation are often based on domain knowledge. However, if such domain knowledge is scarce, finding a good representation of a given dataset can be quite tedious. The research area of *representation learning* thus focuses on automated techniques for discovering a suitable representation. More specifically, one can define representation learning as "learning representations of the data that make it easier to extract useful information when building classifiers or other predictors" [56]. A good representation thus should make a subsequent learning task easier [172, Chapter 15].

*Difference to
dimensionality
reduction*

Representation learning is closely related to the topic of dimensionality reduction discussed in Section 5.3, since we are interested in finding an alternative feature space \mathcal{Z} (sometimes called *latent space* in the representation learning context), which corresponds to a more compact representation than the original input space \mathcal{X} . However, the two topics have a different focus: Dimensionality reduction techniques typically assume the existence of a set of handcrafted candidate features. The main task of dimensionality reduction can then be framed as finding a more compact way of representing the information from this feature set, either by identifying an optimal subset (feature selection, cf. Section 5.3.1), or by finding a projection into a lower-dimensional space (feature extraction, cf. Sections 5.3.2 and 5.3.3). Representation learning, on the other hand, takes the raw perceptual input, which has not yet been preprocessed, and aims to learn useful features based on this low-level information. In a computer vision task, for example, dimensionality reduction starts with a large set of handcrafted features (e.g., edge detectors), while representation learning starts from raw pixel values.

In the following, we first discuss some general desiderata for good representations in Section 6.1.1, before introducing different ways of evaluating the quality of learned representations in Section 6.1.2.

6.1.1 Desiderata for Good Representations

Given an input \vec{x} , representation learning tries to discover a mapping function ϕ , such that the representation $\vec{z} = \phi(\vec{x})$ is useful for downstream tasks. This learned representation \vec{z} (with the respective underlying feature space \mathcal{Z}) is also referred to as *latent code* or *hidden representation*, and its features are also called *latent variables*. The mapping function ϕ is typically expected to have some desirable properties. In the following, we will summarize a few of these desiderata.

Representation learning as function learning

The first property, which is often required from the mapping function ϕ , is that it is *smooth*: If we have two similar inputs $\vec{x} \approx \vec{x}'$, then also their representation should be similar, i.e., $\phi(\vec{x}) \approx \phi(\vec{x}')$ [56]. Minor changes in the observable properties of the data (e.g., small changes of pixel intensities in an image) should thus only lead to small changes in the extracted representation. This smoothness prior is implicitly used by most machine learning algorithms in order to interpolate between training examples (cf. Section 5.1.4).

Smoothness: similar inputs lead to similar representations

Moreover, a good representation should be *expressive* in the sense that a relatively small number of variables in the extracted representation can capture a relatively large number of possible input configurations [56]. This motivates the use of *distributed representations* instead of *one-hot representations*: In one-hot representations, the different variables of the representation are mutually exclusive which typically means that only one of them can be nonzero. A representation with n features can thus only represent n different configurations. In a distributed representation, the different variables are assumed to refer to different properties of the data. Using a representation with n features that can take v different values each, one can thus distinguish v^n different input configurations [172, Chapter 15]. One may assume, that generalization is made possible through shared properties, which provides another motivation for distributed representations, namely, to capture these properties [172, Chapter 15]. Using a distributed representation thus gives rise to a similarity space [172, Chapter 15].

Expressiveness and distributed representations

One can also argue that a good representation requires a certain *depth* or *hierarchical structure*: Since the input (e.g., pixel values of images) is often related to the abstract explanatory factors (such as categories of the objects in the image) in quite complex ways, one may need multiple steps to extract these factors from the raw input [172, Chapter 15]. Simpler representations of the input are then used to define more abstract representations, which tend to be more invariant to local changes [56].

Depth: multiple hierarchy levels

Manifolds: only small parts of the input space are populated

Another typical assumption made in representation learning is that the observations occupy only small subregions of the overall input space. For instance, photographs of human faces occupy only a very small region in the overall space of possible combinations of pixel values. These small subregions are called *manifolds* (cf. Section 5.1.4). In representation learning, one thus assumes that the learned representations correspond to local coordinate systems on these manifolds [172, Chapter 15]: They describe for instance all changes to a photograph of a face which ensure that it still "counts" as a picture of a face. All other possible variations (which would destroy the semantic content of the image) do not need to be considered in the learned representation.

Disentanglement: individual features correspond to individual semantic properties

A special focus in this chapter will be on *disentangled* representations (cf. Section 5.3.2): Individual variables of the extracted representation should correspond to individual underlying factors of variation in the data, which can vary independently from each other [56]. These underlying factors of variation can be thought of as high-level semantic properties of the respective input. One can argue that such a representation is desirable for downstream tasks: If this downstream task targets one underlying factor (e.g., class membership) which is among the factors contained in the representation (or closely related to one or some of these), then it should be very straightforward to learn the task based on the given representation [172, Chapter 15]. Moreover, one can argue that information about individual factors can be readily accessed in disentangled representations and is relatively robust to irrelevant changes in the input [416]. Furthermore, disentangled representations can be expected to be easily interpretable, which relates strongly to the goal of explainable AI [113, 133, 274] and the interpretable dimensions of conceptual spaces (cf. Sections 1.2.1 and 2.1.1).

Three criteria for disentanglement: modularity, compactness, and explicitness

When it comes to defining disentanglement in more detail, three criteria are commonly used [416]: *Modularity* requires that each variable in the representation captures only a single factor of variation, while *compactness* requires that all information about a single factor of variation is only represented in a very small number of variables, preferably one. Taken together, these two criteria aim at a one-to-one mapping between variables in the representation and factors of variation. The third criterion requires that the representation should be *explicit* in the sense that a mapping between variables and true underlying factors corresponds to a simple linear model.

Sparsity: most features are irrelevant for most inputs

Based on the assumption of disentangled representations, one can furthermore argue that a good representation should be *sparse* [172, Chapter 15]: If the different variables of the learned representation correspond to independent features, then one may expect that most features are irrelevant for most inputs. Hence, only a small subset of features should be "active" (i.e., have nonzero values). Please note that extreme sparsity leads to a one-hot representation, which (as we discussed above) is, however, not desirable, since it prevents an expressive representation. Sparsity and expressiveness can thus be to some

extent conflicting desiderata for a good representation – in practice, one therefore needs to aim for a good trade-off between these two goals.

As we have seen in this brief overview, there are different desiderata a good representation should fulfill. Unfortunately, it is quite difficult in practice to establish a clear training objective targeting all of the desiderata mentioned above [56]. In the next section, we will give an overview of different evaluation metrics to judge the quality of an extracted representation.

6.1.2 Evaluating Representations

Evaluating the quality of a learned representation can be done in different ways. For the purpose of our discussion, we would like to distinguish extrinsic from intrinsic evaluation approaches.

In an *extrinsic* evaluation, the representations are not evaluated directly, but based on their performance in a downstream task. For instance, one can train a simple linear classifier on top of the extracted representation and measure its classification accuracy [56]. Presumably, representations that enable a higher classification accuracy contain more useful pieces of information than representations with a lower accuracy. This may, however, not reflect the quality of the representation accurately, if only a single narrow task is used for evaluation. It would therefore be desirable to evaluate a given representation with respect to multiple different tasks. This can, however, become computationally demanding. In addition to supervised downstream task, one can also evaluate unsupervised objectives. For instance, one can measure how well the original input can be reconstructed from the extracted representation.

In an *intrinsic* evaluation, the performance in downstream tasks is not considered. Instead, statistical properties of the learned representation itself are used to quantify its quality. For instance, in order to quantify the sparsity of a given representation, one can compute the average fraction of non-zero variables over a set of test examples. In the following, we will focus on different intrinsic evaluation metrics for measuring disentanglement.

Higgins et al. [196] proposed the β -VAE score, which can be considered one of the first measures for disentanglement. They assume the existence of a dataset, where each data point is labeled with the values of its true generating factors. They propose to use pairs (\vec{x}, \vec{x}') of data points, which agree on exactly one of the generative factors, but whose values differ with respect to all other generative factors. Higgins et al. then extract the hidden representations $\vec{z} = \phi(\vec{x})$ and $\vec{z}' = \phi(\vec{x}')$ for these two data points and compute their difference vector $\vec{z}_{diff} = |\vec{z} - \vec{z}'|$. A linear classifier is then trained to classify based on \vec{z}_{diff} which generative factor was held constant. The β -VAE score corresponds to the accuracy of this classifier. On an intuitive level, the β -VAE score quantifies whether we are able to identify the generative factors as interpretable directions

*Extrinsic vs.
intrinsic evaluation*

*Extrinsic evaluation:
performance in a
downstream task*

*Intrinsic evaluation:
statistical properties
of the representation
itself*

*β -VAE score:
quantifying
explicitness based on
difference vectors*

in the latent space \mathcal{Z} induced by the learned representation. It can thus be interpreted as a way to measure explicitness (cf. Section 6.1.1).

Criticism of the β -VAE score

While being a crucial first step in the right direction, the β -VAE score is clearly not optimal. Kim and Mnih [221] for instance have criticized, that it might be too sensitive to the hyperparameters of the linear classifier (e.g., its learning rate). Moreover, they highlight, that the β -VAE score does not enforce that the dimensions of the latent code are aligned with the true generative factors: Even if the latent code consists of linear combinations of the true generative factors, the disentanglement metric will return a high score. Thus, modularity and compactness (cf. Section 6.1.1), which together ensure a one-to-one mapping between learned features and factors of variation, are not covered by the β -VAE score. Finally, Kim and Mnih identify a failure mode of this disentanglement metric: Even if only $K - 1$ of K factors have been disentangled, the linear classifier can reach 100% accuracy by learning that the K th factor corresponds to the absence of all other factors.

FactorVAE score: quantifying modularity based on feature variance

Based on their criticism, Kim and Mnih [221] propose the *FactorVAE* score. Here, multiple examples $\vec{x}^{(j_1)}, \dots, \vec{x}^{(j_k)}$ are sampled, where one of the generating factors is kept fixed, while all others are varied. Now, instead of computing difference vectors for pairs of induced latent codes $\vec{z}^{(j)}$, Kim and Mnih compute the normalized variance of each latent dimension over all representations $\vec{z}^{(j_1)}, \dots, \vec{z}^{(j_k)}$. They then determine the index of the latent dimension with the least variance and train a simple majority vote classifier which needs to predict the generating factor that has been kept fixed solely from this index. The FactorVAE score is then defined as the error rate of this simple classifier. The classifier in this setup can only be successful, if the latent dimension with the least variance corresponds to the given ground truth factor. The FactorVAE score thus quantifies modularity.

Criticism of the FactorVAE score

Again, one can criticize that the FactorVAE score only covers one of the three criteria for disentangled representations (cf. Section 6.1.1) and ignores both compactness and explicitness. For instance, also redundant representations, where each generative factor is represented by multiple features, can obtain a very good FactorVAE score, despite being not very desirable in practice. Moreover, as Kim and Mnih, however, acknowledge, their improved disentanglement metric is still not suitable for data with non-independent factors of variation.

SAP score: quantifying compactness based on prediction performance

Kumar et al. [237] propose to use *separated attribute predictability (SAP)* for quantifying disentanglement. In order to compute the *SAP score*, one needs to construct an $n \times K$ matrix S showing the connection from each of the n latent variables to the K generative factors. Kumar et al. propose to compute S_{ik} as the accuracy of predicting the k th generative factor based on the i th latent variable only. For continuous generative factors, they propose to use the R^2 score of a linear regression (cf. Sections 5.1.2 and 5.2.1), whereas for categorical generative factors, they propose to use the balanced classification accuracy obtained by

a simple thresholding on i . Now for each generative factor k (i.e., for each column of S), Kumar et al. compute the difference d_k of the two largest entries, i.e. $d_k = S_{i_1k} - S_{i_2k}$ where i_1 and i_2 index the two rows with the largest values in column k . The final SAP score corresponds to the arithmetic mean over all these values. Thus, a large SAP score can only be obtained if each column of S has exactly one large value. This means, that each generative factor is captured only by a single latent variable, making the SAP score a metric that measures compactness.

Just as for the β -VAE score and the FactorVAE score, we can again criticize that only a single criterion for disentanglement is being operationalized – both modularity and explicitness are not directly taken into account. The SAP score therefore does not allow us to discriminate between a generative factor k , that is represented by multiple features (leading to multiple large entries in the k th column, i.e., a small difference d_k), and a generative factor k' , that is not covered by the learned representation at all (thus having only small entries in the k' th column, leading also to a small difference $d_{k'}$).

Criticism of the SAP score

Eastwood and Williams [142] propose a systematic framework with three metrics, which aim to cover all three criteria of disentanglement, namely, modularity, compactness, and explicitness, respectively.¹ These disentanglement metrics are all based on training a regressor to map from the given latent representation to the values of the true generative factors. Eastwood and Williams assume that we can derive a matrix R of relative importance values from our trained regressor. Each entry R_{ik} of this matrix represents the relative importance of the i th latent variable for predicting the k th generative factor. Eastwood and Williams propose two types of regressors, namely, a lasso regressor (cf. Section 5.2.1) and a random forest regressor (cf. Section 5.2.4). In the case of a lasso regressor, the matrix of learned weights is used for defining R . In the case of a random forest regressor, the entries of R are based on the frequency of a given latent dimension being used as a decision node in one of the trees. While a lasso regressor results in a sparse linear model, a random forest regressor is able to discover also nonlinear mappings between latent dimensions and generative factors.

Eastwood and Williams: systematic framework based on regression

The first evaluation metric proposed by Eastwood and Williams is called *informativeness* I and corresponds to the overall prediction error (measured for example with the mean squared error, cf. Section 5.1.2). This metric measures how much information about the K generative factors is contained in the overall latent code \vec{z} . Informativeness based on a lasso regressor quantifies explicitness, since a lasso regressor uses a linear model (cf. Section 5.2.1). It can thus be linked to the β -VAE score [196] discussed above. Since a random forest regressor is not based on a linear model (cf. Section 5.2.4), informativeness cannot give any insights about a linear relationship between learned features and

Informativeness: prediction error of the regressor

¹ An implementation of their evaluation framework is publicly available at <https://github.com/cianeastwood/qedr>.

generative factors, if a random forest is used as a regressor.

*Disentanglement:
quantifying
modularity based on
feature importance*

A second metric called *disentanglement* D measures, whether each latent variable captures at most one generative factor. It thus refers to the modularity criterion and is therefore related to the FactorVAE score [221] introduced above. In order to compute the overall disentanglement score D , a local disentanglement score D_i is computed for each latent variable. It is based on the entropy of the probability distribution induced by the i th row of R . Recall from Section 5.2.4, that the entropy $H(A)$ of a random variable A measures the degree of uncertainty associated with A . In our context, we thus measure, whether the entries in the i th row of R are uniformly distributed (resulting in a high entropy), or whether they are very unevenly distributed (resulting in a low entropy). The highest value of $D_i = 1$ is obtained, if the entropy becomes zero, i.e., if the i th latent variable is only useful for predicting a single generative factor. The overall disentanglement score D is then computed as a weighted sum over the individual D_i , where the weights are based on the overall predictive power of the respective latent variable.

*Completeness:
quantifying
compactness based
on feature
importance*

The third proposed evaluation metric is called *completeness* C and measures whether each generative factor is only captured by a single latent dimension. It thus refers to the compactness criterion and is hence related to the SAP score [237]. Completeness is defined in a way analogous to the disentanglement D , but focusing on the columns of R instead of its rows. A high local completeness value C_k is thus only obtainable for a very uneven distribution of entries in the k th column, corresponding to a scenario, where information about a given generative factor is only represented by a single feature.

*Criticism of the
framework: mutually
exclusive and
uncorrelated
generative factors*

The evaluation framework proposed by Eastwood and Williams seems to be the most comprehensive one developed so far since it explicitly considers all three aspects of disentanglement. However, it does not come without weaknesses: The generative factors are assumed to be mutually exclusive and uncorrelated. For correlated or overlapping generative factors, the disentanglement metric D automatically becomes smaller, because a single latent dimension then naturally is useful for predicting multiple generative factors. This might impair the comparability of the disentanglement metric D across different datasets. The usage of a lasso regressor rather than an unregularized linear regression might reduce this problem, because the L_1 penalty encourages the usage of only a single source of information (cf. Section 5.2.1). It might, however, be difficult in practice to determine the amount of regularization needed to counteract correlated generative factors.

*Criticism of the
framework: monotone
correlations*

Moreover, one typically would like to find a monotone mapping from latent dimensions to generative factors, i.e., an increase in the generative factor should be matched by an increase in the respective feature. This is fulfilled when using a lasso regression, because a linear mapping is always monotone. However, linear models require also, that this relationship needs to be linear – an assumption, which may be too restrictive in some cases. For instance, when considering disks, one generative factor may correspond to the overall area, while the learned

representation may contain a feature corresponding to the radius of the dist. These two are clearly related, but their relation is quadratic, i.e., nonlinear. Using a random forest regressor also allows for such nonlinear mappings, but does not require monotonicity any more – a random forest can also learn a U-shaped function mapping features to generative factors. Thus, when using a random forest regressor, both the regression performance (i.e., the informativeness score I) and the entries of the matrix R (and hence also the disentanglement D and the completeness C) do not necessarily reflect a monotone, but rather an arbitrary nonlinear decoding of the latent code.

Chen et al. [100] have criticized classifier-based disentanglement metrics in general due to their dependence on specific hyperparameter settings: Since the choice of a classifier’s hyperparameters (such as the regularization strength in the case of a lasso regression) might strongly influence its performance, it might thus also have a heavy impact on the disentanglement metric. Moreover, the optimal hyperparameters for a classifier-based disentanglement metric might depend also on the dataset and on the model being evaluated, making it more difficult to compare disentanglement scores to each other.

Chen et al. [100] therefore propose the *MIG score* as a classifier-free disentanglement metric based on mutual information.² They first compute the mutual information between all pairs of latent variables and generative factors, i.e., the amount of uncertainty about one of the two random variables, which is removed when observing the other (cf. Section 5.3.2). The *mutual information gap* (MIG) is then defined for each generative factor by computing the difference between the two latent dimensions with the highest mutual information. The arithmetic mean over all generative factors then results in the final MIG score. Through the focus on finding a single latent variable for each generative factor, the MIG score is another metric referring to compactness. The procedure of comparing the two largest values for each generative factor relates it closely to the SAP score [237] as discussed above.

Again, one may criticize, that the MIG score only considers compactness, but ignores modularity and explicitness. Just as for the SAP score [237], the mutual information gap thus cannot discriminate, whether a given generative factor is captured by multiple features or by no feature at all. As we have seen in the discussion of the disentanglement D and the completeness C as proposed by Eastwood and Williams [142], one could, however, also derive a MIG score focused on latent features rather than generative factors, which would then quantify modularity.

Van Steenkiste et al. [416] have evaluated the representations extracted by different representation learning algorithms on an abstract visual reasoning task, which consisted of identifying the right continuation for an incomplete sequence of images. When associating the values of the different evaluation metrics discussed above with performance

General criticism of classifier-based metrics

MIG score: quantifying compactness based on mutual information

Criticism of the MIG score

Empirical comparison of disentanglement scores

² See <https://github.com/rtqichen/beta-tcvae> for their implementation.

in their downstream task, van Steenkiste et al. found that the FactorVAE score showed the highest correlation among all proposed evaluation metrics. This may indicate that for downstream tasks, modularity is more important than compactness or explicitness. However, they may still play an important role for human interpretability and storage efficiency and are thus in our opinion still worthwhile to consider. Van Steenkiste et al. found, that in general, representations with high disentanglement scores enabled quicker learning from a smaller number of examples. Their results thus provide empirical evidence both for the usefulness of disentangled representations in downstream tasks and for the soundness of the proposed evaluation metrics.

*The need for
unsupervised
disentanglement
metrics*

All of the metrics discussed so far assume, that the true generative factors and their respective values are already known. However, when applying representation learning to a novel problem, these factors may not be known in practice, motivating the usage of representation learning in the first place (cf. Section 6.1). Instead of using the disentanglement scores from an annotated dataset to estimate the expected quality of a learned representation on a novel problem, it would thus be highly desirable to quantify disentanglement directly on the novel problem. We therefore now briefly introduce a recent proposal for an unsupervised disentanglement metric by Li et al. [257].³ They assume that the overall latent code is structured into meaningful subspaces of variation, where all semantic variations of the same type (e.g., manipulating the rotation of an object) take place in the same subspace. Li et al. further assume that real examples and artificial examples generated from the latent representation are nearly indistinguishable from each other and that therefore the real examples are close to the space spanned by the generated examples.

*The approach by Li et
al.: quantifying
disentanglement
based on subspace
clustering*

In order to measure the performance of a given representation learning system, Li et al. [257] start with a given latent code and generate a sequence of sample data points by manipulating only a single latent variable. All sequences which are based on the manipulation of the same latent variable (but on different original latent codes) are assigned to one cluster. This dataset of sequences is then fed to a so-called *subspace clustering algorithm*, which defines a cluster as a set of sequences lying in the same subspace of data. Li et al. use a normalized version of mutual information to compare the resulting clustering found by the subspace clustering algorithm to the original clustering induced by the creation of their dataset. They argue, that if the subspace clustering algorithm is able to recover the clusters (i.e., the latent variable that has been changed), then varying this latent variable must have a systematic and meaningful effect on the generated data points. As a second part of their evaluation metric, Li et al. use the mean distance of the original data points to the overall subspace spanned by the artificially generated data points. This is intended to measure how realistic the generated samples look. Their overall metric is then a convex combination of the

³ See <https://github.com/ZejianLi/analogical-training> for their implementation.

two aforementioned quantities.

While their evaluation approach is indeed unsupervised in nature, it can be criticized for different reasons. Firstly, the overall score combines a measure of disentanglement with a measure of output quality. These are, however, two conceptually separate ways of evaluating a representation learning system. In order to better distinguish these aspects, it is hence preferable to use separate metrics. Moreover, the performance of the subspace clustering algorithm might depend heavily on its hyperparameters (including also the sizes of the sequences and clusters). It is also not obvious that a good performance of this subspace clustering algorithm indicates that no latent dimension involves multiple generative factors and that all generative factors are represented in the latent code. This means, that their approach does not directly target modularity, compactness, or explicitness. So while the work by Li et al. is an important first step into the direction of unsupervised evaluation methods for disentanglement, it also highlights that much work still remains to be done and that we especially need a clear conceptual and mathematical foundation for unsupervised disentanglement metrics.

Criticism of this approach

6.2 ARTIFICIAL NEURAL NETWORKS

Artificial neural networks (ANNs) are a powerful, but complex machine learning technique, which can be used to approximate arbitrary functions with arbitrarily small error. Since we will apply artificial neural networks in our machine learning experiments in Part III of this dissertation, and since these models can become quite complex, we now introduce them in more detail than the other machine learning algorithms from Chapter 5. Our introduction is mainly based on the textbook "Deep Learning" by Goodfellow, Bengio, and Courville [172].

Artificial neural networks are often motivated as being (loosely) inspired by neurobiology, since the brain can be thought of as a complex network of interconnected neurons. In a quite simplified model, each of these individual units represents a single vector-to-scalar function, which takes a number of real-valued inputs (raw perceptual data or activations of other units) and outputs its own activation based on these inputs [172, Chapter 6]. In the ANN model, these activation values are computed based on a weighted sum of the inputs and a subsequent nonlinear transformation. An *artificial neural network* can then be defined as a collection of such individual artificial neurons [341, Section 18.7]. One should, however, note that the actual goal of ANNs is not to provide an accurate model of brain function, but to solve machine learning tasks [172, Chapter 6]. For instance, the output of the individual units is a single real number instead of the chain of spikes typically observed in biological neurons [295, Chapter 4].

Biological inspiration for artificial neural networks

In addition to this biological inspiration, one can also motivate artificial neural networks from an engineering perspective [172, Chapter

Transforming the feature space

6]: Linear models such as logistic regression (cf. Section 5.2.2) are easy to fit, but limited to simple linear hypotheses – they can only solve tasks, for which the examples are linearly separable. In order to allow for nonlinear hypotheses, one can transform the input \vec{x} with a mapping function $\phi(\vec{x})$ into a different feature space, where the examples can be linearly separated, and then train a linear model in this derived feature space. This procedure is used by support vector machines, where a generic kernel function $\kappa(\vec{x}^{(j_1)}, \vec{x}^{(j_2)})$ is used to compute the inner product of two vectors in the transformed feature space (cf. Section 5.2.3). Another option consists in manually engineering the mapping function $\phi(\vec{x})$ by defining useful features based on domain knowledge – this is called feature engineering (cf. Sections 5.3 and 6.1) and has been the predominant approach in computer vision for many decades.

Learning the mapping function

Artificial neural networks, on the other hand, follow a third approach by *learning* $\phi(\vec{x})$ – the classification function thus becomes $f(\vec{x}; \theta, \vec{w}) = g(\vec{w} \cdot \vec{z}) = g(\vec{w} \cdot \phi(\vec{x}; \theta))$, where the model parameters \vec{w} and the activation function g are used for making a linear classification, and where the mapping function ϕ comes with its own set of free parameters θ . Both \vec{w} and θ need to be learned in order to solve the overall problem. Artificial neural networks therefore implicitly solve a representation learning problem by estimating the parameters θ of ϕ . If the individual entries of $\phi(\vec{x})$ are defined as a logistic regression (i.e., $\sigma(\vec{w} \cdot \vec{x})$, cf. Section 5.2.2), one naturally arrives at a layered network of interconnected units. One can thus think of ANNs as stacked logistic regressions, which give rise to a nonlinear model with an intermediate hidden feature space \mathcal{Z} .

General properties of ANNs

Mitchell [295, Chapter 4] lists some important general properties of artificial neural networks: They are useful on noisy, complex sensor data as provided by cameras or microphones, where the training examples may contain errors. Moreover, ANNs can be used with arbitrary target functions and are thus applicable to binary classification, multi-class classification, and regression. They usually require long training times, but are fast to evaluate. A major disadvantage is that the hypothesis learned by a neural network is often very difficult to interpret, since it is represented by a large number of numeric weights. Deep ANNs using a large number of hidden layers have in recent years excelled at many different tasks, ranging from image classification [193] over language modeling [83] to reinforcement learning [371]. They have therefore proven to be quite powerful tools for solving different types of machine learning tasks.

Important design choices

Goodfellow et al. [172, Chapter 11] give some general recommendations for applying artificial neural networks in practice. They first emphasize the importance of using an appropriate network architecture. We will consider two basic variants in our introduction, namely, fully connected feedforward networks (Section 6.2.1) and convolutional neural networks (Section 6.2.2). Furthermore, Goodfellow et al. recommend to introduce some mild regularization techniques (cf. Section 5.1.3), such as weight decay, dropout, or transfer learning. We will give an

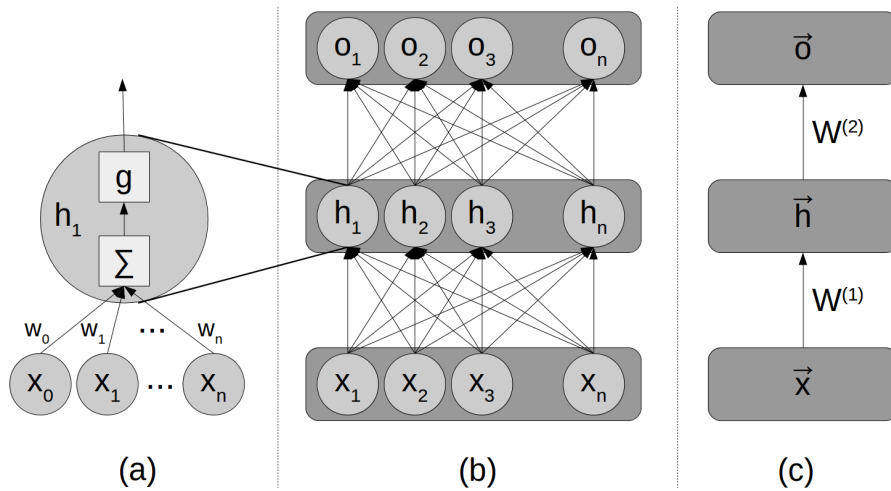


Figure 6.1: Illustration of a multilayer feedforward network. (a) Individual hidden unit. (b) Network of units arranged in layers. (c) Layer-based visualization.

overview of regularization in ANNs in Section 6.2.3. Moreover, Goodfellow et al. suggest to use stochastic gradient descent with momentum and a decaying learning rate, or Adam [223] as optimization algorithms for estimating the network’s weights, and in Section 6.2.4, we give a brief introduction to the backpropagation algorithm which is used to compute the gradient for the overall network. In Section 6.2.5, we then discuss different variants of gradient descent for training neural networks as well as different termination criteria like early stopping. Goodfellow et al. further recommend using advanced techniques like batch normalization (especially for convolutional neural networks), which will be described in further detail in Section 6.2.6.

6.2.1 Fully Connected Feedforward Networks

In this section, we introduce the most traditional architecture used for artificial neural networks, namely, layered feedforward networks. Figure 6.1 illustrates such a network on three different scales of granularity: The structure of an individual unit, the composition of units, and the abstraction to layers.

As we can see in Figure 6.1a, the output of each individual unit is based on a weighted sum of its inputs, i.e., $g(\sum_{i=0}^n w_i x_i)$, where g is a nonlinear activation function [341, Section 18.7]. If the sigmoid function σ is chosen as activation function, then an individual unit corresponds to a logistic regression (cf. Section 5.2.2. Please note, that the input x_0 is fixed to a constant value of 1, such that the weight w_0 represents the bias of the linear function (cf. Section 5.2.1).

Figure 6.1b illustrates how these individual units are connected to each other: In the exemplary three-layered architecture, the inputs x_i are used to determine the activation of a set of hidden units h_j whose activation in turn is used as input to the output units o_k . As we can see,

Individual units

A layer-wise network of units

the network architecture forms a directed acyclic graph [295, Chapter 4], which means that information can only flow in one direction [172, Chapter 6] – hence the term *feedforward*. Moreover, we can observe that connections are only inserted between subsequent layers. Furthermore, two neighboring layers are always *fully connected*, which means that the activation of all units in a given layer l are used as an input for every unit in the subsequent layer $l + 1$.

Thinking in layers

The overall activation of a complete layer can be represented as a vector, whose entries correspond to the activations of its individual units. It is therefore also common to conceptualize artificial neural networks in terms of complete layers as visualized in Figure 6.1c. The individual weights of the individual units (which were expressed as individual weight vectors \vec{w}) can then be represented by a matrix W of weights. If we apply the activation function g element-wise, we can thus for example specify the activation of the hidden layer by $\vec{h} = g(W^{(1)}\vec{x})$. This is not only a convenient notation, but also important for practical implementations, since the underlying matrix multiplications can be implemented quite efficiently.

ANNs as function composition

We can use this insight furthermore to understand the behavior of a neural network as function composition [172, Chapter 6]: In our example, the function $f(\vec{x})$ computed by the overall network can be specified as $f(\vec{x}) = f_2(f_1(\vec{x}))$, where f_1 corresponds to the hidden layer and f_2 corresponds to the output layer. It is important to note at this point that this function composition is only useful from a representational point of view, if the functions computed by the individual layers are nonlinear, since the composition of multiple linear functions again results in a linear function [295, Chapter 4].

Deep learning uses multiple hidden layers

At this point, we would like to highlight that for a long time, artificial neural networks have been limited to the three-layered structure illustrated in Figure 6.1. In recent years, however, progress in hardware, software, and neural networks theory has made it possible to train much deeper networks containing many more hidden layers. This is the reason, why neural networks are mostly labeled as *deep learning* in current publications. Deeper architectures with more layers correspond to a deeper hierarchy of increasingly abstract representations, i.e., a longer chain of nested function compositions.

Universal approximation theorem: ANNs can approximate any function

The *universal approximation theorem* states, that a fully connected feedforward network with a single hidden layer can approximate any continuous function of the given inputs with arbitrary accuracy [341, Section 18.7]. However, the number of hidden units may need to grow exponentially with the number of inputs in order to reach this goal. For instance, consider an arbitrary boolean function of n inputs. For each of the 2^n possible input combinations, we can create one hidden unit that is only active for this specific input. The output unit can then be defined as a logical disjunction over all hidden units for which the output should be TRUE [295, Chapter 4]. Since this exponential size of the hidden layer may quickly become infeasible for practical applications, one often decides to use deeper models (i.e., more hidden layers) with

a smaller number of neurons per layer [172, Chapter 6].

Moreover, the universal approximation theorem only guarantees that a given function can be represented by a neural network, but it does not state, whether or how the parameters of the network can be learned from examples in order to approximate this function [172, Chapter 6]. In practice, the network's parameters (i.e., the weights connecting the different units to each other) are estimated by minimizing a loss function (cf. Section 5.1.3) using gradient descent (cf. Section 5.2.2). Due to the composition of multiple nonlinear functions, the loss function is, however, in general not convex, therefore a convergence of the optimization cannot be guaranteed [172, Chapter 6]. We will consider neural network training in more detail in Sections 6.2.4 and 6.2.5.

Estimating the weights

Overall, the hypothesis space that can be explored by artificial neural networks is continuous and very high-dimensional, because a large number of real-valued connection weights need to be estimated [295, Chapter 4]. Since only the output layer is directly constrained by the training examples, the behavior of the hidden layer(s) can be chosen rather freely in order to optimize task performance – i.e., the hidden representation emerges based on what is useful for the given task [172, Chapter 6]. The basic inductive bias of artificial neural networks can be described as a smooth interpolation between data points [295, Chapter 4]. As already stated in Sections 5.1.4 and 6.1.1, different ANN architectures, however, also include additional biases based on manifolds, distributed representations, depth, and location invariance.

Hypothesis space and inductive bias

So far, we have left the type of *activation function* g and the *loss function* J to be minimized by the network relatively unspecified. As Goodfellow et al. [172, Chapter 6] have argued, these two design decisions are tightly connected to each other and should therefore be considered jointly. Traditionally, the sigmoid function has been used in combination with the mean squared error [295, Chapter 4], but this combination has been discouraged more recently due to saturation effects.

Activation function and loss function

As part of a more general approach for defining the activation function for output units and their respective appropriate loss functions, Goodfellow et al. [172, Chapter 6] interpret the overall network as providing a conditional probability distribution $\mathbb{P}(y|\vec{x};\theta)$ over the possible output values y , given the input vector \vec{x} and the network's parameters θ . They propose to then use the principle of *maximum likelihood* (cf. Section 5.1.3) to derive a loss function based on the type of activation function used for the network's output units. More specifically, maximizing the likelihood of $\mathbb{P}_{model}(y|\vec{x};\theta)$ corresponds to minimizing the negative log-likelihood, i.e., the cross-entropy between the training data (represented as probability distribution \mathbb{P}_{data}) and the model's distribution \mathbb{P}_{model} . Thus, the loss function should in general have the following form (cf. Equation 5.2 from Section 5.1.3):

Deriving the loss function from maximum likelihood

$$J(\theta) = -\mathbb{E}_{\vec{x}, y \sim \mathbb{P}_{data}} \log \mathbb{P}_{model}(y|\vec{x};\theta) \quad (6.1)$$

Depending on the form of $\mathbb{P}_{model}(y|\vec{x};\theta)$ (which is based on the

Loss function depends on activation function

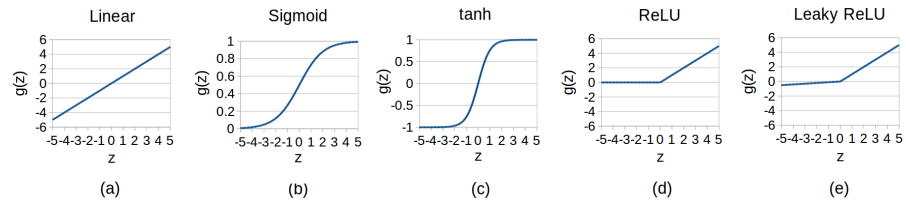


Figure 6.2: Different activation functions commonly used in artificial neural networks: linear (a), sigmoid (b), hyperbolic tangent (c), ReLU (d), and leaky ReLU (e).

type of activation function in the output units), Equation 6.1 can be transformed into a more specific cost function. Let us now consider the application of this approach to different types of output units, based on the derivations by Goodfellow et al. [172, Chapter 6].

Linear output units for regression

In the most straightforward case, we can use a *linear unit* with $\hat{f}(\vec{h}) = \vec{w} \cdot \vec{h}$, i.e., the identity function as activation function $g(z) = z$ (Figure 6.2a). This is especially applicable to regression tasks, where the output of the network needs to be real-valued. From a probabilistic point of view, the output of a linear unit can be interpreted as the mean of a conditional Gaussian distribution $\mathbb{P}(y|\vec{h}) = N(y; \hat{f}(\vec{h}), I)$. In this case, maximizing the log-likelihood corresponds to minimizing the mean squared error (cf. Sections 5.1.2 and 5.2.1).

Sigmoid output units for binary classification

Sigmoid units with $\hat{f}(\vec{h}) = \sigma(\vec{w} \cdot \vec{h})$ and thus $g(z) = \sigma(z) = \frac{1}{1+e^{-z}}$ (Figure 6.2b) are mostly used for predicting the probability of a binary variable y being true, e.g., in a binary classification problem. As already stated above, a sigmoid unit is thus equivalent to a logistic regression (cf. Section 5.2.2). The usage of the sigmoid function for binary classification can be motivated by assuming that the probability of different values for y depends both on the respective values of y (either 0 or 1) and $z = \vec{w} \cdot \vec{h}$ (which is a real number). Let us for now consider an unnormalized probability distribution $\hat{\mathbb{P}}(y|z)$, whose individual values $\hat{\mathbb{P}}(y = 0|z)$ and $\hat{\mathbb{P}}(y = 1|z)$ do not sum to one. If we assume that logarithm of $\hat{\mathbb{P}}(y|z)$ is linear in both y and z , this can be written as $\log \hat{\mathbb{P}}(y|z) = y \cdot z$. By exponentiating and normalizing, we can then obtain a valid probability distribution $\mathbb{P}(y|z)$:

$$\begin{aligned} (\log \hat{\mathbb{P}}(y|z) = yz) &\Leftrightarrow (\hat{\mathbb{P}}(y|z) = e^{yz}) \\ &\Leftrightarrow \left(\mathbb{P}(y|z) = \frac{e^{yz}}{\sum_{y'=0}^1 e^{y'z}} = \frac{e^{yz}}{e^{0 \cdot z} + e^{1 \cdot z}} = \frac{e^{yz}}{1 + e^z} \right) \end{aligned}$$

As one can easily see, we thus get $\mathbb{P}(y = 1|z) = \frac{e^z}{1+e^z}$. By multiplying both the numerator and the denominator with e^{-z} , we can see, that this is equivalent to $\mathbb{P}(y = 1|z) = \frac{1}{1+e^{-z}} = \sigma(z)$. Analogously, we find that $\mathbb{P}(y = 0|z) = \frac{1}{1+e^z} = \sigma(-z)$. We can summarize both cases as $\mathbb{P}(y|z) = \sigma((2y - 1)z)$.

We can now apply the maximum likelihood principle in order to derive an appropriate loss function based on Equation 6.1, where we use that $\log \sigma(z) = \log \frac{1}{1+e^{-z}} = \log 1 - \log(1 + e^{-z}) = -\log(1 + e^{-z})$:

$$\begin{aligned} J(\theta) &= -\mathbb{E}_{\vec{x}, y \sim P_{data}} \log \mathbb{P}_{model}(y|z) = -\mathbb{E}_{\vec{x}, y \sim P_{data}} \log \sigma((2y - 1)z) \\ &= \mathbb{E}_{\vec{x}, y \sim P_{data}} \log(1 + e^{-(2y-1)z}) = \mathbb{E}_{\vec{x}, y \sim P_{data}} \log(1 + e^{(1-2y)z}) \end{aligned}$$

This loss function only saturates (i.e., provides a very small gradient), if $(1 - 2y)z$ becomes very negative. This can only happen for $y = 1, z \gg 0$ and $y = 0, z \ll 0$, i.e., if the model output is correct. Other loss functions such as the mean squared error may saturate whenever $\sigma(z)$ saturates (i.e., when z becomes very small or very large).

In the case of a multi-class classification problem, *softmax units* can be used to estimate a probability distribution over a discrete variable with k possible values. They use the activation function $softmax(z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$. Please note that the activation of output unit i now depends also on all other output units j – the softmax activation function ensures that the activations of all output units sum to one. This can be viewed as a way to introduce "competition" between output units, providing a "softened" version of the arg max function. Based on a similar analysis as for the sigmoid units, one can gain the insight that maximizing log-likelihood for softmax units corresponds to maximizing $z_i - \log \sum_j e^{z_j}$. This expression cannot saturate due to the linear dependence in z_i . Optimizing this expression corresponds to increasing the correct z_i while decreasing all incorrect z_j . Again, other loss functions such as the mean squared error may saturate since they are (unlike maximum likelihood) not able to "undo" the exponentiation.

After having considered the three most important types of output units and their corresponding maximum likelihood losses, let us now consider different types of hidden units in more detail. They are called *hidden* units since their output is only available within the network, but not visible from the outside [295, Chapter 4].

Traditionally, default choices for the activation function of hidden units were the *sigmoid* $g(z) = \sigma(z)$ (with $g(z) \in (0, 1)$) and the *hyperbolic tangent* $g(z) = \tanh(z) = \frac{e^{2z} - 1}{e^{2z} + 1}$ (with $g(z) \in (-1, 1)$) [172, Chapter 6], illustrated in Figure 6.2b and c, respectively. One can show that these two activation functions are related by the simple equation $\tanh(z) = 2\sigma(2z) - 1$. Both types of activation functions saturate across most of their domain and may therefore provide only very small gradients, making gradient-based learning difficult. In general, their usage as hidden units is therefore discouraged. The hyperbolic tangent is slightly preferable to the logistic sigmoid, since its behavior is more similar to the identity function for small z (e.g., $\sigma(0) = 0.5$, but $\tanh(0) = 0$).

In recent years, the *rectified linear unit (ReLU)* $g(z) = \max(0, z)$ (Figure 6.2d) along with its numerous variants has become the dominant choice for hidden units [172, Chapter 6]. It is easy to optimize, since it behaves like a linear unit for $z > 0$ and thus does not saturate.

Applying maximum likelihood to sigmoid units

Saturation behavior of the loss function

Softmax units for multi-class classification

Activation functions for hidden units

Sigmoid and hyperbolic tangent

Rectified linear units

However, its derivative becomes zero for $z < 0$, which makes learning impossible for examples on which the unit is not active. In order to alleviate this shortcoming, several variants such as the *leaky ReLU* with $g(z) = \max(0, z) + \alpha \cdot \min(0, z)$ (Figure 6.2e) have been proposed, which have a nonzero derivative for $z < 0$. ReLUs and their variants are not differentiable at all points (e.g., the derivative of $\max(0, z)$ is undefined for $z = 0$), but this is usually not a problem in practice, since $z = 0$ occurs only rarely, and in this case either the left or the right derivative can be used.

Design decisions for feedforward networks

Unfortunately, there is not good theory about how to design a good, let alone optimal neural network architecture [341, Section 18.7]. Therefore, many design decision need to be informed by prior experience, other models, or structured experiments. Important design decisions include the number of layers, their respective size, and the way these layers are connected with each other. In addition to a chain of layers, one can for instance also consider so-called *skip connections*, which go from layer l to layer $l + 2$ or higher, making it easier for the gradient to flow to lower layers of deep networks [172, Chapter 6]. A related idea is to attach extra copies of the output units to earlier layers in the network to ensure that the gradient for making updates is sufficiently large [172, Chapter 8]. Moreover, considering only sparse connections between layers (such as in convolutional neural networks, see Section 6.2.2) can be an alternative to using fully connected designs [172, Chapter 6].

Adapting the network structure

Some approaches propose to change the network structure during or after learning, by either starting with few hidden units and growing the network or by starting with a large network and removing unnecessary connections (see [295, Chapter 4] and [341, Section 18.7]). However, in modern deep learning systems, it is more common to use large networks and to restrict their effective capacity by applying appropriate regularization methods (see Section 6.2.3).

6.2.2 Convolutional Neural Networks

Convolutional neural networks for image data

Convolutional neural networks (CNNs) date back to LeCun et al. [246] and Waibel et al. [423]. In recent years, they have been quite successfully applied to computer vision tasks such as image classification [193], since they are able to learn location-independent features. CNNs can thus be regarded as the state of the art approach for processing image data and as one of the most successful variants of artificial neural networks. Our following introduction into CNNs is mainly based on the respective book chapter by Goodfellow et al. [172, Chapter 9].

Convolutions in general

Convolutional neural networks are based on a mathematical operation called *convolution*. Goodfellow et al. [172, Chapter 9] introduce convolution by using the example of a tracking task, where a noisy sensor gives the position $x(t)$ of a given object at time t . In order to get a more stable estimate, one can smooth the sensor readings with a

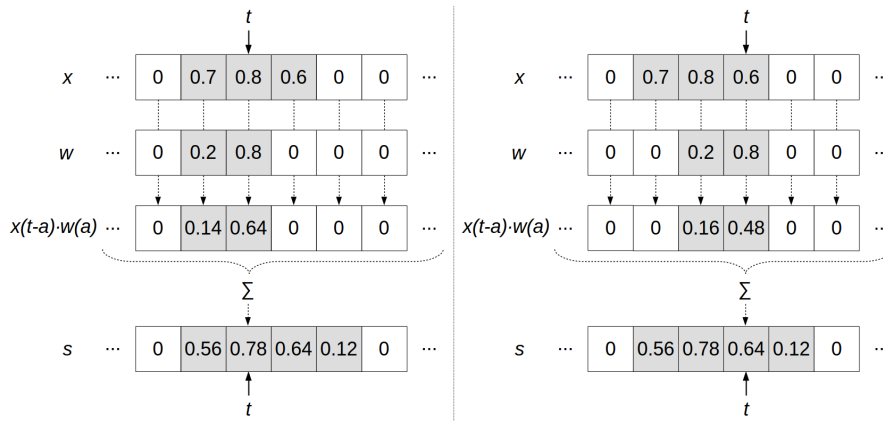


Figure 6.3: Illustration of a one-dimensional discrete convolution for smoothing a noisy sensor at two different time indices t .

weight function $w(a)$ which depends on the age a of the measurement. The overall smoothed estimate $s(t)$ can be written as follows, where the symbol $*$ is used to denote the convolution operation:

$$s(t) = (x * w)(t) = \int_{-\infty}^{\infty} x(a) \cdot w(t - a) da$$

In practice, the time index t often needs to be treated as a discrete quantity, leading to the following notion of a *discrete convolution* (where also the age a is assumed to be discrete):

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a) \cdot w(t - a)$$

Since the convolution operation is commutative, we can rewrite this as follows, which corresponds more to our intuitive formulation of a being the age that specifies the weight w :

$$s(t) = (x * w)(t) = (w * x)(t) = \sum_{a=-\infty}^{\infty} x(t - a) \cdot w(a)$$

In machine learning applications, we furthermore assume that the functions x and w are zero unless specified otherwise, which makes it possible to compute a discrete convolution through a finite sum over the given values. In the context of convolutional neural networks, the first argument of the convolution (in our case x) is called the *input*, while the second argument (in our case w) is called a *kernel*. The output of the convolution operation is often referred to as *feature map*.

Figure 6.3 illustrates such a discrete convolution for the example of a noisy sensor. Both the function values for the sensor output x and the weight function w are shown in the form of arrays which are zero for most of their entries. Please note that for illustrative purposes, we have shown w in such a way that its entries align correctly with the respective entries of x for the multiplication operation. This means that w has been

Discrete convolution

A simple reformulation

Convolutions in machine learning

Example for a one-dimensional convolution

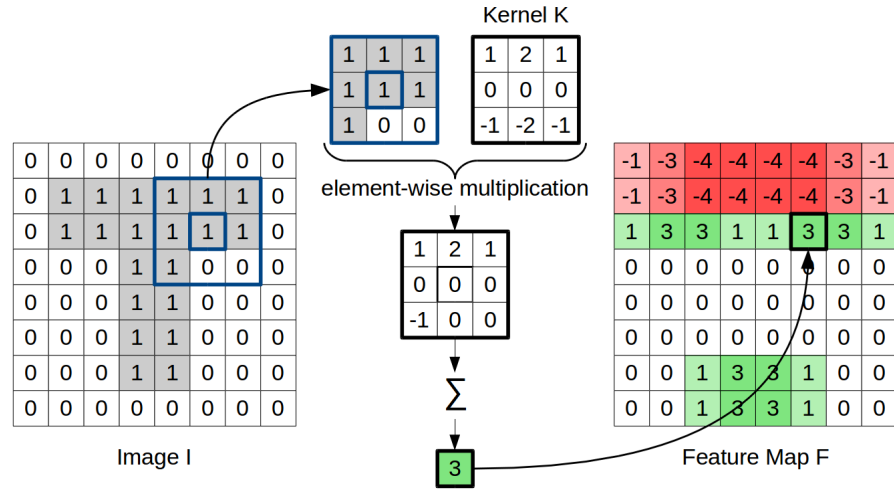


Figure 6.4: Illustration of a two-dimensional convolution with a 3×3 kernel detecting horizontally oriented edges.

horizontally flipped in our visualization, since we assume that w takes the age a as an input and that therefore $w(0) = 0.8$ and $w(1) = 0.2$. As you can see in Figure 6.3, the convolution $(x * w)(t)$ results in a smoothed version $s(t)$ of the original sensor readings $x(t)$. More importantly, it illustrates that we can visualize the convolution operation as sliding w over x , computing an element-wise multiplication, and summing the results for every point t .

Two-dimensional convolution

Convolutions can of course also be applied to more than one axis. For instance, if we consider a two-dimensional image I and a two-dimensional kernel K , we can compute the resulting feature map based on a two-dimensional discrete convolution:

$$F(i, j) = (I * K)(i, j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I(m, n) \cdot K(i - m, j - n)$$

Example for a two-dimensional convolution

As mentioned above, convolutional neural networks have been especially successful on image data, therefore this extension to two-dimensional inputs and kernels is of crucial importance. Figure 6.4 illustrates the convolution of a small two-dimensional image I (showing the letter T) and a 3×3 kernel K , which extracts a feature map F , that highlights horizontally oriented edges from the original image I . Again, applying the convolution can be interpreted by sliding the kernel over the image, applying an element-wise multiplication with the respective image patch, and computing the sum over the individual multiplication results in order to get the value of the feature map at the current location.

Convolutional layers constrain the links and weights

The convolution operation can be integrated into artificial neural networks by introducing a so-called *convolutional layer* [172, Chapter 9]. Figure 6.5a illustrates a fully connected layer, where each input unit x_i is connected to each hidden unit h_j with an individual weight w_{ij} . Here, we have a total number of $5 \cdot 5 = 25$ connections between the two layers

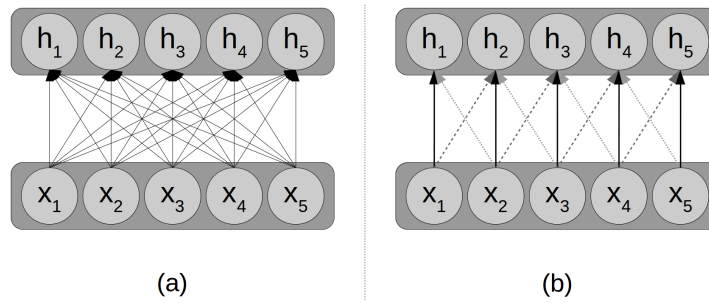


Figure 6.5: Illustrating the difference between a fully connected layer (a) and a convolutional layer (b).

with 25 individual weights. In Figure 6.5b, we see a convolutional layer. Here, the input layer \vec{x} corresponds to the input of the convolution, the weights \vec{w} represent the kernel of the convolution, and the hidden layer \vec{h} corresponds to the feature map resulting from the convolution of \vec{x} and \vec{w} . The convolutional layer depicted in Figure 6.5b has a kernel width of three, which means that each hidden unit h_j only receives three inputs, namely from the input units x_{j-1} , x_j , and x_{j+1} . In other words, we assume that our kernel w has only nonzero values for $w(-1)$, $w(0)$, and $w(1)$. The small kernel size drastically reduces the number of connections from 25 to 13 in our example. Moreover, the fact that the same kernel function w is applied at all positions is reflected through weight sharing: The weights of all hidden units are shared, which is indicated by the three different line styles. This means for example that $w_{12} = w_{23} = w_{34} = w_{45} = w(-1)$. Overall, we thus only have three different weights in the convolutional layer, much less than the 25 individual weights used in the fully connected layer.

Instead of storing a 5×5 matrix of weights, we can thus simply store a vector of three weights, which represents our convolutional kernel. As Figure 6.5b suggests, this can, however, still be re-expressed as a sparse 5×5 weight matrix, which has entries of zero for all non-existing connections, and where the same three weights appear multiple times. While this matrix-based representation is thus not very efficient from an implementational point of view, it can nevertheless be necessary for computing the gradient. Moreover, if one wants to reconstruct the input from the convolution's output and its kernel, a matrix-based representation is very helpful.

Storing the weights

In general, convolutional neural networks use three stages in each of their layers [172, Chapter 9]: They first perform a *convolution* to produce a set of linear activations. They then apply a *nonlinear activation function* $g(z)$ on the results of the convolution in order to ensure that the overall network is able to learn nonlinear functions. Usually, rectified linear units (ReLU) and their variants (e.g., leaky ReLU) are used in CNNs. The combination of convolution and nonlinearity is often referred to as *detector stage*. In a third step, neighboring outputs are aggregated with each other by using a so-called *pooling function*.

Three stages of a convolutional layer

Pooling functions

With this pooling function, the output at a certain location is replaced by a summary statistic of its local neighborhood (usually a rectangle around the given unit). The most popular choice is *max pooling*, where the maximum across the neighborhood is returned, but other pooling operations such as computing a (possibly weighted) average are also possible in principle. The goal of using a pooling function is to make the representation approximately invariant to minor translations. By using fewer pooling units than detector units (so-called *pooling with downsampling*), one can additionally reduce the size of the output. In this case, one has to specify both the *pool width* (i.e., the size of the area to aggregate over) and the so-called *stride* (i.e., the step size between two neighboring centers of pooling). Finally, pooling can also be used to deal with inputs of arbitrary size by specifying the desired output dimensionality, and computing adequate values for pool size and stride dynamically based on the size of the input.

General CNN architectures

In many practical applications, the first few layers of a convolutional neural network consist of such convolutional layers (including both the feature detector and the pooling stage). The size of the representation typically decreases with an increasing depth of the network (using the downsampling technique mentioned above), and the final layers used for making a classification are fully connected. One can view the convolutional layers as feature detectors, that report, whether a given feature is present in the image for each individual location of the image. While the first convolutional layers often learn kernels that correspond to simple edge detectors, higher-level features such as eyes or faces emerge at later convolutional layers. The final fully-connected layers of the overall network then take both the detected features and their locations and aggregate them into a final classification decision.

Using multiple kernels in parallel

Individual convolutional layers usually do not consider only a single kernel (which corresponds to a single feature that can be extracted from the input), but multiple such kernels in parallel, yielding multiple feature maps as output. Also the input to a convolutional layer often consists of multiple channels (e.g., the three color channels for an input image, or the different feature maps from a prior convolutional layer). Overall, a typical convolutional layer should thus be considered as having three-dimensional tensors (with two spatial coordinates for the location within the image and a third coordinate for the different channels) both as input and as output.

Pooling across feature maps

The existence of multiple feature maps per convolutional layer also offers the possibility to apply pooling not only across spatial coordinates within one feature map, but across different feature maps for a single spatial location. This can then be seen as an aggregation over different variants of the same feature. In some sense, the features can then learn to which transformations they should become invariant. For example, if each feature map focuses on detecting slightly differently rotated digits, a max pooling across these feature maps can correspond to a rotation-invariant feature, which detects digits in general.

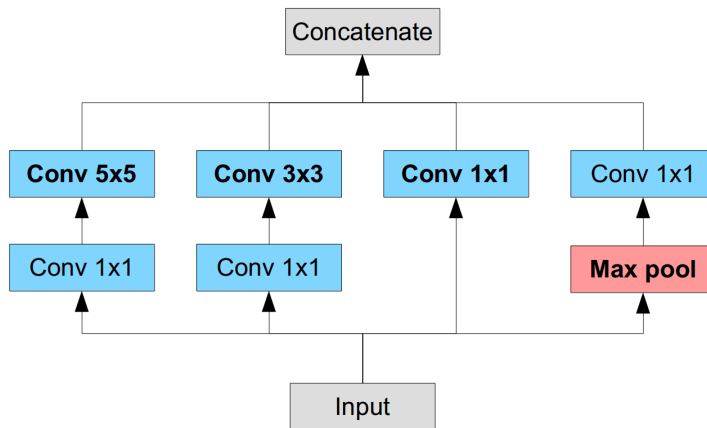


Figure 6.6: Original inception module as described in [399].

When introducing the pooling operation, we also mentioned the possibility of downsampling by specifying a stride between centers of the pooling regions. This downsampling can also be applied to the convolutional stage by specifying a stride s which specifies in which step size the convolution should be evaluated.

The definition of both the weight function w and the input function x being zero for all unspecified locations can be reflected in convolutional neural networks by introducing *zero padding*: The input is enlarged by appending zeros in all directions. This way, the convolution can also be applied to the edges of the input, ensuring that the output of the convolutional layer can have in principle the same size as its input. This is implicitly done in our illustration from Figure 6.5b – otherwise the hidden units h_1 and h_5 would have to be removed, since they only receive two inputs.

In our experiments reported in Chapters 9 and 12, we will make use of a specific CNN, called the *inception* architecture [399, 400]. This architecture is based on stacking so-called *inception modules*, which compute multiple convolutions with different kernel sizes in parallel. The original structure of such an inception module is illustrated in Figure 6.6 [399]. As one can see, it consists of four parallel channels whose output is then concatenated: Convolutional layers with kernel sizes 5×5 , 3×3 , and 1×1 , as well as a max pooling layer. The intuition behind using different kernel sizes in parallel is to allow for receptive fields of different sizes in order to allow the network to learn both local and non-local features. The additional convolutions of size 1×1 shown in Figure 6.6 are used to reduce the dimensionality of the representation. Their receptive field is limited to a single pixel, but they are able to aggregate across multiple channels. By reducing the number of channels before applying the 5×5 and the 3×3 convolution, the number of model parameters is drastically reduced. This also applies to the dimensionality reduction after the max pooling, which ensures that the concatenated output of the inception module (which is used as input to the next module) does not surpass a desired maximal size. The

Strided convolutions

Zero padding

The inception architecture and GoogLeNet

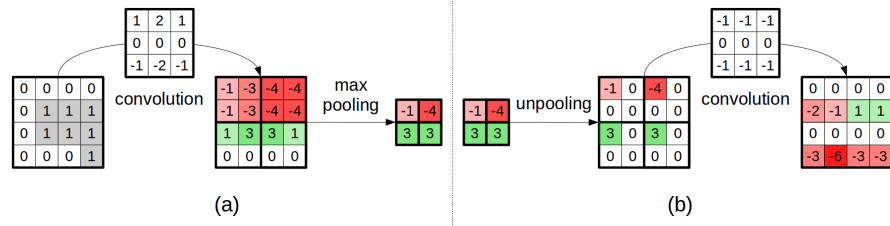


Figure 6.7: (a) Convolution followed by max pooling. (b) Unpooling followed by convolution.

CNN based on these original inception modules is called *GoogLeNet* and contains 27 layers in total [399].

The inception-v3 network

The *inception-v3* network [400] is a further refinement of the inception architecture, using two main insights: Firstly, Szegedy et al. [400] note that 5×5 convolutions are computationally relatively expensive, but that they are approximately equivalent to two 3×3 convolutions, which are applied in sequence: The overall receptive field still has the same size (namely, $5 \cdot 5 = 25$ pixels), but only $2 \cdot 3 \cdot 3 = 18$ instead of $5 \cdot 5 = 25$ parameters are needed. Secondly, Szegedy et al. also note that an arbitrary $k \times k$ convolution can be replaced by a $1 \times k$ convolution and a subsequent $k \times 1$ convolution. Again, the receptive field has the same size, but the number of parameters drops from k^2 to $2k$. In the inception-v3 network, both of these insights are applied to some of the inception modules, allowing for a considerably deeper network of 42 layers without a considerable increase in the number of parameters or higher computational cost. We will use a pretrained inception-v3 network in Chapters 9 and 12 for our machine learning experiments.

Increasing the representation size

Typical convolutional networks start from a very high-dimensional input (namely, images) and reduce the representation size in multiple steps through max pooling or strided convolutions, until a fairly small representation is reached, which can then be used for classification through a softmax layer. However, in some settings one is also interested in the opposite direction: Creating a high-dimensional image from a low-dimensional hidden representation. This corresponds to adding information in contrast to discarding it [172, Section 20.10.6].

Unpooling as approximate inverse to max pooling

While convolutions can still be applied for transforming images based on local kernels, the pooling step can in general not be easily inverted, since it typically comes with a loss of information [172, Section 20.10.6]: When using max pooling with both a width and a stride of 2, we only keep the maximum value for each 2×2 patch of the feature map (cf. Figure 6.7a). Since three out of the four values are discarded completely, it is impossible to reconstruct them. In practice, one therefore needs to approximate the inverted pooling function with so-called *unpooling* steps. In the most straightforward (and perhaps therefore also most popular) variant, one simply replaces each entry of the feature map by a block of size $s \times s$, where the original value is copied to the top left corner and all other entries of the block are set to zero [135] (cf. Figure 6.7b). Using such an unpooling step followed by a convolution can be seen as

an approximate inverse of computing a convolution and a subsequent pooling [135]. While the unpooling increases the dimensionality of the output, the convolution kernels applied afterwards learn to replace the zero entries with more meaningful values [172, Section 20.10.6].

Different terms are used in the research community for the combination of unpooling and convolutions: While many researchers speak of *deconvolutions*, this is strictly speaking not correct (since they are not the actual inverse of a convolution operation). Therefore, others prefer the terms *upconvolutions*, *transposed convolutions*, or *fractionally-strided convolutions*. We will use the term *upconvolutions* from now on.

Combination of unpooling and convolution

Upconvolutional layers have been applied in various contexts, e.g., for image generation [135], autoencoding images [452], semantic image segmentation [302], and for visualizing and studying the representations extracted by convolutional neural networks [134, 451].

Usage of upconvolutional layers

Finally, one can loosely link CNNs to neuroscientific results. For example, Hubel and Wiesel [203] found, that individual neurons in the early visual system of a cat's brain respond most strongly to very specific patterns of light such as precisely oriented bars. This can be mathematically modeled as kernels of a convolution. Also in the human brain, the primary visual cortex is assumed to be arranged in the form of a spatial map containing simple cells with a small receptive field and more complex cells which are invariant to small shifts. This can be linked to the detector stage and the pooling stage of CNNs, respectively [172, Chapter 9]. Moreover, the feature maps learned by deep CNNs have been recently linked to the activation patterns at different stages of the human visual stream [106, 443]. We will reconsider these results in Chapter 10 in the context of the SHAPE domain.

CNNs and neuroscience

However, one should caution against interpreting CNNs as a faithful model of the human visual system, since they typically do not include important biological aspects such as saccades, feedback from higher layers, and a tight integration with other senses [172, Chapter 9].

CNNs do not model human vision

We will consider convolutional neural networks in our studies in Chapters 9, 11, and 12 both as a baseline for predicting human dissimilarity ratings and as pretrained feature extractors for a regression from images to points in a similarity space.

Outlook

6.2.3 Regularization

In its general form, regularization has already been introduced in Section 5.1.3. Please recall that the general idea of regularization is to constrain a given machine learning system in order to reduce the generalization error, possibly at the cost of an increased training error. Regularization can come in different forms [172, Chapter 7]: It can be formulated as *hard constraints* (which explicitly remove a part of the hypothesis space) or as *soft constraints* (which introduce a preference over different parts of the hypothesis space). This corresponds to the distinction of

Regularization in general

inductive biases into restriction biases and preference biases (cf. Section 5.1.1). Regularization can encode a *generic preference for simpler models* based on Occam’s razor and considerations about model capacity (cf. Section 5.1.3), or it can leverage *specific prior knowledge* about the given problem domain. Since proper regularization plays an important role in artificial neural networks, we will now give an overview of different regularization strategies. This section is mainly based on the respective book chapter by Goodfellow et al. [172, Chapter 7].

Dataset augmentation: creating additional training data

Let us first consider some general regularization techniques, which are applicable to most machine learning models, and which typically modify the content of the training set. A first general strategy for decreasing the generalization error simply consists in using more data for training. If the available data for the given task is, however, limited, one can use *dataset augmentation* to create additional fake data points based on the original ones [172, Chapter 7]. These additional data points can, for instance, be created by simply copying existing data points (which is called *oversampling*), by linear interpolations between existing data points, or by adding random noise [192]. They can then be added to the training set in order to provide more data for the estimation of the model’s parameters.

Augmentation for computer vision

Dataset augmentation has been particularly useful for computer vision tasks such as object recognition, since images have many factors of variation (small translations and rotations, different zoom levels, variations with respect to brightness and focus, etc.) which can often be easily simulated [172, Chapter 7]. Additional images can thus be created by applying such small variations to existing images. An additional advantage of images in the context of dataset augmentation is that the created variations can be easily visualized. We will apply dataset augmentation techniques in our machine learning experiments in Chapters 9 and 12.

Semi-supervised learning: including unlabeled data points

Another general regularization strategy is called *semi-supervised learning*, where also unlabeled examples are used in a supervised learning setting [172, Chapter 7] (cf. Section 5.1.2). Essentially, these unlabeled examples (which are usually more abundant, since they are easier to collect) are used to learn a transformation into a hidden representation, before the labeled examples (which are often less numerous) are then used to learn the target function based on this hidden representation. For instance, a principal component analysis (see Section 5.3.2) can be used as an unsupervised preprocessing step, before applying a classifier on the projected data points.

Multitask learning: learning multiple tasks at once

In the context of (deep) neural networks, semi-supervised learning can be modeled by having a generative model $\mathbb{P}(\vec{x})$ and a discriminative model $\mathbb{P}(y|\vec{x})$, which share part of the network structure (e.g., the first few layers). By optimizing both models at the same time, the gradient from estimating $\mathbb{P}(\vec{x})$ constrains the shared parts of the network and thus helps the network to learn a useful hidden representation. This can be seen as a special case of *multitask learning*, where the same network

is trained on multiple tasks at once, and where the different tasks share the lower layers of the network [172, Chapter 7]. This multitask learning approach is, however, only helpful, if the different tasks are semantically related. We will use such a hybrid network architecture in Chapter 12.

Another approach, which also considers multiple related tasks, is called *transfer learning* [172, Chapter 15]. In transfer learning, these tasks are, however, not being optimized simultaneously, but in a sequential way: The system is first trained on a task T_1 (using a large dataset), before being fine-tuned on a different task T_2 , for which only relatively little data is available. Here, the underlying motivation is, that training the system on T_1 results in a set of parameter values θ_1 , which are a good starting point for good performance with respect to T_2 . Especially in gradient-based learning, starting the optimization in a promising region of the hypothesis space can lead to better results in a shorter amount of time. We will consider such a transfer learning approach in our machine learning experiments in Chapter 12.

*Transfer learning:
meaningful
initialization of
model parameters*

The idea of creating *ensembles* of predictors has already been discussed in Section 5.2.4, when we introduced random forests: Different models may tend to make different errors on the test set, and can thus to some extent cancel out each others' weaknesses. This technique can be applied to any kind of machine learning algorithm: For instance, one can train different versions of the same classifier on different datasets, which are created by sampling with replacement from the original dataset. This approach is called *bagging*. In the context of ANNs, one can also create different versions of the same network based on different random initializations or different hyperparameter settings, while using the same underlying training set.

*Ensembles:
combining the
predictions of
different models*

A final general technique is called *adversarial training* [172, Chapter 7]. In this approach, an optimization procedure is used to find an input \vec{x}' , which is very similar to a given example \vec{x} , such that the output of the model changes, but that a human observer would still expect the same behavior. The generation of such adversarial examples has been highly successful for deep neural networks on image classification tasks [401]. The idea of adversarial training is now to include these adversarial examples into the training set and to retrain the system in order to make it more robust. While being in principle applicable to any kind of machine learning algorithm, adversarial examples have mostly focused on (deep) neural networks, since the derivation of the adversarial examples can be performed with gradient descent.

*Adversarial training:
cherry-picking bad
inputs to increase
robustness*

Let us now consider several regularization techniques, which are more specialized towards artificial neural networks. A first group of regularization approaches can be summarized as *parameter norm penalties*. Here, we simply add a penalty based on the norm of the parameters to the optimization objective. Parameter norm penalties have already appeared in Section 5.1.3 under the name *weight decay*. We have already seen the two most popular variants in Section 5.2.1, when discussing linear regression, namely L_1 regularization (using

*Parameter norm
penalties: weight
decay*

$\sum_i |w_i|$ as regularization term) and L_2 regularization (corresponding to $\sum_i |w_i|^2$). At this point, we should note that in deep learning, such parameter norm penalties are often only applied to the weights, but not to the bias terms (i.e., the intercepts). The reasoning behind this is that weights control how two units interact, while biases only influence a single variable [172, Chapter 7]. The bias terms can thus usually be estimated using less data points and are therefore less likely to overfit.

Constraints on individual weights: parameter tying, parameter sharing, and removing connections

Parameter norm penalties work on the overall size of *all* parameters and encode a preference for models, whose weights are on average small. We can also encode more specific constraints with respect to individual parameters, if such constraints are provided by domain knowledge [172, Chapter 7]: For instance, one can use *parameter tying* as a soft constraint to ensure that two or more parameters take on similar values by including a parameter norm penalty on their (pairwise) difference. This encodes a preference bias on the hypothesis space of possible weight combinations and thus can help to reduce overfitting. If we prefer a hard constraint instead, we can enforce *parameter sharing*, which forces a set of parameters to be equal. This parameter hence only needs to be stored once in memory and thus can drastically reduce the number of unique model parameters. Shared weights can be updated by using the average gradient across all of their respective locations in the network. A third variant of constraints with respect to individual parameters is to completely *remove connections* from the overall network's architecture. This corresponds to forcing the corresponding weights to a value of zero. Convolutional neural networks as introduced in Section 6.2.2 use a combination of parameter sharing and removing connections to drastically reduce the size of their hypothesis space, based on the assumption of location invariance.

Noise injection and label smoothing

When discussing data augmentation, we already touched upon the idea of adding noise to the training examples in order to improve generalization. In artificial neural networks, *injecting noise* is, however, also possible at the level of hidden activations or weights, and may increase the robustness of the overall system [172, Chapter 7]. It is also possible to add noise to the output targets, since most datasets contain a small amount of incorrect labels anyways. One variant of this is called *label smoothing* and can for instance be conducted by using $1 - \epsilon$ and ϵ as target values in a binary classification task instead of 1 and 0 [172, Chapter 7]. Since sigmoid units are limited to the interval $(0, 1)$, using the smoothed variants can prevent ever-increasing weights, which try to drive the output of the sigmoid further and further into saturation.

Sparsity constraints: aiming for sparse representation

In addition to adding noise to hidden activations, one can also put other soft constraints on the hidden representation learned by a neural network. One example for this are *sparsity constraints*, where a penalty on the activations of the units is introduced (e.g., using an L_1 regularization). The network is thus incited to use only a small number of hidden units for any given input, which reduces its hypothesis space and leads to sparse representations (cf. Section 6.1.1). This puts also indirect constraints onto its weights, since the activation of each unit is determined by those weights. It is important to note that encouraging

a sparsity of weights (e.g., with L_1 weight decay) is different from encouraging a sparsity of activations, since sparse weights can lead to non-sparse activation patterns, while sparse activation patterns may require non-sparse weights.

A more radical regularization approach related to both noise injection and sparse representation is called *dropout* [388]: For each training example, a random binary mask is sampled and applied to all input and hidden units of the network. This means that a certain, randomly selected portion of the network is disabled (forced to an activation of zero) for each training example. Importantly, since the binary mask changes for every training example, different subsets of units are deactivated at each time step. Usually, input units are discarded with a probability of 0.2, while hidden units are disabled with a probability of 0.5. By following this dropout procedure, one implicitly trains an ensemble consisting of all subnetworks, that can be formed by removing individual units from an underlying base network. Once the model has been trained, predictions can either be made by averaging across a number of binary masks or by evaluating the overall network, where the weights leaving unit i are multiplied by the probability of unit i being included in the mask. Usually, the latter approach is chosen, since it is more straightforward to implement. Using dropout is computationally not expensive and works well with any sufficiently large model. Implicitly, it encourages hidden units to perform well regardless of which other hidden units are included in the subnetwork – dropout therefore ensures a certain redundancy in the hidden code.

Dropout: turning off randomly selected units

6.2.4 Backpropagation

Artificial neural networks are usually trained by using gradient descent. In order to compute the gradient with respect to all weights in the network, a technique called *backpropagation* is used: After propagating information forward through the network (from inputs to outputs) and computing a scalar loss, the information about the loss is backpropagated from the outputs to all weights in the network in order to compute their respective derivatives [172, Chapter 6]. It is important to emphasize that backpropagation is an algorithm for computing the gradient, but that it does not make any parameter updates. Backpropagation can thus be thought of as an internal helper algorithm for gradient descent on neural networks.

Backpropagation for computing gradients

In Section 5.2.2, we have already seen, how the chain rule can be applied to a computation graph of a logistic regression in order to compute an overall derivative based on the derivatives of individual steps. Backpropagation essentially consists of a recursive application of the chain rule, while storing intermediate results for increased efficiency. It can thus be linked to dynamic programming techniques used in optimization problems [172, Chapter 6]. The key insight of backpropagation is that many subexpressions may be repeated several

Backpropagation as smart application of the chain rule

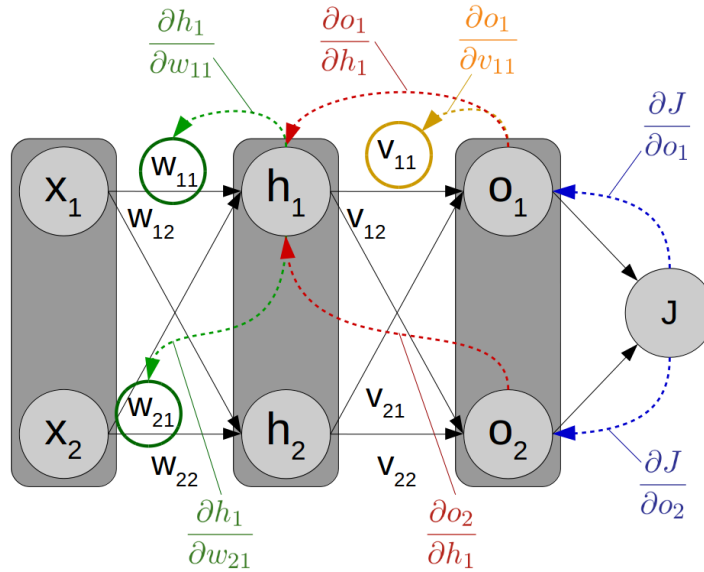


Figure 6.8: Illustration of backpropagation for a small example network.

times within the overall expression of the gradient and that temporarily storing the values of these subexpressions can save a considerable amount of computation time compared to a naive approach, which computes every entry of the gradient individually.

An illustrative example

We will illustrate the idea of backpropagation with the simple neural network illustrated in Figure 6.8, consisting of two input nodes, two hidden units, and two output units. Figure 6.8 also illustrates the different steps needed to compute the respective partial derivatives for three selected weights w_{11} , w_{21} , and v_{11} . For the sake of simplicity, we ignore all other weights as well as the bias terms. When applying the chain rule to $\frac{\partial J}{\partial w_{11}}$, $\frac{\partial J}{\partial w_{21}}$, and $\frac{\partial J}{\partial v_{11}}$, one can easily see that some subexpressions appear in the partial derivatives for multiple weights:

$$\frac{\partial J}{\partial w_{11}} = \frac{\partial J}{\partial o_1} \frac{\partial o_1}{\partial h_1} \frac{\partial h_1}{\partial w_{11}} + \frac{\partial J}{\partial o_2} \frac{\partial o_2}{\partial h_1} \frac{\partial h_1}{\partial w_{11}} = \underbrace{\left(\frac{\partial J}{\partial o_1} \frac{\partial o_1}{\partial h_1} + \frac{\partial J}{\partial o_2} \frac{\partial o_2}{\partial h_1} \right)}_{\frac{\partial J}{\partial h_1}} \frac{\partial h_1}{\partial w_{11}}$$

$$\frac{\partial J}{\partial w_{21}} = \frac{\partial J}{\partial o_1} \frac{\partial o_1}{\partial h_1} \frac{\partial h_1}{\partial w_{21}} + \frac{\partial J}{\partial o_2} \frac{\partial o_2}{\partial h_1} \frac{\partial h_1}{\partial w_{21}} = \underbrace{\left(\frac{\partial J}{\partial o_1} \frac{\partial o_1}{\partial h_1} + \frac{\partial J}{\partial o_2} \frac{\partial o_2}{\partial h_1} \right)}_{\frac{\partial J}{\partial h_1}} \frac{\partial h_1}{\partial w_{21}}$$

$$\frac{\partial J}{\partial v_{11}} = \frac{\partial J}{\partial o_1} \frac{\partial o_1}{\partial v_{11}}$$

Reoccurring subexpressions

As we can see, the subexpression $\frac{\partial J}{\partial o_1}$ appears in all three individual equations, and the expressions $\frac{\partial J}{\partial o_1} \frac{\partial o_1}{\partial h_1}$ and $\frac{\partial J}{\partial o_2} \frac{\partial o_2}{\partial h_1}$ (which can be summarized to $\frac{\partial J}{\partial h_1}$) are relevant for both w_{11} and w_{21} . If we were to naively compute all entries of the gradient independently of each other, we would thus have to re-compute the values of these subexpressions multiple times. While this is not much of an issue in our small example,

the computational overhead can become quite considerable once the network consists of thousands or millions of parameters.

Backpropagation thus starts at the value J of the loss function and computes the partial derivatives for the parameters by iteratively moving one step back in the network. This means that we first compute $\frac{\partial J}{\partial o_1}$ and $\frac{\partial J}{\partial o_2}$ and temporarily store these values. They can be used to compute the partial derivatives for the weights connecting the hidden layer to the output layer, e.g., $\frac{\partial J}{\partial v_{11}} = \frac{\partial J}{\partial o_1} \frac{\partial o_1}{\partial v_{11}}$. Moreover, we can compute the partial derivatives $\frac{\partial J}{\partial h_1}$ and $\frac{\partial J}{\partial h_2}$ based on the previously stored values for $\frac{\partial J}{\partial o_1}$ and $\frac{\partial J}{\partial o_2}$ and the respective missing parts $\frac{\partial o_k}{\partial h_j}$. Finally, the computation for the input weights of the hidden layer can then be computed based on these locally cached values, e.g., $\frac{\partial J}{\partial w_{11}} = \frac{\partial J}{\partial h_1} \frac{\partial h_1}{\partial w_{11}}$. By doing a single backward pass through the network, we can ensure that each subexpression is evaluated only once, since we visit every node in the computation graph exactly once. At this point, we should point out that computing partial derivatives such as $\frac{\partial o_2}{\partial h_1}$ actually involves again multiple steps, since the unit o_2 internally uses a weighted sum and a nonlinear activation function to convert its inputs to an output. The partial derivative $\frac{\partial o_2}{\partial h_1}$ thus crucially involves the derivative of the activation function g and the weight v_{12} connecting the units h_1 and o_2 . This is analogous to the derivation of the gradient for the logistic regression (cf. Section 5.2.2).

We leave our discussion of backpropagation at this relatively high-level stage and refer the interested reader to [295, Chapter 4] and [341, Section 18.7] for a mathematical derivation of the partial derivatives in three-layer networks using sigmoid activations, and to [172, Chapter 6] for a generalized version of backpropagation, which is applicable to arbitrary tensors and operations.

6.2.5 Training Algorithms

After having sketched a way of deriving the gradient in a computationally efficient way, let us now put our focus on different learning algorithms. As mentioned already in Section 6.2.1 and 6.2.4, the learning algorithms for ANNs use different variants of gradient descent as introduced in Section 5.2.2 for the logistic regression. In that context, we have already discussed some differences between *batch gradient descent* (where the loss is computed over all examples from the training set) and *stochastic gradient descent* (where only a relatively small minibatch of examples is used to estimate the gradient for the next update). Both variants are not guaranteed to converge to a global minimum, and stochastic gradient descent may even fail to terminate in a local minimum unless the learning rate α is decreased over time. A simple way of reducing the learning rate is to use a *linear decay* until a fixed number τ of iterations is reached [172, Chapter 8]. In general, picking good values for the initial and the final learning rate can be quite difficult, but has an important influence on the overall performance of the network: If the learning rate is set too large, the individual updates are too large to

Backpropagation in a nutshell

Further references

Gradient descent and learning rate

converge, while a very small learning rate may cause the network to make too little progress (cf. also Section 5.2.2).

Minibatch size

When considering stochastic gradient descent, one furthermore needs to select an appropriate size for the *minibatches* based on the following considerations [172, Chapter 8]: A large minibatch size allows for a more accurate estimation of the gradient, but comes at increased computational cost. Smaller batch sizes can offer a stronger regularization effect, since they introduce additional noise into the estimation of the gradient. However, if batch sizes become too small, then multicore hardware architectures may not be fully utilized. It is furthermore common practice to choose batch sizes as a power of two, since this is more convenient when training the network on modern graphics cards. Finally, one should note, that the minibatches need to be drawn from the training set in a random way to ensure both that the individual gradient estimates are unbiased, and that two subsequent gradient estimates are independent from each other. This can be achieved by shuffling the dataset, before iterating over it.

Different termination criteria

A crucial decision when using gradient descent-based methods is the choice of the *termination criterion* [295, Chapter 4]: By default, gradient descent terminates, once the gradient becomes sufficiently close to zero, indicating that a local minimum has been reached (cf. Section 5.2.2). Especially for stochastic gradient descent, this may, however, be suboptimal, since the algorithm may be in a local minimum for the loss with respect to the current minibatch, but not with respect to the overall loss function when computed across all training examples. Another possible termination criterion is a *threshold on the training error*: As soon as the error on the training set becomes sufficiently small, training is halted. However, since gradient descent on a deep neural network is usually prone to overfitting, this should be accompanied with suitable regularization methods. Also a *fixed number of iterations* or a *limit on the elapsed time* can be used in order to decide, when to terminate the training procedure. In all of these cases, it is, however, hard to know a priori which level of training error, which number of iterations, or which time limit to impose. Since the main optimization goal is usually the generalization error, one can thus simply monitor the network's performance on a separate validation set and terminate training, once the validation set error starts to increase again, indicating that the network has started to overfit the data. This is called *early stopping* and can be implemented by always keeping a copy of the network's parameters that have yielded the best performance on the validation set so far, and by terminating, if no improvement on the validation set has been observed for a reasonably large number of iterations.

Challenges for gradient-based learning

We have already highlighted two general challenges for gradient-based learning in neural networks, namely the choice of the learning rate and the termination criterion. There are, however, also additional difficulties that arise from the repeated application of nonlinearities and the resulting highly non-convex loss function [172, Chapter 8]:

For a convex function, there is only a single local minimum, which is also a global minimum. However, for non-convex functions as used in neural networks, multiple *local minima* may arise, for example, due to weight space symmetry: If we switch both the ingoing and the outgoing weights of two units i and j (which belong to the same layer), then the output of the network and thus also the loss value do not change, even though we jumped to a different point in the hypothesis space. More problematic are, however, local minima, whose loss value is considerably higher than the loss value for the global minimum. If gradient descent gets stuck in such a "bad" local minimum, then the overall performance of the network may be sub-optimal.

Local minima

Also *plateaus, saddle points, and other flat regions* may become problematic for gradient descent, since the gradient approaches zero in these regions, even though no local minimum is present. This may cause gradient descent to get trapped. It seems that while local minima are more problematic in low-dimensional parameter spaces (i.e., relatively small networks), saddle points are much more common in high-dimensional parameter spaces and thus more of an issue in deep networks.

Flat regions

A third type of "formation" often observed for non-convex loss functions are *cliffs* in the parameter space, which typically result from the multiplication of several large weights. Since the loss function has a relatively sharp drop, the gradient becomes very large, and a single update based on the gradient can move the parameters extremely far away from their current location. A simple heuristic for dealing with such cliffs is called *gradient clipping* and limits the maximal size of the gradient to a predetermined threshold. The underlying assumption of this heuristic is that the main information coming from the gradient should be the direction of the update, but not so much its size (which is rather controlled by the learning rate).

Cliffs and gradient clipping

In order to overcome problems with respect to very small gradients (local minima, plateaus, saddle points, and flat regions), a so-called *momentum* term can be introduced into the update step of gradient descent [295, Chapter 4]. The update rule for a given weight w_i is then computed in two steps:

Gradient descent with momentum

$$v_i = \beta v_i - \alpha \frac{\partial J}{\partial w_i}$$

$$w_i = w_i + v_i$$

Here, the variable v_i contains an exponentially decaying moving average over the past gradients (based on an additional hyperparameter β) and can be interpreted as the velocity of the update [172, Chapter 8]. The introduction of the momentum term thus causes the gradient descent algorithm to "remember" previous updates. Momentum therefore incites gradient descent to keep moving into the same direction as before – if it has been "going downhill" for some time, it keeps speeding up, which may cause it to step through local minima or to pass faster through flat regions [295, Chapter 4].

Velocity as moving average

Automatically
adapting the learning
rate

As noted above, the learning rate is a hyperparameter, which is difficult to tune manually, but which has a large impact on the optimization procedure. In order to solve this dilemma, different variants of gradient descent have been proposed, which try to automatically adapt the learning rate during training.

AdaGrad: keeping
track of all past
gradients

A first example for such adaptive learning rate algorithms is *AdaGrad* ("Adaptive gradient algorithm") [140]. Its underlying idea is to accumulate squared values of the gradient over time via simple summation and to use this vector \vec{r} of accumulated squared gradients for adapting the learning rate:

$$r_i = r_i + \frac{\partial J}{\partial w_i} \cdot \frac{\partial J}{\partial w_i}$$

$$w_i = w_i - \frac{\alpha}{\delta + \sqrt{r_i}} \cdot \frac{\partial J}{\partial w_i}$$

Individual learning
rate for each weight

Here, α is the initial learning rate and δ is a small positive constant ensuring that a division by zero does not happen for $r \approx 0$ (i.e., in the first iteration). One can easily see, that the step size is decreased more heavily for parameters with large partial derivatives, which causes more progress in more gently sloped dimensions of the parameter space. In *AdaGrad*, each weight w_i thus has an individual learning rate, which is based on the past "steepness" of the loss function with respect to this weight. However, Goodfellow et al. [172, Chapter 8] note, that the decrease in the learning rate can be too high in practical applications: Since each r_i can only grow over time, *AdaGrad* always reduces the learning rate, but is never able to increase it again. If the drop in the learning rate is too sudden, then training may be unnecessarily slowed down.

RMSProp: moving
average over past
gradients

Another popular training algorithm is *RMSProp* ("Root Mean Square Propagation") [198], which is very similar to *AdaGrad*, but uses an exponentially weighted moving average instead of a simple summation:

$$r_i = \rho r_i + (1 - \rho) \frac{\partial J}{\partial w_i} \cdot \frac{\partial J}{\partial w_i}$$

$$w_i = w_i - \frac{\alpha}{\delta + \sqrt{r_i}} \cdot \frac{\partial J}{\partial w_i}$$

Moving average
allows learning rates
to increase

By introducing a new hyperparameter ρ for specifying the exponential decay, *RMSProp* is able to discard the distant past, causing the decrease of the learning rate to focus on the more recent history. This means, that the step size can also grow again for a specific parameter, if there have been large updates initially, but only small updates recently. *RMSProp* is an effective and practical optimization algorithm and considered to be one of the standard procedures these days [172, Chapter 8].

Adam: considering
first and second
moments

Let us conclude our discussion of adaptive learning rate algorithms with *Adam* ("Adaptive Moments") [223]. It estimates both the first

moment \vec{s} (i.e., the average) and the second moment \vec{r} (i.e., the variance) of the gradient with an exponentially weighted moving average:

$$s_i = \rho_1 s_i + (1 - \rho_1) \frac{\partial J}{\partial w_i}$$

$$r_i = \rho_2 r_i + (1 - \rho_2) \frac{\partial J}{\partial w_i} \cdot \frac{\partial J}{\partial w_i}$$

$$\hat{s}_i = \frac{s_i}{1 - \rho_1^t}$$

$$\hat{r}_i = \frac{r_i}{1 - \rho_2^t}$$

$$w_i = w_i - \alpha \frac{\hat{s}_i}{\delta + \sqrt{\hat{r}_i}}$$

Adam can thus be interpreted as combining RMSProp (which uses the second moment r_j) with momentum (which uses the first moment s_j). Adam is in general a very robust learning algorithm, which is not sensitive to the choices of the hyperparameters ρ_1 , ρ_2 , α , δ , and which is widely applied in practice [172, Chapter 8]. We will make use of Adam in our machine learning experiments in Chapter 12.

*Combining
RMSProp with
Momentum*

There are of course also other optimization techniques than gradient descent that can be used for training neural networks. For instance, approaches based on *line search* use the gradient to determine the direction of the update, and then find the minimum of the loss function on the straight line implied by this direction [295, Chapter 4]. Other approaches use information about *second derivatives* in order to understand, how fast the gradient is changing locally, and to make better updates based on this knowledge [172, Chapter 8]. It seems, however, that at least currently, the additional computational cost implied by these more complex approaches outweighs their performance advantages. Optimization approaches based on gradient descent are therefore still the default choice for training neural networks.

Other approaches

6.2.6 Other Considerations

The initialization of the model's parameters can heavily influence, whether the model converges at all, how fast it converges, which final loss value it is able to obtain, and how large the generalization error will be [172, Chapter 8]. If all parameters were initialized to the same fixed value, then all units in the same layer would compute the same function and hence all parameters would be updated in the same way. In order to break this symmetry between different units, one often uses a random initialization [172, Chapter 8].

*Importance of weight
initialization*

While larger initial weights break this weight symmetry in a clearer way, they may cause the activation functions to saturate already at the beginning of the training procedure (causing very small updates) or gradients to explode (causing unreasonably large updates) [172, Chapter 8]. One therefore prefers small random weights, such that

*Motivation for small
initial weights*

the activation functions are approximately linear and provide a good gradient. Since we expect local minima especially in regions of the parameter space, where the network's function is highly nonlinear, this can also prevent gradient descent from getting trapped in such a local minimum at the beginning of the training procedure [295, Chapter 4].

Strategies and considerations

Goodfellow et al. [172, Chapter 8] provide an overview of different weight initialization strategies such as *normalized initialization* (which tries to ensure, that all layers have a comparable variance with respect to their activations and gradients) and *sparse initialization* (where each unit is restricted to k nonzero weights). They also note that biases should be initialized separately using a constant predetermined value: In most cases, setting the bias to 0 is a good choice, but for ReLUs, a small positive value such as 0.1 is usually preferred, since it ensures that the unit does not saturate at the beginning of training. For output units, the bias terms mainly drive the respective unit's activation in the beginning of training. They thus should be chosen in such a way, that the output of the respective unit corresponds to the average desired output over the training set, giving the network a reasonable starting point with its predictions.

Batch normalization due to parallel updates

At this point, we would like to introduce an important technique called *batch normalization* [205], which is often used as part of the overall training procedure. Goodfellow et al. [172, Chapter 8] point out, that the gradient tells us, how to update each individual parameter under the assumption that all other parameters remain unchanged. In practice, however, we update all parameters at once, which can lead to unexpected results especially in complex deep models. Also simple statistics such as the mean and standard deviation of a given unit's output may change drastically under such complex updates, since they are determined by a complex interaction of lower layers.

Normalizing the output of a layer

In order to normalize the output of a given layer l , one can collect its activations for the current minibatch in a design matrix H , and replace it with a normalized version $H' = \frac{H - \vec{\mu}}{\vec{\sigma}}$, where $\vec{\mu}$ and $\vec{\sigma}$ contain the mean and the standard deviation of the given layer's units with respect to the current minibatch, respectively. These vectors $\vec{\mu}$ and $\vec{\sigma}$ can be easily extracted from the design matrix H . The next layer $l + 1$ of the network is then presented with the normalized activations H' . Since we can simply add the normalization procedure as additional nodes to the computation graph of the network, backpropagation will take these additional steps into account – the gradient will thus never propose an operation that acts simply to increase or decrease the standard deviation or the mean of a hidden unit h_i [172, Chapter 8]. When applying the trained network on an individual example, one typically estimates $\vec{\mu}$ and $\vec{\sigma}$ based on running averages collected during training.

Mean and variance as additional parameters

Normalizing both the mean and the standard deviation of a hidden unit can considerably reduce its expressive power and thus also the expressive power of the overall network. A more complex variant of batch normalization therefore uses $\vec{\gamma}H' + \vec{\beta}$ as an input to the next

layer instead of H' . The two additional parameters $\vec{\gamma}$ (i.e., the desired variance) and $\vec{\beta}$ (i.e., the desired mean) can then also be learned with gradient descent. This ensures that the expressive power of the network remains unchanged. However, the mean and standard deviation of each unit are now controlled by an explicit parameter instead of being determined by complex interactions of other units, causing improved learning dynamics [172, Chapter 8].

We should also note, that the usage of batch normalization can act as a form of regularization, since the estimate of both the mean $\vec{\mu}$ and the standard deviation $\vec{\sigma}$ is subject to noise from sampling the underlying minibatch. This can make it unnecessary to use additional noise-based regularization methods such as dropout [172, Chapter 11].

*Batch normalization
as regularization*

The learning rate and the number of training iterations are two important hyperparameters of a deep learning model. In Section 6.2.5, we have already introduced adaptive learning rate algorithms and early stopping for tuning these two hyperparameters as part of the training procedure. Other hyperparameters, however, need to be optimized in an offline fashion – either by using the experimenter’s intuitions and experience, or by resorting to techniques such as grid search or random search (cf. Section 5.1.2). As Goodfellow et al. [172, Chapter 11] point out, many of these hyperparameters influence the effective capacity of the model. Hyperparameter tuning thus corresponds to changing the effective capacity in such a way, that it matches the complexity of the task as closely as possible.

*Optimization of
hyperparameters*

Using a larger *number of units in the network* (both influenced by the number of layers and the number of units per layer) leads to an increase of the model’s representational capacity. This comes, however, at increased computational cost and increased memory demands. Related to this is the kernel width in convolutional neural networks (cf. Section 6.2.2), which can also be increased for a larger representational capacity, at the additional cost of a smaller output dimension (unless zero padding is used).

*Increasing model
capacity*

Regularization schemes such as weight decay or dropout (cf. Section 6.2.3) lower the effective representational capacity of the model and can thus prevent overfitting. While weight decay limits the size of the weights (and thus the degree of nonlinearity of the overall model), dropout reduces the degree of interaction between individual units.

*Decreasing model
capacity*

In order to gain additional insights into the training process, Goodfellow et al. [172, Chapter 11] also suggest to monitor histograms of activations and gradients. This can for instance be used to visualize how often hidden units saturate. As a rule of thumb, they state that the magnitude of a parameter update across a minibatch should be around one percent of the magnitude of the parameters.

6.3 NETWORK ARCHITECTURES FOR REPRESENTATION LEARNING

ANNs for representation learning

Artificial neural networks are an obvious candidate for representation learning: Even standard feedforward networks can be thought to implicitly learn an internal representation of their inputs, which is more convenient for a given classification task. While the raw input may not be linearly separable, each layer in the network learns another transformation step (i.e., a higher-level representation), which culminates in an abstract representation in the second-to-last layer, where the different classes are then separated with a simple linear classifier [172, Chapter 15]. Moreover, the class labels assigned to the individual examples already give a strong clue about one of the generating factors (namely, class membership). It is furthermore easily possible in a neural network setting to also include additional constraints such as sparsity by introducing different types of regularization, for instance by using convolutional layers instead of fully connected ones [56] (cf. Section 6.2.2), or by adding a sparsity penalty to the loss function (cf. Section 6.2.3). Also the usage of dataset augmentation (cf. Section 6.2.3) can be seen as a way of improving the robustness of the internal representations, since small random deformations are not expected to change the overall classification result [56].

Greedy layer-wise pretraining

Artificial neural networks also allow for semi-supervised learning. In the context of representation learning, this can correspond to learning a representation from unlabeled data and then learning a given supervised task on top of this [172, Chapter 15] (cf. Section 6.2.3). An example for this is *greedy layer-wise unsupervised pretraining*: Instead of training the whole network at once, one first trains a small network with only a single hidden layer on an unsupervised task (such as reconstructing the original input). The weights of this hidden layer are then used to initialize the lower layer of new network with two hidden layers. This network is then again trained on the same task. This procedure of using the weights of a network with l layers to initialize the bottom part of a network with $l + 1$ layers is repeated, until the full network depth is reached. The final network is then trained on the supervised task, using the pretrained weights as initialization. This layer-wise pretraining scheme thus aims to find a good initialization of a deep network's weights for a supervised task.

Multitask learning

In current research, the semi-supervised nature of representation learning tasks is mostly implemented by training on both the supervised and the unsupervised objective at once instead of using pretraining [172, Chapter 15]. This *multitask learning* approach (cf. Section 6.2.3) has the advantage, that the relative influence of the unsupervised and the supervised objective can be explicitly controlled through their respective weight in the overall loss function.

Transfer learning

Also *transfer learning* tasks (where the network is first trained on task T_1 and afterwards on task T_2 , cf. Section 6.2.3) relate strongly to representation learning: Transfer learning can only succeed if the

representation learned in order to solve T_1 extracts factors of variation which are also relevant for T_2 [56]. In the context of neural networks, transfer learning can be simply implemented by training the network on T_1 , and then fine-tuning the pretrained weights in order to achieve good performance on T_2 .

Finally, one should mention that many representation learning algorithms also contain a *generative component*, which maps from the given representation to the input space [56]. This generative component can be used during training to constrain the learned representation (e.g., by enforcing an accurate reconstruction of the input) and later during evaluation to visualize the effect of manipulating individual entries in the learned representation. While a visual inspection of such generated examples may be insightful for designing further experiments, standardized evaluation metrics for comparing different generative models pose a difficult research problem in themselves [172, Chapter 20].

Generative components

In the following, we will introduce four selected types of neural representation learning approaches. We start by presenting *autoencoders* in Section 6.3.1, whose main training objective is to reconstruct their input, while using only a limited representational capacity. In Section 6.3.2, we will then discuss *word embeddings*, i.e., a distributed representation of natural language semantics learned from large text corpora. We then introduce *generative adversarial networks* (GANs) in Section 6.3.3, which use a game-theoretic setting involving two competing networks in order to learn a good generative model. In this context, we put a special emphasis on InfoGAN [101], which explicitly aims to obtain disentangled representations. Finally, we focus on the class of *variational autoencoders* (VAEs) in Section 6.3.4, which have also been shown to successfully learn disentangled representations from unlabeled data.

6.3.1 Autoencoders

Autoencoders are an important unsupervised neural network architecture, which is commonly used for dimensionality reduction and feature extraction (cf. Section 5.3.2). Autoencoders are typically trained on the task of *reconstructing* their input at the output layer, while using only a constrained internal representation. Figure 6.9 illustrates the general structure of an autoencoder, which can be split into an *encoder* network that translates the original input \vec{x} into a hidden representation $\vec{h} = f(\vec{x})$, and a *decoder*, which reconstructs the original input from this hidden representation, i.e., $\vec{r} = g(\vec{h})$. Autoencoders are trained to minimize the difference between the reconstruction $\vec{r} = g(f(\vec{x}))$ and the original input \vec{x} . Since copying the input to the output is a trivial task by itself, autoencoders need to have additional constraints in order to enforce a useful transformation of the input in a more desirable format. These constraints can come in the form of hard or soft regularization and will be considered in more detail below.

The reconstruction objective

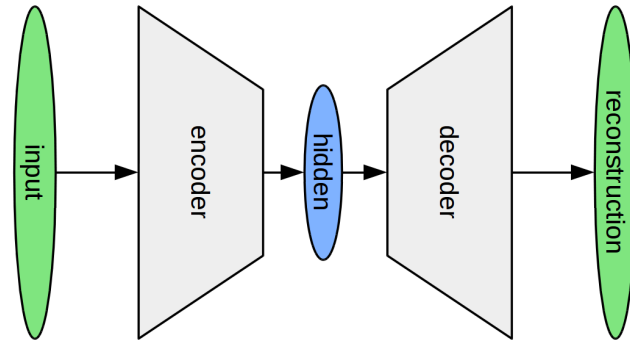


Figure 6.9: Visualization of the general autoencoder architecture.

Depth of autoencoders

While being commonly used with a single-layer fully connected encoder and a single-layer fully connected decoder [56], deeper autoencoders are often capable of achieving a greater compression of their input than their shallow counterparts [172, Chapter 14]. Such deep autoencoders can be trained in a greedy way by stacking shallow autoencoders on top of each other, following the pretraining scheme described in Section 6.3.

Autoencoders and manifolds

If we assume that data points tend to concentrate around a small number of low-dimensional manifolds in the overall input space, then autoencoders can be interpreted as learning local coordinate systems for these manifolds [172, Chapter 14]: They typically need to make a compromise between good reconstruction (which encourages sensitivity to changes on the surface of the manifold) and the fulfillment of their regularization constraints (which encourage insensitivity to changes in all other input directions) [56]. Therefore, only variations of \vec{x} , that stay on the same manifold, need to be represented as changes in \vec{h} .

Loss function

In order to obtain a suitable training objective for an autoencoder, Goodfellow et al. [172, Chapter 14] propose to consider decoders as providing a conditional probability distribution $\mathbb{P}_{\text{decoder}}(\vec{x}|\vec{h})$. One can then apply the principle of maximum likelihood (cf. Sections 5.1.3 and 6.2.1) by minimizing $\mathbb{E}_{\vec{x} \sim \mathbb{P}_{\text{data}}(\vec{x}) \vec{h} \sim \mathbb{P}_{\text{encoder}}(\vec{h}|\vec{x})} - \log \mathbb{P}_{\text{decoder}}(\vec{x}|\vec{h})$. If the input \vec{x} is real-valued, this means that linear output units should be used, which motivates the usage of the mean squared error as training objective (cf. Section 6.2.1). If the input \vec{x} is on the other hand binary or limited to the interval $[0, 1]$, then sigmoid output units seem to be more adequate [56], which implies the loss function $\log(1 + e^{(1-2\vec{x})\vec{h}})$ based on maximum likelihood considerations (cf. Section 6.2.1).

Weight sharing

In principle, autoencoders allow for different weight matrices in the encoder and in the decoder. However, in practice, one often introduces weight sharing (cf. Section 6.2.3) as a type of hard regularization by ensuring that the decoder's weight matrix equals the transpose of the encoder's weight matrix [56].

Undercomplete autoencoder

A straightforward way of constraining an autoencoder's capability to reconstruct its input lies in constraining the size of the hidden

representation: By ensuring that \vec{h} has considerably less entries than \vec{x} , we impose a hard constraint on the network, essentially limiting its representational capacity. Such *undercomplete autoencoders*, which make use of a so-called *bottleneck layer*, are probably the most common autoencoder variant. One can show that if such an autoencoder is trained with the mean squared error as training objective, and if the decoder is a linear function of the hidden representation, then the overall autoencoder learns the same subspace as a principal component analysis [172, Chapter 14]. If both encoder and decoder are allowed to be nonlinear, an undercomplete autoencoder can thus be interpreted as a more powerful, nonlinear generalization of PCA.

Another autoencoder variant may use a larger hidden representation \vec{h} , but introduce a sparsity penalty as soft regularization. Such a *sparse autoencoder* effectively encourages the hidden representation to be sparse, i.e., to have only very few units active for any input (cf. Sections 6.1.1 and 6.2.3). This can be achieved by adding for example a regularization term in form of the L_1 loss on the activation of the hidden layer [172, Chapter 14]. It is important to note that in this case, the L_1 loss is computed based on the outputs of the hidden units, not based on their weights as in the weight decay scheme (cf. Section 6.2.3). As Bengio et al. [56] note, this option is, however, rarely chosen in practice. More popular variants include the so-called Student-t penalty ($\log(1 + h_j^2)$) and a constraint tying the average activation of each hidden neuron over a complete minibatch to a desired fixed value. The latter variant can either be realized using the MSE with respect to a target value ρ , or the KL-divergence (cf. Section 5.1.3) with respect to a binomial distribution with a small probability ρ for the outcome 1. In this way, for instance by setting $\rho = 0.05$, one can explicitly encode, that about 5% of the hidden units should be active on average.

Sparse autoencoder

Denoising autoencoders follow a slightly different paradigm, where the network is regularized by adding noise to the inputs, which needs to be removed by the autoencoder [172, Chapter 14]. More formally, instead of being presented with the original input \vec{x} , the autoencoder receives a corrupted version \vec{x}' , but still has to reconstruct the original input \vec{x} . Typical types of noise used for the corruption step include additive Gaussian noise and (in the case of gray-scale images) salt and pepper noise (i.e., setting random pixels to minimal or maximal intensity) [56]. Adding noise to the input creates a larger variability in the examples, and having to reconstruct the non-corrupted version makes the reconstruction task more difficult, acting therefore as an indirect form of regularization [172, Chapter 14]. In the manifold learning interpretation, the denoising objective corresponds to moving the corrupted input \vec{x}' back to the closest point that lies on the data manifold [56]. One can observe empirically, that denoising leads to qualitatively better features as reflected in better classification performance [56].

Denoising autoencoder

If we interpret the decoder of a denoising autoencoder as a generative network, we may also be interested in drawing samples from

Sampling the decoder

the distribution, which this autoencoder has learned. Goodfellow et al. [172, Chapter 20] describe the following iterative sampling technique: Starting with a given input \vec{x} , one computes the hidden representation $\vec{h} = f(\vec{x})$ and the output of the decoder as $\vec{r} = g(\vec{h})$. If the autoencoder is trained with the MSE objective, this reconstruction \vec{r} can be interpreted as the mean of a normal distribution (cf. Section 6.2.1). In order to sample a new example \vec{x}' from this distribution, one can therefore simply apply Gaussian noise with a variance based on the MSE of the reconstruction objective. This whole process can be iteratively repeated in order to sample a set of data points from the distribution implicitly learned by the autoencoder.

Contractive autoencoders

As a final variant, we want to introduce *contractive autoencoders*, which try to learn a continuous encoding function [172, Chapter 14]: If \vec{x} changes slightly, also \vec{h} should not change much. This desirable behavior is incited by using a regularization term of the form $\sum_j \|\nabla_{\vec{x}} h_j\|^2$ in the training objective. This means, that large derivatives of the learned code are discouraged, which often causes the hidden units of a contractive autoencoder to saturate, i.e., to be close to the extremes \vec{r} of their range [56]. Contractive autoencoders seem to be empirically superior to denoising autoencoders and Bengio et al. [56] explain this empirical observation by three important differences: Firstly, the contractive penalty is applied directly to the learned representation and not to the reconstructions. Secondly, the contractive penalty is analytic unlike the stochastic regularization introduced by denoising autoencoders. Thirdly, the relative influence of the regularization can be explicitly controlled for contractive autoencoders, while it can be only indirectly modified through the type and level of noise in denoising autoencoders.

Sampling the decoder

Since contractive autoencoders are explicitly designed to recover an estimate of the data manifold's tangent plane, repeated encoding and decoding with injected noise corresponds to a random walk along the surface of the manifold [172, Chapter 20]. Thus, the same sampling procedure as described above for denoising autoencoders can again be applied to contractive autoencoders.

Other variants

Of course, it is in principle possible to combine these different approaches with each other, e.g., by training denoising autoencoders, which make use of a bottleneck architecture. When using autoencoders as part of a multitask learning approach, one can furthermore use different parts of the bottleneck layer for different sets of tasks as illustrated by the work of Salakhutdinov and Hinton [345] in the context of metric learning, which has been introduced in Section 5.3.3.

Evaluating autoencoders

When evaluating the representation learned by an autoencoder, one can make use of an extrinsic evaluation based on the test set reconstruction error. However, since a larger capacity of the autoencoder (e.g., through a larger bottleneck layer) typically also leads to a lower reconstruction error even on the test set, the usability of this type of evaluation is clearly limited [56].

As Goodfellow et al. [172, Chapter 15] point out, autoencoders optimizing the MSE on images may not represent small objects in their learned hidden representation, since the absence of such small objects has only a minor influence on the reconstruction error. Goodfellow et al. note that other architectures such as generative adversarial networks (which will be introduced in Section 6.3.3) do not suffer from this problem, since they try to capture any kind of structured pattern in the data. We would also like to add, that the representations learned by regular autoencoders are usually not disentangled and thus hard to interpret by humans.

Limitations of autoencoders

We will make use of a denoising autoencoder with a bottleneck layer in Chapter 12 as part of both a transfer learning and a multitask learning approach, when trying to map images onto points in psychological similarity spaces.

Outlook

6.3.2 Word Embeddings

The field of *distributional semantics* tries to capture the meaning of words based on their usage, building on the *distributional hypothesis*, which states, that words with similar linguistic contexts have similar meanings [251]. The context under consideration for each word can be based on a window of a fixed size (i.e., k words to the left and to the right), on syntactic information (e.g., grammatical dependencies from a parse tree), or on larger units of texts (e.g., paragraphs or documents). Word-based contexts tend to result in representations, that focus on attributional similarity (e.g., CAR being similar to VAN), while document-based contexts tend to emphasize topical similarity (e.g., CAR being similar to DRIVING) [251].

The distributional hypothesis

The meaning of a given word w_i is represented by a so-called *word vector* or *word embedding* \vec{v}_i . In contrast to classical, symbolic definitions of word similarity, this approach is graded and distributed, since it uses continuous values on multiple dimensions to represent each word [251]. Word embeddings thus provide an automated way of quantifying the semantic similarity of words, which can be highly useful in natural language processing (NLP) applications. The similarity of two words w_i and w_j is typically quantified by measuring the cosine of the angle α between their respective vectors \vec{v}_i and \vec{v}_j :

Word embeddings

$$\text{Sim}(w_i, w_j) = \cos \alpha = \frac{\vec{v}_i \cdot \vec{v}_j}{\|\vec{v}_i\| \cdot \|\vec{v}_j\|}$$

Broadly speaking, one can distinguish two types of model families for extracting word embeddings from text corpora [251, 317]: *Matrix-based models* count the overall co-occurrences of words and contexts, and apply dimensionality reduction techniques on the resulting global co-occurrence matrix. *Prediction-based models*, on the other hand, make use of a local context window and try to predict a word given its context. Approaches based on (usually quite shallow) neural networks mostly fall into the second category.

Approaches for obtaining word embeddings

Matrix-based models

The *global co-occurrence matrix* X used in *matrix-based models* contains the raw co-occurrence counts X_{ik} between a given word w_i and a given context c_k . Since both the size V of the overall vocabulary (i.e., the number of different words w_i) and the number K of different contexts c_k is usually quite large, and since individual words tend to occur only in a relatively small number of contexts, this global co-occurrence matrix is usually quite sparse [251]. In order to counteract undesired effects based on word frequency, the values of this co-occurrence matrix are usually normalized before computing similarities. In recent years, the *positive pointwise mutual information (PPMI)* has become a commonly used weighting function [251]:

$$PPMI(w_i, c_k) = \max\left(0, \log \frac{\mathbb{P}(w_i, c_k)}{\mathbb{P}(w_i) \cdot \mathbb{P}(c_k)}\right)$$

Positive pointwise mutual information

PPMI measures how much the probability of a word-context pair (w_i, c_k) is greater than one would expect, if they occurred independently of each other. It can thus be related to mutual information as introduced in Section 5.3.1. One can in theory use the rows of the normalized co-occurrence matrix as word vectors, and compare them using the cosine similarity. However, due to the large size of this matrix and its sparsity, this is quite inefficient both from computational and memory usage perspectives.

Latent semantic analysis

Instead, one often applies dimensionality reduction techniques in order to compress the information from the co-occurrence matrix X and to identify the underlying latent variables determining semantic similarity. For instance, a popular procedure called *Latent Semantic Analysis (LSA)* [243] uses a singular value decomposition (cf. Section 5.3.2) $X = U\Phi V^T$ on a $V \times K$ word-document matrix X , where V specifies the size of the vocabulary and K the number of documents. LSA then uses $M = U\Phi$ as lower-dimensional representation of size $V \times n$ (with $n \ll K$), where each word is now represented by an n -dimensional embedding. While such a compressed representation is advantageous from a computational and storage point of view, the individual dimensions of the resulting word vectors are often no longer interpretable [251]. Nevertheless, this dimensionality reduction step is often associated with large gains in performance and hence considered a core feature of distributional semantics [185].

Word2vec

Two popular examples of the prediction-based models have been proposed by Mikolov et al. [290, 291]. Their *word2vec* approach uses shallow neural networks with a single hidden layer to learn word embeddings based on a prediction task.

Continuous bag of words

Figure 6.10a illustrates the network structure used in the *continuous bag-of-words (CBOW)* model: The network has to predict the current word w_i based on its context, i.e., its surrounding words $w_{i-k}, \dots, w_{i-1}, w_{i+1}, \dots, w_{i+k}$ for a given window size k . The input words are represented by so-called *one-hot vectors* (cf. Section 6.1.1). This means, that the input for each context position is represented by V binary inputs (one for each word from the vocabulary). Out of these

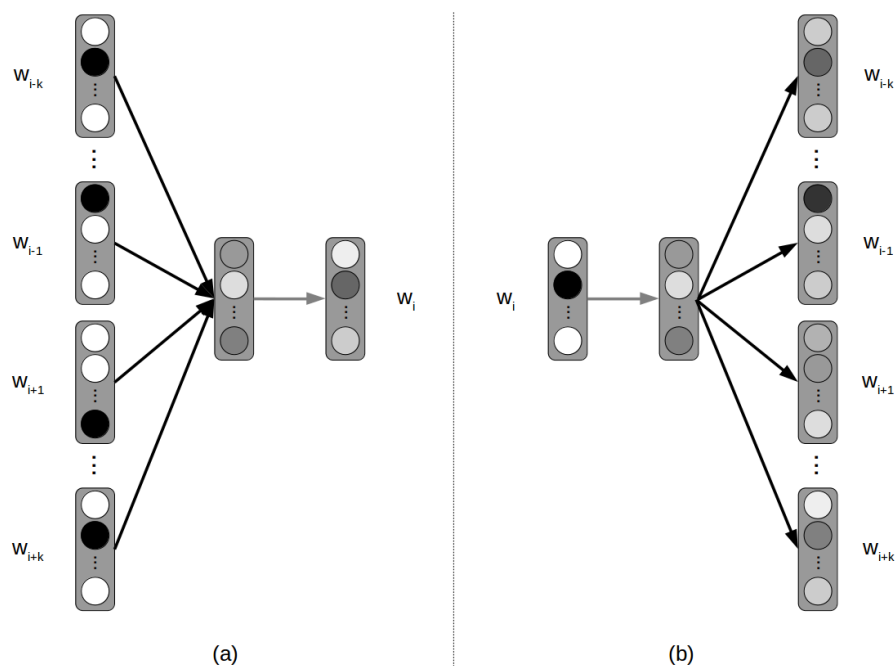


Figure 6.10: Illustration of the shallow network structures used by the CBOW model (a) and the Skip-gram model (b).

V binary inputs, only the one corresponding to the word found at this position in the current context is active, while all others are inactive. This representation is very sparse and high-dimensional, but necessary in order to ensure that the network perceives the individual words as unrelated. If we used only a single input unit and V different input values to encode the V different words, the ordering chosen for mapping words onto values would already imply a similarity relation. Just as the input layer, also the output layer makes use of V different outputs for predicting the target word, implemented as softmax units. The CBOW model uses weight sharing (cf. Section 6.2.3) between the input layer and the hidden layer across all word positions. These weights can then be interpreted as word embeddings, once the network has been successfully trained.

Figure 6.10b shows the structure of the *Skip-gram* model, which uses the inverse prediction task: Given a target word w_i , predict the words $w_{i-k}, \dots, w_{i-1}, w_{i+1}, \dots, w_{i+k}$ from its context. Again, one-hot encodings, softmax outputs, and shared weights are used. Since the number of output units is very large for the Skip-gram model, computing gradients can become computationally challenging. Therefore, different extensions have been proposed to reduce the computational burden. For instance, in a hierarchical softmax output layer, a binary tree with V leaf nodes is used, where each internal node stores the relative probability distribution over its children. This results in $\log V$ instead of V computation steps for obtaining the gradient.

Mikolov et al. [290, 291] were able to show, that the word vectors obtained with CBOW and Skip-gram are not only able to reflect semantic similarity of word pairs through the cosine similarity, but they are also

Skip-gram

*Semantic properties
of word2vec*

able to produce meaningful results for vector additions: For example, \vec{v}_{QUEEN} is the closest vector to the result of $\vec{v}_{\text{KING}} - \vec{v}_{\text{MAN}} + \vec{v}_{\text{WOMAN}}$. Moreover, $\vec{v}_{\text{GERMANY}} + \vec{v}_{\text{CAPITAL}} \approx \vec{v}_{\text{BERLIN}}$. This (perhaps surprising) behavior can be used in analogy tasks, such as completing the sentences "MAN is to WOMAN AS KING is to ..." (targeting semantic structures) or "BIG is to BIGGER AS SMALL is to ..." (targeting syntactic structures).

Global vectors

The ability of word vectors to capture such syntactic and semantic regularities has been used as a motivation for the *GloVe* (Global Vectors) model [317], which attempts to unify aspects from matrix-based methods (namely, their focus on global co-occurrence statistics) and prediction-based methods (namely, their ability to perform good in analogy tasks).

Ratios of
co-occurrence
probabilities

Given two *focus words* w_i and w_j , Pennington et al. [317] argue, that the ratio of their co-occurrence probabilities with other *context words* w_k should be used as a starting point for defining semantic similarity. For instance, if $w_i = \text{ICE}$ and $w_j = \text{STEAM}$, then the fraction $\frac{\mathbb{P}(w_k|w_i)}{\mathbb{P}(w_k|w_j)}$ (where $\mathbb{P}(w_k|w_i)$ and $\mathbb{P}(w_k|w_j)$ are estimated from the co-occurrence matrix X) should roughly equal one for context words w_k , which are related to both words (e.g., $w_k = \text{WATER}$) or to neither of them (e.g., $w_k = \text{FASHION}$). However, for context words such as $w_k = \text{SOLID}$ or $w_k = \text{GAS}$, which are related to only one of the target words, the fraction $\frac{\mathbb{P}(w_k|w_i)}{\mathbb{P}(w_k|w_j)}$ is expected to be relatively large or small, respectively.

Modeling
co-occurrence
probabilities with
vectors

Pennington et al. propose to model the conditional probabilities $\mathbb{P}(w_k|w_i)$ based on the inner product of the embedding \vec{v}_i of the focus word w_i and the embedding \vec{v}'_k of the context word w_k , using a nonlinear transformation function F :

$$\mathbb{P}(w_k|w_i) = F(\vec{v}_i^T \cdot \vec{v}'_k)$$

A more specific model

This means that the same word w_j has two different embeddings \vec{v}_j and \vec{v}'_j , depending on whether it is being used as a focus word or as a context word. By using $F(x) = e^x$ and by approximating $\mathbb{P}(w_k|w_i)$ based on the entries X_{ik} of the co-occurrence matrix X , Pennington et al. obtain the following model:

$$\left(e^{\vec{v}_i^T \cdot \vec{v}'_k} = \frac{X_{ik}}{\sum_{k'=1}^K X_{ik'}} \right) \Leftrightarrow \left(\vec{v}_i^T \cdot \vec{v}'_k = \ln(X_{ik}) - \ln \left(\sum_{k'=1}^K X_{ik'} \right) \right)$$

Main constraint for
the embeddings

Since $\sum_{k'=1}^K X_{ik'}$ is independent of w_k , Pennington et al. replace the term $\ln \left(\sum_{k'=1}^K X_{ik'} \right)$ with a constant bias term b_i , and also introduce another bias term b'_k for the context word w_k , leading to their final constraint for the embeddings \vec{v}_i and \vec{v}'_k :

$$\vec{v}_i \cdot \vec{v}'_k + b_i + b'_k = \ln(X_{ik}) \quad (6.2)$$

In order to ensure, that rare co-occurrences are not underweighted and large co-occurrences are not overweighted, Pennington et al. introduce a weighting function f for the individual matrix entries X_{ik} , using a threshold x_{max} and a scaling strength α :

Weighting function

$$f(X_{ik}) = \begin{cases} \left(\frac{X_{ik}}{x_{max}}\right)^\alpha & \text{if } X_{ik} < x_{max} \\ 1 & \text{if } X_{ik} \geq x_{max} \end{cases}$$

The overall optimization objective for their GloVe model can then be formulated as a least squares problem by using Equation 6.2 and a weighted sum over all pairs of words in the vocabulary:

Overall loss function

$$J = \sum_{i=1}^V \sum_{k=1}^V f(X_{ik}) (\vec{v}_i^T \vec{v}'_k + b_i + b'_k - \ln(X_{ik}))^2$$

The optimal embeddings \vec{v}_i and \vec{v}'_k as well as the optimal bias terms b_i and b'_k can then be found using gradient descent. Pennington et al. were able to show, that their GloVe model is indeed able to outperform both CBOV and Skip-gram with respect to analogy and word similarity tasks, as well as in named entity recognition as an exemplary NLP task.

Results

While the similarity judgments provided by word embeddings have been identified as useful features in various areas of natural language processing, Lenci [251] has noted that from the view of computational linguistics, there are still open problems associated with distributional semantics. For instance, he notes that word embeddings often seem to confound semantic similarity, syntactic information, and general topic-based relatedness. Stripping apart these different notions of similarity in a systematic way is still an ongoing area of research. Moreover, Lenci discusses the problem of compositionality: Embeddings are usually created for individual words or short phrases (such as WHITE HOUSE). It is, however, unclear how to combine the embeddings of these atomic terms in order to model the meaning of complete sentences or paragraphs. In practice, taking the average across all individual words is a common approach, which is, however, not able to distinguish the semantically quite different events described by the phrases "dog bites man" and "man bites dog".

Limitations of word embeddings

Günther et al. [185] on the other hand argue, that word embeddings have been successfully applied in various areas of cognitive science, including artificial intelligence, psycholinguistics, and cognitive neuroscience. According to Günther et al., the portrayal of these models is often based on common misconceptions. For example, they refer to several studies demonstrating that word embeddings can also be learned in an online fashion (rather than the psychologically implausible batch processing), that different word meanings can be extracted based on clustering techniques, and that word embeddings can be extended to a multimodal setting, where for instance also word-image co-occurrences are considered in addition to the classical word-word or word-document statistics. Moreover, Günther et al. argue, that also

Merit of word embeddings

humans presumably leverage statistical regularities in language learning. Finally, since language is often used to communicate about the real world, Günther et al. note that word embeddings are indirectly embodied and can thus in principle also capture non-linguistic aspects of meaning.

*Word embeddings
and semantic
properties*

Sommerauer and Fokkens [382] have investigated to which extent semantic properties are reflected by word embeddings. They collected human ratings with respect to both interaction-based properties (such as object functions or activities, in which they often partake) and perceptual properties (such as COLOR or SHAPE) for a set of selected words. They assumed that interaction-based properties are well reflected by word embeddings, but that perceptual properties are not. Sommerauer and Fokkens used a leave-one-out evaluation on both a logistic regression and a feedforward neural network trained on top of pretrained Skip-gram embeddings to classify whether a property applies to a word or not. They compared the performance of these classifiers to a simple centroid-based approach, where the centroid of the positive examples was computed, its k nearest neighbors were classified as positive, and all other vectors as negative.

*Word embeddings
capture interactions
better than
perception*

Sommerauer and Fokkens observed that perceptual properties such as BLACK, or WOODEN were not recoverable from the embeddings, which indicates, that the textual evidence for perceptual properties is too sparse. Their classifier-based approach was in general more successful than the centroid-based approach. For instance, the classifiers were able to identify a sub-group of vehicles to which HAS WHEELS applies, as well as words outside the categories of weapons and carnivorous animals, which had been labeled as DANGEROUS. Also a distinction into separate word senses (such as CHICKEN, which can both refer to an animal or to food) was possible for the classifiers, while the centroid-based approach was mostly bound by the majority class. Overall, although Sommerauer and Fokkens state, that their results were not yet conclusive, their study illustrates that word embeddings lack perceptual information, and that other semantic properties can be extracted in principle, but require additional classifiers instead of being directly accessible through the cosine similarity between a query vector and a prototype.

*Word embeddings
and numerical
attribtues*

In a related strand of research, Gupta et al. [176] have investigated, whether specific attributes (such as the number of inhabitants of a country) can be extracted from pretrained word embeddings. They used a pretrained Skip-gram model and trained a logistic regression for predicting both categorical and numeric attributes of countries and cities (the latter were rescaled to the interval $[0, 1]$ to allow for the application of a logistic regression). According to their results, the logistic regression performed considerably better than simple baselines (predicting the majority class and the average numeric value, respectively), indicating that word embeddings are indeed able to capture also more structured information to at least some degree.

*Word embeddings
and adjectival scales*

Kim and de Marneffe [222] have shown, that it is also possible to recover several adjectival scales such as TERRIBLE < BAD < OKAY < GOOD < EXCELLENT from pretrained word embeddings: They started by identify-

ing two endpoints of such a scale (e.g., TERRIBLE and EXCELLENT) and by retrieving their respective word vectors \vec{v}_a and \vec{v}_b . They then defined a midpoint $\vec{v}_2 = \vec{v}_b + (\vec{v}_a - \vec{v}_b)/2$ as well as two quarter points \vec{v}_1 and \vec{v}_3 and identified the words, whose embeddings were most similar to these three intermediate points. Their analysis shows, that in most cases, these intermediate points described a meaningful scale. Kim and de Marneffe furthermore used this approach to classify indirect responses to yes/no questions such as "Is Obama qualified? - I think he is young" by checking whether YOUNG lies on the positive or on the negative side of the scale induced by the endpoints QUALIFIED and UNQUALIFIED. Their work thus suggests, that while the individual coordinates of a word embedding are not necessarily interpretable, it is possible to identify directions in the embedding space, which correspond to meaningful adjectival scales.

Rothe and Schütze [338] propose to decompose a given embedding space into interpretable orthogonal subspaces. These subspaces can then be used to make targeted modifications to a given word embedding. More specifically, they consider subspaces for polarity (which allows one to find antonyms, e.g., FRIEND-FOE), concreteness (allowing for abstraction, e.g., FRIEND-FRIENDSHIP), frequency (used to find less frequent synonyms, e.g., FRIEND-COMRADE), and part of speech (reflecting grammatical changes, e.g., FRIEND-BEFRIEND). By using orthogonal subspaces of the overall space, they leave the structure of the global embedding space intact and tacitly assume that the different subspaces are independent of each other. Their work indicates, that such interpretable subspaces can be extracted for at least some domains of interest from pretrained embedding spaces as provided by the word2vec model.

Instead of analyzing a pretrained space, Jameel et al. [207, 208] have proposed an approach, which constrains the training of word embeddings explicitly in such a way, that each semantic type is represented by a particular subspace. They used the GloVe model as a starting point and introduced additional constraints based on knowledge graph embeddings. Overall, their loss function consists of three parts:

- J_{text} is equivalent to the loss function used by GloVe, but models the co-occurrence of entities and words instead of the word-word co-occurrence used by the original GloVe model.
- J_{type} ensures, that all entities of a given semantic type (e.g., HUMAN OR ANIMAL) can be written as a convex combination of $n + 1$ support points in the overall n -dimensional embedding space. This constraint by itself has a trivial solution, but it is complemented by a regularization term J_{type}^{reg} , which requires the space spanned by these $n + 1$ support points to be as low-dimensional as possible. Taken together, J_{type} and J_{type}^{reg} thus ensure, that each semantic type is represented by a relatively low-dimensional subspace.
- J_{rel} incorporates relations from a knowledge graph into the model. It considers tuples of the form (e, r, f) , which represent that the

Decomposing the embedding space into orthogonal subspaces

Learning orthogonal subspaces based on GloVe

relation r holds between the entities e and f , and ensures that f lies in the same subspace as the result obtained by $e + r$. Again, a regularization term J_{rel}^{reg} ensures, that this relation-induced subspace is as low-dimensional as possible.

Experimental results

Jameel et al. were able to show that their model performs better with respect to ranking, induction, analogy making, and knowledge graph embedding than both pure word embedding models (namely, CBOW and Skip-gram) and pure knowledge graph embedding models. Their approach thus exemplifies that a domain-like structure can in principle be learned from textual data.

Learning domain structures based on global co-occurrences

McGregor et al. [286] have also attempted to construct interpretable subspaces of a distributional model. Unlike the other approaches discussed so far, they, however, start with a global co-occurrence matrix of word-word combinations, weighted with PPMI. In order to define a new domain (such as CARNIVORES), they use a small set of relevant words (such as LION, BEAR, and TIGER), and identify the columns of this global co-occurrence matrix, which have non-zero entries for all of these seed words. These columns correspond to context words (such as KILL, HUNT, and PREY), which co-occur with all of the seed words. After normalizing the respective columns (in order to counteract frequency effects), McGregor et al. select a small subset of these columns based on their average value for the given seed words. This means, that they select the context words, which are strong indicators for the seed words. The underlying intuition of this step is that features with a high mean value are very salient for the given conceptual domain. In order to identify other members of the given conceptual domain (e.g., WOLF), McGregor et al. first project the overall co-occurrence matrix onto the subspace spanned by the selected feature subset. Afterwards, they propose to either use the Euclidean distance to the prototype (i.e., the average among the seed words) or to use the distance from the origin (compared to the respective distance of the seed words) to determine concept membership. Their experimental evaluation shows that their approach achieves similar performance levels as Skip-gram and GloVe on a concept membership list completion task, and that the behavior of their model aligns also relatively well with human conceptualizations.

Word embeddings and conceptual spaces

Let us now consider word embeddings from the perspective of conceptual spaces. Remember that conceptual spaces are spanned by a small number of interpretable dimensions, which are based on sensory input such as vision. Moreover, the overall conceptual space is structured into different semantic domains based on different perceptual modalities (e.g., COLOR vs. SHAPE). Semantic similarity is inversely related to the Euclidean or Manhattan distance, depending on whether the dimensions involved belong to the same or to different domains.

Differences

Word embeddings, on the other hand, are purely based on textual data and thus not directly related to sensory information. The embedding spaces themselves are usually very high-dimensional and their dimensions are not interpretable. Finally, the cosine of the angle

between two vectors is used in order to measure their similarity, not the Euclidean or Manhattan distance of the points they represent.

Despite these obvious differences, one can, however, also see certain parallels between word embeddings and conceptual spaces: Both approaches use a geometric approach for defining semantic similarity, and in both cases, these spaces support classification and commonsense reasoning strategies such as analogical reasoning. Moreover, as illustrated with the various approaches discussed above, it also seems to be possible to extract or enforce domain-like structures in word embedding spaces. One may thus view word embeddings as a more language-centric and engineering-oriented variant of the more cognitively and psychologically motivated conceptual spaces framework.

This relation between the two approaches has been illustrated by the work of Derrac and Schockaert [123] (cf. Section 1.2.5), who have created conceptual spaces based on the cosine similarities extracted from co-occurrence matrices. Essentially, they collected a co-occurrence matrix for entities (corresponding to target words) and tags (corresponding to contexts) and normalized this matrix using PPMI. Instead of applying singular value decomposition for reducing the dimensionality of the representation, they used the cosine similarity of the PPMI vectors for all pairs of entities to define a global similarity matrix. Derrac and Schockaert then used multidimensional scaling (to be introduced in Chapter 8) to find a lower-dimensional representation of the entities in a Euclidean space, enforcing that the Euclidean distance between two entities was inversely related to the cosine similarity of their PPMI vectors. Their work can thus be interpreted as a way of translating word embeddings to conceptual spaces. We will discuss their approach in greater detail in Chapter 8 in the context of multidimensional scaling.

Similarities and links

*Obtaining
conceptual spaces
from word
embeddings*

6.3.3 Generative Adversarial Networks

Similar to autoencoders, *generative adversarial networks (GANs)* [173] consist of two models, which are implemented by neural networks (cf. Figure 6.11): The *generative model* $G(\vec{z}, \theta_G)$ (implemented as *generator network*) maps its noise input \vec{z} (which follows some probability distribution $\mathbb{P}(\vec{z})$) onto the feature space, which is used to represent the data points \vec{x} . The family of possible mapping functions is given by the network architecture, and its parameters θ_G are optimized in order to create realistic samples. The *discriminative model* $D(\vec{x}, \theta_D)$ (implemented as *discriminator network*) tries to discriminate between samples, that were drawn from the real data distribution \mathbb{P}_{data} , and samples $G(\vec{z})$ created by the generative model. $D(\vec{x}, \theta_D)$ outputs a single scalar representing the probability that \vec{x} came from the real data distribution \mathbb{P}_{data} . The parameters θ_D are optimized in order to maximize classification accuracy. In most cases, GANs are applied to image data.

*Generative
adversarial networks:
generator and
discriminator*

The overall GAN setup can be interpreted in game-theoretic terms as a *minimax game*: There is a single *value function*, which is being

*GANs and game
theory*

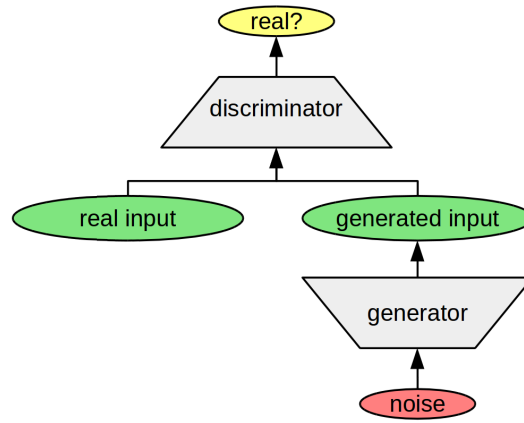


Figure 6.11: Structure of generative adversarial networks [173] with generator and discriminator.

maximized by one player (in our case the discriminator), while being minimized by another player (in our case the generator). While the discriminator tries to distinguish real inputs from generated ones, the generator tries to fool the discriminator into accepting its generated data points as "real". The overall value function of the GAN framework is defined as follows, where \mathbb{P}_{data} is the probability distribution of the real data points \vec{x} from an unlabeled dataset:

$$V(D, G) = \mathbb{E}_{\vec{x} \sim \mathbb{P}_{data}(\vec{x})} [\log D(\vec{x})] + \mathbb{E}_{\vec{z} \sim \mathbb{P}(z)} [\log(1 - D(G(\vec{z})))]$$

Interpreting the value function

The first term of this equation is based on the output of the discriminator for data points from the real dataset (e.g., for real images) and grows monotonically with $D(\vec{x})$. The second term of this equation is based on the output of the discriminator for data points, which were created by the generator based on the noise variables. This term grows as the value of $D(G(\vec{z}))$ becomes smaller.

Objectives of generator and discriminator

The generator G tries to minimize the overall value of $V(D, G)$. This can only be done through the second term, i.e., by maximizing $D(G(\vec{z}))$. Hence, the generator updates its weights in such a way, that the discriminator misclassifies its generated images as real. The discriminator D , on the other hand, tries to maximize the overall value of $V(D, G)$. This corresponds to maximizing $D(\vec{x})$ and minimizing $D(G(\vec{z}))$, i.e., distinguishing real data points from generated ones.

Optimal solution to the minimax game

Goodfellow et al. showed that this overall minimax game has a global optimum for $\mathbb{P}_G = \mathbb{P}_{data}$, i.e., when the generator's output distribution equals the distribution of the real data points. One can guarantee that this optimum is reached, if both the generative and the discriminative model have unlimited capacity. There is, however, no guarantee that a practical implementation of the GAN framework with networks of finite capacity will reach convergence.

The parameters of the two neural networks are estimated by using gradient descent. The overall training procedure consists of two nested loops: In each iteration of the outer loop, the weights of the generator are updated. This is done, after the weights of the discriminator have been updated k times in the inner loop. Goodfellow et al. note that $k = 1$ is sufficient in most cases. Please note, that although the discriminator is trained on a classification task, the overall system works in an unsupervised way: The dataset containing the real examples $\vec{x}^{(j)}$ does not need any class labels $y^{(j)}$, since the classification task of the discriminator involves distinguishing real from generated inputs. The labels needed for this classification task can be generated on the fly, by labeling all real examples \vec{x} with the positive class and all generated examples $G(\vec{z})$ with the negative class

Training procedure

In general, the generator might learn very slowly in the beginning of the training procedure, when its outputs are of low quality. In such a scenario, the discriminator can easily distinguish real outputs from generated ones, leading to a low value of $D(G(\vec{z}))$. This in turn makes $1 - D(G(\vec{z}))$ relatively large. Since the derivative of the logarithmic function decreases with increasing input, the gradient of $\log(1 - D(G(\vec{z})))$ used to update the generator's weights might be relatively small, thus leading to only small improvements. In order to speed up initial training, Goodfellow et al. suggest to maximize $\log D(G(\vec{z}))$ for the generator instead of minimizing $\log(1 - D(G(\vec{z})))$. In both cases, the generator tries to increase $D(G(\vec{z}))$, but for small values of $D(G(\vec{z}))$, $\log D(G(\vec{z}))$ provides a much steeper gradient than $\log(1 - D(G(\vec{z})))$.

Speeding up initial generator training

Training GANs is generally considered to be a difficult task [14]. It seems, that convergence to an equilibrium state is not always possible and depends on many factors [172, Chapter 20]. Among others, using dropout (cf. Section 6.2.3) in the discriminator seems to be an important ingredient for successful training [172, Chapter 20]. These training difficulties have sparked a variety of modifications to the original GAN framework. For instance, Durugkar et al. [141] showed, that by using multiple discriminator networks at the same time, one can obtain a faster convergence than with standard GANs. Moreover, modifying the minimax objective for accelerating early training was no longer necessary in this setting. Zhao et al. [458] interpret the discriminator as providing an energy function, which assigns low energies to regions near the data manifold. They propose to model this energy function as the reconstruction error of an autoencoder which replaces the standard discriminator network in their proposal. Also they observed more stability in training compared to the original GAN setup. Moreover, they argue, that their modified setup can be interpreted as a denoising autoencoder (cf. Section 6.3.1), where the corrupted examples are not obtained through the application of random noise, but sampled from the generator network.

GAN variants with improved training stability

Despite its difficulties in training, the GAN framework has been highly successful in various application domains. Radford et al. [326] have used convolutional layers (cf. Section 6.2.2) in order to apply

Applications of GANs

GANs to image data. Their results showed that the latent code (i.e., the noise vector \vec{z}) covers the space of possible images in a meaningful way. They demonstrate this by using simple vector arithmetics similar to what has been discussed for word embeddings in Section 6.3.2. Wu et al. [440] have applied GANs to the generation of three-dimensional objects, using volumetric convolutional layers. They have furthermore trained a variational autoencoder (to be introduced in Section 6.3.4) on two-dimensional images and have used its latent code as an input to the generator network, allowing them to generate a three-dimensional representation based on a two-dimensional input. Recently, Brock et al. [80] have scaled up the GAN framework to higher-resolution images of size 128×128 by noting, that the trade-off between sample quality and sample variety can be controlled through truncating the distribution of the latent variables \vec{z} after training. Zhu et al. [459] have introduced CycleGAN, which aims to learn a mapping between two domains (e.g., images of horses and images of zebras) by employing a GAN-like setting: For each translation direction, they train a generator (mapping from source to target domain) and a discriminator (trying to distinguish real images from translated ones). By furthermore introducing a reconstruction loss (enforcing that translating from source to target and back should result again in the original image), they were able to successfully learn such mappings without needing paired data.

GAN representations are not interpretable

One important disadvantage of GANs is that the individual dimensions of the noise vector \vec{z} are not meaningful. Although we can manipulate them in order to generate new images (e.g., by interpolating between two noise vectors), it is difficult to generate an image with certain desired semantic properties, because these properties cannot be readily accessed or directly specified.

InfoGAN: learning disentangled representations

Chen et al. [101] have proposed an information-theoretic extension to GANs, which is called *InfoGAN*. Its explicit goal is to learn disentangled representations (cf. Section 6.1.1) in a completely unsupervised manner. Their proposed network architecture follows the overall GAN principle, and adds an incentive for the generator to maximize the mutual information between a subset of the noise variables and its output.

Splitting the generator input

Chen et al. modify the GAN framework by splitting the generator input into two parts: On the one hand, there is a vector \vec{z} , which is treated as the source of incompressible noise. The noise coming from \vec{z} allows the generator to create example data points, which share the same underlying semantic features, but may differ in their surface level features. On the other hand, a second vector \vec{c} (called the *latent code*) is used to target salient structured semantic features of the dataset. Chen et al. assume that \vec{c} is a concatenation of individual latent variables, whose probability distributions are independent of each other.

Mutual information

If we only use \vec{c} as an additional input to the generator without modifying the training objective, the generator might simply discard it. Chen et al. propose to solve this problem by introducing an information-theoretic regularization term into the overall GAN loss function. This additional regularization term tries to ensure, that the mutual infor-

mation between the latent code \vec{c} and the distribution of the generator output $G(\vec{z}, \vec{c})$ is high. Please recall from Section 5.3.1, that the mutual information MI between two random variables A and B is defined as follows, based on the entropy H :

$$MI(A, B) = H(A) - H(A|B) = H(B) - H(B|A)$$

Please recall from Section 5.2.4, that the entropy $H(A)$ measures the amount of uncertainty associated with a random variable A and is defined as $H(A) = \mathbb{E}_{A \sim P(A)}[-\log(P(A))]$. Mutual information therefore quantifies how much of this uncertainty about A is removed if we are allowed to observe B . In the case of InfoGAN, we are interested in measuring how much uncertainty about the latent code \vec{c} can be removed by observing the generator output $G(\vec{z}, \vec{c})$. We can include this new regularization term into the GAN loss function as follows:

Adding a regularization term

$$\begin{aligned} V_{MI}(D, G) &= V(D, G) - \lambda \cdot MI(\vec{c}, G(\vec{z}, \vec{c})) \\ &= \mathbb{E}_{\vec{x} \sim P_{data}(\vec{x})}[\log D(\vec{x})] + \mathbb{E}_{\vec{z} \sim P(\vec{z})}[\log(1 - D(G(\vec{z})))] \\ &\quad - \lambda \cdot MI(\vec{c}, G(\vec{z}, \vec{c})) \end{aligned}$$

Chen et al. argue, that we need access to the posterior distribution $\mathbb{P}(\vec{c}|\vec{x})$ in order to compute $MI(\vec{c}, G(\vec{z}, \vec{c}))$. This can be seen by first applying the definitions of mutual information and entropy, respectively:

Computing the mutual information term

$$\begin{aligned} MI(\vec{c}, G(\vec{z}, \vec{c})) &= H(\vec{c}) - H(\vec{c} | G(\vec{z}, \vec{c})) \\ &= H(\vec{c}) - \mathbb{E}_{\vec{x} \sim P_G(\vec{z}, \vec{c})}[\mathbb{E}_{\vec{c}' \sim \mathbb{P}(\vec{c}|\vec{x})}[-\log \mathbb{P}(\vec{c}'|\vec{x})]] \\ &= H(\vec{c}) + \mathbb{E}_{\vec{x} \sim P_G(\vec{z}, \vec{c})}[\mathbb{E}_{\vec{c}' \sim \mathbb{P}(\vec{c}|\vec{x})}[\log \mathbb{P}(\vec{c}'|\vec{x})]] \end{aligned}$$

The posterior distribution $\mathbb{P}(\vec{c}|\vec{x})$ is, however, not available in practice. Chen et al. solve this problem by defining an auxiliary distribution $\mathbb{Q}(\vec{c}|\vec{x})$, which approximates $\mathbb{P}(\vec{c}|\vec{x})$. Since \mathbb{Q} is only an approximation of \mathbb{P} , we introduce an approximation error, which can be captured with the Kullback-Leibler (KL) divergence D_{KL} . Please recall from Section 5.1.3, that D_{KL} measures the difference between two probability distributions and is defined as follows:

Introducing an approximation

$$D_{KL}(\mathbb{P}(X)||\mathbb{Q}(X)) = \mathbb{E}_{X \sim \mathbb{P}(X)}[\log \mathbb{P}(X) - \log \mathbb{Q}(X)]$$

We can thus rewrite our formula for the mutual information as follows:

$$\begin{aligned} MI(\vec{c}, G(\vec{z}, \vec{c})) &= H(\vec{c}) + \mathbb{E}_{\vec{x} \sim P_G(\vec{z}, \vec{c})}[\mathbb{E}_{\vec{c}' \sim \mathbb{P}(\vec{c}|\vec{x})}[\log \mathbb{P}(\vec{c}'|\vec{x})]] \\ &= H(\vec{c}) + \mathbb{E}_{\vec{c}' \sim \mathbb{P}(\vec{c}|\vec{x})}[\log \mathbb{Q}(\vec{c}'|\vec{x})] \\ &\quad + \mathbb{E}_{\vec{x} \sim P_G(\vec{z}, \vec{c})}[D_{KL}(\mathbb{P}(\vec{c}|\vec{x})||\mathbb{Q}(c|\vec{x}))] \end{aligned}$$

Since the KL divergence is always non-negative, we can derive a lower bound for $MI(\vec{c}, G(\vec{z}, \vec{c}))$ by removing D_{KL} from our formula:

Obtaining a lower bound

$$\begin{aligned} MI(\vec{c}, G(\vec{z}, \vec{c})) &= H(\vec{c}) + \mathbb{E}_{\vec{x} \sim P_G(\vec{z}, \vec{c})}[\mathbb{E}_{\vec{c}' \sim \mathbb{P}(\vec{c}|\vec{x})}[\log \mathbb{Q}(\vec{c}'|\vec{x})]] \\ &\quad + D_{KL}(\mathbb{P}(\vec{c}|\vec{x})||\mathbb{Q}(c|\vec{x})) \\ &\geq H(\vec{c}) + \mathbb{E}_{\vec{x} \sim P_G(\vec{z}, \vec{c})}[\mathbb{E}_{\vec{c}' \sim \mathbb{P}(\vec{c}|\vec{x})}[\log \mathbb{Q}(\vec{c}'|\vec{x})]] \end{aligned}$$

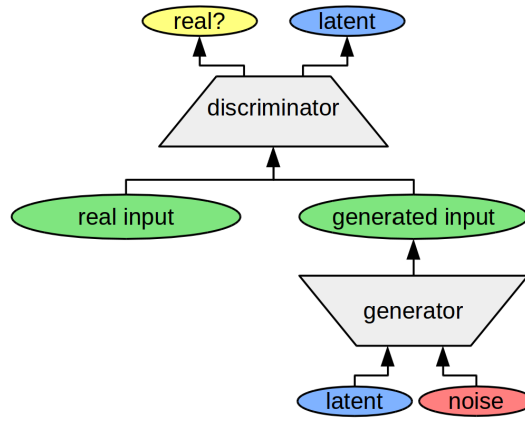


Figure 6.12: Structure of InfoGAN [101], which adds a reconstruction objective for part of the noise vector to the general GAN framework.

Rewriting the lower bound

Finally, Chen et al. [101] were able to prove that we can collapse the two nested expected values into a single term:

$$\begin{aligned}
 MI(\vec{c}, G(\vec{z}, \vec{c})) &\geq H(\vec{c}) + \mathbb{E}_{\vec{x} \sim P_G(\vec{z}, \vec{c})} [\mathbb{E}_{\vec{c}' \sim P(\vec{c}|\vec{x})} [\log Q(\vec{c}'|\vec{x})]] \\
 &\stackrel{(v)}{=} H(\vec{c}) + \mathbb{E}_{\vec{c} \sim P(\vec{c}), \vec{x} \sim P_G(\vec{z}, \vec{c})} [\log Q(\vec{c}|\vec{x})] \\
 &= L_{MI}(G, Q)
 \end{aligned}$$

Maximizing the lower bound

Instead of maximizing $MI(\vec{c}, G(\vec{z}, \vec{c}))$ as part of the InfoGAN loss function, we can now use the lower bound $L_{MI}(G, Q)$:

$$\begin{aligned}
 V_{MI}(D, G) &= V(D, G) - \lambda \cdot L_{MI}(G, Q) \\
 &= \mathbb{E}_{\vec{x} \sim P_{data}(\vec{x})} [\log D(\vec{x})] + \mathbb{E}_{\vec{z} \sim P(\vec{z})} [\log(1 - D(G(\vec{z})))] \\
 &\quad - \lambda \cdot (\mathbb{E}_{\vec{c} \sim P(\vec{c}), \vec{x} \sim P_G(\vec{z}, \vec{c})} [\log Q(\vec{c}|\vec{x})] + H(\vec{c}))
 \end{aligned}$$

Implementation with neural networks

In practice, the auxiliary distribution Q is implemented through a neural network. Chen et al. propose that this network Q should share all but the last layer with the discriminator D , which adds only negligible overhead to the regular GAN framework. This network Q then predicts $Q(\vec{c}|\vec{x})$, i.e., the conditional probability of a latent code \vec{c} for a given data point \vec{x} . For categorical variables in \vec{c} , a softmax activation is used for the output of Q , while for continuous variables Chen et al. propose to use a Gaussian distribution which is determined by its mean and variance. The output of the network Q is then trained on the data points created by the generator, and by trying to minimize the difference between its prediction for \vec{c} and the actual \vec{c} used to generate the data point. The overall resulting network structure is illustrated in Figure 6.12. As one can see, this reconstruction of \vec{c} introduces a structure similar to an autoencoder. Chen et al. recommend to fix the hyperparameter λ to one, if \vec{c} contains categorical variables, and to lower it (up to a factor of 10), if \vec{c} contains only continuous variables in order to ensure that $L_{MI}(G, Q)$ and $V(D, G)$ are on the same scale.

Experimental results

In their paper, Chen et al. report results on five different datasets, including MNIST (a dataset of hand-written digits) and CelebA (a

dataset of celebrity faces). On the MNIST dataset, the latent code \vec{z} consisted of one categorical variable c_1 with ten possible values and two continuous variables c_2 and c_3 with a uniform distribution on the interval $[-1, 1]$. The InfoGAN architecture was able to discover that there are ten different classes in the dataset (namely, the digits from ZERO to NINE). Moreover, it extracted the interpretable dimensions of digit ROTATION (c_2) and digit WIDTH (c_3). On the one hand, these are quite impressive results, because they were obtained in an unsupervised way. On the other hand, Chen et al. use quite a large amount of prior knowledge in setting up the experiment, e.g., by defining that c_1 should consist of exactly ten categories. While for their MNIST example, all latent variables proved to be semantically meaningful at the end of the training procedure, only a subset of \vec{z} became meaningful in their remaining experiments.

InfoGAN was among the first neural architectures claiming to learn disentangled representations. As such, it is clearly of interest when aiming to learn a mapping from the subsymbolic to the conceptual layer, i.e., when trying to ground conceptual spaces in sensory data. However, due to the instabilities in training inherited from the general GAN framework, its applicability in practice remains unclear. Moreover, the disentanglement claims by Chen et al. are solely based on visual inspection without using any formal evaluation criteria like the ones introduced in Section 6.1.2. Finally, at least the MNIST results presented by Chen et al. seem to be to some extent cherry-picked, since we were unable to reproduce them, even when using their original source code. In the context of learning disentangled representations, most researchers have focused on architectures based on variational autoencoders, which will be introduced in the upcoming section.

Outlook

6.3.4 Variational Autoencoders

Variational autoencoders (VAE) can be interpreted as a recent probabilistic variant of regular autoencoders. Kingma and Welling [224] base their derivation of variational autoencoders on a simple graphical model shown in Figure 6.13. They treat the observable data points as a random variable X , and assume, that this observable random variable is generated based on the value of an unobservable latent variable Z . The values of this random variable Z are drawn from a prior distribution $\mathbb{P}_\theta(\vec{z})$ with parameters θ . The conditional probability distribution $\mathbb{P}_\theta(\vec{x}|\vec{z})$ describes the process of generating a data point \vec{x} based on concrete values \vec{z} of the underlying latent variables. The solid lines in Figure 6.13 illustrate this generation process. In our case, \vec{x} represents an image and \vec{z} represents the latent code describing semantic features of this image.

The graphical model of variational autoencoders

Given this graphical model, we would like to estimate the parameters θ , as well as the probability distribution $\mathbb{P}_\theta(\vec{x})$ (corresponding to the true distribution over data points) and the conditional probability $\mathbb{P}_\theta(\vec{z}|\vec{x})$,

The learning and inference problem

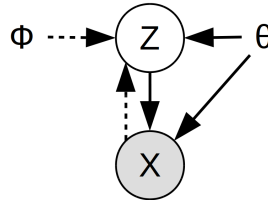


Figure 6.13: Graphical model underlying variational autoencoders.

i.e., a way of extracting semantic features from perceptual inputs. Since this problem is intractable in practice, Kingma and Welling propose to use a recognition model $Q_\phi(\vec{z}|\vec{x})$ (illustrated by dotted lines in Figure 6.13), which approximates the intractable true posterior probability $P_\theta(\vec{z}|\vec{x})$. It makes use of the parameters ϕ , which are in general different from the parameters θ of the true probability distribution $P_\theta(\vec{z}|\vec{x})$.

The autoencoder view

When framing this in the terms of an autoencoder, the unobserved variable Z can be interpreted as a hidden representation. The recognition model $Q_\phi(\vec{z}|\vec{x})$ can then be thought of as an encoder, because it produces for a given data point \vec{x} a distribution over all possible values of \vec{z} , that could have generated this observation \vec{x} . Conversely, the conditional probability distribution $P_\theta(\vec{x}|\vec{z})$ corresponds to the decoder of an autoencoder, since it produces a distribution over possible values of \vec{x} for a given latent code \vec{z} .

Deriving the loss function

The following derivation of the objective function used in VAEs is based on the detailed tutorial by Doersch [130]. Overall, we try to maximize the probability of the observed data points \vec{x} under the whole generative process, i.e., we want to maximize the following expression:

$$P_\theta(\vec{x}) = \int P_\theta(\vec{x}|\vec{z})P_\theta(\vec{z})d\vec{z}$$

Considering the KL divergence

Unfortunately, this formula is intractable, and can therefore not be directly optimized in practice. Doersch starts the derivation of the objective function by computing the Kullback-Leibler divergence D_{KL} (cf. Section 5.1.3) of the recognition model $Q_\phi(\vec{z}|\vec{x})$ and the true posterior distribution $P_\theta(\vec{z}|\vec{x})$:

$$D_{KL}(Q_\phi(\vec{z}|\vec{x})||P_\theta(\vec{z}|\vec{x})) = E_{\vec{z}\sim Q_\phi(\vec{z}|\vec{x})} [\log Q_\phi(\vec{z}|\vec{x}) - \log P_\theta(\vec{z}|\vec{x})]$$

Applying Bayes' theorem

If one applies Bayes' theorem $P(A|B) = \frac{P(B|A)\cdot P(A)}{P(B)}$ to $P_\theta(\vec{z}|\vec{x})$, one can rewrite this as follows:

$$\begin{aligned} D_{KL}(Q_\phi(\vec{z}|\vec{x})||P_\theta(\vec{z}|\vec{x})) &= E_{\vec{z}\sim Q_\phi(\vec{z}|\vec{x})} \left[\log Q_\phi(\vec{z}|\vec{x}) - \log \left(\frac{P_\theta(\vec{x}|\vec{z}) \cdot \log P_\theta(\vec{z})}{\log P_\theta(\vec{x})} \right) \right] \\ &= E_{\vec{z}\sim Q_\phi(\vec{z}|\vec{x})} [\log Q_\phi(\vec{z}|\vec{x}) - (\log P_\theta(\vec{x}|\vec{z}) + \log P_\theta(\vec{z}) - \log P_\theta(\vec{x}))] \end{aligned}$$

We can now use the fact, that a sum of expected values corresponds to the expected value of this sum to restructure our current expression:

Restructuring the expected values

$$\begin{aligned} D_{KL}(\mathbb{Q}_\phi(\vec{z}|\vec{x})||\mathbb{P}_\theta(\vec{z}|\vec{x})) &= \mathbb{E}_{\vec{z}\sim\mathbb{Q}_\phi(\vec{z}|\vec{x})} [\log \mathbb{Q}_\phi(\vec{z}|\vec{x}) \\ &\quad - (\log \mathbb{P}_\theta(\vec{x}|\vec{z}) + \log \mathbb{P}_\theta(\vec{z}) - \log \mathbb{P}_\theta(\vec{x}))] \\ &= \mathbb{E}_{\vec{z}\sim\mathbb{Q}_\phi(\vec{z}|\vec{x})} [\log \mathbb{Q}_\phi(\vec{z}|\vec{x}) - \log \mathbb{P}_\theta(\vec{z})] \\ &\quad - \mathbb{E}_{\vec{z}\sim\mathbb{Q}_\phi(\vec{z}|\vec{x})} [\log \mathbb{P}_\theta(\vec{x}|\vec{z})] + \log \mathbb{P}_\theta(\vec{x}) \end{aligned}$$

As you may notice, the first expected value corresponds to the KL divergence of $\mathbb{Q}_\phi(\vec{z}|\vec{x})$ and $\mathbb{P}_\theta(\vec{z})$. We can use this insight to rewrite our expression as follows:

Using the KL divergence

$$\begin{aligned} D_{KL}(\mathbb{Q}_\phi(\vec{z}|\vec{x})||\mathbb{P}_\theta(\vec{z}|\vec{x})) &= \mathbb{E}_{\vec{z}\sim\mathbb{Q}_\phi(\vec{z}|\vec{x})} [\log \mathbb{Q}_\phi(\vec{z}|\vec{x}) - \log \mathbb{P}_\theta(\vec{z})] \\ &\quad - \mathbb{E}_{\vec{z}\sim\mathbb{Q}_\phi(\vec{z}|\vec{x})} [\log \mathbb{P}_\theta(\vec{x}|\vec{z})] + \log \mathbb{P}_\theta(\vec{x}) \\ &= D_{KL}(\mathbb{Q}_\phi(\vec{z}|\vec{x})||\mathbb{P}_\theta(\vec{z})) \\ &\quad - \mathbb{E}_{\vec{z}\sim\mathbb{Q}_\phi(\vec{z}|\vec{x})} [\log \mathbb{P}_\theta(\vec{x}|\vec{z})] + \log \mathbb{P}_\theta(\vec{x}) \end{aligned}$$

Finally, by rearranging the terms of the resulting overall equation, we get the following result:

Rearranging terms

$$\begin{aligned} \log \mathbb{P}_\theta(\vec{x}) - D_{KL}(\mathbb{Q}_\phi(\vec{z}|\vec{x})||\mathbb{P}_\theta(\vec{z}|\vec{x})) &= \mathbb{E}_{\vec{z}\sim\mathbb{Q}_\phi(\vec{z}|\vec{x})} [\log \mathbb{P}_\theta(\vec{x}|\vec{z})] - D_{KL}(\mathbb{Q}_\phi(\vec{z}|\vec{x})||\mathbb{P}_\theta(\vec{z})) \quad (6.3) \end{aligned}$$

The left part of Equation 6.3 gives us our maximization target: We want the log likelihood of the observed data to be high, while the KL divergence between the recognition model $\mathbb{Q}_\phi(\vec{z}|\vec{x})$ and the true posterior distribution $\mathbb{P}_\theta(\vec{z}|\vec{x})$ should be low. This means, that the parameters of the model should be trained in such a way, that the data we actually observe in our dataset is very likely under this model. Moreover, the recognition model should be a good approximation of the true posterior distribution.

The optimization target

The left part of Equation 6.3 cannot be directly optimized as noted before, because $\mathbb{P}_\theta(\vec{z}|\vec{x})$ is not available. However, it is equivalent to the right part of Equation 6.3, which is easier to optimize and consists of two parts: The first term computes the log likelihood of the data given the latent code created by the recognition model $\mathbb{Q}_\phi(\vec{z}|\vec{x})$. By maximizing this term, we force our model to make accurate reconstructions of the original data. This again relates VAE to regular autoencoders, which try to minimize the reconstruction error (cf. Section 6.3.1). The second term on the right part of Equation 6.3 is the Kullback-Leibler divergence between the recognition model and the prior probability distribution over the latent code. In order to maximize the overall equation, we need to minimize this KL-divergence, which in turn means, that the recognition model should be as similar as possible to the prior distribution. In terms of neural networks, one can think of this as a regularization term.

The loss function

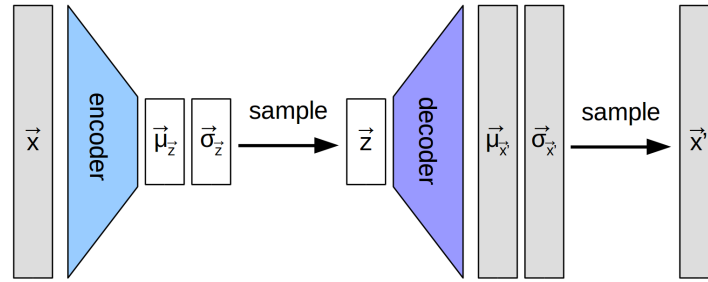


Figure 6.14: Implementation of variational autoencoders with neural networks for $\vec{x} \in \mathbb{R}^n$.

Training procedure

In order to train the VAE model, Kingma and Welling make use of the so-called *reparametrization trick*: Instead of computing the probability distribution over \vec{z} directly, they draw an auxiliary noise variable $\vec{\epsilon}$ and compute $\vec{z} \sim \mathbb{Q}_\phi(\vec{z}|\vec{x})$ as $\vec{z} = g_\phi(\vec{\epsilon}, \vec{x})$. In their experiments, they assume, that \vec{z} follows a centered isotropic multivariate normal distribution, i.e., $\mathbb{P}_\theta(\vec{z}) = N(\vec{z}; 0, I)$. They then express \vec{z} as $z_j = \mu_j + \sigma_j \cdot \epsilon_j$ with $\epsilon_j \sim N(0, 1)$. The expected value for any function $f(\vec{z})$ can be approximated by drawing k samples of $\vec{\epsilon}$ and computing the average across $f(g_\phi(\vec{\epsilon}, \vec{x}))$ for these k samples. Kingma and Welling note that in practice $k = 1$ is sufficient, if the batch size is chosen large enough (e.g., 100 examples per minibatch). For training, any optimization algorithm (like gradient descent or AdaGrad) can be used to optimize the parameters ϕ and θ based on the gradient of the loss function from Equation 6.3.

Implementation with neural networks

For practical experiments, Kingma and Welling use fully connected feedforward networks with a single hidden layer for both $\mathbb{P}_\theta(\vec{x}|\vec{z})$ and $\mathbb{Q}_\phi(\vec{z}|\vec{x})$. They assume that $\mathbb{P}_\theta(\vec{z}) = N(\vec{z}; 0, I)$ and that $\log \mathbb{Q}_\phi(\vec{z}|\vec{x}) = \log N(\vec{z}; \vec{\mu}_z, \vec{\sigma}_z^2 I)$. Since both the prior distribution and the approximate posterior distribution over \vec{z} are Gaussians, their KL-divergence can be computed in closed form. Also $\mathbb{P}_\theta(\vec{x}|\vec{z})$ is assumed to be a multivariate normal distribution (for continuous values in \vec{x}) or a Bernoulli distribution (for binary values in \vec{x}). If we focus on the continuous case only, the resulting network structure can be seen in Figure 6.14. The encoder network (representing $\mathbb{Q}_\phi(\vec{z}|\vec{x})$) takes as input a data point \vec{x} and produces as output the mean $\vec{\mu}_z$ and the standard deviation $\vec{\sigma}_z$ of its distribution. Instead of obtaining a fixed value for the latent code, we therefore obtain a probability distribution over different latent codes. One can then take k samples of this multivariate normal distribution based on the reparametrization trick, and use them as input to the decoder, which represents $\mathbb{P}_\theta(\vec{x}|\vec{z})$. Again, the output of the network consists of the mean $\vec{\mu}_{x'}$ and the standard deviation $\vec{\sigma}_{x'}$ of the output distribution, which can be used to draw different samples \vec{x}' .

Experimental results

Kingma and Welling report experiments on both the MNIST dataset and a dataset of faces. For both experiments, the encoder and decoder had the same number of hidden neurons. For the MNIST dataset, they used 500 hidden units, and for the dataset of faces they reduced the number of hidden neurons to 200 in order to prevent overfitting. Experimental results showed, that the VAE networks tended to converge

quite fast. Moreover, they were able to generate images of relatively high quality despite using only a single hidden layer. Furthermore, the individual entries of the latent code \vec{z} tended to correspond to interpretable semantic properties as judged by manual visual inspection.

Variational autoencoders are generally considered as a simple and elegant approach, that is quite successful in learning manifolds [172, Chapter 20]. Also using their decoder as a generative model of the underlying domain is quite straightforward, since its output already consists of a probability distribution over possible data points, from which one can easily draw samples. However, sample images generated by variational autoencoders tend to be somewhat blurry in practice, which is considered a main drawback in comparison to GANs [172, Chapter 20]. Recently, however, Vahdat and Kautz [414] have proposed a deep convolutional variant of VAE, which is able to produce high-quality images with a size of 256×256 pixels. Their work demonstrates, that given the right architectural choices, also variational autoencoders are in principle capable of producing high-quality samples.

In Section 6.3.3, we mentioned the work by Wu et al. [440], who used the latent code obtained from a variational autoencoder as input to the generator of a GAN. This setup was inspired by the prior work of Larsen et al. [244], who proposed to collapse the decoder of a VAE and the generator of a GAN into a single network. They noted, that VAEs are usually trained using an element-wise reconstruction objective, which is, however, not very useful when applied to images, since for example small translations can cause large element-wise errors, while only causing negligible visual differences. The discriminator in a GAN, on the other hand, learns high-level features for distinguishing real inputs from generated ones. Larsen et al. therefore used the output of the discriminator as a feature-wise reconstruction objective for the variational autoencoder. Their experiments showed an improved visual fidelity over standard VAEs, while still extracting an embedding with interpretable directions.

Higgins et al. [196] have further modified the VAE framework by giving a larger weight β to the regularization term than to the reconstruction error. According to their observations, VAEs are able to extract a disentangled representation from simple datasets, but fail to do so on more complex datasets. Higgins et al. make a slight change to the graphical model on which the VAE framework is based (cf. Figure 6.13): They split up the latent random variable Z into two parts, namely, a set of conditionally independent generative factors V , and a set of conditionally dependent generative factors W . The observed data X is then generated based on both of these factors according to the probability distribution $\mathbb{P}(X|V, W)$. Higgins et al. aim to build a generative model, that learns a hidden representation \vec{z} (which is at least as large as the number of independent generative factors in V), such that $\mathbb{P}_\theta(\vec{x}|\vec{z}) \approx \mathbb{P}(X|V, W)$. Just as in the original VAE framework, a model $\mathbb{Q}_\phi(\vec{z}|\vec{x})$ is used for inferring the posterior probability of the

General properties of VAEs

Combining VAEs with GANs

β -VAE: emphasizing disentangled representations

latent factors \vec{z} given an observation \vec{x} . Again, this can be thought of as an encoder mapping a data point onto a hidden representation.

Conditional independence for the latent code

Ideally, we want $\mathbb{Q}_\phi(\vec{z}|\vec{x})$ to result in a disentangled representation that recovers the original generative factors. This means, that the conditionally independent factors V should be represented in a disentangled manner within a subset of \vec{z} , while the conditionally dependent factors W can be represented in an entangled way within the remaining entries of \vec{z} . Higgins et al. argue, that we can encourage disentanglement by aligning $\mathbb{Q}_\phi(\vec{z}|\vec{x})$ with a prior distribution $\mathbb{P}_\theta(\vec{z})$ over the latent code, which incorporates the assumption of conditional independence. Concretely, they propose to use an isotropic unit normal distribution.

A constrained optimization problem

In addition to extracting a disentangled representation, we would also like to optimize the model in such a way, that the likelihood of the observed data points is maximized. If we use disentanglement as a constraint for optimizing the likelihood of the data given the model, we arrive at the following constrained optimization problem:

$$\begin{aligned} & \text{maximize } \mathbb{E}_{\vec{x} \sim \mathbb{P}_{data}(x)} [\mathbb{E}_{\vec{z} \sim \mathbb{Q}_\phi(\vec{z}|\vec{x})} [\log \mathbb{P}_\theta(\vec{x}|\vec{z})]] \\ & \text{under the constraint } D_{KL}(\mathbb{Q}_\phi(\vec{z}|\vec{x}) || \mathbb{P}_\theta(\vec{z})) < \epsilon \end{aligned}$$

Interpreting the constrained optimization problem

The quantity to be maximized corresponds to the reconstruction fidelity already seen for the standard VAE: We try to maximize the probability of the data points \vec{x} when first extracting the underlying latent code \vec{z} from \vec{x} and then reconstructing \vec{x} from \vec{z} . The optimization constraint measures the Kullback-Leibler divergence between $\mathbb{Q}_\phi(\vec{z}|\vec{x})$ (i.e., the inferred values of \vec{z} for a given \vec{x}) and the assumed prior distribution $\mathbb{P}_\theta(\vec{z})$ (i.e., the isotropic unit Gaussian). By choosing a value for ϵ , we can decide how closely $\mathbb{Q}_\phi(\vec{z}|\vec{x})$ must approximate $\mathbb{P}_\theta(\vec{z})$. The problem from above can be transformed into a single optimization function in the form of a Lagrangian (cf. Section 5.2.3, where a similar approach was used for support vector machines):

$$\begin{aligned} F(\theta, \phi, \beta; \vec{x}, \vec{z}) &= \mathbb{E}_{\vec{z} \sim \mathbb{Q}_\phi(\vec{z}|\vec{x})} [\log \mathbb{P}_\theta(\vec{x}|\vec{z})] \\ &\quad - \beta \cdot (D_{KL}(\mathbb{Q}_\phi(\vec{z}|\vec{x}) || \mathbb{P}_\theta(\vec{z})) - \epsilon) \\ &= \mathbb{E}_{\vec{z} \sim \mathbb{Q}_\phi(\vec{z}|\vec{x})} [\log \mathbb{P}_\theta(\vec{x}|\vec{z})] \\ &\quad - \beta \cdot D_{KL}(\mathbb{Q}_\phi(\vec{z}|\vec{x}) || \mathbb{P}_\theta(\vec{z})) + \beta \cdot \epsilon \\ &\geq \mathbb{E}_{\vec{z} \sim \mathbb{Q}_\phi(\vec{z}|\vec{x})} [\log \mathbb{P}_\theta(\vec{x}|\vec{z})] - \beta \cdot D_{KL}(\mathbb{Q}_\phi(\vec{z}|\vec{x}) || \mathbb{P}_\theta(\vec{z})) \\ &= L(\theta, \phi, \beta; \vec{x}, \vec{z}) \end{aligned} \tag{6.4}$$

The β -VAE loss function

We would like to maximize the overall term $F(\theta, \phi, \beta; \vec{x}, \vec{z})$, which can be done by maximizing its lower bound $L(\theta, \phi, \beta; \vec{x}, \vec{z})$, because $\beta \cdot \epsilon$ is a positive constant. Please note, that for $\beta = 1$, $L(\theta, \phi, \beta; \vec{x}, \vec{z})$ is equivalent to Equation 6.3, i.e., the loss function of a regular VAE. One can interpret the Lagrangian multiplier β here as regulating the trade-off between reconstruction accuracy (first term) and the independence of the latent factors (second term). For $\beta > 1$, we put a stronger emphasis on the independence of the latent factors, hence making a stronger constraint on the bottleneck layer of the overall autoencoder.

However, this also reduces the focus on the reconstruction term, leading often to less accurate reconstructions. In the limit (i.e., $\beta \rightarrow \infty$), the reconstruction term does not play any role, and the autoencoder does no longer auto encode the data. Given the value of ϵ (i.e., the degree to which the distribution over the latent codes is allowed to differ from the assumed prior distribution), one can compute the optimal value for β . Unfortunately, ϵ is typically unknown and depends on both the network architecture and the underlying dataset. Therefore, β needs to be manually optimized. The value for β is thus the major hyperparameter of the β -VAE architecture.

Higgins et al. showed through some qualitative experiments on three datasets (CelebA, chairs, and faces), that β -VAE with a suitably tuned value for β (250, 5, and 20, for the three datasets, respectively) leads to better results than both standard VAE and InfoGAN. They note that β -VAE and InfoGAN tend to discover overlapping, but not identical sets of underlying generative factors. They also observed, that increasing β sometimes causes the latent code to lose the representation of some generative factors, especially factors, which lead to only minor changes in pixel space. Higgins et al. also noted, that for a larger hidden code \vec{z} , a larger value of β is usually needed. Moreover, good reconstructions often led to entangled representations, whereas disentangled representations were often accompanied by blurry reconstructions, highlighting again the inherent trade-off in the β -VAE loss function from Equation 6.4.

Overall, Higgins et al. claim that their approach is robust with respect to different architectures, optimization parameters, and datasets. Unlike InfoGAN, it is stable to train, makes few assumptions about the underlying dataset, and relies mainly on tuning the hyperparameter β .

Burgess et al. [84] have analyzed β -VAE from an information-theoretic perspective. They note the similarity of the β -VAE loss function from Equation 6.4 and the *information bottleneck principle* $\max[MI(Z, Y) - \beta MI(X, Z)]$, which targets the maximization of the mutual information between the latent bottleneck Z and the task Y , while discarding all the irrelevant information in X , that is uninformative about Y . When interpreting the β -VAE loss function in this context, $Q_\phi(\vec{z}|\vec{x})$ corresponds to the information bottleneck Z , and the task Y is to reconstruct the original input. Burgess et al. argue, that a highly weighted KL term in the β -VAE loss function encourages high mutual information between the data X and the latent code Z .

From an information-theoretic perspective, the individual entries z_i of the latent code can be interpreted as channels with additive white noise. If the KL divergence becomes zero, then for all of these channels $\mu = 0$ and $\sigma = 1$, i.e., all channels have zero capacity to transmit information about \vec{x} . Burgess et al. note, that the KL divergence can be minimized by reducing the spread of the means or by broadening the variances of $Q_\phi(\vec{z}|\vec{x})$. This in turn causes the posterior distributions to overlap. They further argue, that this encourages the network to learn a mapping, where the similarity of data points is preserved in the latent space, because this is the only way of reaching a small reconstruction error.

Experimental results

General properties of β -VAE

β -VAE and the information bottleneck principle

Latent variables as noisy channels

Based on the idea of the z_i being information channels, Burgess et al. make the observation, that the capacity C of these channels determines how much information is kept. They propose to gradually increase this capacity over the course of training in order to encourage the network to first learn the most important factors and then later to "flesh them out" with more details. Their proposed updated optimization function looks as follows:

$$\mathbb{E}_{\vec{z} \sim \mathbb{Q}_\phi(\vec{z}|\vec{x})} [\log \mathbb{P}_\theta(\vec{x}|\vec{z})] - \gamma \cdot |D_{KL}(\mathbb{Q}_\phi(\vec{z}|\vec{x})||\mathbb{P}_\theta(\vec{z})) - C|$$

Experimental results

As one can see, instead of using the KL divergence directly, Burgess et al. use the absolute difference between the KL divergence and the current capacity C of the latent channel. Their experiments showed, that this loss function leads to disentangled representations with a good reconstruction capability.

*β -VAE variants
based on total
correlation*

Both Kim and Mnih [221] and Chen et al. [100] have independently from each other developed a variant of β -VAE, which puts its emphasis on the information-theoretic quantity of *total correlation*. Total correlation measures the dependency between multiple random variables and is used in this context to quantify to what extent the individual entries of \vec{z} are independent from each other. Chen et al. [100] note, that the KL divergence term in Equation 6.4 can be broken down as follows:

$$\begin{aligned} D_{KL}(\mathbb{Q}_\phi(\vec{z}|\vec{x})||\mathbb{P}_\theta(\vec{z})) &= D_{KL}(\mathbb{Q}_\phi(\vec{z}, \vec{x})||\mathbb{Q}_\phi(\vec{z}) \cdot \mathbb{P}_\theta(\vec{x})) \\ &\quad + D_{KL}(\mathbb{Q}_\phi(\vec{z})||\prod_j \mathbb{Q}_\phi(z_j)) \\ &\quad + \sum_j D_{KL}(\mathbb{Q}_\phi(z_j)||\mathbb{P}_\theta(z_j)) \end{aligned} \quad (6.5)$$

*Splitting up the KL
divergence*

The first term describes the mutual information between the data \vec{x} and the latent variables \vec{z} based on their empirical joint data distribution $\mathbb{Q}_\phi(\vec{z}, \vec{x})$. The second term describes the total correlation of the latent variables. This term becomes zero, if $\mathbb{Q}_\phi(\vec{z}) = \prod_j \mathbb{Q}_\phi(z_j)$, i.e., if the entries of \vec{z} are statistically independent of each other. The third term measures for each of the latent variables in \vec{z} how far it departs from its prior distribution.

*Penalizing only total
correlation*

The β -VAE optimization function puts a large penalty on the overall term $D_{KL}(\mathbb{Q}_\phi(\vec{z}|\vec{x})||\mathbb{P}_\theta(\vec{z}))$, i.e. on all three terms equally. As Kim and Mnih [221] note, β -VAE therefore implicitly also punishes the mutual information between \vec{x} and \vec{z} . The network is thus discouraged from encoding information about \vec{x} in \vec{z} , which leads to poor reconstructions for large values of β . Instead of penalizing all three of these terms equally, both Kim and Mnih [221] and Chen et al. [100] propose to put a larger penalty on the second term (i.e., the total correlation), since it directly measures, whether the different entries z_i of \vec{z} are statistically independent from each other.

Kim and Mnih [221] propose the following objective function for their *FactorVAE* network:

The FactorVAE loss function

$$\begin{aligned} \mathbb{E}_{\vec{z} \sim \mathbb{Q}_\phi(\vec{z}|\vec{x})} [\log \mathbb{P}_\theta(\vec{x}|\vec{z})] - D_{KL}(\mathbb{Q}_\phi(\vec{z}|\vec{x}) || \mathbb{P}_\theta(\vec{z})) \\ - \gamma D_{KL}(\mathbb{Q}_\phi(\vec{z}) || \prod_j \mathbb{Q}_\phi(z_j)) \end{aligned}$$

The first two terms correspond to the loss function of the original VAE framework from Equation 6.3, which is supplemented by an additional penalty on the total correlation. The hyperparameter γ controls the importance of the independence assumption.

Chen et al. [100] propose the following objective function for their β -TCVAE network:

The β -TCVAE loss function

$$\begin{aligned} \mathbb{E}_{\vec{z} \sim \mathbb{Q}_\phi(\vec{z}|\vec{x})} [\log \mathbb{P}_\theta(\vec{x}|\vec{z})] - \alpha \cdot D_{KL}(\mathbb{Q}_\phi(\vec{z}, \vec{x}) || \mathbb{Q}_\phi(\vec{z}) \cdot \mathbb{P}_\theta(\vec{x})) \\ - \beta \cdot D_{KL}(\mathbb{Q}_\phi(\vec{z}) || \prod_j \mathbb{Q}_\phi(z_j)) \\ - \gamma \cdot \sum_j D_{KL}(\mathbb{Q}_\phi(z_j) || \mathbb{P}_\phi(z_j)) \end{aligned}$$

Based on Equation 6.5, one can easily see that for $\alpha = \gamma = 1$ (which are the settings used by Chen et al. in their experiments), this is equivalent to the FactorVAE loss function.

In practice, the computation of the term $D_{KL}(\mathbb{Q}_\phi(\vec{z}) || \prod_j \mathbb{Q}_\phi(z_j))$ is computationally not tractable, hence it must be approximated. This is where FactorVAE differs from β -TCVAE: In FactorVAE [221], an additional discriminator network D is trained to distinguish inputs from $\mathbb{Q}_\phi(\vec{z})$ and $\prod_j \mathbb{Q}_\phi(z_j)$, leading to the following approximation:

Approximation of total correlation in FactorVAE

$$D_{KL}(\mathbb{Q}_\phi(\vec{z}) || \prod_j \mathbb{Q}_\phi(z_j)) \approx \mathbb{E}_{\vec{z} \sim \mathbb{Q}_\phi(\vec{z})} \left[\frac{D(\vec{z})}{1 - D(\vec{z})} \right]$$

In β -TCVAE [100], however, $D_{KL}(\mathbb{Q}_\phi(\vec{z}) || \prod_j \mathbb{Q}_\phi(z_j))$ is estimated by sampling a minibatch of data points \vec{x} , and by approximating $\mathbb{Q}_\phi(\vec{z})$ with the weighted average of $\mathbb{Q}_\phi(\vec{z}|\vec{x})$ for this minibatch.

Approximation of total correlation in β -TCVAE

Experiments showed, that both FactorVAE and β -TCVAE yield better performance and especially a better trade-off between disentangled representations and faithful reconstructions than standard VAE, β -VAE, and InfoGAN. In the experiments conducted by Chen et al. [100], β -TCVAE also yielded better performance than FactorVAE.

Experimental results

Kumar et al. [237] propose yet another variant of variational autoencoders called the *disentangled inferred prior VAE (DIP-VAE)*. They consider $\mathbb{Q}_\phi(\vec{z}) = \int \mathbb{Q}_\phi(\vec{z}|\vec{x}) \mathbb{P}_\theta(\vec{x}) d\vec{x}$, i.e., the inferred prior distribution over \vec{z} based on the recognition model. Ideally, $\mathbb{Q}_\phi(\vec{z})$ should be factorizable, which can be achieved by minimizing its KL divergence to the prior distribution $\mathbb{Q}_\theta(\vec{z})$. This KL divergence can be written as follows:

DIP-VAE: considering the inferred prior

$$\begin{aligned} D_{KL}(\mathbb{Q}_\phi(\vec{z}) || \mathbb{P}_\theta(\vec{z})) &= D_{KL}(\mathbb{E}_{\vec{x} \sim \mathbb{P}_{data}(\vec{x})} \mathbb{Q}_\phi(\vec{z}|\vec{x}) || \mathbb{E}_{\vec{x} \sim \mathbb{P}_{data}(\vec{x})} \mathbb{P}_\theta(\vec{z}|\vec{x})) \\ &\leq \mathbb{E}_{\vec{x} \sim \mathbb{P}_{data}(\vec{x})} D_{KL}(\mathbb{Q}_\phi(\vec{z}|\vec{x}) || \mathbb{P}_\theta(\vec{z}|\vec{x})) \end{aligned}$$

*Why true prior and
inferred prior may
differ*

In general, the prior $\mathbb{P}_\theta(\vec{z})$ and the posterior $\mathbb{Q}_\phi(\vec{z})$ are different, but Kumar et al. argue, that standard VAE encourages inferring factors, which are close to being disentangled. It has been empirically observed, that standard VAEs are able to discover disentangled factors on simple datasets, but fail to do so on more complex ones (cf. our discussion of β -VAE [196]). Kumar et al. hypothesize, that there might be two reasons for that: On the one hand, the real data distribution $\mathbb{P}_{data}(\vec{x})$ and the distribution $\mathbb{P}_\theta(\vec{x})$ learned by the model might differ to a large degree. This would then also cause differences between $\mathbb{P}_\theta(\vec{z})$ and $\mathbb{Q}_\phi(\vec{z})$, because $\mathbb{P}_\theta(\vec{x}) = \int \mathbb{P}_\theta(\vec{x}|\vec{z})\mathbb{P}_\theta(\vec{z})d\vec{z}$. This problem can potentially be solved by making the VAE network more powerful (e.g., by adding more layers). On the other hand, the optimum for the VAE objective might not necessarily be an optimum for minimizing the difference between $\mathbb{P}_\theta(\vec{z})$ and $\mathbb{Q}_\phi(\vec{z})$. In order to address this second issue, Kumar et al. therefore propose the following loss function:

$$\mathbb{E}_{\vec{z} \sim \mathbb{Q}_\phi(\vec{z}|\vec{x})}[\log \mathbb{P}_\theta(\vec{x}|\vec{z})] - D_{KL}(\mathbb{Q}_\phi(\vec{z}|\vec{x})||\mathbb{P}_\theta(\vec{z})) - \lambda \cdot D(\mathbb{Q}_\phi(\vec{z})||\mathbb{P}_\theta(\vec{z}))$$

*The DIP-VAE loss
function*

As one can see, the first two terms correspond to the standard VAE objective from Equation 6.3, while the third term quantifies the difference between $\mathbb{Q}_\phi(\vec{z})$ and $\mathbb{P}_\theta(\vec{z})$. Kumar et al. note, that using the KL divergence for the third term makes it computationally intractable, because there exists no closed form solution. Instead, they propose to match the covariance of the two distributions by computing D as the L2 difference of their covariance matrices. In their evaluation, Kumar et al. showed improved performance, when comparing their DIP-VAE to standard VAE and to β -VAE.

Outlook

As our discussion has highlighted, there exists a plethora of VAE variants for learning disentangled representations. Most of these proposals have been accompanied by a novel evaluation metric for disentanglement (cf. Section 6.1.2). This tight connection between variational autoencoders and disentanglement as well as the simplicity of the underlying graphical model and the absence of training instabilities (especially when compared to GANs) makes them an interesting candidate for learning conceptual domains from perceptual data.

6.4 REPRESENTATION LEARNING FOR CONCEPTUAL SPACES

After having covered the representation learning in sufficient detail, we will now relate it to the process of grounding the dimensions of a conceptual space. We first establish a general connection between representation learning and conceptual spaces in Section 6.4.1, before considering rectangles as a simple example domain in Section 6.4.2.

6.4.1 General Considerations

The cognitive framework of conceptual spaces [179] makes several claims about the properties of the geometric representation at the conceptual level. We will now briefly summarize some of its core assumptions (which have been treated more extensively in Sections 1.2 and 2.1) and show to which extent they are reflected by the representation learning approaches discussed so far.

Properties of conceptual similarity spaces

First of all, Gärdenfors assumes, that the conceptual space is spanned by *interpretable dimensions*, which can be linked to cognitively meaningful aspects. As discussed in Sections 6.3.3 and 6.3.4, the representation learning community has recently made various proposals for network architectures, which try to extract a disentangled representation of the underlying generative factors. Based on experimental results from the literature, the individual latent variables also tend to be interpretable. Section 6.1.2 has provided an overview of different evaluation metrics for quantifying the degree of disentanglement, which may thus be interpreted as a proxy for measuring interpretability. Moreover, the vector arithmetics discussed both for word embeddings (cf. Section 6.3.2) and the GAN framework (cf. Section 6.3.3) illustrate, that the learned latent spaces contain a meaningful structure. Overall, it thus seems, that representation learning is able to accommodate this crucial requirement of the conceptual spaces framework.

Interpretable dimensions

In a conceptual space, we furthermore use a *distance-based notion of similarity*: Two stimuli are considered to be similar, if their distance with respect to all individual dimensions (and hence also in the overall similarity space) is small. Gärdenfors proposes to use the Euclidean distance within a given domain, and the Manhattan distance for combining different domains. The smoothness assumption in representation learning (cf. Sections 5.1.4 and 6.1.1) states, that points with small distance in the input space should also have a small distance in the latent space. This means, that a distance-based notion of similarity in the latent space is semantically meaningful, because it reflects perceptual similarity. Word embeddings (cf. Section 6.3.2) are the only representation learning approach discussed in this chapter, where the pairwise similarity of two inputs is explicitly considered. However, as discussed in Section 6.3.2, one typically uses the Cosine similarity (which is based on the angle between two vectors) rather than an exponential decay of Euclidean or Manhattan distances, as proposed by Gärdenfors. On the other hand, the related area of metric learning (cf. Section 5.3.3) explicitly targets a meaningful distance metric in the latent space learned by the system, often taking into account explicit similarity constraints during the training process. Overall, there seems to be a reasonably good match between representation learning and conceptual spaces with respect to a distance-based notion of similarity.

Similarity based on distance

Moreover, conceptual spaces provide a *geometric interpretation of conceptual betweenness* – betweenness can be defined based on the distance in the space (namely, by requiring $d(x, y) + d(y, z) = d(x, z)$), and any point y lying between x and z in the conceptual space can be

Conceptual betweenness

considered an intermediate case. As discussed in Section 6.3.2, Kim and de Marneffe [222] found, that one can obtain meaningful adjectival scales from word embeddings by searching for points between the vectors representing the two extremes of such a scale. Also in the context of the GAN framework (cf. Section 6.3.3), Radford et al. [326] found, that linear interpolations in the latent space correspond to a meaningful "morph" between generated images in the input space. These meaningful interpolations show, that also latent spaces from representation learning give rise to a meaningful betweenness relation.

Convex conceptual regions

The geometric notion of semantic betweenness also gives rise to Gärdenfors' description of *concepts as convex regions* – if both x and z belong to a convex concept C , then also any y between x and z must belong to C . Typical representation learning approaches do not make any specific assumptions about the shape of conceptual regions in the learned representation. However, similar to metric learning (cf. Section 5.3.3), they usually aim to extract a representation, on which a simple classifier can obtain good levels of performance (cf. Section 6.1). A prototype-based classification rule as proposed by Gärdenfors [179] is an example for such a simple classifier, and has, for instance, been used in ProtoNet [381] in the context of metric learning (cf. Section 5.3.3). Also the studies by Sommerauer and Fokkens [382] and Gupta et al. [176] discussed in Section 6.3.2 used simple classifiers such as a logistic regression on top of pretrained word embeddings in order to make classification and regression predictions, respectively. It thus seems, that the latent spaces from representation learning could be easily adapted to give rise to well-formed conceptual regions.

Domain structure

Furthermore, the conceptual space is *structured into semantic domains* such as COLOR, SHAPE, or SIZE. While such an explicit structure is usually not enforced in representation learning approaches, we have seen various attempts to extract or enforce such structures in the context of word embeddings in Section 6.3.2. In principle, one should be able to generalize these approaches also to representation learning in the visual domain. Enforcing such a domain structure may thus be a valuable contribution from conceptual spaces to representation learning.

Perceptual grounding

Finally, for the scope of this dissertation, we assume that the dimensions of a conceptual space are *grounded in perception*, since we restrict our considerations to physical objects. Word embeddings are based on textual data, and thus only indirectly related to perception [185, 382] (cf. Section 6.3.2). However, most other representation learning approaches such as GANs (cf. Section 6.3.3) and VAEs (cf. Section 6.3.4) have been applied to image data, which corresponds to visual perception. The general architectures can in principle also be applied to other types of perceptual input (such as auditory perception), but may need to be adapted in order to reflect the temporal nature of such domains. Overall, representation learning thus provides a clear perceptual grounding of the learned representations.

How to learn a conceptual space

Now that we have established a suitable mapping between the desiderata for a conceptual space and the properties of representation

learning, let us discuss the overall setup for obtaining the dimensions of a conceptual space with representation learning approaches.

We propose to start by picking the domain of interest (e.g., *shape*) and by obtaining a suitable dataset for this domain. In order to ensure, that the neural network does not learn to represent information about other domains (such as *COLOR*), one should preprocess this dataset in such a way, that most to all information about such other domains is removed. For instance, when interested in learning a similarity space for the *SHAPE* domain, one should ensure, that the images are converted to greyscale, that they contain only individual objects on a neutral background, and that these objects are all of approximately the same size. Also variations with respect to the texture should ideally be removed, e.g., by using silhouettes. By preprocessing the dataset in such a way, the variance of the inputs with respect to irrelevant domains is removed, hence preventing the network to discover the associated factors of variation. Moreover, one could potentially introduce an additional term into the loss function, which measures the correlation between the learned latent representation and dimensions from other (already manually defined) domains. This would cause a stronger error signal, if the network starts to re-discover already known dimensions from other domains, and should therefore drive the network away from learning redundant representations. This may be interpreted as a safety measure, which can potentially handle factors of variation, that the experimenter forgot to remove from the dataset or that could not be removed by simple preprocessing steps.

Data preprocessing

After an appropriate dataset has been obtained, one can use any of the representation learning algorithms discussed in this chapter to extract a latent representation from the data. It may be more convenient to use unsupervised approaches as the ones in Section 6.3, since unlabeled data is more readily available. However, if also labeled data points are available, one may use those labels to enforce additional constraints. For instance, one could follow a prototype-based approach like ProtoNet [381] (cf. Section 5.3.3) to ensure, that concepts in the learned space form convex regions. Training such a neural network involves the fine-tuning of many hyperparameters, including the size of the latent code. Just as for dimensionality reduction (cf. Section 5.3), one should aim for the smallest representation, which still accurately describes the data. This can be quantified based on the final loss values, performance in downstream tasks, and various disentanglement metrics (cf. Section 6.1.2). Additionally, visual inspection of the results should be used to then validate their quality.

Training the network

Finally, when the network has been successfully trained, one can interpret each entry of the latent code as one dimension of the extracted similarity space. The encoder network can then be used to translate from raw perceptual inputs to points in this similarity space, providing therefore an automated way of bridging the subsymbolic and the conceptual layer. If a model with a generative component was chosen (such as the decoder of an autoencoder), this component can be used to translate in the opposite direction, enabling an easy visualization of conceptual

Using the learned representation

structures for debugging purposes. Moreover, the translation from the conceptual to the subsymbolic layer can potentially be interpreted as an act of imagination, which may be useful for an agent, which wants to run simulations of potential future scenarios.

Interpretable directions for CelebA

We will now briefly summarize two recent approaches, which have used representation learning for obtaining similarity spaces, and which then interpreted the learned representation as conceptual spaces for downstream tasks. White and Loh [432] used the latent space of a VAE for generating animations through sketching. They trained a VAE on the CelebA dataset, which consists of portraits of celebrities. White and Loh tried to find meaningful directions in this space by employing different techniques. All of these directions were based on the labels contained in the CelebA dataset (such as *SMILING*). As a first technique, as detailed in [431], they computed the difference vector between the average of all positive examples and the average of all negative examples for a given label. For instance, to find the *SMILING* direction, the average of the latent vectors representing non-smiling images was subtracted from the average of the vectors representing smiling images. In order to deal with correlated labels (for instance, in CelebA, women are more likely to smile than men), White and Loh rebalanced the dataset in such a way, that these correlations were eliminated. As a second approach, White and Loh also trained a linear support vector machine to separate the two classes. They then used the normal vector of its decision hyperplane as a direction for the given label. This second approach had previously been proposed by Derrac and Schockaert [123] in the context of conceptual spaces, and will be discussed in more detail in Chapter 8.

Using the conceptual space for creating animations

Based on the extracted directions, White and Loh provide a two-dimensional representation of facial expressions using e.g., *smiling* as one of the dimensions. By creating a path in this two-dimensional space, one can generate an animation of a change in facial expression: For each frame, a latent code on this path is used to generate an image. Since interpolations in the latent space lead to meaningful interpolations in the image space, putting the images created along the path right after another yields a smooth animation.

Relevance

The work by White and Loh is relevant in our context, because they explicitly equate the latent space of a generative model with a conceptual space as proposed by Gärdenfors. They therefore share our intuitive understanding of using the hidden representation of a neural network as a basis of a conceptual similarity space. However, White and Loh do not explicitly consider semantic similarity or conceptual regions, nor do they distinguish different conceptual domains. Their work can thus be seen as an important first step, which needs to be followed up by further studies.

Modified β -VAE for an artificial domain

Higgins et al. [197] used a two-stage approach to learn both a disentangled representation of visual input and a description of concepts based on this representation. In a first step, they applied β -VAE to a dataset of artificial images, which were generated based on four visual

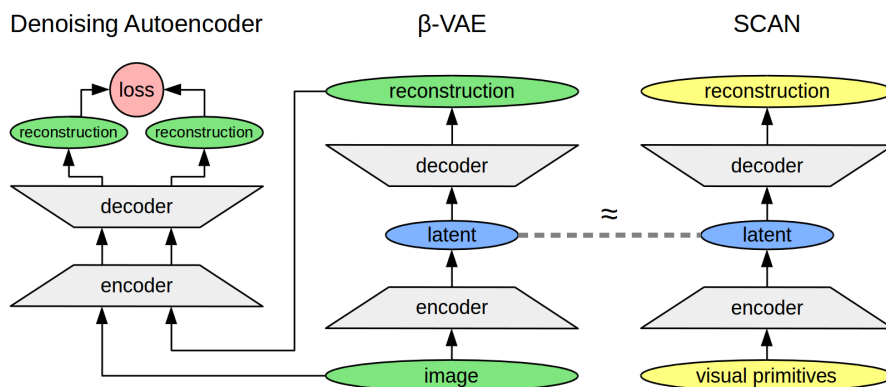


Figure 6.15: Combination of two variational autoencoders and a denoising autoencoder as proposed by Higgins et al. [197].

primitives, namely OBJECT IDENTITY, OBJECT COLOR, FLOOR COLOR, and WALL COLOR. Higgins et al. used a modified version of the β -VAE, where they replaced the pixel-level reconstruction term with the L2 loss of a denoising autoencoder when comparing the original input \vec{x} to the reconstruction \vec{x}' sampled from the β -VAEs decoder.

In order to learn concepts, Higgins et al. labeled each example \vec{x} with a vector \vec{y} indicating the values for each of the four visual primitives. Their proposed *symbol-concept association network* (SCAN) is another modified variational autoencoder, which uses the same size of hidden units as the β -VAE trained unsupervisedly on the input images. Again, the hidden representation is constrained to a multivariate Gaussian distribution with a diagonal covariance matrix. Through an additional loss term, Higgins et al. ensure that the latent distribution learned by SCAN matches the one of the pretrained β -VAE for all data points. The overall setup is illustrated in Figure 6.15.

Their overall system can be used to translate from images to a symbolic description by using first the encoder of the β -VAE and then the decoder of SCAN. A translation from a symbolic description to a visualization can be obtained by using the encoder of SCAN and then the decoder of the β -VAE. Higgins et al. also note the important role of compositionality for cognition and propose to incorporate three operations for concept creation into their architecture, namely set union, set intersection, and set difference. For instance, in order to obtain disjunctive concepts such as APPLE \vee PEAR, they first use SCAN's encoder to obtain a latent representation of both concepts based on their high-level description vector \vec{y} . They then use a convolution operator to combine these two latent representation into a single latent representation for the composite concept, which can then be again decoded into a symbolic form (using SCAN's decoder) or into visualizations (using the decoder of the β -VAE).

The work by Higgins et al. is quite elegant since it uses the same underlying architecture (namely, variational autoencoders) for most of its components. While the unsupervised β -VAE learns to disentangle different factors of variation from the data, the SCAN trained on top of this learns an appropriate combination of these factors for reflect-

*Symbol-concept
association network*

*Translation
directions and
compositionality*

Relevance

ing concepts. Moreover, their system allows for a translation between perceptual input, a latent similarity space, and high-level concept definitions, which can be related to the subsymbolic, conceptual, and symbolic layer, respectively (cf. Sections 1.2.4 and 1.2.3). Unlike conceptual spaces, their system does, however, not explicitly take into account different subspaces for individual cognitive domains. Also the mapping between abstract concept definitions and the latent representation is relatively opaque, which makes it difficult to interpret in contrast to the simple prototype-based approach used in conceptual spaces. Finally, we should remark, that the diagonal covariance matrix used for defining concepts implicitly asserts that different properties of a concept are not correlated to each other. This is only used as a soft constraint, but discourages the system to discover systems of correlations, which we deemed to be very important in Part I of this dissertation.

*ANN experiments
are complex and
time-consuming*

Let us now finally address some potential shortcomings of using ANNs for extracting the dimensions of a conceptual space. First of all, ANNs in general require very large amounts of data and computing time in order to achieve good performance. Since they typically contain millions of tunable parameters, training can be quite complicated and might include significant amounts of hyperparameter tuning. Obtaining a similarity space with representation learning techniques can thus become easily a quite time-consuming task.

*ANNs can be
in stable*

Moreover, the existence of adversarial examples [401], i.e., pairs of inputs with a minimal difference, which lead to a large difference in the output, casts some doubt on the stability of neural networks. Even the same network trained on the same dataset might learn quite different internal representations, based on the random initialization of its weights. Also this lack of feature stability seems to be a disadvantage of ANNs in the conceptual spaces setting.

*Representation
learning does not
guarantee
interpretability*

While there have been promising first results with respect to the interpretability of the extracted dimensions, one cannot simply assume that this is always the case. For instance, we have tried to reproduce the MNIST results reported in the InfoGAN paper [101], using the authors' original source code, but were unable to obtain a disentanglement anywhere as "clean" as presented in the paper. If the interpretability of the resulting similarity space is crucial, one should therefore be aware of the fact that ensuring disentanglement may involve considerable amounts of additional work in practice.

*Limited psychological
plausibility*

Finally, when viewing conceptual spaces as a psychologically motivated and cognitively plausible framework of human cognition, ANNs trained to optimize a carefully crafted mathematical loss function in a batch-like setting on thousands or millions of inputs may not be very satisfactory – the psychological plausibility of the extracted dimensions is questionable at best. Therefore, additional measures need to be taken in order to validate, whether the similarity spaces by neural networks are a good reflection of human representations. We will return to this issue in more detail in Chapters 8 and 10.

6.4.2 The Rectangle Domain

In order to investigate the applicability of representation learning techniques for obtaining the dimensions of a conceptual space, we propose to start with a relatively simplistic setup. In the domain under consideration, one should be able to describe the class of shapes under investigation by a well-defined, small set of parameters. Since we are mainly interested in the domain of shapes, we therefore propose to select a very simple type of shapes for initial studies.⁴

Simple shapes as starting point

Newson et al. [300] have investigated the behavior of convolutional autoencoders, when being trained on a dataset of disks. They only considered black and white images of centered disks, varying only with respect to their RADIUS. Their autoencoder used a single neuron in the bottleneck layer and was trained to minimize the MSE between the input and the reconstruction. Newson et al. found, that the bottleneck neuron reflected the overall AREA of the disk and not its RADIUS. Since AREA, RADIUS, and DIAMETER of a disk are monotonically related, it is irrelevant which of these encodings is used.

An autoencoder for disks

Based on this simple setup, Newson et al. introduced various changes and observed their effects. Removing the bias terms from the neurons in the encoder did not have any major effects, but removing them from the neurons in the decoder led to blurry reconstructions. When presenting the autoencoder with disks, which were larger than the ones seen during training, the autoencoder failed to extrapolate correctly: Its output was limited to the largest disk that had been seen during training. Also if the dataset contained a gap (i.e., if the training set consisted only of very small and very large disks), the network failed to interpolate correctly.

Ablation and generalization

Newson et al. investigated different regularization techniques in order to allow the autoencoder to correctly interpolate even in the case of gaps in the dataset. Forcing the autoencoder to maintain the L2 distance between two images also in the latent space did not improve the autoencoder's generalization capability. Newson et al. hypothesize, that this is caused by the fact, that there are no close neighbors on the edge of an unobserved region. Regularizing the weights of both the encoder and the decoder improved the interpolation ability of the network, but led to instabilities of the hidden code's amplitude. In their experiments, the most helpful option was to regularize only the weights of the encoder, which led to improved interpolation capabilities without any undesired side effects.

Effects of regularization

Since disks have already been investigated and since they can be specified by using a single generative factor, we propose to focus on a slightly more complex domain, namely, the domain of rectangles, as a test bed for representation learning. More specifically, if we only consider centered, unrotated rectangles without texture or color, then

Motivation for the rectangle domain

⁴ Thanks to Brendan Ritchie for making the suggestion of using a parametrically describable class of shapes as a starting point.

this dataset can be described with two variables. It still consists of visual input (as expected by most representation learning approaches), and since it contains two generative factors, it can also be used to test disentanglement. The domain of disks can for instance not be used to investigate the disentanglement capabilities of different architectures, since only a single latent variable needs to be recovered. We therefore think that the domain of rectangles is a suitable minimal problem domain for continuing the investigations by Newson et al. with slightly more complex stimuli.

Combinations of generative factors

Almost all of the disentanglement metrics presented in Section 6.1.2 assume, that the true generative factors are explicitly given. Since all of these metrics explicitly measure to which extent these true generative factors align with the variables from the extracted representation, they tacitly assume, that these generating factors are uniquely determined. However, already in the simple domain of rectangles, this tacit assumption is not fulfilled: One can describe a rectangle either by specifying its `WIDTH` and `HEIGHT`, or by specifying its `AREA` and its `SHAPE` (i.e., its aspect ratio). In fact, any two of these four dimensions can be used to fully and uniquely describe a given rectangle. Our proposed domain can therefore also be used to investigate, whether any of those four potential factors are more frequently recovered by the representation learning algorithms than the others.

Human perception of rectangles

There have been various studies about the human perception of rectangles. A concise summary is given by Borg and Groenen [71, Section 17.4]. It seems that there is somewhat contradicting evidence about whether humans tend to use a `WIDTH` × `HEIGHT` space to represent rectangles, or whether they use a `AREA` × `SHAPE` space (where `AREA` = `WIDTH` · `HEIGHT` and `SHAPE` = `WIDTH`/`HEIGHT`). While it is mathematically also possible to use, for instance, a space spanned by `WIDTH` and `AREA`, such combinations seem not to be explored in psychological research. As multiple authors point out, if a logarithmic scale is used, then `AREA` and `SHAPE` can be obtained through a rotation of the `WIDTH` × `HEIGHT` coordinate system by 45 degrees [72, 227]:

$$\begin{aligned}\log(\text{AREA}) &= \log(\text{WIDTH} \cdot \text{HEIGHT}) = \log(\text{WIDTH}) + \log(\text{HEIGHT}) \\ \log(\text{SHAPE}) &= \log\left(\frac{\text{WIDTH}}{\text{HEIGHT}}\right) = \log(\text{WIDTH}) - \log(\text{HEIGHT})\end{aligned}$$

Evidence for AREA and SHAPE

Krantz and Tversky [227] collected pairwise dissimilarity ratings on a dataset of rectangles carefully designed to be equally spaced both in the `log(WIDTH)` × `log(HEIGHT)` and in the `log(AREA)` × `log(SHAPE)` space. On the one hand, Krantz and Tversky used the two different coordinate systems to make predictions about the relations between pairs of entries from the matrix of aggregated dissimilarity ratings. On the other hand, they used multidimensional scaling (which will be introduced in Chapter 8) to extract a two-dimensional similarity space from this dissimilarity matrix. Their results indicate that neither of the two coordinate systems seems to be satisfactory, but that the `AREA` × `SHAPE` approach is somewhat superior to `WIDTH` × `HEIGHT`.

Borg and Leutner [72] also collected pairwise dissimilarity ratings for a slightly different dataset of rectangles. They used multidimensional scaling (MDS) for analyzing possible two-dimensional similarity spaces, which could explain the dissimilarity ratings. Borg and Leutner found, that the resulting configurations were better explained by a logarithmically transformed $\text{WIDTH} \times \text{HEIGHT}$ space.

Evidence for WIDTH and HEIGHT

Ashby and Gott [16] have not explicitly considered rectangles, but stimuli consisting of a horizontal and a vertical line that met at the top left corner. They studied different categorization models using a $\text{WIDTH} \times \text{HEIGHT}$ space and compared them to human categorization decisions. As Gärdenfors [179, Section 4.10] points out, their results contain some observations, which cannot be readily explained based on the chosen similarity space. According to Gärdenfors, these problems can be largely mitigated if one assumes a similarity space spanned by AREA and SHAPE .

Study on perpendicular lines

Finally, Borg and Groenen [71, Section 17.4] note in their general overview, that the analysis results can change, if the data of these experiments is processed with different MDS algorithms and different random initializations. Moreover, they emphasize, that different participants may be using different strategies for making their similarity judgments – some may use WIDTH and HEIGHT , while others may focus on AREA and SHAPE . Despite being such a simple visual domain, the perception of rectangles is thus still not fully understood.

General considerations

The rectangle domain therefore seems to be a good candidate for further investigations: It is relatively simple, and the stimuli can be explicitly generated. One needs two dimensions to describe the domain, which is enough for testing disentanglement, but there are at least four possible candidate dimensions, which makes evaluation more complex. Also from a psychological point of view, it is still largely unclear which combination of dimensions is used by humans, although the debate seems to concentrate on the combinations $\text{WIDTH} \times \text{HEIGHT}$ and $\text{AREA} \times \text{SHAPE}$.

Rectangle domain as ideal testbed

We propose to apply both InfoGAN [101] and β -VAE [196] to this simple domain and to investigate different aspects: Which dimensions do they tend to extract? Are they consistent in their behavior or does the type of resulting space depend heavily on random initializations and hyperparameter settings? How do the results change, if the probability distribution generating the rectangles is changed?

Possible experiments

Moreover, due to the unclear nature of the ground truth, we propose to also compare the different disentanglement metrics from Section 6.1.2. One can use the collected data to investigate, how well these metric are able to cope with a domain, which has multiple possible underlying factors of variation, and whether any modifications need to be made in such a scenario.

Evaluation metrics

6.5 SUMMARY

Lessons learned

In this chapter, we have introduced artificial neural networks as a powerful machine learning algorithm that lends itself towards learning abstract representations of its perceptual inputs. We have covered different representation learning approaches as well as evaluation metrics for quantifying the degree of disentanglement achieved in the resulting representations. Finally, we have argued, that representation learning may be a good fit for discovering the dimensions of a conceptual space in a bottom-up fashion. We have proposed to use the domain of rectangles as a first test bed for such an approach.

Outlook

After having covered general topics in machine learning in Chapter 5 and neural approaches with a focus on representation learning in the current chapter, we will now put our focus on cognitively plausible ways of learning conceptual regions with machine learning. In this context, we will consider knowledge-based constraints, concept formation, and simulated linguistic interaction in more detail. Chapter 7 will thus conclude our background on machine learning, will then be supplemented in Chapter 8 by a discussion of multidimensional scaling as an optimization technique used for deriving similarity spaces based on psychological data.

7

LEARNING CONCEPTS IN A COGNITIVE WAY

7.1	Concept Learning in Conceptual Spaces	359
7.1.1	Learning Concepts from Few Examples	359
7.1.2	The Bayesian Perspective	361
7.1.3	The Machine Learning Perspective	367
7.2	Learning Concepts under Knowledge-Based Constraints	370
7.2.1	General Approaches	372
7.2.2	Logic Tensor Networks	377
7.2.3	Towards Conceptual Logic Tensor Networks	384
7.3	Learning Concepts from Unlabeled Data	393
7.3.1	Classical Clustering Techniques	393
7.3.2	Hierarchical Concept Formation with COBWEB	400
7.3.3	Adaptive Resonance Theory	406
7.3.4	Incremental Category Learning with SUSTAIN	410
7.4	Learning Concepts through Communication	414
7.4.1	General Considerations	415
7.4.2	Vertical Transmission	420
7.4.3	Horizontal Transmission	423
7.5	Summary	431

In Chapters 5 and 6, we have introduced general machine learning techniques and artificial neural networks, respectively. While standard machine learning techniques have been applied quite successfully in practice, they differ considerably from the way in which humans learn: Machine learning algorithms typically require large amounts of labeled examples, and consume them passively in a batch-like manner. Humans, on the other hand, can successfully acquire new concepts from a relatively small amount of examples, continue to refine what has been learned, and are actively involved in the learning process, both through explicit allocation of their attention, and through embodied interactions with their environment [447].

For instance, Carey and Bartlett [87] observed, that already three-year-olds are in principle able to partially learn a new, previously unknown color term based only on a single or very few encounters. Zaadnoordijk et al. [447] have recently compared insights from developmental psychology and neuroscience to current approaches in machine learning. They note, that "[t]here is more structure to constrain and guide infants' learning processes", and that "[i]nfants' learning opportunities are more flexible [...] and richer" when compared to most of the current approaches in machine learning. They furthermore argue, that by increasing the human-likeness of machine learning algorithms in these respects, one may be able to learn more robust representations from fewer examples and to obtain a better generalization across

Machine learning is not cognitive

Aspects of learning in humans

tasks. Since we attempt to use machine learning in the context of the cognitive framework of conceptual spaces, this chapter will give an overview of different approaches, which try to incorporate different cognitive aspects into machine learning. For each of these approaches, we will furthermore try to provide a link to the conceptual spaces approach.

*Concept learning in
the conceptual spaces
literature*

We begin our discussions in Section 7.1 with an overview of concept learning mechanisms discussed in the context of conceptual spaces. This includes Gärdenfors' original Voronoi-based approach, various considerations from a Bayesian perspective, and machine learning experiments from the literature.

*Constraints from
background
knowledge*

In Section 7.2, we then consider the usage of background knowledge as a way to reduce the dependency on large amounts of labeled examples. This background knowledge may come in an explicit and structured format or in the form of useful starting conditions and biases. It can in general be linked to top-down information from the symbolic layer, which influences the way, in which conceptual regions are learned. Moreover, it can be viewed as an attempt to implement the knowledge view on concepts [298, Chapter 3] (cf. Section 1.1.1) in machine learning systems. We put our focus on the framework of logic tensor networks [21, 358], which combines bottom-up example-driven learning with top-down rule-based constraints. Moreover, we sketch, how logic tensor networks can be combined with the conceptual spaces framework.

*Unsupervised and
incremental learning*

Section 7.3 then focuses on unsupervised learning approaches, which do not require explicit labels for the observations. Concepts are in this case discovered based on statistical regularities in the environment: One tries to identify clusters of observations, which are very similar to each other, while being dissimilar from other examples. An especially interesting approach in this context are incremental algorithms, which exhibit a continual learning behavior also observed in humans [447]. After introducing clustering algorithms in general, we will consider COBWEB [159], ART [90], and SUSTAIN [270] as three exemplary incremental approaches in more detail.

Language games

We then consider concept learning through communication in Section 7.4: It is well known that the application of a concept in cognitive tasks has an important influence on its representation [298, Chapter 5]. Also from a developmental perspective, learning typically incorporates multiple modalities and active involvement with other agents and the environment [447]. We will focus our discussion on so-called *language games* [389], where multiple agents try to align their vocabulary and conceptualization of a given environment over a series of interactions.

Finally, in Section 7.5, we summarize the main insights from this chapter and sketch, how the discussed aspects can be realized in the context of the conceptual spaces framework.

Our discussion of logic tensor networks in Sections 7.2.2 and 7.2.3 has previously been published in [38].

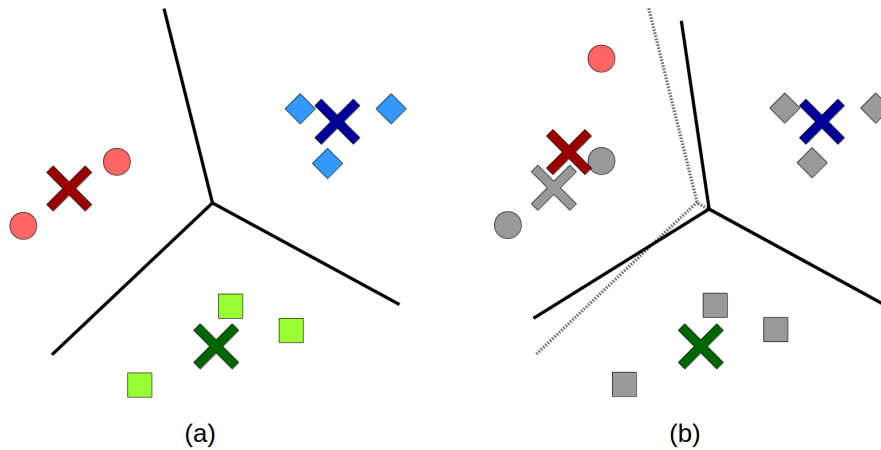


Figure 7.1: (a) Learning prototypes in a conceptual space based on few examples. (b) Incremental update of one prototype and the resulting change in the Voronoi tessellation.

7.1 CONCEPT LEARNING IN CONCEPTUAL SPACES

In this section, we will consider the different existing approaches to concept learning in conceptual spaces. These approaches typically harness the structure of the conceptual space itself with its different semantic domains, its interpretable dimensions, and its notion of semantic similarity as distance in the space. We will start in Section 7.1.1 by considering some relatively straightforward mechanisms for concept learning based on a small number of examples. Afterwards, we will discuss Bayesian approaches to induction in the context of conceptual spaces in Section 7.1.2. Finally, in Section 7.1.3 we introduce the application of machine learning techniques in conceptual spaces.

7.1.1 Learning Concepts from Few Examples

As already discussed in Section 1.2.2, Gärdenfors [179, Chapter 3] establishes a link between the theory of conceptual spaces and the prototype theory of concepts [336]: If concepts are represented as convex regions, the centroid of this region can be interpreted as a prototype. Conversely, given a set of prototypes, one can create a Voronoi tessellation of the conceptual space by assigning each point to its closest prototype, which results in convex regions.

Since Gärdenfors thus essentially represents concepts by their prototypical point, learning a concept corresponds to learning its prototype. Gärdenfors [179, Section 4.5] also proposes a concrete procedure for constructing such a prototype from a limited number of examples: Since each example $x^{(j)}$ of the target concept C is represented as a point in the

*Conceptual spaces
and prototypes*

Learning prototypes

conceptual space, the prototype p can be obtained by simply averaging the coordinates of the example points on all dimensions:

$$p_d = \frac{1}{N} \sum_{j=1}^N x_d^{(j)}$$

Voronoi tessellation

The set of all prototypes gives then rise to a Voronoi tessellation of the space (see Figure 7.1a). This approach is quite straightforward, and Gärdenfors notes, that it is also able to generalize quite well:

The additional information required for the generalization is *extracted* from the geometrical structure of the underlying conceptual space required for the calculation of prototypes and for the Voronoi tessellation. [179, Section 4.5] (emphasis in original)

Updating prototypes

Since a prototype p is simply an average over the examples seen for the given concept C , it can also be incrementally updated based on a novel example x , essentially resorting to a moving average:

$$\Delta p_d = \frac{x_d - p_d}{N + 1}$$

Updating conceptual prototypes

Once one of the prototype changes, also the resulting Voronoi tessellation of the space changes (see Figure 7.1b). As Gärdenfors notes, this simple learning procedure can already explain over-generalization effects in children: When an infant learns the word "dog", they typically also apply it to other animals such as horses or cows. This can be explained by the child having only very few prototypes in their conceptual space, thus resulting in very large regions in the Voronoi tessellation. Over time, the child may learn additional words with their associated prototypes, thus leading to finer-grained distinctions in the Voronoi tessellation of the conceptual space and hence a reduction in over-generalization.

Relation to machine learning

One should note that Gärdenfors' approach to concept learning is somewhat similar to the k nearest neighbor algorithm introduced in Section 5.2.5, which also uses distances in the similarity space as main source of information. However, while k nearest neighbor retains all examples in memory and makes its classification based on the class information of the k nearest neighbors, Gärdenfors first compresses the information from all examples into a small set of prototypes, and then uses a 1 nearest neighbor rule to obtain the Voronoi tessellation of the space. Gärdenfors' proposal is thus also similar to approaches such as ProtoNet [381] (cf. Section 5.3.3).

One shot learning with correlations

Frommelt [161] considered learning concepts with geometrically encoded correlations from a single example. This is an especially challenging problem, since a single example is not able to convey any information about correlations between domains and dimensions. Moreover, it is impossible to make any deductions about the variance of individual features, and hence about size of the concept.

Frommelt used the formalization of conceptual spaces presented in Part I of this dissertation. In order to learn a new concept \tilde{S} such as $\tilde{S}_{\text{MAMMAL}}$ based on a single example \vec{x} , he leveraged prior knowledge extracted from its sibling concepts, e.g., \tilde{S}_{FISH} and \tilde{S}_{BIRD} . For each sibling concept \tilde{S}_k , he identified the direction $\vec{z}^{(k)}$ of greatest variance (and thus of greatest correlation) by taking the first component of a principal component analysis (which has been introduced in Section 5.3.2), which was applied to the midpoints of all cuboids of that concept. Moreover, he also recorded the order of magnitude of this variance $\sigma^{(k)}$ as a measure of the concept's size. Frommelt then averaged these directions $\vec{z}^{(k)}$ of greatest variance as well as the magnitudes $\sigma^{(k)}$ of the variances across all sibling concepts \tilde{S}_k to obtain a rough estimate of both the direction of correlation (provided as average vector $\vec{z}^{(avg)}$) and the variance of examples for the novel concept (stored as average variance $\sigma^{(k)}$). The new concept \tilde{S} was then initialized with two cuboids, which met at the coordinates of the given example \vec{x} , and whose remaining corner points were defined based on the expected direction $\vec{z}^{(avg)}$ of correlation and the expected variance $\sigma^{(avg)}$ along that direction.

Estimating correlations with PCA

In his experiments, Frommelt used the dataset provided by Lieto et al. [261], which was also used as a test bed for the Dual-PECCS system [265] discussed in Section 1.2.2. Frommelt used animal concepts at the family level (i.e., MAMMAL, BIRD, FISH, etc.) as a test case and created their respective conceptual region by defining each example from the dataset as a trivial cuboid and then applying the repair mechanism, which has been proposed in Sections 3.1.1 and 3.2.1 for both the intersection and the unification operators.

Experiments

His evaluation results in a leave-one-out procedure on both a binary and a multi-class classification task showed, that his approach performed better than initializing the concept as spanning the whole conceptual space. However, his results were often worse than the ones obtained by the trivial concept, which only includes the given example and excludes all other points. This indicates, that the extension of the conceptual region based on the expected correlations led to more false positives than true positives. Frommelt also reported, that artificially translating the learned concept to its optimal position considerably improved results, highlighting that outlier data points led to a poor estimate of the concept's central region, and thus prevented generalization. While the proposal by Frommelt was thus not successful in an empirical evaluation, he lists several possibilities for improving his algorithm, which has so far only be implemented in a very naive form.

Results

7.1.2 The Bayesian Perspective

Let us now turn to a view, which considers concept learning as induction and uses a form of Bayesian inference. *Bayesian inference* can be defined as starting with a prior probability distribution $\mathbb{P}(h)$ over a set of statistical

Bayesian inference

hypotheses $h \in H$, and then using Bayes' theorem to estimate a posterior distribution $\mathbb{P}(h|\{e_1, \dots, e_n\})$ in the light of pieces of evidence e_i :

$$\mathbb{P}(h|\{e_1, \dots, e_n\}) = \frac{\mathbb{P}(h) \cdot \mathbb{P}(\{e_1, \dots, e_n\}|h)}{\mathbb{P}(\{e_1, \dots, e_n\})}$$

Since the denominator $\mathbb{P}(\{e_1, \dots, e_n\})$ serves mainly a normalization purpose, most analyses focus on the likelihood $\mathbb{P}(\{e_1, \dots, e_n\}|h)$.

*Carnap's work:
families, predicates,
and attribute spaces*

The studies considered in this section are all based on Carnap's "Basic System of Inductive Logic" [88, 89]. We will therefore begin by introducing the relevant notions of Carnap's system based on the summary of Sznajder [403]: A *language* is a set of *families*, which consists of unary *predicates*, that are defined on the same *domain*, and that are mutually exclusive and jointly exhaustive. In other words, the predicates of a family provide a partition of the underlying domain. For instance, the family of COLOR words contains the predicates RED, GREEN, and BLUE. Such a family is by itself a purely syntactic construct. Carnap now associates each family with a so-called *attribute space*, where distances are inversely related to perceived similarities, and where each predicate maps onto one region in this space. One can easily see, that this notion of attribute spaces is tightly connected to conceptual spaces, which also assume a distance-based notion of similarity. Moreover, the partitioning of the domain by predicates proposed by Carnap can be linked to Gärdenfors' Voronoi tessellations. However, while Gärdenfors argues that conceptual regions should be convex, Carnap does not consider any constraints on conceptual regions.

*Confirmation
functions*

In his framework, Carnap considers so-called *confirmation functions* $\mathbb{P}(H|E)$ over pairs of propositions H and E . The value of such a confirmation function represents the rational degree of belief in H in a situation, where the agent's only information is given by E . Alternatively, one can characterize $\mathbb{P}(H|E)$ as the degree of inductive support for H by E . Carnap now uses the aforementioned attribute spaces to derive values of the confirmation functions for some simple propositions. The mechanism relating attribute spaces to confirmation functions can be broken down into two rules:

*The γ rule for prior
probabilities*

The γ rule considers the prior probability $\mathbb{P}(A)$ for observing an object, which can be described with the predicate A . Instead of using a uniform distribution over all predicates as a prior, Carnap postulates that $\mathbb{P}(A)$ should equal the normalized size of the region representing A in the attribute space. Predicates covering larger fractions of the domain should thus have a larger prior probability than predicates, which are defined in a more narrow way.

*The η rule for
analogical learning*

The η rule on the other hand concerns analogical reasoning within a domain: Let us assume that predicate A is more similar to predicate B than to predicate C (based on the distances of the respective regions in the attribute space). If we now observe an object a of type A , the expectation of the next observed object b belonging to predicate B is higher than the expectation of said object b to belong to type C . This

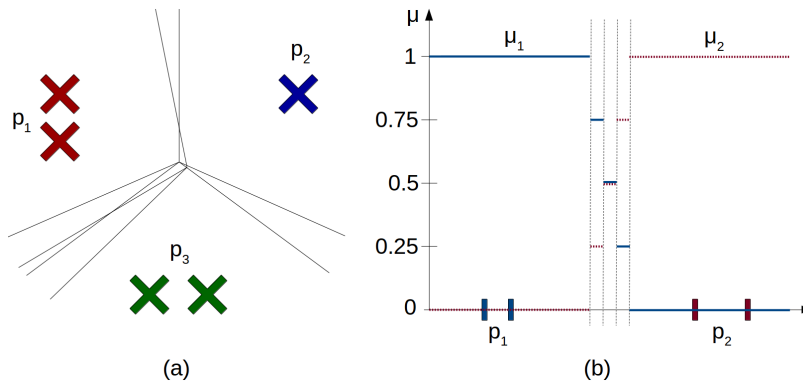


Figure 7.2: (a) Collated Voronoi diagram of a two-dimensional conceptual space. (b) Membership function based on a collated Voronoi diagram in a one-dimensional space.

can be formalized as follows, using the notation $A(a)$ to denote, that the observation a belongs to the predicate A :

$$\eta_{AB} = \frac{P(A(a) \cap B(b))}{P(A(a))P(B(b))} > \frac{P(A(a) \cap C(b))}{P(A(a))P(C(b))} = \eta_{AC} \quad (7.1)$$

Essentially, probabilistic dependence is thus grounded in the similarity of predicates in the attribute space. The so-called η curve is a continuous positive function of the distance in the attribute space. By integrating this η curve over the regions corresponding to the individual predicates, one can obtain the probabilities for the next observations as described in Equation 7.1.

Obtaining η values from attribute spaces

Decock et al. [118] relate both Carnap’s work and the conceptual spaces framework to the principle of indifference. The *principle of indifference* states, that all propositions in a set of mutually exclusive and jointly exhaustive propositions should have the same prior probability. Decock et al. note, that while this may be intuitively appealing, it depends on the number of alternatives under consideration: When drawing a colored ball from an urn with unknown content, one can derive a probability of 0.5 for drawing a red ball if considering the propositions RED and NOT RED. However, when considering the alternatives RED, BLUE, and OTHER, the principle of indifference would lead to a probability of one third for drawing a red ball.

The principle of indifference

Decock et al. note that Carnap’s γ rule can be interpreted as a geometric variant of the principle of indifference, which uses the relative size of a proposition in attribute space to determine its prior probability. In the example from above, the prior probability of drawing a red ball would be the size of the region denoting RED divided by the overall size of the COLOR domain. Since the propositions in Carnap’s framework are assumed to partition the attribute space, this prior probability does not change when considering different sets of alternatives.

Relation to Carnap’s γ rule

Decock et al. now point out that attribute spaces are relatively undetermined and thus come with large degrees of freedom. They thus

Conceptual spaces and collated Voronoi diagrams

propose to use conceptual spaces instead, which come with a larger amount of cognitive constraints, such as the alignment with results from psychological experiments. They resort to *collated Voronoi diagrams* as introduced by Douven et al. [136] (cf. Section 2.4.1), where concepts are represented with prototypical regions instead of prototypical points, and where the partitioning of the conceptual space is obtained by overlaying all possible Voronoi tessellations based on individual prototypes. Figure 7.2a shows such a collated Voronoi diagram in a two-dimensional similarity space. Decock et al. [118], argue, that one can obtain a soft degree of membership for a point x to a concept C from this collated Voronoi diagram by dividing the number of diagrams for which x lies inside the region for C by the overall number of diagrams. This is illustrated in Figure 7.2b for a one-dimensional example. Decock et al. now use the integral over these membership functions to compute the size of a conceptual region. Dividing this size of C by the size of the overall conceptual space can thus provide prior probabilities in a way analogous to Carnap's γ rule. Decock et al. furthermore note, that it is easily possible to take into account multiple domains by simply considering their product space.

*Observations as
points in attribute
spaces*

Also Sznajder [402, 403] has noted the similarity of Carnap's attribute spaces to the conceptual spaces framework. At the same time, she argues that Carnap does not make full use of the geometric structures, since observations are only made on the level of predicates, but are not specified as coordinates in the attribute space. By assuming that individual observations are represented as points in the similarity spaces, Sznajder generalizes Carnap's original proposal. For instance, instead of centering Carnap's η curves on the middle of the conceptual region, one can then center them on the current data point, resulting in different predictions for the generalization behavior of prototypical observations and borderline observations [402, Section 3.5].

*From region-based to
point-based
predicates*

In her later work [403], Sznajder further generalizes Carnap's approach by replacing region-based predicates with point-based predicates, essentially treating hypotheses not on the predicate level, but rather as distributions over points in the similarity space. Carnap's prediction methods for a finite number of region predicates correspond to so-called *Dirichlet priors*, which are a multivariate generalization of the beta distribution on sets of multinomial hypotheses (where each hypothesis corresponds to a particular probability distribution over the predicates). This can be generalized to an infinite number of point-based predicates with so-called *Dirichlet processes*, where each hypothesis represents a probability density function over the infinitely many values in the continuous similarity space.

*Updating conceptual
regions*

Sznajder [403] furthermore points out, that Carnap partitioned the attribute space into predicates, but did not consider revisions of this partitioning. This can, however, easily be done with her extension, since it does not rely on fixed predicates, but represents a general probability density function over the similarity space. This probability density function may implicitly show a cluster structure by distinguishing

regions with higher density from regions with lower density. It can be incrementally updated based on new observations, thus potentially changing the different regions that are being distinguished. However, Sznajder cautions against the complete removal of predicates from the overall system: She argues that a core aspect of induction is to formulate useful generalizations and that region-based predicates offer a way of capturing such generalizations.

In order to illustrate the advantage of her proposal over Carnap's original work, she considers two predicates (BLACK and WHITE), which are defined on the LIGHTNESS dimension of the COLOR space (represented as unit interval, which is split into half by the two predicates). She furthermore assumes that the following observations e are made:

$$e = \{0.41, 0.42, 0.43, 0.44, 0.45, 0.55, 0.56, 0.57, 0.58, 0.59\}$$

Carnap's original rules provide us with $\mathbb{P}(\text{WHITE}|e) = \mathbb{P}(\text{BLACK}|e) = 0.5$, indicating that the distribution of colors in the world is quite uniform with respect to the two predicates. Sznajder's extended approach, however, is able to capture the statistics of the given observations much better, allowing us for instance to make the distinction into $\mathbb{P}([0, 0.4]|e) = \mathbb{P}([0.6, 1]|e) \approx 0.036$ and $\mathbb{P}([0.4, 0.6]|e) \approx 0.927$. This finer-grained view shows, that the examples form a cluster in the center of the similarity space, indicating that one may need to add more categorical distinctions in this region.

Also Poth [323, 324, 325] approaches the concept learning problem by combining Bayesian inference with the conceptual spaces framework. She does so by extending the categorization model of Tenenbaum et al. [406, 441]. In this model, learners use Bayes' theorem to estimate $\mathbb{P}(h|e)$ for each hypothesis h based on the given evidence e . In this context, each hypothesis describes the overall region occupied by stimuli of a common category. Moreover, the model by Tenenbaum et al. introduces the so-called *size principle*, which states, that the likelihood term $\mathbb{P}(e|h)$ is proportional to $\left(\frac{1}{|R_h|}\right)^N$, where N denotes the number of examples and R_h corresponds to the conceptual region expressed by hypothesis h . This size principle results in two empirical predictions: On the one hand, if the number of examples is held constant, the learner should prefer smaller regions. On the other hand, if the perceived similarities are held constant, a larger number of examples further emphasizes this preference on smaller regions. For example, three example images for the word "fep" are provided, and all three examples happen to be Dalmatians, then the probability for "fep" referring to DALMATIAN should be higher than for mapping it to DOG OR OBJECT WITH BLACK DOTS: For the latter two categories, one would simply expect a larger variance in the (presumably randomly sampled) examples.

Poth [324] notices three weaknesses of the proposal of Tenenbaum et al. and proposes a solution for each of them: Firstly, similarities are completely replaced by probabilities in this Bayesian setting, although both exemplar variability and the description of geometric regions

A simple example

A finer-grained view

The size principle

Adding similarities to the size principle

require such a notion of similarity. Poth proposes to remedy this problem by grounding the Bayesian approach in conceptual spaces, where a geometric representation of similarities is given. She proposes the following reformulation of the size principle, which measures the relative overlap of the evidence e with the candidate region R_h :

$$\mathbb{P}(e|h) \propto \left(\frac{|e \cap R_h|}{|R_h|} \right)$$

The likelihood term $\mathbb{P}(e|h)$ can thus be replaced with relative proportions of regions in a conceptual space.

Adding prior probabilities

Secondly, Poth argues, that the proposal by Tenenbaum et al. is incomplete, since it focuses only on the likelihood $\mathbb{P}(e|h)$, but ignores the prior probabilities $\mathbb{P}(h)$. She proposes to use the prior probabilities as defined by Decock et al. [118] based on the relative size of the conceptual regions (cf. our discussion from above).

Counteracting under-generalization with well-formed regions

Thirdly, she notices, that the size principle by itself may lead to under-generalizations, since it quite aggressively favors (very) small conceptual regions, and since no constraint is put on the shapes of such regions. According to Poth, this problem with under-generalizations can be implicitly solved through the definition of conceptual regions as being convex, or (as a special case) based on a Voronoi tessellation: Since conceptual spaces thus do not allow for regions, which minimally fit the given examples, but at least require a convex hull, a certain degree of over-generalization is automatically incited, which may counteract the under-generalization tendency of the size principle.

The complex first paradox

Poth and Brössel [325] use this modified version of the size principle as a way to explain the *complex first paradox*. The complex first paradox states, that children tend to acquire complex concepts such as DOG earlier than simple concepts like BROWN, even though neuroscientific theories indicate, that the representation of complex concepts is more widely distributed in cortex [430]. Their explanation for this paradox is based on the assumption, that complex concepts like DOG are smaller than simple concepts like BROWN: Firstly, one can assume that the extension of DOG (i.e., the set of possible examples falling under this category) is much smaller than the extension of BROWN. Secondly, the objects falling under BROWN are very similar with respect to their COLOR, but may differ considerably with respect to other aspects such as SHAPE or SIZE. Members of the DOG concept, however, are similar to each other with respect to many aspects (such as COLOR, SHAPE, and SIZE).

Limitations of the extended size principle

Poth and Brössel acknowledge, that this view is not able to account for the basic level bias [298, Chapter 7] (i.e., that children tend to learn basic level categories such as DOG earlier than subordinate concepts such as DALMATIAN, cf. Section 1.1.2) and for the shape bias [212] (i.e., the preference for shape-based concepts, to be discussed in more detail in Chapter 10). They speculate, that both of these biases may not be innate, but rather learned, since caregivers tend to teach basic-level concepts, and since the SHAPE domain is more useful than other domains (such as COLOR) in order to make classifications.

Our summaries in this section have illustrated, that conceptual spaces have been successfully linked to philosophical and probabilistic account of concept learning and induction. The studies discussed above thus provide further support to the general conceptual spaces framework as an interdisciplinary and cognitive approach. From the viewpoint of artificial intelligence, the considerations from this section are, however, too abstract in the sense, that they do not provide explicit algorithms and empirical evaluations, which are, however, necessary for an actual application of conceptual spaces in a technical system.

Outlook

7.1.3 The Machine Learning Perspective

Let us now consider conceptual spaces from the viewpoint of machine learning. As already discussed in Section 1.2.5 and at various points in Chapter 5, conceptual spaces are closely related to the feature spaces commonly used in machine learning. However, conceptual spaces are typically spanned by a small set of interpretable and cognitively meaningful dimensions, while no such constraint applies to feature spaces in general: They can incorporate a large number of dimensions, whose main aim is to provide a good distinction between different classes without targeting psychological plausibility. Nevertheless, one can easily envision the application of standard machine learning techniques in conceptual spaces, as already noted at various points in Chapter 5.

*Conceptual spaces
and feature spaces*

Lee and Portier [247] describe an experiment, in which they analyzed the effect of the conceptual space's domain structure on learning. For a small dataset of fruits, they compared the performance of different nearest-neighbor approaches on two kinds of feature spaces: In the first type of feature space, the Euclidean metric was used as an overall distance measure for combining all dimensions. In the second type of space, however, the dimensions were grouped into semantic domains, and distance within a domain was measured by the Euclidean metric, while the distances of different domains were combined by using the Manhattan metric. The results obtained by Lee and Portier indicate, that using a domain structure can aid the learning process for nearest neighbor classifiers. However, their results should be interpreted with caution, since their study was conducted on a single, very small dataset.

*Domain structure
may aid learning*

Derrac and Schockaert [123] have not only defined various notions of conceptual betweenness (cf. Section 4.5.1) and extracted conceptual spaces from textual data (cf. Section 6.3.2, to be discussed in more detail in Chapter 8), but they have also investigated the performance of different types of classifiers in these similarity spaces. Derrac and Schockaert argue, that similarity-based classification has important limitations, which can be overcome, if information about conceptual betweenness and about salient directions are taken into account. For instance, similarity-based classification is only applicable, if sufficiently similar instances are available. Since conceptual betweenness does not

*Comparing classifiers
in conceptual spaces*

directly depend on closeness in the similarity space, it is also applicable in the absence of very similar instances.

Betweenness-based classifiers

While a similarity-based classifier can be equated with a k nearest neighbors classifier (cf. Section 5.2.5), a *betweenness-based classifier* works as follows: When trying to classify a new observation x , one searches for pairs of examples (a, c) , which belong to the same concept, and for which the degree of betweenness $B(a, x, c)$ is high. The pair of points (a, c) , which maximizes this degree of betweenness then determines the classification of the observation x . This implicitly uses the convexity assumption of the conceptual spaces framework: If both a and c are classified as RED, then also all points between them must be labeled as RED. In analogy to a k nearest neighbor classifier, one can also take into account the top k pairs of points a, c with highest betweenness values, and determine the classification of x based on a (weighted) majority vote among those k pairs.

Competitors

Derrac and Schockaert compared the performance of such a betweenness-based classifier to several baselines, including support vector machines (cf. Section 5.2.3), decision trees (cf. Section 5.2.4), and k nearest neighbors (cf. Section 5.2.5). Moreover, they analyzed the performance of rule-based classifiers, which extract rules in the form of "if x is MORE SCARY than THE SHINING, then x is a HORROR MOVIE" based on interpretable directions such as SCARY in their conceptual spaces.

Experimental results

Derrac and Schockaert conducted experiments on three of datasets, which considered the domains of PLACE TYPE, MOVIE, and WINE, respectively, and which had been extracted from word co-occurrence statistics (cf. Section 6.3.2). Derrac and Schockaert made the following observations: If there was only little data available (such as for the smaller datasets of the PLACE TYPE and WINE domains), traditional classifiers did not perform very well and could be outperformed by betweenness-based classifiers. However, for the larger dataset of the MOVIE domain, support vector machines achieved the best performance. There, also rule-based classifiers, which used interpretable directions in the conceptual space, showed a reasonably good level of performance. One advantage of the rule-based classifiers is of course, that they can automatically generate interpretable explanations for their classification behavior, which is more difficult for classifiers like support vector machines. Notably, betweenness-based classification is not easily applicable to large datasets – if there are too many candidate pairs (a, c) to consider, then evaluating all of them requires prohibitively large amounts of computation time. Derrac and Schockaert noted that in most cases, classification performance was relatively insensitive to the number of dimensions of the underlying conceptual space. They also emphasize, that the k nearest neighbors classifier was quite sensitive to the choice of k , while the betweenness-based classifier was considerably more robust in this respect.

Comparison to human data

Finally, they used the dataset of the PLACE TYPE domain to compare the performance of different classifiers to that of human subjects. In their study, human participants rated statements in the form of " X is a kind of Y " as correct, partially correct, or incorrect. They did not have

access to a conceptual spaces representation and made the decision solely based on the statement and their world knowledge. Derrac and Schockaert compared the results of this study with the predictions of the different classifiers and found, that betweenness-based classification achieved a similar precision and accuracy as humans (when using a domain ontology as gold standard), but had a considerably lower recall. This means, that a betweenness-based classifier tended to result in more false negative classifications (cf. Section 5.1.2) than humans. The similarity-based k nearest neighbors classifier, however, was not able to achieve competitive results on any of these metrics. Derrac and Schockaert interpret this as further support for betweenness-based classification over similarity-based approaches.

Bouraoui and Schockaert [76] considered concept learning from a small number of examples. They note, that especially in high-dimensional spaces, the resulting concepts might be quite unreliable. In order to solve this problem, they proposed to use background knowledge in the form of a description logics knowledge base as additional constraint. They modeled each concept C with a single Gaussian distribution G_C , and assumed, that this distribution's covariance matrix is diagonal. This means, that they did not consider correlations between domains or dimensions. For each concept, they maintained a probability distribution over the concept's mean μ_C and covariance matrix Σ_C , which was iteratively updated. These distributions were both constrained by the available training data and by prior knowledge from the knowledge base. For instance, if C_1 is a subconcept of C_2 (based on the information from the knowledge base), Bouraoui and Schockaert assume, that $G_{C_2}(\mu_{C_1})$ is high, i.e., that the prototype of C_1 is a typical member of C_2 . Moreover, they assume, that in this case $\sigma_{C_1}^2 \leq \sigma_{C_2}^2$ for all entries in Σ_{C_1} and Σ_{C_2} , i.e., that the superordinate concept has a larger variability. Their experimental results showed, that this approach is able to outperform several baselines (including for instance SVMs) on the task of knowledge base completion.

*Conceptual spaces
and background
knowledge*

As you can see, there has so far only been limited work in applying machine learning for identifying conceptual regions in conceptual spaces. Nevertheless, the existing studies highlight, that such a combination may be quite fruitful: The (rather anecdotal) results by Lee and Portier [247] indicate, that the domain structure of a conceptual space may provide a useful constraint for nearest neighbor classifiers. The quite extensive study by Derrac and Schockaert [123] highlights the merits of rule-based and betweenness-based classifiers, which harness the structure of the underlying conceptual space with its interpretable dimensions and its meaningful betweenness relation. Finally, the work by Bouraoui and Schockaert [76] exemplifies, that conceptual spaces are also capable of incorporating knowledge-based constraints. This important topic will be covered in more detail in the upcoming section.

Outlook

7.2 LEARNING CONCEPTS UNDER KNOWLEDGE-BASED CONSTRAINTS

Inductive and deductive approaches

Many purely *inductive machine learning* models, especially deep neural networks, heavily depend on the availability of large-scale datasets. However, collecting large amounts of labeled data is expensive and not always possible, for instance, if some classes occur only very infrequently in practice. On the other hand, *deductive methods* rely completely on prior knowledge (usually expressed in a logical format), and derive hypotheses in an entirely deductive fashion without taking into account training examples. This distinction into inductive and deductive learning has already been introduced in Section 5.1.

Hybrid approaches

Mitchell [295, Chapter 12] notes, that most practical problems seem to lie between these two extremes, since they involve both a limited amount of data and a limited amount of prior knowledge. This motivates the need for combining inductive bottom-up approaches, which are based on statistical regularities in the data, with deductive top-down methods, which are based on formalized domain knowledge. The resulting *hybrid approaches* can be expected to combine imperfect prior knowledge with a limited set of examples to rapidly and robustly generalize to new observations. Also humans are able to learn new concepts, even if presented only with a very small amount of examples [87]. Moreover, the knowledge view on concepts (cf. Section 1.1.1 emphasizes, that humans make use of prior knowledge when acquiring new concepts [298, Chapters 5, 6, 10] (cf. also Section 1.1.2).

Few shot learning

The research area of *few shot learning* attempts to replicate this human ability in the context of machine learning [427]. Please recall from Section 5.1, that machine learning can in general be defined as follows:

Definition 5.1 (Machine Learning)

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E . [295, Chapter 1]

Formal definition

Wang et al. [427] now define few shot learning as a special case of Definition 5.1, where E contains only a limited number of examples with supervision information for the target task T . However, they note that E may also contain prior knowledge, which can be used to substitute the missing examples to some extent. Mitchell [295, Chapter 12] formalizes this assumption by considering E as consisting of a set of training examples D and a domain theory B .

One shot learning and zero shot learning

Two important special cases of few shot learning are *one shot learning* (where E contains only a single labeled example per concept) and *zero shot learning* (where E does not contain a labeled examples at all). In Section 7.1.1, we have reviewed Gärdenfors' original concept learning algorithm [179, Section 4.5], which is capable of few shot learning, and Frommelt's proposal for a one shot learning algorithm [161], which tries to estimate correlations between domains based on background knowledge about related concepts. Moreover, in Sections 3.6.2 and 4.6.1,

we have sketched, how our formalization of the conceptual spaces framework can be used for zero shot learning via concept combination.

Please recall from Section 5.1.1, that a given model can be said to represent a hypothesis $h = \hat{f}(\vec{x})$, which maps from observations to desired outputs (e.g., class labels). The performance measure P for a given task T can be formulated as a loss function J , which measures the difference between the model's prediction $\hat{f}(\vec{x})$ and the ground truth $y = f(\vec{x})$ for all labeled training examples \vec{x} (cf. Sections 5.1.2 and 5.1.3). This loss function should be minimized in order to achieve good performance on the task.

Machine learning as loss minimization

Wang et al. [427] define the *expected risk* of making incorrect predictions as $R(h) = \int J(\hat{f}(\vec{x}), y) d\mathbb{P}(\vec{x}, y)$, where $\mathbb{P}(\vec{x}, y)$ is the true probability distribution over examples \vec{x} with their corresponding label y . The expected risk therefore denotes the expected loss over all possible examples, taking into account their respective probability. The optimal hypothesis h^* minimizes this expected risk $R(h)$.

Expected risk

In general, no machine learning model can represent arbitrary hypotheses – it is always constrained by its inductive bias to a so-called *hypothesis space* H of possible hypotheses $h = \hat{f}(\vec{x}; \theta)$, which can be represented by this model, if its free parameters are set to θ (cf. Section 5.1.1). Thus, the optimal hypothesis h^* , which minimizes the expected risk $R(h)$, may not lie inside of H . The best hypothesis available to the model is thus $\hat{h} \in H$, which minimizes the expected risk using the given model and the parameters θ^* . If $h^* \neq \hat{h}$, then the model is bound to make an *approximation error* $E_{app} = |R(h^*) - R(\hat{h})|$ due to its limited hypothesis space. This is illustrated in Figure 7.3a.

Hypothesis space and approximation error

Moreover, since $\mathbb{P}(\vec{x}, y)$ is unknown in practice, one cannot directly minimize $R(h)$ in order to obtain \hat{h} . One therefore needs to use the *empirical risk* $R_N(h) = \frac{1}{N} \sum_{j=1}^N J(\hat{f}(\vec{x}^{(j)}), y^{(j)})$ instead, which is based on the N training examples available for optimizing the algorithm. We denote by $h_N \in H$ the hypothesis, which minimizes the empirical risk using the given model and the parameters θ_N . Especially if N is small (as in the few shot learning problem), we can expect that $h_N \neq \hat{h}$. This introduces an *estimation error* $E_{est} = |R(h_N) - R(\hat{h})|$, which is based on the fact, that the machine learning model is not able to find the optimal parameters θ^* , since it has only access to a limited amount of N examples. Also this is illustrated in Figure 7.3a.

Empirical risk and estimation error

Wang et al. [427] argue, that the prediction errors made by any type of machine learning model can be viewed as a combination of approximation error (due to the limited hypothesis space) and estimation error (due to insufficient data). In the few shot learning context, one can expect especially the estimation error to be quite high. In order to arrive at better models, one therefore needs to devise methods for reducing this estimation error. Wang et al. argue, that this can be most efficiently done by incorporating prior knowledge.

Minimizing the estimation error for few shot learning

In the following subsections, we will take a closer look at different ways of incorporating prior knowledge into the learning process, help-

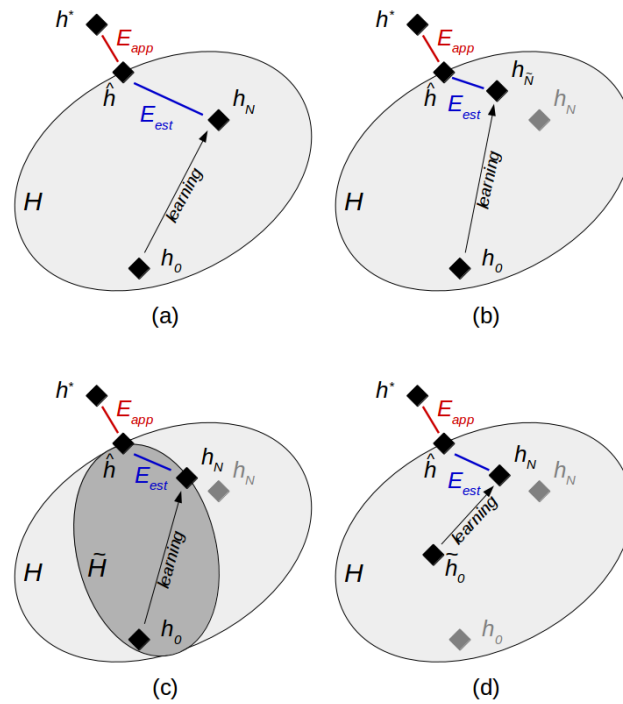


Figure 7.3: Illustration of learning as a search through the hypothesis space. (a) Approximation and estimation error. (b) Improvement through data augmentation. (c) Improvement by reducing the hypothesis space. (d) Improvement through initialization.

ing to solve the few shot learning problem. In Section 7.2.1, we review some general approaches as discussed by Mitchell [295, Chapter 12] and Wang et al. [427]. Afterwards, we present the framework of logic tensor networks [21, 358] in Section 7.2.2, a neural network model, which incorporates logical constraints in the form of fuzzy rules. In Section 7.2.3, we then argue, that logic tensor networks are a useful tool for modeling concept learning in conceptual spaces under constraints from the symbolic layer.

7.2.1 General Approaches

Modifying the search through the hypothesis space

In the following, we will give some more details on different ways for incorporating prior knowledge in order to alleviate the underlying problem of few shot learning. These different ways are motivated by viewing the learning problem as a search through a hypothesis space. Wang et al. [427] distinguish three different types of approaches based on the component of the learning problem, which is being changed:

Data augmentation: increasing the number of examples

Data augmentation techniques (Figure 7.3b) attempt to increase the number of available training examples from N to $\tilde{N} \gg N$. One can then apply standard machine learning techniques on this enlarged dataset without further modifications. Prior knowledge is used to decide how the additional training examples are created.

Constraining the hypothesis space (Figure 7.3c) reduces the (effective) capacity of the model by using soft or hard regularization (cf. Sections 5.1.3 and 6.2.3), and thus discourages the exploration of certain parts of the hypothesis space, or removes them altogether. Prior knowledge is used to determine which parts of the hypothesis space to exclude.

Regularization: constraining the hypothesis space

Modified learning algorithms (Figure 7.3d) search the hypothesis in a different way, either by starting from a different initial hypothesis h_0 , or by using modified search operators. Prior knowledge is used to define the initial hypothesis, or to specify additional search operators.

Modified algorithms: different starting point or additional search operators

All of these approaches have the common goal to reduce the estimation error E_{est} by inductively finding a hypothesis h_N , which lies closer to the optimal hypothesis \hat{h} representable by the model. In the following, we will describe these three basic approaches in more detail and provide some concrete examples.

Reducing the estimation error

The first way of incorporating background knowledge consists in *augmenting the training set*, i.e., creating additional artificial training examples based on the existing training examples and prior knowledge (cf. Section 6.2.3). This data augmentation step is also frequently used for classification problems with imbalanced class frequencies [192]. Wang et al. [427] distinguish the following approaches:

Dataset augmentation

First of all, one can use *hand-crafted rules* for dataset augmentation. For instance, when training a classifier on images, one can modify the image's pixels by applying small translations or rotations of the image. Also additive noise can be used to change the images on the pixel-level while leaving high-level categorical information intact.¹ Wang et al. [427] argue, that such hand-crafted rules are highly dependent on domain knowledge, and are therefore not always applicable. We would like to add, that also an interpolation between neighboring examples in the feature space is a possible way of creating additional data points [192]. In this case, however, one tacitly assumes convexity of categories in the given feature space, which may not always be fulfilled.

Augmentation with hand-crafted rules

Another possibility for obtaining additional data points is to apply a *learned transformation* to examples from the given training set. Again, the idea is to use a transformation function $t(\vec{x}, z)$ with auxiliary input z for transforming each training example $(\vec{x}^{(j)}, y^{(j)})$ into multiple artificial examples $(t(\vec{x}^{(j)}, z), y^{(j)})$. However, the transformation function t is now not handcrafted, but based on automatically identified variations between samples from the given training set or from a larger dataset with similar content. These learned variations are then applied to the training examples in order to generate additional data points. In contrast to handcrafted rules, no explicit prior knowledge is needed, which may, however, also reduce the interpretability of the augmentation step.

Augmentation through learned transformations

Wang et al. also argue, that one can make use of other weakly labeled or unlabeled datasets. For instance, one can train a classifier on the given training set, and then apply its hypothesis h_N onto data points

Augmentation through self-training

¹ In fact, such pixel-wise changes have been used to construct attacks against deep neural networks in order to make them misclassify a given image [401], cf. Section 6.2.3.

$(\vec{x}', -)$ from an unlabeled dataset. The data points $(\vec{x}', h_N(\vec{x}'; \theta))$ with the highest prediction confidence can then be added to the training set as additional examples for training an updated version of the classifier. This procedure is called *self-training*. However, one needs to be careful with this approach, since confident misclassifications made by h_N decrease the quality of the training set.

Augmentation with similar datasets

Finally, it is also possible to take samples from *similar datasets* and adapt them accordingly. However, it is in general not possible to directly copy the examples, for example, because different class labels are used. Also a different distribution of the examples (e.g., photographs versus line drawings) may cause the need for adaption. Overall, the samples from the supplementary dataset thus need to be suitably transformed, often by aggregating across multiple samples. Wang et al. mention for instance generative adversarial networks [173] as a possible way to translate samples between datasets, e.g., by using CycleGAN [459] (cf. Section 6.3.3), if a manual rule-based translation is not possible.

Properties of data augmentation

Wang et al. [427] note, that data augmentation is often a relatively straightforward process, which is easy to understand. Especially in the case of images as input, one can easily visualize the artificially created examples and verify their quality. However, for other types of data, which are not that easy to visualize, it may be more complex to ensure, that the additional training examples have a sufficiently high quality. Moreover, Wang et al. argue, that the transformation function is often tailor-made for a specific dataset and can thus hardly be reused.

Constraining the hypothesis space

A second way of constraining machine learning algorithms through background knowledge consists in *reducing the hypothesis space H* in such a way, that the distance between \hat{h} and h_N becomes smaller. This reduction should, however, ensure, that the distance between h^* and \hat{h} remains small (see Figure 7.3c) [427]. In other words, one can try to exclude hypotheses $h \in H$, for which prior knowledge predicts poor performance. This can be done by either removing parts of the hypothesis space completely, or by discouraging the exploration of certain parts of the hypothesis space (cf. hard vs. soft regularization, Sections 5.1.3 and 6.2.3).

Multitask learning

One popular approach used in the deep learning literature is *multitask learning* [427] (cf. Sections 6.2.3 and 6.3). Here, a single model is trained to solve multiple related tasks at the same time. Re-using model parameters (e.g., by sharing the lower layers between tasks) serves as an additional constraint on the hypothesis space H . In this case, prior knowledges comes in the form of related tasks and their respective datasets.

Explicit regularization term

If prior knowledge is available in a more direct format, one can directly incorporate this knowledge through an *additional term in the loss function*. For instance, Mitchell [295, Chapter 12] describes the *EBNN* (*explanation-based neural network*) framework, which uses derivatives as additional constraints to the learning problem: One first constructs an auxiliary neural network ANN_1 based on a given domain theory: For each term in the domain theory, an individual neuron is created and edges between neurons are inserted based on the relations between

terms in the domain theory. The weights of this neural network are initialized in such a way that its output is identical to the predictions by the domain theory. Now a new neural network ANN_2 is trained using the given dataset, while the auxiliary network ANN_1 remains unchanged. For each training example, the derivatives of the auxiliary network's output with respect to each of its nodes in the input layer are computed. If the auxiliary network ANN_1 (and thus the domain theory) correctly predicts the label of the given example, an additional term in the loss function of ANN_2 is used to ensure, that also the input-output derivatives of ANN_2 match the ones by ANN_1 (and hence the predictions made by the domain theory). This soft constraint ensures, that parts of the hypothesis space, which are in line with the background knowledge, are more likely to be explored.

In Section 5.1.4, we have already discussed, that the dimensionality of the feature space plays a crucial role in determining the number of free parameters in the machine learning model and thus the size of the hypothesis space. Another way of reducing the hypothesis space therefore consists in reducing the number of features, for instance through *feature selection* (cf. Section 5.3.1) and *feature extraction* (cf. Section 5.3.2). If *metric learning* [236] (cf. Section 5.3.3) is used for dimensionality reduction, one can often make use of simpler classifiers (e.g., k NN or a simple prototype-based classifier [381]). Since simpler classifiers have less free parameters, this further reduces the size of the hypothesis space. Also the area of representation learning as discussed in Sections 6.1 and 6.3 has similar aims.

Lampert et al. [241] have introduced a specific variant of restricting the hypothesis space, which they call *attribute-based classification*. They explicitly target zero shot learning, where observations need to be classified with respect to a new class, for which no training examples are available. The only information available for this new class is assumed to be a high-level description in terms of semantic attributes. Lampert et al. propose to first learn a mapping from the original input (e.g., images) to these semantic attributes by training one separate classifier per attribute. Afterwards, one can define a zero shot classifier by using the predictions of the attribute classifiers to infer the presence or absence of the respective attributes, and comparing this to the high-level description of the class. The work by Lampert et al. thus uses a dimensionality reduction approach, because the number of semantic attributes is considerably lower than the dimensionality of the original input. Since zero shot classes are defined as a conjunction of attributes, their approach results in a quite interpretable overall system. We will revisit their work in Chapter 8 in the context of multidimensional scaling and interpretable similarity spaces.

Recently, Clay et al. [107] have proposed a few shot learning approach based on pretrained neural networks. They considered three setups for training the same encoder network, and investigated, how well one can then extract concepts on top of the learned representation. As input, they used RGB images of a video game world, and the encoder network was either trained in a fully supervised classification task, as part of

*Dimensionality
reduction*

*Attribute-based
classification*

*Training neural
networks for few shot
learning*

an autoencoder in an unsupervised reconstruction task, or as part of a larger deep reinforcement learning network. In the latter case, the network was able to control the in-game avatar through its outputs and optimized a curiosity objective, which rewarded the network for making unexpected observations, and hence for exploring its environment.

Fast concept mapping

Clay et al. then used a procedure they called *fast concept mapping* (FCM) in order to extract concepts from the learned embedding based on a small number of examples: For each example, they created a graph of all neurons in the embedding layer, where edges indicated pairs of jointly active neurons. By summing the connection strengths over all examples for a given concept, they obtained a weighted connection graph, where the weight of each connection was based on the number of times it was present in the individual graphs. Clay et al. then defined a concept by taking the k strongest connections. Classification decisions for novel inputs were reached by creating the connectivity graph of the novel input, and by then computing the weighted overlap between this connectivity graph and the concept definition in order to obtain a confidence value. Clay et al. found, that their FCM approach reached a good classification performance already for a small number of examples, and that the embodied reinforcement learning network clearly outperformed the unsupervised autoencoder, being only marginally worse than the fully supervised network. Like the attribute-based classification by Lampert et al. [241], the FCM approach by Clay et al. [107] can be interpreted as using a dimensionality reduction approach.

Modifications of the learning algorithm

A third way of adding background knowledge to machine learning is concerned with *modifications of the optimization algorithm*. Wang et al. [427] assume, that the hypothesis h_N is found by an iterative optimization of the model's free parameters θ , i.e., $\theta_t = \theta_{t-1} + \Delta\theta_{t-1}$. This mainly applies to artificial neural networks (cf. Section 6.2), but not necessarily to other classifiers like support vector machines (cf. Section 5.2.3 or k nearest neighbors (cf. Section 5.2.5).

Initialization through transfer learning

This iterative optimization can be modified by initializing the model's parameters θ_0 (and hence the initial hypothesis h_0) based on prior knowledge (illustrated in Figure 7.3d). For instance, one can first train the model on other tasks with large datasets to obtain θ_0 , which is then subsequently fine-tuned based on the new task. This approach is called *transfer learning* in the literature (cf. Section 6.2.3).

Initialization through explicit prior knowledge

Also explicit prior knowledge can be used to obtain an initial hypothesis. The *KBANN* (*knowledge-based artificial neural networks*) framework discussed by Mitchell [295, Chapter 12] for instance uses rule-based domain knowledge to initialize the structure and the weights of a neural network similar to the EBNN approach discussed above. However, instead of training a second neural network from scratch, the weights of the initialized network are updated with gradient descent based on the examples from the training set, adapting the imperfect domain theory to the empirical observations. As Mitchell points out, the individual nodes of the network are initialized with a specific meaning, but their interpretation may change as the weights are updated.

Another possibility is to modify the update steps for the search through the hypothesis space, i.e., to change $\Delta\theta_{t-1}$. In this respect, Wang et al. [427] mention *meta-learning* as a possible, yet relatively complex approach. Here, a meta-model is trained to output search steps for the optimizer of the actual model to be trained. This means that the meta-model takes the current parameters θ_{t-1} as an input and outputs proposed updates $\Delta\theta_{t-1}$. Changes in the direction or the size of the update are simple examples for such an approach (cf. e.g., the momentum rule from Section 6.2.5).

Modified update steps with meta-learning

In case on non-neural learning algorithms, background knowledge can also be used to expand the set of possible search operators. Mitchell [295, Chapter 12] for instance presents FOCL as a variant of FOIL, which is an inductive rule learning algorithm. In both FOCL and FOIL, hypotheses are represented through a set of classification rules, where a given term is defined based on the presence or absence of other (more elementary) terms. In both cases, the system starts with the most general rule and then greedily specializes this rule by looking at all possible candidate specializations, and by choosing the one with the greatest information gain (cf. decision tree learning in Section 5.2.4). FOIL creates candidate specializations by adding only a single literal to the preconditions, whereas FOCL also uses its domain theory to create more complex specializations, thus allowing for larger steps in the hypothesis space and faster learning.

Additional search operators for rule-based learning

As one can see, there are many different variants of using prior knowledge in machine learning tasks in order to reduce the number of training examples necessary for obtaining good performance. In almost all cases, prior knowledge is used to create some form of regularization. Determining the most applicable approach for a given machine learning problem depends heavily on the learning task, the training examples, and the background knowledge available. Of course, one can in principle also combine multiple of these approaches with each other, e.g., by using both data augmentation and dimensionality reduction.

Outlook

7.2.2 Logic Tensor Networks

In the context of neural-symbolic learning and reasoning [111], *logic tensor networks (LTNs)* [21, 358, 359, 360] have been recently proposed as a principled way of using neural computations to connect feature spaces with symbolic rules through fuzzy logic.² Logic tensor networks integrate knowledge representation, learning, and reasoning using a differentiable fuzzy first-order logic language called *real logic*. They are especially useful for hybrid domains, where both numerical (subsymbolic) information and relational (symbolic) knowledge are available. From the symbolic perspective, LTNs provide a spatial embedding for fuzzy logics, while from the subsymbolic perspective, they are a

Logic tensor networks in general

² See <https://github.com/logictensornetworks/logictensornetworks> for the implementation of this framework.

machine-learning algorithm, which is based on neural networks and guided by additional logical constraints.

Real logic in a nutshell

Real logic is a first-order logical language, which contains *constant symbols* (representing individuals such as BOB or PARIS), *function symbols* (representing mappings between observations, e.g., HOMETOWNOF), *predicate symbols* (representing concepts and relations such as LAWYER and LIVESIN), and *variable symbols* (representing lists of observations) [21]. All of these language constituents are typed with respect to a set of *domains* \mathcal{D} : We can require, that BOB belongs to the domain of PEOPLE, and PARIS to the domain of CITIES. Moreover, we can require that the function HOMETOWNOF takes only PEOPLE as input and returns CITIES, and that the predicate LAWYER is only applicable to the domain of PEOPLE. The individual parts of the language can now be combined into formulas such as LAWYER(BOB), or $\forall x : (\text{LAWYER}(x) \rightarrow (\text{HOMETOWNOF}(x) = \text{PARIS}))$. These formulas are constructed using logical connectives (such as \rightarrow) and quantifiers (such as \forall) and have a fuzzy degree of truth in the interval $[0, 1]$.

Grounding constants and variables

In order to relate the semantics of the logical language to actual data points, real logic makes use of a so-called *grounding function* G_θ , which maps terms (i.e., constants, variables, and results of function applications) onto tensors in the real field.³ In the original formulation of the framework [358, 359, 360], the grounding function G_θ mapped any term onto a point in \mathbb{R}^n , i.e., to a vector of feature values. In the recent reformulation [21], also tensors of higher or lower order are allowed. Constants from the same domain $D \in \mathcal{D}$ are always mapped onto tensors of the same order and size. Variables (which express lists of observations) are mapped to a finite sequence of tensors in the respective domain.

Grounding functions and predicates

Moreover, functions and predicates are mapped onto real functions or tensor operations⁴, which are implemented with neural networks. The networks implementing predicates are required to return a value from the interval $[0,1]$, and can thus be interpreted as defining a fuzzy membership function of the respective concept or relation in the given feature space. Relations such as LIVESIN are implemented as fuzzy regions in a product space of domains (in this case PEOPLE and CITIES). Overall, the grounding G_θ associates any term in the language with a tensor and any formula expressible in the language with a real number in $[0,1]$, representing its degree of truth.

Grounding formulas

In order to determine the truth value of a formula, one uses an element-wise application of the grounding (see Figure 7.4): First, all terms (such as BOB) are grounded into tensors, and all function and predicate symbols (such as HOMETOWNOF and LIVESIN) are grounded into their respective neural networks. The structure of the symbolic formula

³ Tensors are a generalization of vectors and matrices to an arbitrary number of dimensions: A tensor of order 0 corresponds to a scalar, a tensor of order 1 is a vector, and a tensor of order 2 is a matrix. Higher-order tensors then correspond to multidimensional arrays with more than two dimensions.

⁴ Tensor operations are a generalization of vector and matrix operations.

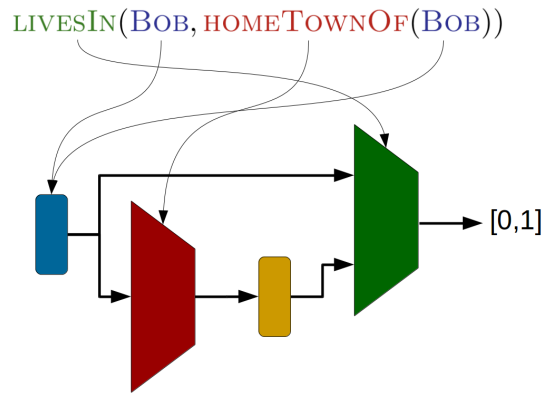


Figure 7.4: Grounding of a formula containing a function symbol (`HOMETOWNOF`), a predicate symbol (`LIVESIN`), and a constant symbol (`BOB`).

then determines, which neural network is applied to which tensor in order to compute the respective output tensor (in the case of a function symbol like `HOMETOWNOF`) or the degree of truth (in the case of a predicate symbol like `LIVESIN`). If predicates such as `MARRIED(x, y)` are applied to variables such as $x = (\text{BOB}, \text{JOHN})$ and $y = (\text{ALICE}, \text{MARY}, \text{SUSAN})$, the resulting grounding is a matrix containing the degree of truth for each possible combination of observations [21]. This also applies to functions and more complex formulas, resulting in output tensors whose dimensionality depends on the number of variables being used.

In order to ground logical connectives such as \wedge , \vee , \neg , and \rightarrow , the corresponding operators from fuzzy logic are used. Examples of such fuzzy operators have already been introduced in Part I of this dissertation (cf. Definitions 3.3, 3.7, 3.11, 4.4, and 4.5) and include $N(x) = 1 - x$ for the negation and $T_{\min}(x, y) = \min(x, y)$ for the conjunction. However, Badreddine et al. [21] note, that many standard operators from fuzzy logic are not well-suited for the context of neural networks, since they may cause vanishing or exploding gradients. A recent thorough study by van Krieken et al. [415] reaches similar conclusions and notes that non-standard combinations of fuzzy logical operators often lead to improved learning performance, while no longer satisfying the usual logical laws. Badreddine et al. [21] recommend using the product norm $T_{\text{prod}}(x, y) = x \cdot y$ for implementing the conjunction and its complement $S_{\text{prod}}(x, y) = x + y - x \cdot y$ for the disjunction. Moreover, they note that edge cases with $x, y \in \{0, 1\}$ are still problematic, and propose to solve these issues by using a projection function to slightly contract the interval $[0, 1]$ before applying logical connectives. For the negation, $N(x) = 1 - x$ is used.

Grounding logical connectives

Since real logic is a first-order language, it also needs to provide a grounding for the universal and the existential quantifier. Mapping $\forall x$ to the minimum over all entries of the variable x may be a straightforward choice, but does not tolerate exceptions. It may thus not be suitable for real-life applications, if one assumes a certain amount of noise both in

Grounding quantifiers

the background knowledge and in the empirical data (e.g., incorrect labels) [131, 360]. Instead, the generalized mean $mean_p(x_1, \dots, x_k) = \left(\frac{1}{k} \sum_{i=1}^k x_i^p\right)^{\frac{1}{p}}$ is used, where the parameter p controls the "strictness" of the aggregator.⁵ The current version of the framework [21] proposes to use different variants of the generalized mean for grounding both the universal and the existential quantifier. This breaks the duality between the existential and the universal quantifier, but seems to be necessary to enable robust gradient-based learning. However, one could in principle use both an exception-tolerant and a rigid classical version for both quantifiers by employing two different values of p . One would then need to specify which quantifier version to apply in which contexts.

Aggregate functions

Recently, Badreddine and Spranger [23] have proposed to extend the LTN framework by introducing more general *aggregate functions* such as computing the sum, average, maximum, minimum, or standard deviation for a given list of values. Please note, that these values do not need to represent degrees of truth (as for the quantifiers), but may, for instance, also reflect individual feature values such as the HEIGHT for members of the PEOPLE domain. Badreddine and Spranger note, that most of the common aggregate functions are differentiable and can thus be easily implemented in the context of LTNs. Enriching real logic and LTNs with such aggregate functions increases their expressive power, both with respect to the evaluation of queries (which can now be formulated in a similar way as for relational databases), and with respect to additional constraints during learning.

Satisfiability

Satisfiability specifies the degree, to which a grounding G_θ satisfies a given set K of formulas, by simply aggregating the truth values of all formulas $\phi \in K$ [358, 359, 360]. Donadello et al. [131] propose to re-use the generalized mean for this purpose, since defining satisfiability based on a simple conjunction over the formulas can lead to undesired behavior in gradient-based optimization.

Knowledge and constraints in LTNs

The knowledge represented in logic tensor networks consists of both the formulas ϕ in the logical language (corresponding to symbolic top-down information) and the grounding G_θ obtained from observations (corresponding to subsymbolic bottom-up information) [21]. These two types of knowledge can be used to encode different types of constraints into the system: For instance, one can explicitly define a grounding for some of the symbols (e.g., mapping a given constant to a concrete feature vector, or defining a binary predicate $\text{sim}(x, y)$ as the Cosine similarity of the feature vectors representing x and y). Also a parametric definition of functions and predicates is possible. In this case, one specifies the structure of the respective network, but leaves the exact parameter settings undetermined. Moreover, different kinds of logical formulas can be used to constrain the system: *Factual propositions* such as $\text{LAWYER}(\text{BOB})$ encode facts about individual constants (which

⁵ Note that $mean_1$ corresponds to the standard arithmetic mean, and $mean_{-1}$ to the harmonic mean. If $\vec{x} = (x_1, \dots, x_d)$ is a difference vector, then $mean_p(x)$ is equivalent to a Minkowski metric (cf. Definition 2.1).

corresponds to providing labels for training examples), while *generalized propositions* such as $\forall x : (\text{LAWYER}(x) \rightarrow \text{HOMETOWNOF}(x) = \text{PARIS})$ use quantified variables to represent general background knowledge.

Learning in logic tensor networks takes place through gradient descent on the parameter values θ of the grounding G_θ in order to maximize the satisfiability of the overall set of formulas K [21]. In practice, maximizing satisfiability may need to be accompanied by a regularization term on the parameters θ to prevent overfitting [21, 360].

Once a grounding has been established, it can be used for answering concrete queries about the truth value of a given formula or about the embedding of a given term. Moreover, one can use reasoning processes in order to determine, whether a new formula ϕ is a logical consequence of the given theory $T = \langle K, G_\theta \rangle$. Badreddine et al. [21] formalize this by checking whether the confidence in ϕ is at least as high as the overall satisfiability of T . They distinguish two types of reasoning modes: In *brave inference*, this comparison is made only for groundings G_θ , which maximally satisfy K . In *refutation-based inference*, the system on the other hand tries to find a counter-example by minimizing the satisfiability of ϕ , while ensuring that the satisfiability of K does not fall below a given threshold. If no such counterexample can be found, then ϕ is deemed to be a logical consequence of T .

When considering logic tensor networks in the context of few shot learning, they can be described according the taxonomy proposed by Wang et al. [427] (cf. Section 7.2.1) as constraining the hypothesis space: One can interpret LTNs as performing multitask learning, where one task is the correct classification of a given set of data points (i.e., satisfying factual propositions), while the other task is to enforce the validity of the given logical rules (i.e., satisfying general propositions). Since both tasks share the same set of underlying parameters, they constrain each other and thus reduce the set of possible hypotheses.

Badreddine et al. [21] have given a principled overview of different tasks, which can be solved with LTNs. These include classification (where predicates are used to model membership to different classes), as well as regression (where the grounding of a function symbol needs to be learned) and clustering (where logical constraints reflect, that clusters should be disjoint, not empty, etc.). Badreddine et al. also discuss semi-supervised pattern recognition, using the example of single digit addition on handwritten digits, where two digit images are used as input and the system needs to predict the sum of the two digits as a scalar output. Also the task of embedding learning and knowledge completion can be solved as illustrated with the well-known smoker-friends example [358, 359]. LTNs have also been used to learn transitive predicates (such as *HYPONYMOF*) for simple ontologies based only on one-hop examples [62]. Moreover, Bianchi et al. [63] have recently illustrated the capability of LTNs to connect pretrained entity embeddings with a subset of the DBpedia ontology [20].

Learning in LTNs

Querying and reasoning in LTNs

LTNs and few shot learning

Applications of LTNs

LTNs for semantic image interpretation

The first larger-scale application of LTNs was implemented by Donadello et al. [132, 360] on the task of *semantic image interpretation*. They focused on the classification of bounding boxes (representing individual objects in an image) and the detection of PARTOF relations between pairs of such bounding boxes. In this context, constants referred to individual bounding boxes, and unary predicates were used for the different object classes, while a binary predicate was used for the PARTOF relation. Each bounding box was represented by a set of features containing the coordinates of its top left and its bottom right corner, as well as classification scores from an off-the-shelf object classifier.

Constraints on the PARTOF relation

In order to harness the possibility offered by logic tensor networks to include prior knowledge, Donadello et al. introduce several mereological axioms, which constrain the PARTOF relation. These constraints include for instance asymmetry ($\forall x, y : \text{PARTOF}(x, y) \rightarrow \neg \text{PARTOF}(y, x)$) and the observation, that in their application scenario, parts cannot have further sub-parts ($\forall x : \text{TAIL}(x) \Rightarrow (\forall y : \neg \text{PARTOF}(y, x))$). As Donadello et al. note, such constraints can be obtained from existing ontologies like WordNet [292].

First results

Donadello et al. found in their experiments that the LTN was able to outperform two simple baselines for the bounding box classification and the detection of the PARTOF relationship. Moreover, they observed that the presence of mereological axioms made the LTN more robust to artificially increased noise levels in the training data.

Generalization to arbitrary relations

In a follow-up study, Donadello et al. [131] have generalized their approach to detecting arbitrary visual relationships, which can be written as triples (*subject, relationship, object*), where both *subject* and *object* are bounding boxes in the image. Due to the large number of possible combinations, any dataset is necessarily incomplete, making generalization difficult. Donadello et al. propose to use logic tensor networks in order to leverage additional background knowledge in the form of logical constraints. In their experiments, they limit themselves to negative constraints in the form of "clothes cannot ride", while leaving other rules such as mutual exclusivity, symmetry, reflexivity, or isa-hierarchies for future work. They furthermore include additional features for describing the relation between two bounding boxes, such as the size of their intersection, or the angle between their centroids.

Experimental results

In several ablation studies, in which they disabled different components of their overall system, Donadello et al. were able to show, that adding prior knowledge in the form of negative constraints and using additional features improved the LTN's performance. When comparing to other state of the art systems, Donadello et al. found, that their LTN model achieved competitive results even though other models used more powerful object classifiers.

Fairness and the neural-symbolic cycle

Recently, Wagner and Garcez [422] have proposed to use LTNs in order to incorporate fairness constraints into deep neural networks. Their approach follows a general *neural-symbolic cycle*, where a neural network is trained on a classification task and then queried in order to extract the learned knowledge. This knowledge is then consolidated and fed back

to the network in the form of additional constraints for further training. This general approach increases the interpretability of the system and can potentially help to prevent *catastrophic interference* [285], i.e., the tendency of neural networks to un-learn previously learned knowledge when being trained on additional tasks and data. The knowledge, which Wagner and Garcez extracted from the LTN, focused on group fairness, i.e., differences in treatment between groups, that differ with respect to a protected attribute such as gender or race. Wagner and Garcez were able to show, that by using the aforementioned fairness constraints, the trained LTN exhibited less bias based on protected attributes without having to sacrifice classification performance.

Badreddine and Spranger [22] have employed logic tensor networks in the context of *reinforcement learning (RL)*, where an artificial agent repeatedly needs to choose between multiple possible actions, which change the state of its environment, and where this agent receives a (potentially delayed) reward based on its actions. They propose to speed up the learning process by providing additional semantic information (such as "object x is an enemy" or "object y is a target location") as input to the system. In their experiment, they consider a simple 5 by 5 grid populated with simple geometric shapes: The agent is represented by a plus sign and other objects are circles, squares, and crosses. These other objects can be assigned to the types `TARGET` and `AVOID`. The goal of the agent is then to collect all `TARGET` objects, while avoiding all `AVOID` objects. As an input to a deep reinforcement learning network, Badreddine and Spranger use a concatenation of an image of the current environment (50 by 50 pixels), as well as prior semantic knowledge. This prior semantic knowledge contains the output of classifiers for the different shapes for each cell of the grid. Moreover, for each cell of the grid, it contains the values for the general predicates `AVOID` and `TARGET` which are specified by logical rules (e.g., $\forall x : \text{CIRCLE}(x) \leftrightarrow \text{TARGET}(x)$). Badreddine and Spranger use LTNs to evaluate these general predicates for each of the grid cells. The deep reinforcement learning network then learns to map these input vectors to one of four possible actions, namely moving up, down, left, or right.

In their experiments, Badreddine and Spranger changed the visual and semantic properties of the environment every 50 epochs by changing object and background colors, or the assignment of objects to the `TARGET` and `AVOID` types. They noticed, that the RL network was able to recover from performance drops at these context changes much faster, if semantic predicates (which were updated according to the changes in the environment) were provided by the LTN. Moreover, in this case the performance drops became smaller after each context change, indicating that the RL network learned to rely more on the semantic information than on the raw perceptual input. Although the study by Badreddine and Spranger uses LTNs only as a small ingredient in their reinforcement learning system, it highlights, that LTNs can be used in a variety of contexts.

*LTNs for
reinforcement
learning*

Experimental results

7.2.3 Towards Conceptual Logic Tensor Networks

Properties of LTNs

The knowledge view on concepts from psychology [298, Chapter 6] emphasizes the crucial role of world knowledge in the learning and application of concepts (cf. Section 1.1.1), and can thus be linked to the influence of logical rules on the learning process in LTNs. Moreover, LTNs take into account information about points in a feature space, and the grounding of predicates usually gives rise to a membership function with one or more receptive fields. LTNs can therefore also be related to the prototype [336] and exemplar theories [287] of concepts. In addition to that, logic tensor networks offer the possibility to combine bottom-up information in the form of training examples with top-down information in the form of general rules.

Bridging the conceptual and the symbolic layer

These observations make LTNs quite interesting from the perspective of conceptual spaces: They can implement a two-way interaction between the conceptual and the symbolic layer. Just as with the conceptual spaces framework, observations can be represented by points and concepts can be represented as regions in the feature space. Moreover, LTNs are able to encode the domain structure of a conceptual space. Finally, the usage of fuzzy sets and fuzzy logic allows us to represent vague conceptual boundaries (cf. Section 2.3.3).

Applying LTNs to conceptual spaces

How exactly can we apply LTNs to conceptual spaces? Both properties and concepts can be represented by predicates with a convex membership function. While properties refer only to a single domain, concepts are defined on a concatenation of domains: We can require, that the predicate $\text{RED}(x)$ is defined on the three-dimensional COLOR domain, while the predicate $\text{APPLE}(x = (x_c, x_t, x_s))$ involves the domains of COLOR , TASTE , and SHAPE . Individual observations can then be represented by one point per domain. Function symbols could potentially be used in the context of events, where they could relate to the force vector and the result vector as proposed by Gärdenfors [181, Chapters 8, 9, and 10] (cf. Section 1.2.1). Applying a function symbol like LIFT could then for instance translate into a simple vector addition in the LOCATION domain, which increases the ALTITUDE of the given object. Finally, basic relations like LONGER THAN can be defined by considering regions in product spaces of multiple domains (e.g., $\mathbb{R}^+ \times \mathbb{R}^+$ for representing the length of the two objects under consideration) [179, Section 3.10.1]. In order to represent more complex relational knowledge for concepts such as ROBBER OR SEAT , one could try to use the event structure proposed by Gärdenfors [181, Chapter 9] for the corresponding ROBBING and SITTING events.

LTNs for commonsense reasoning

An advantage of using logic tensor networks for grounding symbolic knowledge (such as for instance encoded in formal ontologies) in conceptual spaces is their large variety of inference and learning methods. In addition to learning conceptual regions from observations, they can for instance also generate an embedding of an unobserved object based on a symbolic description. For example, "object x is a RED APPLE " can be translated into a point in the conceptual space by maximizing the

satisfiability of $\text{APPLE}(x) \wedge \text{RED}(x)$. This spatial representation of object x can then in turn be used to make further inferences, for instance about the TASTE domain (e.g., by evaluating $\text{SWEET}(x)$). Thus, the geometric embedding can give rise to commonsense inferences not easily realizable within the symbolic layer. This approach may thus be a useful data-driven complement to the rule extraction procedure and other commonsense reasoning strategies proposed for our formalization of the conceptual spaces framework (cf. Sections 4.3 and 4.6.2).

Moreover, LTNs do not only provide an embedding of entities and classes, but they are also able to enforce the validity of general rules, which may reduce the required number of training examples. This makes them especially attractive for bridging the conceptual and the symbolic layer, since they can harness the whole expressiveness of formal ontologies and other symbolic languages in order to guide the machine learning process. This is also related to embodied and enactivist approaches to cognition [145], which assume that top-down information strongly influences bottom-up perception and conceptualization of the environment. One may furthermore speculate, that the enforcement of general logical rules can help to prevent catastrophic interference, where continued learning causes a neural network to forget previously learned knowledge [285].

Incorporating top-down information from the symbolic layer

When viewed from the perspective of cognitive science, logic tensor networks can of course not be labeled as a cognitively plausible learning mechanism: They rely on batch-processing large amounts of (typically labeled) data with gradient descent. One can of course argue, that LTNs are not used to model the human concept acquisition process itself, but rather to take a shortcut to the resulting concept inventory. However, it would certainly also be interesting to extend LTNs in such a way that they can work in an incremental way.

Cognitive plausibility

In the following, we will take a look at different membership functions from the conceptual spaces literature and discuss their applicability in logic tensor networks. We will broadly distinguish *partitional membership functions* (which create a partition of the conceptual space) from *non-partitional membership functions* (which allow conceptual regions to overlap and regions in the conceptual space to be "unpopulated"). Since the parameters of the membership functions are optimized through gradient descent, we will especially focus on their derivatives, an approach also followed by the aforementioned study by van Krieken et al. [415], who analyzed fuzzy logical operators in the context of neural networks. For illustration purposes, we will consider a one-dimensional conceptual space with two concepts C_1 and C_2 as well as three data points x_1, x_2, x_3 , which are supposed to belong to C_1 , but not to C_2 .

Analyzing possible membership functions

Let us first consider membership functions, which partition the underlying conceptual space, i.e., which aim to assign each point to exactly one concept. We start with Gärdenfors' approach of identifying concepts with a prototypical point and creating a *Voronoi tessellation*

Voronoi tessellations

of the space [179, Section 3.9] (cf. Sections 1.2.2 and 7.1.1): One starts from a set of prototypical points p_1, \dots, p_K for the K concepts under consideration. Each point x in the conceptual space is then assigned to its closest prototype p_k based on the distances $d(x, p_k)$. As Gärdenfors [179, Section 3.9] argues, a Voronoi tessellation based on the Euclidean metric partitions the overall space into convex regions.

Crisp Voronoi tessellations provide no gradient

Since the Voronoi tessellation gives us a partitioning of the overall space, the membership function of each concept C_i is constant almost everywhere and undefined on the border line to a neighboring conceptual region (see Figure 7.5a). Therefore, the derivative of this membership function with respect to any variable is either zero or undefined, which is highly problematic for gradient descent. For example, consider the point x_3 in Figure 7.5, which is currently misclassified as belonging to C_2 instead of C_1 . In gradient-based optimization, the prototypes p_1 and p_2 are updated by computing the derivative $\frac{d\mu_k(x_3)}{dp_k}$, and then by slightly increasing or decreasing the value of p_k , depending on the sign of the derivative and whether we want to increase or decrease $\mu_k(x_3)$ (cf. Section 5.2.2). In the case of Figure 7.5a, both derivatives are zero – small changes to p_1 and p_2 do not result in any changes to $\mu_k(x_3)$. Thus, gradient descent is incapable of making any update to the prototypes.

Classification rule of Voronoi tessellations

In order to make gradient-based learning possible, we need a soft version of the Voronoi approach. We can express the classification decision of the Voronoi tessellation as follows (where $c > 0$ is a sensitivity parameter):

$$k = \arg \min_{k'} d(x, p_{k'}) = \arg \max_{k'} (-c \cdot d(x, p_{k'}))$$

Using the softmax function

Instead of the crisp arg max function (which results in flat membership values), we can now apply the *softmax* function, which is commonly used in neural networks to provide an output probability distribution over a set of mutually exclusive classes (cf. Section 6.2.1):

$$\text{softmax}(\vec{z})_k = \frac{e^{z_k}}{\sum_{k'} e^{z_{k'}}$$

Soft Voronoi tessellation

Here, $\text{softmax}(\vec{z})_k$ gives the probability for class k , given a vector \vec{z} of raw confidence values. If we combine this with the Voronoi tessellation approach, we obtain a *soft Voronoi tessellation* with the following membership function:

$$\mu_k(x) = \text{softmax}(-c \cdot d(x, p))_k = \frac{e^{-c \cdot d(x, p_k)}}{\sum_{k'} e^{-c \cdot d(x, p_{k'})}$$

Membership as normalized similarity

As we can see, the numerator reflects the semantic similarity of x and p_k (cf. Section 2.1.1), while the denominator is the sum over all similarities to all prototypes. The resulting membership function can thus also be interpreted as a normalized version of semantic similarity. Figure 7.5b illustrates this membership function: We now have a continuous transition from high membership values to low

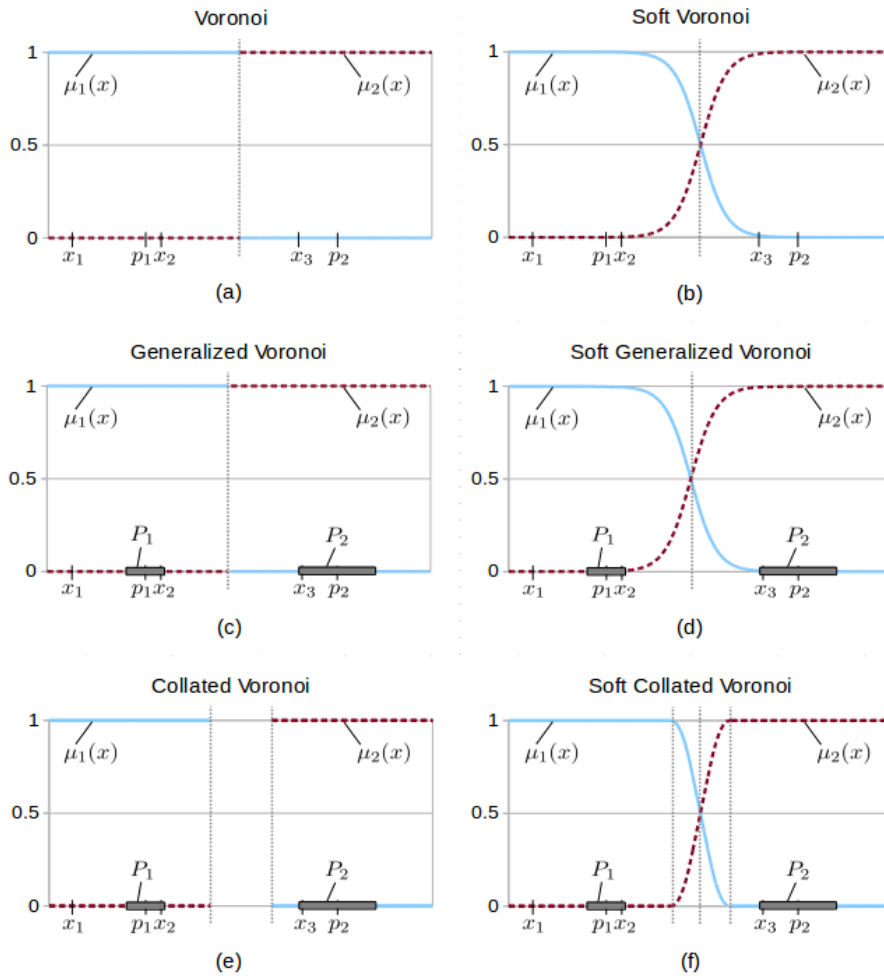


Figure 7.5: Different partitional membership functions in a one-dimensional conceptual space.

membership values. Moreover, the derivative of this membership function is defined on the whole conceptual space and nonzero in all cases - even points such as x_1 have a very small, but nonzero derivative.

We should highlight at this point, that we assume, that the same sensitivity parameter c is used for all concepts. If we allow different values $c_1 \neq c_2$, we can control the size of the respective conceptual regions with smaller values of c leading to larger regions. However, these different values may cause some unintended effects. For instance, Figure 7.6 illustrates a case, where $c_1 \gg c_2$, which causes the membership function $\mu_2(x)$ to be no longer convex.

Considering the sensitivity parameter

Generalized Voronoi tessellations [179, Section 4.9] allow to encode differently sized conceptual regions by considering prototypical regions P_k instead of prototypical points p_k . These prototypical regions are usually represented as disks with a central point p_k and a radius r_k . Based on these prototypical regions, one can now generate a generalized Voronoi tessellation by assigning each point x in the conceptual space to the concept, whose prototypical region is closest. In

Generalized Voronoi tessellations

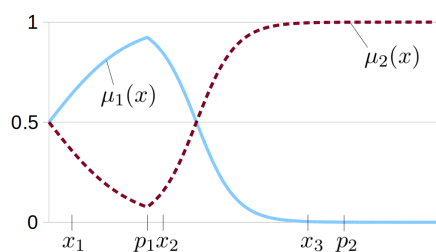


Figure 7.6: Unintended results for the soft Voronoi tessellation and different sensitivity parameters $c_1 \gg c_2$.

the case of disks, this corresponds to finding the concept C_k for which $d(x, P_k) = \max(0, d(x, p_k) - r_k)$ is smallest. Concepts with larger prototypical regions (as reflected through a larger value of r_k) thus result in larger conceptual regions in the generalized Voronoi tessellation (see Figure 7.5c). Again, by using the *softmax* instead of the *argmax* function, this can be generalized to a soft notion of concept membership (see Figure 7.5d).

Collated Voronoi diagrams

Also Douven et al. [136] consider prototypical regions instead of prototypical points. However, they create all possible Voronoi diagrams by picking a single point $p_k \in P_k$ for all prototypical regions P_k . These individual Voronoi tessellations are then aggregated into a so-called *collated Voronoi diagram* (cf. Sections 2.4.1 and 7.1.2): A point x is assigned to concept C_k if and only if it has been assigned to C_k in *all* individual Voronoi diagrams. Douven et al. identify borderline cases as points x that belong to different conceptual regions for different Voronoi diagrams. These borderline points are not assigned to any concept and represent vagueness in concept boundaries. In Figure 7.5e, we again note that the derivative is zero within the conceptual regions and undefined in the border area.

Soft collated Voronoi tessellations

Decock and Douven [117] extended the work of Douven et al. [136] by providing a degree of membership for borderline cases. They define the membership of a point x to a concept C_k as the fraction of individual Voronoi diagrams for which x belongs to the conceptual region of C_k . Decock and Douven note, that if the prototypical regions P_k have an infinite number of points, then the membership function is s-shaped (cf. Figure 7.5f). However, we can observe that the membership function is flat for large parts of the conceptual space, namely, for all non-borderline points. This is again highly problematic for gradient descent.

Best partitional approach: soft generalized Voronoi tessellation

Overall, it thus seems that the best partitional approach for defining membership functions in the context of LTNs is given by the soft generalized Voronoi tessellations: They provide a non-zero derivative for all points in the similarity space and are able to represent differently sized conceptual regions through the radius of their prototypical regions.

Shortcomings of partitional approaches

The usage of Voronoi tessellations for conceptual spaces has not been without challenge in the literature (cf. Sections 1.2.2 and 2.4.1). For

instance, Lewis and Lawry [253] argue, that partitioning the conceptual space may be adequate for individual domains such as COLOR, but that it is not suitable for a combination of multiple domains. It seems implausible, that every single point in a high-dimensional space has to be assigned to exactly one category: On the one hand, some regions in the overall conceptual space may not be covered by any existing concept (see also [395] and [453]). Points in such regions should be recognized as outliers or members of a novel, previously unknown category. On the other hand, conceptual regions may also overlap, for instance in order to represent conceptual hierarchies such as $\tilde{S}_{\text{SKY BLUE}} \subseteq \tilde{S}_{\text{BLUE}}$.

Lewis and Lawry [253] have also made a general proposal for non-partitional membership functions (cf. Sections 2.4.2 and 2.5.3): A point x in the conceptual space is said to belong to concept C_k , if its distance to the prototypical region P_k is not greater than a threshold distance ϵ_k . Lewis and Lawry assume, that this threshold ϵ_k is not known, but that a probability distribution δ_k over its possible values is available. The fuzzy degree of membership of a point x to a concept C_k is then given by the probability of $d(x, P_k)$ being smaller than ϵ_k :

$$\mu_k(x) = \mathbb{P}_{\delta_k}(d(x, P_k) \leq \epsilon_k) = \int_{d(x, P_k)}^{\infty} \delta_k(\epsilon_k) d\epsilon_k$$

Lewis and Lawry are open to different forms for the probability distribution δ_k . If we use $\delta_k(\epsilon_k) = c_k \cdot e^{-c_k \cdot \epsilon_k}$, then concept membership reflects similarity to the prototypical region (cf. Section 2.5.3):

$$\begin{aligned} \mu_k(x) &= \int_{d(x, P_k)}^{\infty} c_k \cdot e^{-c_k \cdot \epsilon_k} d\epsilon_k = \left[-e^{-c_k \cdot \epsilon_k} \right]_{\epsilon_k=d(x, P_k)}^{\epsilon_k \rightarrow \infty} \\ &= 0 - \left(-e^{-c_k \cdot d(x, P_k)} \right) = e^{-c_k \cdot d(x, P_k)} \end{aligned}$$

The shape of the resulting similarity function for $P_k = \{p_k\}$ (i.e., prototypical points) is illustrated in Figure 7.7a. As we can see, all points in the similarity space receive a non-zero membership value. Moreover, the derivative of the membership function is defined for all points except for the prototypes p_1 and p_2 . In practical applications, this theoretical shortcoming can be overcome by defining the derivative in these points to equal zero. Furthermore, we are able to control the size of the conceptual regions by choosing different sensitivity parameters $c_1 \neq c_2$ for the two concepts without any undesired side effects.

However, we can also note that the derivative of the membership function is proportional to the membership value itself: The largest derivatives are observed for the points with the highest membership in the concept. Since gradient descent algorithms typically take into account not only the direction, but also the magnitude of the gradient (cf. Section 5.2.2), this can lead to undesired effects. Consider for instance the point x_2 in Figure 7.7a, which has a fairly high membership to C_1 . The derivative $\frac{d\mu_1(x_2)}{dp_1}$ is quite large, and will thus cause the gradient descent algorithm to increase p_1 by a large amount. In the resulting configuration, $\mu_1(x_2)$ may, however, be smaller than before, since p_1 may

A general framework for non-partitional membership functions

Similarity-based membership

Non-partitional membership with an exponential decay

Shortcomings of an exponential decay

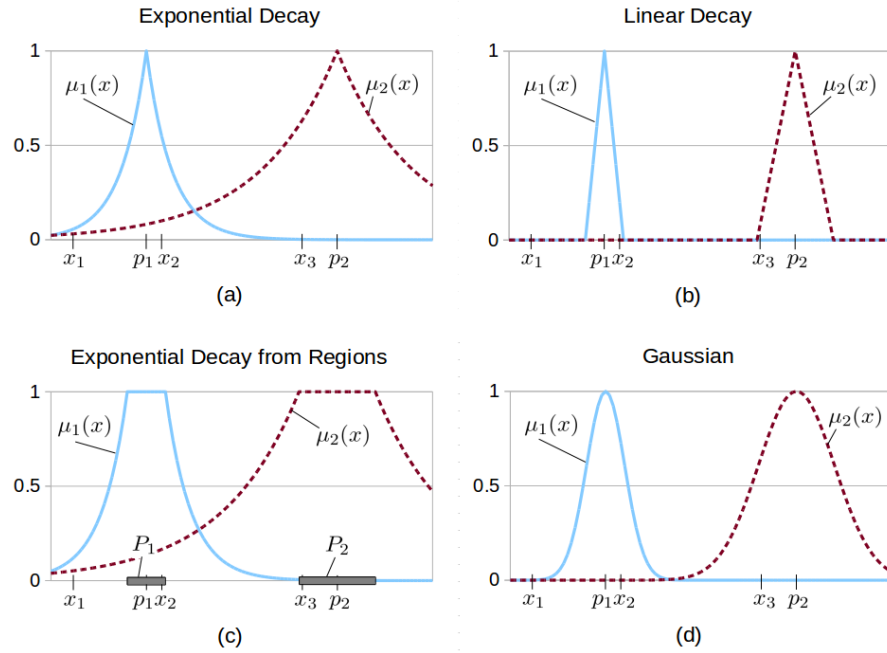


Figure 7.7: Non-partitional membership functions in a one-dimensional conceptual space.

have moved considerably past x_2 . This seems to be a major shortcoming of this similarity-based approach to concept membership.

Considering a linear decay

The examples by Lewis and Lawry [253] often make use of uniform distributions $\delta_k(\epsilon_k) = Uniform(0, r_k)$. As we can see in Figure 7.7b, the membership curve has in this case a triangular shape, and its derivative is therefore constant for all points with a partial membership. However, concept membership and thus also its derivative are zero for most parts of the similarity space.

Considering our own formalization

These considerations can of course also be generalized to prototypical regions P_k , which subsumes our own formalization of the conceptual spaces framework from Part I of this dissertation: There exists a well-defined region with full membership, which in our case is based on the union of axis-aligned cuboids. Membership is then defined as similarity to this prototypical region.

Shortcomings of our membership function

In Figure 7.7c, we can see two problems with this approach: On the one hand, we again have the problem of large derivatives for large membership values as already discussed for Figure 7.7a. On the other hand, the membership function is constant for all points in the prototypical region, hence, the derivative is zero. If an observation such as x_3 is confidently misclassified as belonging to C_2 , then gradient descent is not able to move P_2 away from x_3 .

Artificial derivatives in prototypical regions

The problem of a zero derivative could be circumvented as follows: We define a new membership function $\mu'_k(x) = (1 - \epsilon) \cdot \mu_k(x)$ for some small $\epsilon > 0$. Furthermore, we identify the central point $p_k \in P_k$. The membership value for $x \in P_k$ is then increased based on its distance to p_k , such that $\mu'_k(p_k) = 1$ and that $\mu'_k(x) = 1 - \epsilon$ for points on the border

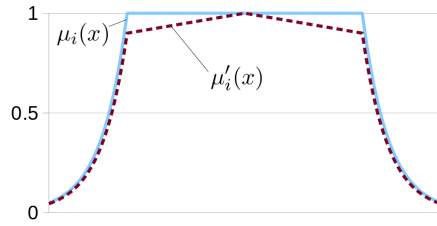


Figure 7.8: Workaround for zero gradient inside prototypical regions.

of P_k . This provides a small slope for the membership function inside the prototypical region and thus a nonzero derivative (cf. Figure 7.8). However, it is unclear, whether this workaround is useful in practice.

Despite these shortcomings, there are, however, reasonably strong arguments for using a membership function like the one proposed in our formalization: Firstly, by using a union of axis-aligned cuboids, our formalization is able to represent correlations between domains. This is an important aspect of human conceptualization [65, 288], which is not captured by any of the aforementioned approaches. Secondly, one can apply a variety of operations defined in Chapters 3 and 4 in order to reason on the learned concepts. For instance, relations such as conceptual similarity and conceptual betweenness are not defined in LTNs, but they become immediately available with the use of our proposed formalization of concepts. Thirdly, logical formulas in LTNs always have to be evaluated on a set of data points, which requires, that one keeps all examples in memory, and which can be quite time consuming. Our formalization on the other hand provides closed formulas for computing the validity of such logical formulas – the original data points are not needed any more and the computation can potentially be faster.

Advantages of our membership function

However, one needs to keep in mind that e.g., the intersection of concepts defined in Section 3.1.2 makes use of a repair mechanism in order to ensure star-shapedness of the result, whereas LTNs work with "raw" fuzzy logic, which does not use such a repair mechanism. Moreover, the operations defined in our formalization are based on the minimum norm, while LTNs are commonly used with the product norm. Therefore, the numeric results of the computations might differ.

Caveats in the context of LTNs

Motivated by the problem of large gradients for large membership values, we also consider multivariate Gaussian functions, whose membership value can be defined as follows with a symmetric, positive semi-definite matrix Σ (cf. Section 2.2.2):

Gaussian membership functions

$$\mu_k(x) = e^{-\frac{1}{2}(x-p_k)^T \Sigma^{-1}(x-p_k)}$$

Figure 7.7d illustrates the usage of such Gaussian functions in our one-dimensional similarity space.⁶ As one can see, this type of membership

Advantages of Gaussians

⁶ We can model this with $\delta_k(\epsilon_k) = \frac{\epsilon_k}{\sigma_k^2} \cdot e^{-\frac{\epsilon_k^2}{2\sigma_k^2}}$ in the one-dimensional case using the approach by Lewis and Lawry [253].

function does not suffer from the gradient size problem as identified in Figure 7.7a: The derivative is small both for points with a very low and for points with a very high membership. It is largest for points with an intermediate level of membership, i.e., points that are currently treated as borderline members. Moreover, multivariate Gaussian functions are able to encode correlations between dimensions as well as different distribution widths through their covariance matrix Σ .

*Theoretical argument
against Gaussians*

However, the usage of Gaussians in the context of conceptual spaces is somewhat unsatisfactory from a theoretical standpoint (cf. Section 2.2.2): The notion of similarity is not based on the Euclidean distance $d_E(x, p_k) = \sqrt{\sum_d (x_d - p_{kd})^2}$, but on the squared Mahalanobis distance $d_M(x, p_k) = \sqrt{(x - p_k)^T \Sigma^{-1} (x - p_k)}$. Applying the Mahalanobis distance corresponds to transforming the similarity space based on the covariance matrix, and then computing the Euclidean metric in the transformed space. This implicit transformation of the similarity space would in our opinion, however, cause a major modification of the original framework. Nevertheless, the simplicity and computational attractiveness of multivariate Gaussians make them an interesting candidate for experimental investigations (cf. the work by Bouraoui and Schockaert [76] discussed in Section 7.1.3), such that one should not hastily dismiss them.

Outlook

Our proposal has so far been only a theoretical one. In order to evaluate its actual merit, practical experiments need to be conducted. Ideally, these experiments should consider all membership functions discussed in this section in order to confirm or refute our theoretical analyses. There are several datasets, which can serve as test beds for a first study, including the conceptual spaces extracted by Banaee et al. [27] (cf. Sections 1.2.5 and 5.3.2) and Derrac and Schockaert [123] (cf. Sections 1.2.5 and 6.3.2), as well as the robotics dataset by Spranger et al. [386] (which will be briefly introduced in Section 7.4.3). Since the strength of LTNs stems from their ability to incorporate top-down rules to compensate for scarce training data, especially the movie spaces from Derrac and Schockaert [123] are relevant: Each movie is annotated with its genres, a set of plot keywords, at its age restriction. Using techniques such as the apriori algorithm [8], one can extract rules from the co-occurrence statistics of the labels and then simulate few shot learning by showing only a small part of the available examples, but providing the general rules as additional constraints to the system. After such initial experiments, studies with more complex knowledge bases (such as formal ontologies) are needed in order to ensure that all relevant pieces of symbolic information (especially relations of varying complexity) can be adequately encoded by the proposed approach.

7.3 LEARNING CONCEPTS FROM UNLABELED DATA

Most classical machine learning algorithms as introduced in Chapter 5 require a large dataset of examples, which are annotated with their respective class labels. This assumption can, however, be quite limiting in practical applications, where labeled data is scarce and costly to acquire. Unlabeled data, on the other hand, is often widely available and hence easy to obtain. Also from a cognitive point of view, supervised learning is limited to environments, where explicit feedback is abundant. However, this is rarely the case in practice. Human children are nevertheless able to form concepts based on only a small amount of labeled examples [87], as already noted in the context of few shot learning (cf. Section 7.2). Thus, learning from unlabeled data is interesting from both an engineering perspective and a cognitive point of view.

*Motivation for
unsupervised
learning*

In this section, we put our focus on *clustering algorithms*, which group data points into different clusters based on their similarity [58, 206]. Essentially, clustering algorithms try to find regions of high density in the overall feature space, which are separated from each other by low-density regions. Clustering algorithms can be naturally applied to the conceptual spaces framework, which is explicitly built upon a spatial representation of similarities. Finding clusters in a conceptual space then corresponds to identifying well-separated conceptual regions. Once the conceptual regions have been identified, one can attach labels to them in order to link them to the symbolic layer.

*Clustering
algorithms*

In Section 7.3.1, we provide an overview of classical clustering techniques such as hierarchical clustering, expectation maximization, or k -means. Most of these approaches process the whole dataset at once and require, that the number of clusters is specified a priori. However, an incremental approach, which processes instances one by one, is much more plausible from a cognitive point of view. Also a method for automatically adjusting the number of concepts to be generated seems desirable. In the remainder of this section, we therefore present three concrete algorithms for *incremental concept formation* and discuss, to what extent they can be readily used in conceptual spaces. This relates to our earlier discussions from Sections 3.6.1 and 4.6.1 in the context of our formalization of the conceptual spaces framework. In Section 7.3.2, we introduce COBWEB as a classical concept formation algorithm, which uses some cognitive inspirations, but is mainly based on an engineering perspective. We then discuss adaptive resonance theory (ART) as a neural model of human information processing in Section 7.3.3. Finally, in Section 7.3.4 we introduce SUSTAIN, which is a psychological model of category learning.

7.3.1 Classical Clustering Techniques

Clustering can be described as the task of finding meaningful groups in a collection of objects based on their pairwise similarities, but without

*Clustering and
conceptual spaces*

having access to any label information. It can thus be regarded as an unsupervised machine learning technique. Intuitively, the objects within one cluster should be more similar to one another than to objects from other clusters [58, 206]. This notion of a cluster is closely related to the well-formedness criterion proposed by Douven and Gärdenfors [137] for natural concepts in conceptual spaces (cf. Section 1.2.1). In most clustering contexts, the objects under consideration are represented as points in multidimensional feature spaces, which may involve both continuous and categorical features. Clustering can thus serve as an unsupervised way for discovering conceptual regions in a conceptual space. Please recall from Section 5.3.2, that clustering can also be interpreted as a form of dimensionality reduction, which leads to a sparse representation of the original data by representing each data point with its cluster ID. This dimensionality reduction aspect makes clustering algorithms also a quite useful tool for data exploration.

Similarity and distance

Since objects shall be grouped based on their similarity, and since they are typically represented as points in a feature space, one usually employs a distance metric for quantifying dissimilarity. The most popular choice for continuous features is the Euclidean distance, which works well for compact and isolated clusters, but requires, that all features have been normalized – this normalization requirement ensures, that none of the features becomes dominant [206]. However, also other distance metrics are possible, such as the squared Mahalanobis distance for correlated features, the Hausdorff distance for sets of points, or the Cosine distance for feature vectors with a unit norm [58, 206]. However, difficulties may arise, if the overall feature space contains different feature types (e.g., both continuous and categorical features) – in this case, the overall distance function needs to be carefully engineered in order to reflect the overall similarity of the objects well [206]. In the context of conceptual spaces, a combination of the Euclidean and the Manhattan metric as introduced in Sections 2.1.1 and 2.3.1 seems to be a reasonable choice. Please recall from Section 5.2.5, that similar considerations for appropriate distance functions also apply to the k nearest neighbors classifier, which bases its classification decisions also directly on the pairwise similarity of examples.

Two main types of clustering algorithms

In general, one can distinguish two broad classes of clustering techniques [58, 206]: *Hierarchical clustering algorithms* create a hierarchy of clusters, where larger clusters are split up into smaller sub-clusters. *Partitional clustering algorithms*, on the other hand, focus on creating a single partitioning of the dataset into mutually exclusive clusters. We will now consider some examples for both types of approaches.

Hierarchical clustering algorithms

Let us first focus on *hierarchical clustering approaches*. The hierarchy obtained by these algorithms is usually represented as a tree (often called dendrogram), whose leaf nodes correspond to the individual data points, and whose internal nodes represent the clusters. This dendrogram can be cut at any desired level of abstraction to obtain a partitioning of the data points. Hierarchical clustering algorithms are relatively flexible with respect to the desired level of granularity and

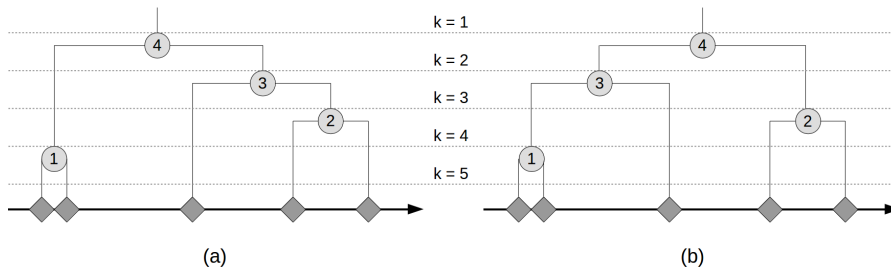


Figure 7.9: Difference between agglomerative hierarchical clustering using the single link method (a) and the complete link method (b).

the type of distance metric used [58]. However, they tend to be computationally expensive [206] and typically do not revisit intermediate clusters, once they have been created [58].

In general, the desired hierarchy of clusters can be obtained either in an agglomerative or in a divisive way [58, 206]: In *agglomerative hierarchical clustering*, one follows a bottom-up procedure by initially placing each data point into its own cluster and then recursively merging the two clusters, which are closest to each other. Thus, the dendrogram is constructed from the leaf nodes upwards. *Divisive hierarchical clustering* on the other hand is a top-down approach, where one starts with a single cluster containing all data points and recursively splits the existing clusters into smaller sub-clusters. In this approach, the dendrogram is constructed from the root node downwards.

For both agglomerative and divisive clustering, it is necessary to measure the distance between pairs of clusters. This information is used to decide which clusters to merge (agglomerative clustering) or how to split a given cluster into sub-clusters (divisive clustering). This distance measure for clusters is often called *linkage metric* [58]. The most common linkage metrics are *single link* and *complete link*, whose cluster-based distances d_{SL} and d_{CL} are defined as follows:

$$d_{SL}(A, B) = \min_{a \in A, b \in B} d(a, b) \quad d_{CL}(A, B) = \max_{a \in A, b \in B} d(a, b)$$

The complete link method tends to produce compact clusters of a convex shape. It is less flexible than single link, which can, however, suffer from producing very elongated clusters [206]. Other potential linkage metrics can be defined as the average over all point-based distances, or as the distance between the cluster centroids [58].

Figure 7.9 illustrates the difference between single link and complete link approaches in an agglomerative setting. We use the same set of example inputs in a one-dimensional feature space in both cases. Figure 7.9 illustrates the resulting dendrograms, where the numbers of the internal nodes denote the order in which they have been created. As we can see, the single link (Figure 7.9a) method merges the data point in the middle with the right cluster, since the distance to the closest cluster member is slightly smaller than for the left cluster. The complete link method (Figure 7.9b), however, selects the left cluster for merging, since the distance to the furthest data point from this cluster is slightly

Agglomerative and divisive clustering

Linkage metrics

Properties of single link and complete link

Linkage metrics and resulting dendrograms

smaller than for the right cluster. In addition to the dendrogram, Figure 7.9 also illustrates different levels in the hierarchy, for which the given number k of clusters can be obtained. The results of the two different linkage metrics are identical for $k \in \{1, 3, 4, 5\}$, but differ for $k = 2$.

Partitional clustering algorithms

Let us now turn to *partitional clustering algorithms*, which create a single partitioning of the data into mutually exclusive clusters. They are often based on optimizing a given local or global criterion in a greedy fashion [206]. Partitional clustering tends to work better for larger datasets, where creating a dendrogram is computationally infeasible. However, one often needs to specify the number of clusters in advance [206]. One can further divide partitional algorithms into *partition relocation methods* and *density-based partitioning methods* [58]. The former group of approaches iteratively relocates the data points between the clusters and tends to create convexly-shaped clusters. The latter group of methods, on the other hand, tries to discover dense connected components of data and is able to form also clusters of irregular shape.

Expectation maximization

We will begin our discussions with the *expectation maximization (EM) algorithm* [119] as a probabilistic variant of a *partition relocation method*. The underlying assumption of such probabilistic models is that each data point is drawn from one of k underlying probability distributions – for each data point, one first randomly selects one of the distributions and then randomly samples a point from that distribution [341, Section 20.3]. In this context, a cluster corresponds to one of these probability distributions, whose parameters need to be estimated [58]. In many cases, multivariate Gaussian distributions (cf. Section 2.2.2) are used to model the individual components [206]. Interpreting the data as being generated from a mixture of distributions is somewhat problematic, since we neither know which data point was sampled from which distribution, nor how the different distributions look like (i.e., which values their parameters have) [341, Section 20.3].

Two steps of the algorithm

Expectation maximization iteratively estimates both the cluster assignment and the parameters of the corresponding distributions. Here, the cluster assignment is treated as a hidden indicator variable. Essentially, the EM algorithm consists of two steps, which are repeated until convergence [341, Section 20.3]: In the *expectation step*, one uses the current hypothesis about the distribution's parameters to infer the expected value of the hidden indicator variables, which denote the assignment of data points to clusters. In the *maximization step*, one then uses the current expected values of the hidden indicator variables to infer the parameters of the probability distributions by maximizing the log likelihood of the data points (cf. Section 5.1.3). One can show, that under some mild constraints the EM algorithm always converges to a local optimum [295, Section 6.12].

k-means as simplified expectation maximization

The popular *k-means* algorithm can be interpreted as a constrained variant of expectation maximization [295, Section 6.12]: If we assume Gaussian distributions with the same prior probability and the same variance, we only need to estimate their means. This can be implemented

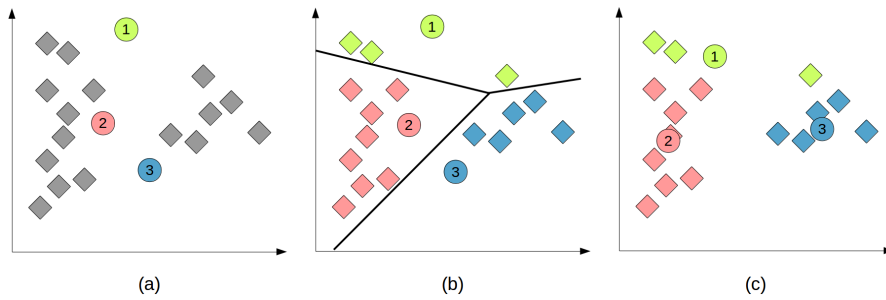


Figure 7.10: Illustration of the k -means algorithm: Random initialization of cluster centroids (a), expectation step (b), and maximization step (c).

by representing each cluster only by its centroid. The expectation step then corresponds to assigning each data point to the cluster with the closest centroid, while the maximization step is realized by recomputing the cluster centroids based on the updated assignment.

From a different point of view, one can characterize the k -means algorithm as minimizing the mean squared error between the data points and the corresponding cluster centroids [206]. The k -means algorithm tends to result in spherical clusters of identical size and works thus only well, if the clusters in the data fulfill this assumption [206]. Moreover, the resulting partitioning depends heavily on the choice of the initial centroids and tends to be sensitive to outliers [58]. Nevertheless, k -means is one of the most popular clustering algorithms due to its simplicity.

Figure 7.10 illustrates the first iteration of the k -means algorithm in a two-dimensional feature space. As one can see, the expectation step (where a reassignment of data points to clusters is made, Figure 7.10b) can be interpreted as a Voronoi tessellation of the feature space based on the cluster centroids. The resulting partitioning of the space is then used to re-estimate the cluster centroids in the maximization step in Figure 7.10c.

One can thus relate the k -means algorithm to the concept learning procedure proposed by Gärdenfors (cf. Section 7.1.1): Instead of using a small number of labeled observations for inferring prototypes and the Voronoi tessellation based on them, we start with a first guess for the prototypes (cf. Figure 7.10a), label them according to the resulting Voronoi tessellation, and then iteratively improve the prototypes until the resulting tessellation of the space becomes stable. The k -means algorithm can thus be interpreted as an unsupervised extension of the supervised learning procedure proposed by Gärdenfors. It has for instance been used by Chella et al. [96] in their work on anchoring for robotics in the context of conceptual spaces (cf. Section 1.2.4). Also the experiments by Derrac and Schockaert [123] with respect to conceptual betweenness (cf. Sections 4.5.1 and 7.1.3) used k -means to obtain representative exemplar points for a given concept based on the complete set of examples.

*Properties of
 k -means*

Example

*k -means and
conceptual spaces*

*Extensions of
k-means*

Of course, there exist several variants and extensions of k -means clustering. For instance, in k -medoid clustering, each cluster is not represented by a centroid, but rather by the most representative of its original data points [58]. Other extensions also allow the algorithm to split clusters with large variance or to merge clusters, whose centroids have a very small distance [206]. Finally, also fuzzified versions of k -means exist (such as the fuzzy c -means algorithm), where the assignment to clusters is represented as a continuous degree of membership rather than a binary variable. A critical issue for such fuzzy extensions consists, however, in defining the membership function [206].

*Density-based
approaches and
DBSCAN*

A second important class of partitioning methods are *density-based partitioning algorithms*. Here, clusters are interpreted as connected regions in the feature space [58]. In order to operationalize the notion of density in the feature space, one often resorts to the neighbors of a given data point, which are closer than a given threshold [206]. Clusters can then be "grown" by starting at any given data point and transitively including all its neighbors [58]. The most well-known example of density-based algorithms is DBSCAN [148]. Its two main parameters are ϵ , which is the distance threshold for determining neighborhood, and $minPts$, which is the minimal number of neighbors a data point needs to have in order to be counted as a *core point*. Clusters in DBSCAN are then identified based on a given core point and the transitive hull of its neighborhood relationship. DBSCAN and density-based algorithms in general are not limited to any particular cluster shape, but require a correct tuning of their thresholds and may produce results, which lack interpretability [58].

*Other clustering
approaches*

There exists a large variety of additional clustering approaches, which we are not able to introduce in more detail [58, 206]: *Graph-based algorithms* construct a neighborhood graph with data points as nodes and neighborhood relations as edges, and then identify clusters as densely connected subgraphs. *Grid-based methods*, on the other hand, partition the feature space based on the Cartesian product of feature subranges and can thus be linked to decision tree learning (cf. Section 5.2.4). *ANN-based methods* such as Kohonen's self-organizing maps [225] try to represent the given data with a lower-dimensional grid, that respects neighborhood relations, and are thus tightly related to both metric learning (cf. Section 5.3.3) and representation learning (cf. Section 6.1.1). Finally, *evolutionary algorithms* maintain a population of solution candidates, which are iteratively changed and combined in order to find more promising solutions.

*Clustering and
background
knowledge*

While basic clustering algorithms have been in practical use for decades, there still remain many open challenges. As in any machine learning task, finding a low-dimensional feature space of high quality is of crucial importance to success in clustering (cf. Section 5.1.4). This makes it often necessary to define features based on domain knowledge and to apply dimensionality reduction techniques such as

feature selection or feature extraction (cf. Sections 5.3.1 and 5.3.2) [206]. Moreover, the feature space needs to be equipped with a useful distance metric, which may be informed by domain knowledge [58]. Another way of incorporating domain knowledge into the clustering process is given by constrained clustering, where additional constraints with respect to pairs of data points (such as "must be in the same cluster" or "must be in different clusters") are taken into account [33].

When being applied in practice on unlabeled data, one also needs to find a good way of evaluating the quality of the resulting clusters [206]. Depending on the clustering algorithm being used, it may also be necessary to explicitly detect and remove outliers [58]. Especially the application to large datasets is difficult and urges for "divide and conquer" approaches (where subsets of data are processed individually by the algorithm and the results are merged afterwards) or incremental clustering algorithms (which process the data points one after another) [206]. Finally, it may often be useful to represent the resulting clusters not as sets of data points, but by using some summary information, such as the cluster centroid or the typical ranges of feature values [206].

Further considerations

Concept formation (as already briefly introduced in Sections 3.6.1 and 4.6.1) can be interpreted as a special type of clustering method [58]: It is an unsupervised learning process, since it does not require class labels to be given. Moreover, the number of classes is typically unknown a priori and the number of clusters is therefore adapted over time [164]. These clusters come with an intensional definition for the underlying concept – they are thus not represented as a collection of points, but based on some underlying parametric formalization [58]. Finally, most concept formation algorithms result in a hierarchical organization of the learned concepts [164].

Concept formation

Fisher [159] argues, that the concept formation problem consists of two interrelated search problems: On the one hand, the algorithm must solve the clustering problem by searching the hypothesis space of possible subsets of data points. On the other hand, the algorithm must at the same time search the hypothesis space of class descriptions for the generated clusters. As with any kind of search algorithm, one can distinguish different approaches based on their search direction (top-down versus bottom-up) and their overall search strategy (exhaustive versus heuristic). As Gennari et al. [164] note, concept formation typically takes place in a top-down manner, where one starts with the most general possible concept and refines it as necessary into sub-concepts.

Two search processes

Fisher [159] furthermore emphasizes, that concept formation algorithms are typically *incremental*, since real world environments provide observations in a sequential order. Processing this sequence of observations in an incremental way allows one to update the internal representation rapidly with each new observation, making it possible to react to changes in the environment. However, incremental processing puts hard constraints on the processing time for each update, making an exhaustive search strategy infeasible in practice. Gennari et al. [164] observe, that most concept formation algorithms follow instead a

Incremental processing

heuristic hill-climbing strategy, where only a single hypothesis is kept in memory. This hypothesis is incrementally modified in order to optimize an explicit or implicit target function. This is usually done by adapting the existing clusters or creating a new one [206]. Since the value of the target function is highly dependent on the current observation, the surface of this optimization function changes with every observation, making concept formation algorithms order-sensitive [164]. A final property related to this incremental processing is the integration of learning and performance [164]: The new observation is compared to the existing clusters, while modifying them at the same time. This differs from the predominant division of training and test phases in standard machine learning techniques (cf. Section 5.1.2).

Evaluating concept formation

Fisher [159] makes the following suggestions for evaluating concept formation algorithms: Although unsupervised in nature, concept formation algorithms can be in principle evaluated based on their classification performance. One can simply add the target class as an additional (nominal) feature and then classify novel instances by finding the most appropriate cluster and using the cluster's intensional description to predict the missing class feature. This procedure can also be applied to any other feature, making also *missing attribute prediction* a possible way of evaluation. Finally, incremental algorithms in general can be evaluated by considering the cost of processing a single observation and by analyzing the number of observations necessary for achieving a stable representation.

Concept formation and conceptual spaces

Concept formation is an important avenue of research when applying the conceptual spaces framework to artificial intelligence. After all, the goal is to find a cognitively plausible way of representing, learning, and applying conceptual knowledge in artificial agents. One can easily argue, that the key characteristics of concept formation approaches (unsupervised and incremental processing, as well as hierarchical and intensional representation) are cognitively more plausible than their alternatives: Since supervision feedback in the form of labels is very scarce in the real world, and since observations are made in a sequential manner, standard machine learning techniques, that rely on batch processing of large labeled datasets, are not very convincing from a cognitive point of view. Also the existence of a conceptual hierarchy with a reasonably well-defined basic level of concepts [298, Chapter 7] (cf. Section 1.1.2) makes a flat partitioning of the (conceptual) feature space less likely than a hierarchical structure. Finally, representing a concept by its intension (such as a prototype) is much more economical than storing all individual observations as exemplars. In the following, we will thus introduce three selected concept formation algorithms.

7.3.2 Hierarchical Concept Formation with COBWEB

General properties of COBWEB

COBWEB [159] is a cognitively inspired clustering method, which uses a hill-climbing search through the space of hierarchical classification schemes. It favors clusters, which maximize the information, that can be

predicted about their members. COBWEB uses a set of operators, which allow the algorithm to incrementally grow or shrink the conceptual hierarchy, enabling a bidirectional search through the hypothesis space. However, it only considers nominal features.

In COBWEB, the cluster hierarchy is stored as a tree, where each inner node corresponds to a cluster, and where each leaf node represents one instance. A cluster is described by three types of conditional probability distributions [164]: The *predictability* $\mathbb{P}(x_i = v_{ij}|C_k)$ represents the distribution of possible values v_{ij} of the feature x_i for members of this cluster C_k , while the *predictiveness* $\mathbb{P}(C_k|x_i = v_{ij})$ denotes, how indicative a given feature value is for the given category. Fisher [159] notes, that a predictability of $\mathbb{P}(x_i = v_{ij}|C_k) = 1$ indicates necessary feature values for category membership, while a predictiveness of $\mathbb{P}(C_k|x_i = v_{ij}) = 1$ represents a feature value, which is sufficient for determining category membership. Finally, the distribution $\mathbb{P}(C_k|C_{k'})$ denotes the relative frequency of observations falling under the category C_k , given that they belong to the parent category $C_{k'}$. As Berkhin [58] notes, this usage of conditional probabilities makes COBWEB's cluster representation similar to a naive Bayes classifier. The three aforementioned probability distributions are estimated by computing relative frequencies among the observations [159]. These co-occurrence counts can be updated in an incremental way, making it possible to adjust the probabilities with each new observation.

Fisher [159] notes, that the classification tree used in COBWEB is somewhat similar to the structure used in supervised decision trees (see Section 5.2.4). However, supervised decision trees check a single feature in an internal node and make a deterministic branching decision. The internal nodes of COBWEB, on the other hand, consider multiple attributes at once, taking into account their conditional probabilities based on the observed frequencies.

When incrementally constructing its tree of clusters, COBWEB tries to optimize a heuristic evaluation measure called *category utility*, which aims for a good trade-off between intra-class similarity and inter-class dissimilarity [159]. This measure is inspired by the basic level of categorization in human classification hierarchies (cf. Section 1.1.2). In COBWEB, intra-class similarity is modeled by the predictabilities $\mathbb{P}(x_i = v_{ij}|C_k)$ of values v_{ij} for the feature x_i within the category C_k , which can be interpreted as the degree, to which members of the same category share the same feature values. Inter-class dissimilarity, on the other hand, is modeled through the predictiveness $\mathbb{P}(C_k|x_i = v_{ij})$, which measures, how well different clusters can be distinguished based on the given feature values. In a first step, Fisher [159] combines these two components into an overall expression, weighting the individual

Representation of clusters with probability distributions

Relation to decision trees

Category utility

feature values v_{ij} by their respective frequency $\mathbb{P}(x_i = v_{ij})$ based on the overall number V_i of possible values for feature x_i :

$$\sum_{k=1}^K \sum_{i=1}^n \sum_{j=1}^{V_i} \mathbb{P}(x_i = v_{ij}) \cdot \mathbb{P}(C_k | x_i = v_{ij}) \cdot \mathbb{P}(x_i = v_{ij} | C_k)$$

A simple rewrite

Based on $\mathbb{P}(A) \cdot \mathbb{P}(B|A) = \mathbb{P}(A, B) = \mathbb{P}(B) \cdot \mathbb{P}(A|B)$, one can easily see, that $\mathbb{P}(x_i = v_{ij}) \cdot \mathbb{P}(C_k | x_i = v_{ij}) = \mathbb{P}(C_k) \mathbb{P}(x_i = v_{ij} | C_k)$. Hence, the formula from above can be rewritten as follows:

$$\sum_{k=1}^K \mathbb{P}(C_k) \sum_{i=1}^n \sum_{j=1}^{V_i} \mathbb{P}(x_i = v_{ij} | C_k)^2$$

Interpretation as feature value guessing

As Fisher [159] observes, $\sum_i \sum_j \mathbb{P}(x_i = v_{ij} | C_k)^2$ can be interpreted as the expected number of feature values, which can be guessed correctly for an arbitrary member of the category C_k , given that both the guessing strategy and the ground truth distribution are based on the observed frequencies. Category utility can now be defined as the *increase* in prediction performance, which arises from knowing the cluster assignment of the data point in comparison to using the overall frequencies of feature values across all clusters:

$$\begin{aligned} CU(\{C_1, \dots, C_K\}) \\ = \frac{1}{K} \sum_{k=1}^K \mathbb{P}(C_k) \left[\sum_{i=1}^n \sum_{j=1}^{V_i} \mathbb{P}(x_i = v_{ij} | C_k)^2 - \sum_{i=1}^n \sum_{j=1}^{V_i} \mathbb{P}(x_i = v_{ij})^2 \right] \end{aligned}$$

Interpretation of category utility

In other words, category utility quantifies, how much the induced category structure helps to predict feature values better than chance [164]. Fisher [159] furthermore points out, that if there exist systems of correlations for some feature values (as for instance indicated by [65, 66, 220], cf. Section 1.1.2), then forming clusters around these correlations is rewarded by category utility.

Control strategy of COBWEB

Algorithm 7.1 shows the overall control strategy of the COBWEB algorithm and Figure 7.11 provides an illustration for its most important operators. The function COBWEB is first called with the root node of the classification tree, meaning that the tree is traversed in a top-down manner. At each visited node, the internal counts are updated based on the novel instance in order to adapt the resulting conditional probabilities (line 1). If the current node is already a leaf node (lines 2–4), we can terminate our tree traversal. Otherwise, the new observation is tentatively put into each of the child nodes (lines 5–8), computing the resulting category utility of the overall classification tree. Then, one of four possibilities is chosen:

Insert: growing the tree

1. *Inserting a new child node (lines 9–12)*: The algorithm tentatively creates a new singleton cluster based on the observation and puts it into the classification tree as a child of the current node (line 9). If the category utility of the resulting classification tree is higher

Algorithm 7.1: Control strategy of the COBWEB algorithm [159].

Input: Observation O , classification tree T , concept C
Output: Updated classification tree T'

- 1 Update counts of C in T
- 2 **if** C is leaf node in T **then**
- 3 | **return** T
- 4 **end**
- 5 **foreach** Child node C_i of C **do**
- 6 | Create classification tree T_i by putting O into C_i and updating C_i 's counts
- 7 **end**
- 8 Select $i_{max} = \arg \max_i CU(T_i)$ and $i'_{max} = \arg \max_{i \neq i_{max}} CU(T_i)$
- 9 Create classification tree T_{insert} by inserting a new node C_{new} based on O as child of C into T
- 10 **if** $CU(T_{insert}) > CU(T_{i_{max}})$ **then**
- 11 | **return** T_{insert}
- 12 **end**
- 13 Create classification tree T_{merge} by merging $C_{i_{max}}$ and $C_{i'_{max}}$ into C_{new} , putting C_{new} as child of C , $C_{i_{max}}$ and $C_{i'_{max}}$ as children of C_{new} , and adding O to C_{new}
- 14 **if** $CU(T_{merge}) > CU(T_{i_{max}})$ **then**
- 15 | **return** $COBWEB(O, T_{merge}, C_{new})$
- 16 **end**
- 17 Create classification tree T_{split} by promoting all children C_j of $C_{i_{max}}$ to be children of C and by removing $C_{i_{max}}$
- 18 **if** $CU(T_{split}) > CU(T_{i_{max}})$ **then**
- 19 | **return** $COBWEB(O, T_{split}, C)$
- 20 **end**
- 21 **return** $COBWEB(O, T, C_{i_{max}})$

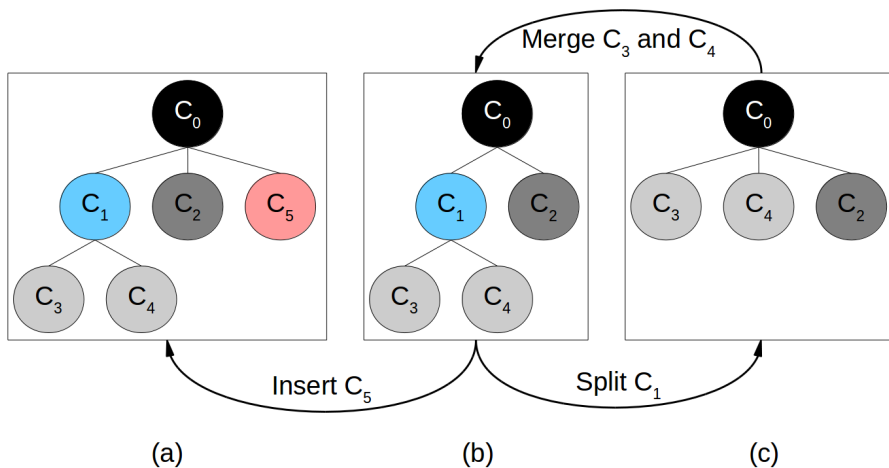


Figure 7.11: Illustration of the COBWEB operators. Splitting and merging can be used as approximately inverse operators (transition from (b) to (c) and backwards), while insertion increases the breadth of the tree.

than what can be achieved by putting the observation into any of the existing sub-clusters, this option is chosen (lines 10–12). Since COBWEB initially starts with a tree, which only consists of the root node, the insertion operator allows it to grow the tree by creating additional nodes. The overall number of clusters can thus emerge during training and does not need to be specified in advance. Insertion can be used to modify the tree in Figure 7.11b in order to obtain the tree in Figure 7.11a.

Merge: reducing breadth, increasing depth

2. *Merging two child nodes (lines 13–16)*: In order to reduce the breadth of the tree while increasing its depth, COBWEB also considers merging two of the child nodes. Only the two best-scoring child nodes are considered in order to reduce computational complexity. These two nodes are tentatively merged (reducing the number of children by one) and used as sub-nodes of the newly created merged node (line 13). If the category utility of the resulting tree is higher than the one of the best-scoring child, this operation is executed, and the observation is placed into the merged node (lines 14–16). Merging transforms the tree in Figure 7.11c into the tree in Figure 7.11b.

Split: reducing depth, increasing breadth

3. *Splitting a child node (lines 17–20)*: As an approximate inverse to the merging operation, COBWEB also considers splitting a child node and promoting its children to the next-higher level (line 17). This results in $c_1 + c_2 - 1$ child nodes, if the current node has c_1 children and the child node to be deleted has c_2 children. The observation is then tentatively put into the best among the c_2 promoted children. If the category utility of the resulting classification tree is higher than the one of the best-scoring child, the split operation is executed (lines 18–20). Splitting transforms the tree in Figure 7.11b into the tree in Figure 7.11c and thus reduces the tree's depth, while increasing its breadth.

No modification to the tree structure

4. *Putting the observation into the best-scoring child (line 21)*: As a default case, COBWEB simply puts the observation into the best-scoring child, if none of the other operations seem to be preferable. This corresponds to simply traversing the tree without any modifications to its structure.

Bidirectional traversal of the search space

The backtracking enabled through the approximately merge and split operators allows COBWEB to move bidirectionally through the search space, which reduces its dependence on the order, in which observations are presented [164].

CLASSIT: considering real-valued features

As mentioned above, COBWEB considers only nominal features. Gennari et al. [164] have proposed CLASSIT as an extension of COBWEB, which focuses on concept formation for physical objects, and which allows only real-valued features. Instead of storing counts for computing the conditional probabilities for each cluster, CLASSIT represents these probabilities as one normal distribution per feature. Since a

normal distribution can be specified by its mean and its variance, one thus only needs to incrementally keep track of the mean and the standard deviation of the respective observations. However, this change in representation, however, also requires a modification of category utility, since the count-based terms $\sum_j \mathbb{P}(x_i = v_{ij}|C_k)$ and $\sum_j \mathbb{P}(x_i = v_{ij})$ need to be replaced with appropriate expressions based on the aforementioned normal distributions. Gennari et al. note, that the probability of a particular feature value corresponds to the value of the probability density function at that value, and that a summation of square probabilities thus corresponds to the integral over the squared normal distribution. They furthermore observe that $\int \frac{1}{\sigma^2 2\pi} e^{-\left(\frac{x-\mu}{\sigma}\right)^2} dx = \frac{1}{2\sigma\sqrt{\pi}}$. Since $\frac{1}{2\sigma\sqrt{\pi}}$ is a constant factor, it can be ignored for our purposes, where we only use category utility for making comparisons. Gennari et al. thus arrive at the following formula for *generalized category utility*, where $C_{k'}$ refers to the parent node and σ_{ik}^2 is the variance of the i th feature in category C_k :

$$GCU(\{C_1, \dots, C_K\}) = \frac{1}{K} \cdot \sum_{k=1}^K \mathbb{P}(C_k) \left[\sum_{i=1}^n \frac{1}{\sigma_{ik}} - \sum_{i=1}^n \frac{1}{\sigma_{ik'}} \right]$$

A problem arises for singleton classes, for which the variance becomes zero. Gennari et al. solve this problem by introducing a hyperparameter called *acuity*, which defines the minimal possible value for σ . In addition to this modification of the representation scheme and the evaluation measure, Gennari et al. also allow the algorithm to terminate at an inner node of the tree, if the respective concept is similar enough to the given instance. This is controlled by a *cutoff* hyperparameter. While this cutoff parameter controls the depth of the tree, the aforementioned acuity controls its branching factor and therefore its breadth. Most importantly, CLASSIT uses the same operators and control logic as COBWEB (cf. Algorithm 7.1).

*Additional
hyperparameters*

CLASSIT is of course not the only extension of the COBWEB algorithm. Other modifications include for instance FUZZ [99], which represents concepts as fuzzy sets, where membership is determined based on the distance to the cluster prototype.

*Other extensions of
COBWEB*

How can we relate COBWEB and its variants to the conceptual spaces framework? As noted above, COBWEB assumes, that all features are of nominal nature. One possible way of grounding these nominal features in conceptual spaces could lie in identifying features with domains and values of these features with properties inside these domains. This would correspond to crisping the domains of the conceptual space and then learning a conceptual hierarchy on top, making it somewhat related to the formalizations of Rickard [329] and Lewis and Lawry [253] (cf. Sections 2.4.2, 2.5.2, and 2.5.3). However, while Rickard only considers a single conceptual level and analyzes the co-occurrence counts for pairs of properties, COBWEB builds a hierarchy of concepts and considers the co-occurrence counts for properties and concepts. And although Lewis and Lawry consider hierarchical conceptual spaces,

*COBWEB and
conceptual spaces*

*CLASSIT and
conceptual spaces*

their hierarchy does not represent conceptual generalization, but rather conceptual combination.

If we consider CLASSIT instead, we do not need to use properties as features, but can instead directly use the dimensions of the conceptual space. The normal distributions used in CLASSIT have a convex membership function and are thus in line with Gärdenfors' convexity requirement. However, since they are defined only for individual dimensions, they are not able to encode correlations. Such correlations could only be encoded indirectly through the sub-clusters. For instance, the correlation of COLOR and TASTE in apples would need to be encoded by the APPLE concept having two sub-concepts, namely RED APPLE (which is RED and SWEET) and GREEN APPLE (which is GREEN and SOUR). While the individual concepts RED APPLE and GREEN APPLE would thus have their means in the RED/SWEET and GREEN/SOUR regions, respectively, the overall APPLE concept would have a prototype lying somewhere in-between. Although this may be a relatively indirect way of encoding correlations, the intrinsic description of concepts through a probability density function (CLASSIT) or a fuzzy membership function (FUZZ) as well as the cognitively inspired category utility make COBWEB and its descendants very interesting as starting points for concept formation in conceptual spaces. One main additional contribution of the conceptual spaces framework may come in the domain structure, which is not considered explicitly by CLASSIT.

*COBWEB and our
formalization*

In theory, one could also adapt COBWEB to use our formalization as presented in Part I of this dissertation (cf. also Sections 3.6.1 and 4.6.1): The creation of a new concept is relatively straightforward (its core can be defined as a single point, and default values for all other parameters can be chosen), and the merge and split operations can be implemented by using the union of concepts (cf. Section 3.2.1) and the axis-parallel cut (cf. Section 3.5.1). The incremental adaption of the conceptual description, however, would be more complex, since we cannot simply increment some counts or update a mean and a standard deviation: In our formalization, we rather need to modify the support points of the cuboids, the weights, and the sensitivity parameter in a meaningful way (cf. Section 3.6.1). Finally, also the (generalized) category utility would need to be adapted to our representation of concepts. Overall, using COBWEB with our proposed formalization of conceptual spaces thus seems to be possible in principle, but would require some considerable modifications.

7.3.3 Adaptive Resonance Theory

*Preventing
catastrophic
interference with
attention*

Adaptive resonance theory (ART) [90, 92, 175] is a neural theory of human cognitive information processing. In its original version (often called ART₁), it is limited to binary input patterns, but various extensions have been proposed, which relax this constraint. The motivation for ART is based on the observation that in most neural network models, learning can be unstable: Continued learning tends to erase previously

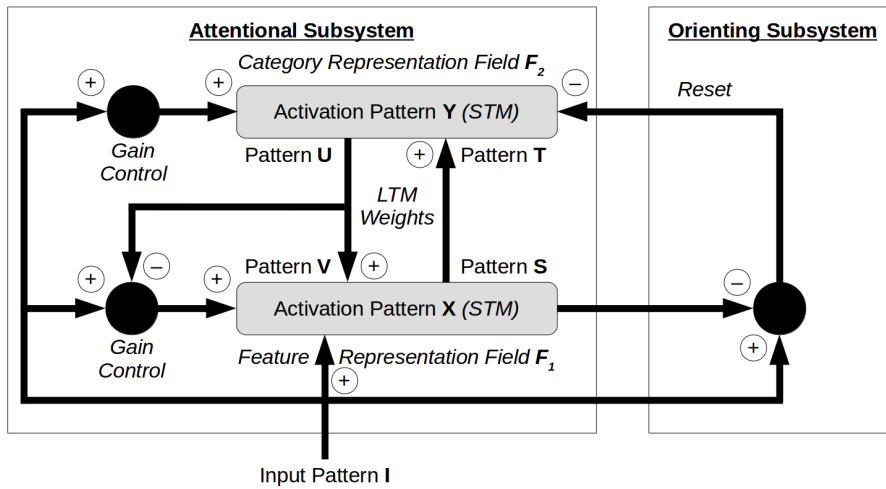


Figure 7.12: Illustration of the overall ART system. Excitatory connections are marked with "+", inhibitory connections with "-".

learned information. This effect is also called *catastrophic interference* or *catastrophic forgetting* [285]. In ART, this problem is overcome through an attentional mechanism, which matches top-down expectations with bottom-up activations. Learning happens only in cases, where such a match occurs, i.e., in a so-called *resonance state*.

The overall structure of ART is illustrated in Figure 7.12. As one can see, ART is comprised of two subsystems: The *attentional subsystem* processes bottom-up activations and top-down expectations, while the *orienting subsystem* decides, whether a resonance state occurs. Within the attentional subsystem, one can distinguish two representational layers: The *feature representation field F_1* essentially represents the input as a point in a feature space, while the *category representation field F_2* encodes the cluster or category associated with the input [91]. The category representation field F_2 is implemented as a competitive network, where typically only one or a few nodes are active at the same time [92]. Familiar inputs are processed with the attentional subsystem, while the orienting subsystem becomes active, if an unfamiliar input occurs, which does not match well with the top-down expectations [175]. The orienting subsystem then resets F_2 in order to search for the next candidate category. The overall matching between top-down expectations (represented as pattern V) and bottom-up activations (represented as input pattern I) occurs within the feature representation field F_1 .

The structure of ART

Let us now take a closer look at the main processing steps occurring in ART (following the description in [175]). We start by considering the *bottom-up mechanism*, which starts with the arrival of a new input pattern I . This input pattern I is transformed into an activation pattern X in F_1 , which corresponds to the representation of I in short term memory (STM). At the same time, I activates the gain controls of F_1 and F_2 (which for the moment can be interpreted as on/off switches) as well as the orienting subsystem. The activation pattern X , however, immediately

The bottom-up mechanism

inhibits the orienting subsystem, preventing it from creating any output. The activation pattern X in F_1 is then transformed into an output pattern S by thresholding the activation of the individual nodes in F_1 . This sparsified output pattern S is then multiplied with weights from long term memory (LTM) in order to create the input pattern T for the category representation field F_2 . This transformation from S to T is called an *adaptive filter*. The input pattern T now leads to an activation pattern Y in F_2 , which is constrained by lateral inhibition of the nodes in F_2 . This lateral inhibition acts as a contrast enhancement. The resulting activation pattern Y can be interpreted as a hypothesis about the input I or as a representation of the best matching category [93].

The top-down mechanism

Once the activation pattern Y has stabilized, the *top-down pass* through the system begins: Again, an output pattern U is created by thresholding the units of F_2 . This pattern U is then transformed into the top-down template V through another adaptive filter. Please note, that the weights used for this transformation are in general different from the ones used in the bottom-up transformation from S to T . Creating the top-down template V can be interpreted as testing the hypothesis V against the input I or as reading out the prototype of the category represented by V [93]. The output pattern U of the category representation field F_2 additionally inhibits the gain control of F_1 . The units inside the feature representation field F_1 now receive two inputs, namely I and V and can only become active, if both inputs match. Thus, F_1 now acts as a matching mechanism for I and V . This behavior is made possible by what Grossberg [175] calls the *2/3 rule*: Each unit in F_1 can only become active if it receives input from at least two of its three inputs (I , V , and gain control). In the first bottom-up pass, the gain control is active and excites all units in F_1 , such that their activation depends only on the input I . During template matching, however, gain control is deactivated and does not provide any input into F_1 . The units of F_1 can thus only be activated, if they receive the same input from both V and I . Since V and I are in general not identical, the new activation pattern X^* , that arises from their interaction, is in general different from the bottom-up activation X .

Finding a resonance state

This similarity between X and X^* now determines, whether we enter a *resonance state*: If X and X^* are sufficiently similar, then the orienting subsystem is still inhibited by X^* : The activation patterns in the attentional subsystem stabilize and persist long enough for learning (i.e., adaption of the LTM weights) to occur [93]. However, if X^* differs considerably from X , this means, that there are fewer active nodes in F_1 , because the template V is necessarily sparser than the gain control input. In this case, the reduced activation of F_1 leads to a reduced inhibition of the orienting subsystem, which is then able to send a strong inhibitory signal to F_2 . This signal resets the activation of F_2 and continues to inhibit the activation pattern Y . Once F_2 has been cleared, the top-down expectation V vanishes and the content of F_1 is again only determined by the input I . This means, that the original representation X is restored, which again is passed to F_2 through adaptive filtering

with the LTM weights. However, since the pattern Y is still inhibited in F_2 , a different activation pattern Y^* will emerge, which then results in a different top-down template V^* . This new template is again matched against the input I in order to determine, whether a resonance occurs, or whether another reset of F_2 is necessary. This whole procedure is repeated, until the system enters a resonance state, either because an existing category matched the input, or because a previously unused pattern Y^* is found, which will then be associated to I .

Learning in ART only occurs within a resonance state. Depending on the result of the matching procedure, it corresponds to refining the representation of a previously learned category, or to learning a new category with a previously unused activation pattern in F_2 [91]. The important distinction between resonant and non-resonant states is controlled by a parameter called *vigilance*, which determines the required degree of similarity between V and I [93]. In general, a low vigilance tends to lead to broad generalizations and abstract prototypes, while a high vigilance results in narrow generalization and exemplar-like categories [93]. While most other neural networks learn through gradient descent and backpropagation (where the contribution of each weight to the overall error is iteratively minimized), ART uses an associative learning mechanism, which requires neither a supervision signal nor a differentiation operation [175].

Learning in ART

Different variants of adaptive resonance theory have been used to analyze and predict data about a variety of cognitive tasks, including perception, conditioning, and decision making [92, 175]. The basic system outlined above has also been extended in various way: The ART2 system [91] uses a more complex setup of F_1 in order to deal with analog input patterns, while Fuzzy ART [93] incorporates fuzzy logic into the overall framework. Distributed ART [92] removes the winner-takes-it-all approach from F_2 and thus allows for a distributed representation of high-level information. Coupling two ART networks with each other results in ARTMAP [92], a system capable of solving supervised classification tasks. Other variants include incremental fuzzy clustering algorithms such as FOSART [30, 31], which uses a simplified version of ART as a starting point. Here, the same matching function is used in the bottom-up and in the top-down pass and the neighborhood relation of the inputs is approximately preserved in the generalizations. RBF activations are used in the category representation field F_2 , which mimics a prototype matching procedure.

Extensions of ART

Like COBWEB, also ART is an incremental clustering approach, which works entirely unsupervised, is capable of creating new clusters as necessary, and can in principle be used for feature prediction tasks. Moreover, it has successfully been extended to classification tasks and continuous input patterns. While the intensional representation of concepts is very explicit in COBWEB and its descendants, it is only implicitly coded through the LTM weights in ART. Moreover, ART does not result in an explicit conceptual hierarchy and has no operators, which correspond to merging and splitting clusters. However, as a

*Comparison to
COBWEB*

ART and conceptual spaces

neural model of human cognitive information processing, it has a much stronger cognitive grounding than COBWEB.

When considering ART from the perspective of conceptual spaces, we can again associate the nominal or binary input values with the presence or absence of properties. The concepts learned by ART would then again be defined on top of the properties, as already discussed for COBWEB in Section 7.3.2, and would thus again relate to the formalizations by Rickard [329] and Lewis and Lawry [253]. The ART2 system is, however, more applicable to the conceptual spaces framework, since it is capable of processing continuous inputs. Its top-down LTM weights can be interpreted as the representation of the respective category's prototype, which relates strongly to Gärdenfors' prototype-based representation of concepts. ART can thus add some neural plausibility to this approach. As far as we can tell, combining our formalization of conceptual spaces as introduced in Part I of this dissertation with adaptive resonance theory is, however, not easily possible due to the very different representational assumptions. Moreover, ART is not explicitly able to model correlations between domains.

7.3.4 Incremental Category Learning with SUSTAIN

Motivation for SUSTAIN

Let us conclude our discussion of concept formation algorithms with *SUSTAIN* (*Supervised and Unsupervised Stratified Adaptive Incremental Network*) [270], a clustering model of human category learning. As a motivation for proposing SUSTAIN, Love et al. [270] note, that category learning in the form of prototype learning may be too simple and inflexible for concepts such as *SPOON*, which consist of multiple sub-concepts, namely *COOKING SPOON* (which is large and wooden) and *TABLE SPOON* (which is small and made of metal). In this case, learning the prototype of *SPOON* as an average instance is not capable of capturing the internal category structure. Exemplar models, which simply store all observations, are on the other hand incapable of extracting a compact summary representation of a category. Love et al. furthermore consider the problem of overfitting and underfitting in machine learning (cf. Section 5.1.2), which indicates that a given model's capacity needs to be correctly adjusted to the learning problem. Finally, Love et al. note, that categorization in humans is heavily influenced by the current task and goals. All of their considerations culminate in the idea, that concepts should be represented by multiple sub-clusters, which are dynamically created, if the model's capacity needs to be increased. These sub-clusters are created as exemplars and may over time evolve into prototypes.

The structure of SUSTAIN

Figure 7.13 illustrates SUSTAIN's overall structure as consisting of three layers. The perceptual input to the system is transformed by the *input layer* into a structured feature vector based on a set of domains, such as *SHAPE*, *BRIGHTNESS*, and *CATEGORY*. In the *cluster layer*, each of these domains comes with a so-called *attentional tuning*, which represents the general importance of the respective domain according to the model's experience. Clusters in SUSTAIN are represented as points in the

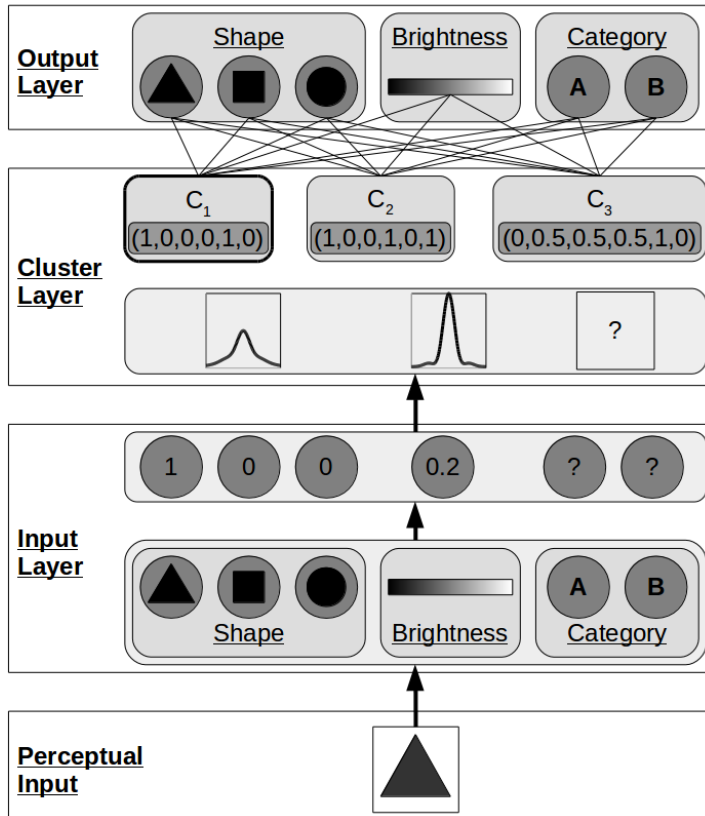


Figure 7.13: Illustration of the overall SUSTAIN model with its three layers.

feature space and are associated with nodes in the *output layer*, which represent the respective cluster's predictions about each dimension in every domains.

The model tries to assign a new instance to one of the existing clusters and to use the category information of this cluster in the output layer to make a classification prediction. If none of the clusters provides a good match, or if the model's prediction was incorrect, a new cluster is created and its prototype is centered on the current observation. SUSTAIN starts with a simple solution (namely, a single cluster) and gradually increases the number of clusters as necessary. Cluster selection is competitive and clusters are formed based on similarity. Learning in SUSTAIN involves both unsupervised and supervised processes. In the following, we will take a closer look at the different processing steps.

Overall processing scheme

Input stimuli are represented as vectors based on the different domains and an additional entry for the category label. Input domains can be both nominal (such as *SHAPE* in Figure 7.13) or continuous (such as *BRIGHTNESS*). In order to represent nominal domains as vectors, Love et al. use a one-hot encoding (cf. Sections 6.1.1 and 6.3.2). For instance, if the possible shapes only include triangles, squares, and circles, then a triangle can be represented as $(1, 0, 0)$. The overall input vector is then denoted by \vec{x} . Its individual entries can be indexed as x_{ij} , where i refers

Obtaining feature vectors

to the underlying domain (such as `SHAPE` and `BRIGHTNESS`) and j indexes the representational dimension within the selected domain.

Representing clusters

Each cluster C_k has one separate receptive field for each domain of the input. This receptive field can be specified by its center $p_{ij}^{(k)}$ with respect to every dimension j of this domain i , and its tuning parameter λ_i for the overall domain. Please note, that while the center of the receptive field may differ between clusters, the tuning parameter is shared across all clusters.

Matching inputs to clusters

In order to match a new input to one of the clusters, one first needs to compute the distances $d_i(\vec{x}, \vec{p}^{(k)})$ between the feature vector \vec{x} and the cluster center $\vec{p}^{(k)}$ with respect to domain i . For continuous domains, this can be easily done by taking the absolute value of their differences on the respective continuous dimension. For nominal domains, Love et al. propose to use the following formula, where V_i represents the number of nominal values for domain i , i.e., the number of one-hot dimensions used to represent it:

$$d_i(\vec{x}, \vec{p}^{(k)}) = \frac{1}{2} \sum_{j=1}^{V_i} |x_{ij} - p_{ij}^{(k)}|$$

Cluster activation

The activation $\mu_k(\vec{x})$ of the k th cluster is then defined as follows:

$$\mu_k(\vec{x}) = \frac{\sum_{i=1}^n (\lambda_i)^r e^{-\lambda_i d_i(\vec{x}, \vec{p}^{(k)})}}{\sum_{i=1}^n (\lambda_i)^r}$$

Picking a winning cluster

Here, n is the overall number of domains and r is an attentional parameter, which controls the relative influence of the attentional tunings: For $r = 0$, all features have the same influence on the overall results, while for larger values of r , more emphasis is put onto features with tight tunings (i.e., large values of λ_i). The cluster with the largest activation value is the winning cluster. The outputs of all other clusters are set to zero, while the output $o_k(\vec{x})$ of the winning cluster is based on its own activation and the activation of its competitors:

$$o_k(\vec{x}) = \frac{(\mu_k(\vec{x}))^\beta}{\sum_{k'=1}^K (\mu_{k'}(\vec{x}))^\beta} \cdot \mu_k(\vec{x})$$

Cluster output

If many clusters are activated, then the output of the winning cluster is inhibited in order to represent the uncertainty in cluster assignment. If the activations of all other clusters are, however, relatively low, then the output of the winning cluster remains high. In the formula above, K refers to the number of clusters and β is a parameter, which controls the degree of cluster competition. For larger values of β , the winning cluster is only weakly inhibited.

Predictions in the output layer

Now in order to predict values for an unknown dimension j in domain i , the output $o_k(\vec{x})$ of the winning cluster C_k is multiplied with the weight $w_{ij}^{(k)}$ connecting it to the output layer:

$$z_{ij}(\vec{x}) = \sum_{k=1}^K w_{ij}^{(k)} \cdot o_k(\vec{x})$$

In order to simulate the decision probabilities in a classification task, SUSTAIN defines the probability of choosing the response j of a nominal domain i based on a softmax function over all other possible values for this domain:

$$P(j|\vec{x}) = \frac{e^{\phi \cdot z_{ij}(\vec{x})}}{\sum_{j'=1}^{V_i} e^{\phi \cdot z_{ij'}(\vec{x})}}$$

*Output probabilities
for nominal domains*

Here, the additional parameter ϕ controls the decision consistency – larger values of d make the probabilities more binary and ensure that the output with the largest activation is almost always chosen.

Decision consistency

Once this forward pass through the model has been computed, learning can take place. In a supervised setting, a new cluster is created, if the model made an incorrect prediction. In an unsupervised setting, a new cluster is created, if the activation of the winning cluster is below a given threshold τ , which indicates, that the observation is not well represented by any of the current clusters. A newly created cluster is always initialized on the current stimulus, and its weights to the output layer are initialized to zero. Afterwards, the following adjustments are made to the position of the winning cluster, the tunings of the dimensions, and the output weights of the winning cluster. For all three updates, the same learning rate α is used. For updating the output weights, t_{ij} denotes the desired target output for the j th dimension on the i th domain.

*Learning in
SUSTAIN*

$$\begin{aligned}\Delta p_{ij}^{(k)} &= \alpha \cdot (x_{ij} - p_{ij}^{(k)}) \\ \Delta \lambda_i &= \alpha \cdot e^{-\lambda_i d_i(\vec{x}, \vec{p}^{(k)})} \cdot (1 - \lambda_i d_i(\vec{x}, \vec{p}^{(k)})) \\ \Delta w_{ij}^{(k)} &= \alpha \cdot (t_{ij} - z_{ij}(\vec{x})) \cdot o_k(\vec{x})\end{aligned}$$

SUSTAIN can be applied to three types of learning tasks, namely classification learning (i.e., predicting the category based on all other features), inference learning (i.e., predicting a missing feature based on all other features), and unsupervised learning (where no explicit prediction needs to be made). Love et al. [270] showed in a variety of experiments, that one can fit SUSTAIN's parameters in such a way, that human behavior in all three of these learning tasks can be accurately predicted. Later, Love and Gureckis [269] were furthermore able to provide a mapping between SUSTAIN and human brain activity.

*Applications of
SUSTAIN*

Like COBWEB and ART, SUSTAIN is an incremental algorithm, which dynamically creates new clusters if necessary, and which is capable of feature prediction. Like ART and unlike COBWEB, SUSTAIN lacks merging and splitting operators and does not create an explicit conceptual hierarchy. However, SUSTAIN comes with an explicit intensional definition of the learned categories through the cluster centers. This relates it to CLASSIT, where clusters are represented by the mean and variance values for each feature. However, in SUSTAIN, only the cluster center is stored with the cluster itself, while the attentional tunings are

*Comparison to
COBWEB and ART*

shared across all clusters. SUSTAIN and ART share the aim to model human conceptualization and are thus cognitively grounded. SUSTAIN differs from both COBWEB and ART by putting a stronger emphasis on supervised learning, using the incremental clustering procedure only as an intermediate processing step.

*SUSTAIN and
conceptual spaces*

When looking at SUSTAIN from the conceptual spaces perspective, we can again associate nominal domains with the set of properties partitioning a given cognitive domain. On the other hand, if we consider continuous domains, then the cluster mean can be interpreted as a prototypical point. This prototype is initially constructed as an exemplar (when a new cluster is created), but is then subsequently generalized based on additional observations. In SUSTAIN, each category can be associated with multiple clusters, which makes it possible to indirectly encode correlations between domains. The distinction between domains and dimensions in SUSTAIN can be roughly mapped into the domain structure of a conceptual space. If we attempted to use SUSTAIN with our formalization of conceptual spaces as proposed in Part I of this dissertation, each cluster would need to be represented by a fuzzy concept. This would make the incremental update in learning more complex than simply moving a prototype. Moreover, SUSTAIN's attentional tunings assume that the same salience weights are used for all clusters, while our formalization proposes to use a separate set of salience weights for each cluster.

7.4 LEARNING CONCEPTS THROUGH COMMUNICATION

*Perspectives on word
semantics*

As Murphy [298, Chapter 11] argues, words get their meanings by being associated to concepts. This stance is commonly referred to as *cognitive semantics*, and needs to be distinguished from *referential semantics*, where words are interpreted as referring to real objects and events. One could say, that in referential semantics, the meaning of a word corresponds to the extent of the concept (e.g., "dog" referring to the set of all possible dogs), while in cognitive semantics, words refer to the intent of the concept (e.g., "dog" referring to the DOG concept, which contains information about the typical SHAPE, SIZE, COLOR, BEHAVIOR etc. of dogs). Murphy furthermore notes, that the relation between concept learning and word learning can be viewed in two principled ways: Either the concept is formed first and the corresponding word is attached later, or a concept is formed based on the usage of a previously unknown word. Especially in the latter condition, the interaction with other individuals is of critical importance for learning. As Gärdenfors [181, Chapter 5] points out, this leads to another perspective on semantics, where meaning is interpreted as something inter-subjective ("meanings are the product of negotiation"). This differs from both referential semantics ("meanings are in the world") and standard cognitive semantics ("meanings are in the head").

In this section, we will take a closer look at some approaches which explicitly focus on the communicative aspects of concept learning. In Section 7.4.1, we introduce some general considerations about the ways in which communication shapes concepts. We introduce Gärdenfors' *language as pointing* and *meeting of minds* approach [181], and present Steels' *language game* methodology [389]. We then structure our further discussion based on the direction, in which information is transmitted [215]: In Section 7.4.2, we focus on *vertical transmission* schemes, where one can identify a clear teacher-student relationship, in which the student adapts its conceptualization to the one of their teacher. Afterwards, we consider *horizontal transmission* schemes in Section 7.4.3, where language learning is viewed as a negotiation between peers in a population, leading to the emergence of a consistent language from local interactions.

Concept learning through communication

7.4.1 General Considerations

We will start our discussion of concept learning in the context of language and communication with Gärdenfors' interpretation of *language as pointing*, which he proposed in [181, Chapter 4] and [184]: Gärdenfors notes, that one of the first ways, in which children communicate with their caregivers, is through pointing. One can distinguish different types of pointing behavior: In *imperative pointing*, the goal of the child is to make the caregiver do something (e.g., give them the toy they are pointing at). The focus on imperative pointing is therefore mostly on the result (e.g., receiving the toy). In *declarative pointing*, on the other hand, the child aims to explicitly direct the attention of the caregiver to the focal object. This also involves checking, whether the caregiver attends to the intended object, thus resulting in joint attention.

Pointing as basic form of communication

Gärdenfors further distinguishes three subtypes of declarative pointing: The goal of *emotive declarative pointing* is to get emotive feedback from the caregiver about the focal object (e.g., smiling or frowning). In this case, the interlocutors need to coordinate in the LOCATION domain in order to identify the focal object based on the pointing gesture. However, also a coordination in the EMOTION domain takes place in order to identify the emotion towards the object. In *information-requesting declarative pointing*, the child attempts to receive non-emotive information about the focal object, usually in the form of a category label ("That is a dog."). In the terms of conceptual spaces, this involves the LOCATION domain (for finding the focal object) and the object category space (i.e., the combination of all domains used to define the concept of the given object, e.g., COLOR, SHAPE, SIZE). From a machine learning perspective, information-requesting declarative pointing can also be viewed as a form of active learning, where the learner explicitly requests a supervision signal for an object. In this case, the learner has a high degree of control about its learning experience in contrast to classical batch-processing of fixed datasets (cf. Section 5.1.2). Finally, *goal-directed declarative pointing* is used to help the caregiver in fulfilling their goal

Different types of declarative pointing

(e.g., if they are looking for an object and the child already spotted it). This involves a combination of the LOCATION domain and the GOAL domain, in which the goal of the caregiver is represented.

Language as pointing in conceptual domains

As Gärdenfors points out, all three subtypes of declarative pointing involve a combination of the LOCATION domain with some other domains of the conceptual space. He furthermore notes, that in some contexts, it may not be entirely clear to the caregiver which object the child is pointing to. In these contexts, verbal language can be used as a way of disambiguation, e.g., by using the category name or by displaying an emotional state (e.g., through crying or laughing).⁷ Thus, language can be used to constrain the search for the focal object in the LOCATION domain by putting constraints on other conceptual domains. Gärdenfors thus views language as "a tool for reaching joint attention by "pointing" to places in our inner worlds" [181, Chapter 4, p. 85]. Once the usage of verbal language has developed to a sufficient degree, children can omit the pointing gesture and use only verbal language to draw the caregiver's attention to an object of interest. Additionally, language can be used to refer to objects, which are not present in the current scene. Gärdenfors concludes his argument by hypothesizing, that the underlying processing steps for creating joint referents are essentially the same in pointing and speaking.

Communication as meeting of minds

If we assume, that the goal of (verbal) communication consist of coordinating the inner worlds of the interlocutors, the question arises, how such a coordination can take place. Gärdenfors (see [181, Chapter 5] and [428]) has analyzed this question under the perspective of fixpoints. He argues that the goal of communication is to obtain a *meeting of minds*, where the interlocutors agree, that their inner worlds are compatible with each other. Communication can then be viewed as a *semantic reaction function* f , which is a concatenation of the speaker's expression function (translating the speaker's conceptualization into language) and the hearer's interpretation function (translating the verbal description into the hearer's conceptualization). A meeting of minds is then achieved, when this semantic reaction function finds a fixpoint x , such that $f(x) = x$. If such a fixpoint is reached, then speaker and hearer have successfully coordinated their inner worlds.

The semantic fixpoint theorem

Gärdenfors uses the three-dimensional COLOR space as an example for illustrating his fixpoint analysis: If we assume, that color concepts such as RED and GREEN are represented by their prototypical points in COLOR space, then the speaker's expression function maps the regions of the resulting Voronoi tessellation onto the different color words. Moreover, the hearer's interpretation function maps the different color words onto their respective prototypes. In this scenario, the prototypes are the fixpoints of color communication. Gärdenfors' *semantic fixpoint theorem* is a consequence of Brouwer's fixpoint theorem [82] and guarantees,

⁷ For example, at the age of two, my son combined pointing to our kitchen's exhaust hood with the word "light" in order to tell me, that I should turn on the exhaust hood's light. This combination of imperative pointing and verbal language prevented me from turning on the exhaust hood's fan or from turning on the ceiling lamp in the kitchen.

that every semantic reaction function, which is a continuous mapping of a convex compact set onto itself, has at least one fixpoint. Since concepts are assumed to be convex regions, and since conceptual spaces are based on similarity, the requirements of a convex compact set and of a continuous mapping function are fulfilled, hence guaranteeing the existence of a fixpoint.

As Gärdenfors notes, the meeting of minds only describes that both interlocutors *think*, that they have reached an agreement. In order to test, whether such a meeting of minds has actually occurred, one usually needs some feedback from the external world. If such a reality check is not possible (for instance, because the conversation is about imaginary events in a fairytale), then the fixpoints may prove to be instable: If the conceptualization of the interlocutors differs, this will result in (slightly) different predictions involving the object or event under discussion. If it becomes apparent over time, that the predictions of one interlocutor are in conflict with the predictions of the other interlocutor, they may have to re-coordinate their inner worlds.

The need for external feedback

As Gärdenfors [181, Chapter 5] notes, the exact form of the semantic reaction function is subject to environmental pressures: It will be adopted by the interlocutors in such a way, that it maximizes the expected communicative success. Moreover, feedback from the environment can act as a tiebreaker in case of multiple possible fixpoints. Jäger and van Rooij [215] argue, that language universals can be explained by the evolution of both brains and culture. More specifically, they identify three important environmental pressures, namely *learnability* (i.e., the ease of learning the language), *parsability* (referring to the ease of cognitive processing), and *usefulness* (i.e., the practical relevance and helpfulness in a given environment or communication setting). Both learnability and parsability incite a simple language, while usefulness prefers a fine-grained language – these pressures are therefore often viewed as pulling in opposite directions [94]. The goal of any semantic category system is then to find a good trade-off between these two objectives, which allows to transmit maximum information for minimal cognitive effort.

Environmental pressures

In their analysis, Jäger and van Rooij [215] furthermore distinguish vertical from horizontal transmission: *Vertical transmission* corresponds to an iterated learning process, where the language is transmitted between generations of agents. This means, that the first agent learns a language, transmits it to the second agent, who after learning transmits it to the third agent, and so on. In each individual communicative setting, there is a clear teacher-student relationship. Jäger and van Rooij argue, that in vertical transmission scenarios, language acquisition is aided by a structured meaning space, which allows for compositionality.

Vertical transmission

On the other hand, *horizontal transmission* focuses on the interactions within a single generation of agents and emphasizes the practical usefulness of a language system. From the perspective of the hearer, utterances need to be parsed rapidly, which limits their possible structural complexity. Moreover, the utterances need to be perceptually distinct

Horizontal transmission

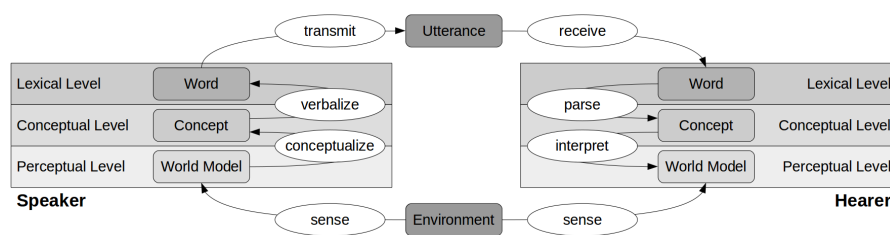


Figure 7.14: Illustration of the semiotic cycle.

from each other in order to decode the audio signal is a robust way. Finally, the semantics of the utterance need to be comprehended quickly and correctly, which incites precise meanings. From the perspective of the speaker, two other considerations come into play: Firstly, the speaker is interested in making utterances containing relevant and useful information, since this can be expected to increase their social status. This consideration therefore encourages a language with a rich semantic repertoire. Secondly, the speaker will choose the least amount of effort to transmit their message, which introduces an economic constraint on the shape of utterances.

*Embodiment as
additional pressure*

Since the interactions between agents take place in the real world, the embodiment of the agents in their environment can be seen as another type of evolutionary pressure on language systems. Bleys et al. [67] have noted, that many computational models of language learning are evaluated on artificially generated data. They furthermore argue, that embodied data differs from such artificial setups in two important ways: Firstly, the same scene is in general perceived slightly differently by two different agents. This perceptual deviation is based on differences in lighting, perspective, and sensor noise. Secondly, embodied data tends to be more structured than artificial data – usually, one is constrained to a relatively small number of objects, where some properties (e.g., the color *TURQUOISE*) are not observed at all, while others (e.g., the color *RED*) occur quite frequently. Artificial datasets, on the other hand, are often constructed in such a way, that the different feature values follow a given ideal distribution (e.g., uniform or normal), and that they cover the underlying feature space in a regular way. Also Spranger et al. [387] consider embodied scenarios as more realistic, but at the same time more challenging due to the larger risk for errors. They note that errors may occur both in natural language processing (e.g., missing words, misunderstood words, or grammatical errors) and in computer vision (e.g., occlusions or errors in visual tracking). Nevertheless, Spranger et al. were able to show through their experiments with humanoid robots, that successful communication about events is possible even under such error-prone conditions.

Language games

As a practical way of evaluating different models of language evolution, different forms of *language games* have been proposed [149, 215, 385, 387, 389, 390]. These language games are based on the so-called *semiotic cycle* [389, 390] depicted in Figure 7.14: Both agents (the speaker

and the hearer) are embodied in the same environment. Each agent perceives this common environment based on their current perspective and sensory channels, leading to a subjective world model. In most applications, this world model corresponds to a feature space, which captures several perceptual properties of the current scene. Based on this world model in the perceptual layer, each agent constructs a set of concepts in the conceptual layer. These concepts are then linked to words in the lexical layer.

A language game can now be defined in the context of this semiotic cycle as a "routinized linguistic interaction in which agents try to achieve communicative goals using their personal conceptual and linguistic knowledge" [67]. It typically involves the following steps: The speaker wants to draw the hearer's attention to one of the stimuli in the jointly observed environment. The speaker thus conceptualizes its perception of the scene (and the focal object), verbalizes the selected concept by selecting an appropriate word, and transmits it to the hearer. The hearer in turn parses the utterance in order to find the appropriate concept, and interprets this concept relative to its current perception by selecting the stimulus, which fits best into this category. Through action in the environment (e.g., by pointing), the hearer can convey their understanding of the utterance, and the speaker can use this nonverbal feedback to decide, whether the hearer has understood the utterance correctly. If this is the case, the conceptualization and verbalization of both agents is reinforced. In case of a miscommunication, the internal representations of the concepts as well as their associations to words may need to be modified. The feedback from these language games thus provides information about the usefulness of the agents' conceptualization in the real world – concepts are grounded in communication. Similar to declarative pointing, also language games can be considered to give a considerably higher degree of control about the learning experience to the individual agents than regular fixed datasets (cf. Section 5.1.2).

Due to the indirect nature of the feedback provided in these language games, the agents face *referential uncertainty*, which is an important aspect of language learning [385]. This means, that the referent (i.e., concept) of a novel word is in general unknown to an agent. Since each observation can be described in an infinite number of ways, it is impossible to be absolutely certain about the meaning of a word, that the agent has never heard before. As Spranger and Beuls [385] note, one can broadly distinguish three models of word learning, which come with their respective definition of referential uncertainty: In *word-object mapping models*, one assumes that the different objects are already known and that the learner only needs to establish a discrete mapping between words and objects. In this case, referential uncertainty can only arise, if multiple objects are present in the environment at the same time: The agent may then be unsure which object is focal. In *combinations of feature models*, word meanings correspond to a combination of symbolic features. In this case, the features are assumed to be already known, but they may need to be combined in novel ways. Here, referential uncertainty can also arise, if multiple features are

Steps in the semiotic cycle

Referential uncertainty

relevant for a given object: The agent may not know which feature is being targeted. Finally, *continuous meaning space models* represent word meanings with continuous vector spaces. In addition to multiple objects and multiple relevant features, the continuous nature of the feature values causes further referential uncertainty. As Eyre and Lawry [149] point out, referential uncertainty can thus result in imprecise concept definitions, where the boundaries between a concept (such as RED) and its negation (e.g., NOT RED) are blurred. Despite the difficulties stemming from referential uncertainty, successful communication is possible, as various examples in Sections 7.4.2 and 7.4.3 will illustrate.

7.4.2 Vertical Transmission

*Vertical transmission
in psychology and
machine learning*

In this section, we introduce two studies, which focus on a vertical transmission in language learning, i.e., where we can identify a clear teacher-student relationship between the agents. This includes the psychological investigations by Carr et al. [94] and the machine learning experiments by Spranger and Beuls [385].

*Iterated learning
with a Bayesian
approach*

Carr et al. [94] investigated, whether the evolutionary pressure coming from language learning favors simplicity or informativeness in the resulting category system. They used an *iterated learning model*, where the evolutionary pressure from learning is iteratively applied, but where other pressures (such as the usefulness in shared tasks) are not present. Carr et al. considered a universe of M discrete meanings, which can be treated as a metric space. They furthermore assumed, that this space can be partitioned into K categories (where partitions can be defined in an arbitrary way), and that each category is associated with one word. A language was then defined as a combination of a partitioning and a lexicon. Carr et al. now modeled each agent as a Bayesian learner, who considers the set of all possible languages as hypothesis space, and who chooses the hypothesis, which matches both the observations made about the teacher and the learner's bias. This bias was represented as a prior probability distribution.

*Simplicity prior and
informativeness prior*

In order to investigate the difference between the simplicity bias and the informativeness bias, Carr et al. used two variants of their model, which make use of different prior distributions: The *simplicity prior* is an exponentially decreasing function of the complexity of the language, which is quantified as the minimum description length in bits (cf. Section 5.1.3) of the underlying partitioning. It thus favors languages with a small number of contiguous categories. The *informativeness prior*, on the other hand, is an exponentially decreasing function of the cost associated with a language, which can be quantified as the probability of a mismatch between an intended meaning and its reconstruction. The informativeness prior thus favors languages with a large number of compact categories.

Simulation results

In their iterated learning scheme, Carr et al. created chains of agents, where the production output of one agent was used as the learning

input to the following agent, starting from a randomly selected initial language. Production was subject to noise (i.e., labels produced by an agent were randomly changed with a small probability), and a bottleneck was enforced on transmission (i.e., not all M examples were transmitted, but only a representative subset). In their simulations with 50 generations of agents, Carr et al. found that the simplicity prior causes a considerable simplification of the category structure and a gradual loss of categories over time. In the resulting languages, categories were mostly defined using only one of the two underlying features. In the simulations with the informativeness prior, however, no reduction in the number of categories was observed. If a very strong emphasis was put on the informativeness prior, then the similarity space was partitioned into four quadrants.

Carr et al. then conducted a psychological study with human learners in the same setup, using Shepard circles as stimuli. These stimuli are circles with a radial line and can vary with respect to both their diameter and the orientation of the line. Carr et al. found that in eleven out of their twelve chains, at least one category was lost over time. Moreover, categories tended to be defined only based on a single dimension. These qualitative effects were predicted by the simplicity prior, but not by the informativeness prior. Carr et al. furthermore estimated optimal parameter settings for their model in order to predict the quantitative observations from the psychological experiment. The model with the simplicity prior was able to achieve a considerably better fit to the experimental data than the model with the informativeness prior. Carr et al. thus conclude, that iterated learning of a language leads to a pressure for simplicity, which eventually leads to a loss of categories and a very simplified category structure, if it is not counterbalanced by other pressures such as informativeness.

Psychological study

Spranger and Beuls [385] have also worked with a vertical transmission setting, however, not in an iterated way. They considered a *description game*, which works as follows: The learner observes an object and a description uttered by the tutor. This description consists of k words, which describe this object.⁸ The learner now has to identify the meaning of the words by integrating information over various trials. In order to evaluate the learning success, Spranger and Beuls prompt both the learner and the tutor with a test object and measure the overlap between the k words chosen by the tutor and the k words chosen by the student for describing this object.

Description games

In their experiments, Spranger and Beuls used a fixed tutor strategy: Objects were represented as points in an n -dimensional feature space, and the tutor represented each word with a prototype $\vec{p} \in [0, 1]^n$ and a binary weight vector $\vec{w} \in \{0, 1\}^n$. The prototypes were randomly drawn from a uniform distribution over $[0, 1]^n$, while the weights were drawn from a binomial distribution, ensuring, however, that at least one entry of the overall weight vectors was nonzero. Given an

The tutor strategy

⁸ In their experiments, Spranger and Beuls used $k = 5$.

example $\vec{x} \in [0, 1]^n$, the tutor then computed the weighted distance $d_{\vec{w}}(\vec{x}, \vec{p}) = \sqrt{\sum_{i=1}^n w_i (x_i - p_i)^2}$ to all prototypes, and chose the k words, whose prototypes had the smallest distance to the example. By using binary weight vectors, Spranger and Beuls ensure that some words are sensitive only to a subset of the features. Moreover, the combination of randomly chosen prototypes and weight vectors makes it likely, that words are used with a different frequency.

*Machine learning
setup*

Based on this fixed tutor strategy, Spranger and Beuls now investigated different learner strategies by applying different types of machine learning algorithms. They note, that the description game can be framed as a supervised, multi-label, online classification problem. They trained different classifiers ranging from logistic regression (cf. Section 5.2.2), over random forests (cf. Section 5.2.4), k nearest neighbors (cf. Section 5.2.5), and naive Bayes to feedforward neural networks with a single hidden layer (cf. Section 6.2.1). Spranger and Beuls used three different datasets, which contained 17 features, 4532 objects, and 100 words known to the tutor: In addition to an artificially created dataset (where feature values were drawn from uniform distributions), they also used two datasets, which were collected based on the sensory input of humanoid robots. In the first of these datasets, learner and tutor made use of the exact same perceptual input, while in the second robot dataset, learner and tutor had a slightly different perception of the same scene.

Experimental results

Spranger and Beuls observed in their experiments, that all machine learning methods performed better on the robot data than on the artificial dataset, which indicates, that they were able to harness the additional structure. In the case of perceptual deviation (i.e., where tutor and learner perceived the same scene, but from different perspectives), a considerable performance drop could be observed. This indicates, that tutor instructions are most helpful, if formulated from the learner's perspective. When exponentially increasing the number of features on their artificial dataset, Spranger and Beuls only noted a linear reduction in performance for most classifiers. Moreover, many classifiers were able to deal well with different degrees of feature sensitivity as controlled by the probabilities used in the binomial distribution of the weight vectors. Spranger and Beuls also found, that frequent words were learned much easier than infrequent ones. Finally, although the different machine learning algorithms used quite different internal representations, all of them worked reasonably well in the description game scenario.

Outlook

While the study by Carr et al. [94] has demonstrated, that iterated learning causes a pressure for simplicity in language, the experiments by Spranger and Beuls [385] show, that many off-the-shelf machine learning algorithms can also be used in the context of language games. However, in both scenarios, the authors assumed a fixed 1:1 mapping between concepts and words. Moreover, the flow of information was unidirectional from teacher to student. In the next section, we will consider studies, which focus on horizontal transmission, where the mapping between words and concepts as well as the shape of the concepts themselves are subject to a negotiation between peers.

7.4.3 Horizontal Transmission

As a prime example of a language game with horizontal transmission of information, Gärdenfors [181, Chapter 5] introduces the *signaling game* investigated by Jäger and van Rooij [215] for the COLOR domain: Speaker and hearer are assumed to have a fixed and finite set of K words as well as a shared world model. This world model corresponds to the COLOR disk, which is spanned by the dimensions HUE and SATURATION. In the signaling game under consideration, nature selects some point in the COLOR space and reveals it to the speaker, but not to the hearer. The speaker can now transmit one of the K words to the hearer, who then has to select one point in the COLOR space as its best guess. The goal of this signaling game is to maximize the average similarity between the original point and the hearer's guess.

A signaling game in the COLOR domain

In game theory, a *Nash equilibrium* is defined as a pair of strategies, which are best responses to each other. This means, that no player can benefit from changing their strategy. Jäger and van Rooij assume, that the COLOR space is convex and compact, and that both the probability distribution used for drawing an example point and the similarity function for pairs of points are continuous. They were able to show, that under these conditions, there exists a Nash equilibrium, which corresponds to a Voronoi tessellation based on K prototypical points: When prompted with nature's chosen stimulus, the speaker selects the word associated to the closest prototype and transmits it to the hearer, who then points to the prototype associated to this word. Unfortunately, if the underlying similarity space is symmetric (e.g., invariant to rotation), an infinite number of equally good Nash equilibria exists. The solution selected by the two agents then heavily depends on the initial conditions.

Voronoi tessellation as Nash equilibrium

What are the dynamics of the process leading to the selection of such a Nash equilibrium? In order to investigate this question, Steels [389, 390, 391] has conducted various studies with large populations of agents, using both computer simulations and robotic experiments. Referring back to the semiotic cycle from Figure 7.14, he splits up the overall *guessing game* into two components, called the discrimination game and the naming game.

Finding the Nash equilibrium

The *discrimination game* is played by a single agent and is concerned with the conceptualization of the world model. Based on a context of several objects and one focal object (which has been selected by nature), the agent tries to find a concept in its conceptual inventory, which matches the focal object, but mismatches all other objects in the current context. The exact procedure of making such a classification depends on the type of concept representation being used, which can for instance be based on prototypes or discrimination trees. If the discrimination game was successful, then the current category system is reinforced and the selected category is returned. If it was not successful, then the category system needs to be adapted by changing existing categories or creating a new category. The exact procedure again depends on the representational format and can for example be based on moving and

The discrimination game

creating prototypes [391], or adding and removing nodes with respect to an existing discrimination tree [389].

The naming game

The *naming game*, on the other hand, is played between the speaker and the hearer and is concerned with the verbalization and parsing of utterances. It assumes, that both players have the same inventory of concepts. In the naming game, nature selects a concept and reveals it to the speaker, but not to the hearer. The speaker verbalizes this concept by looking up the corresponding word. The relation between concepts and words is usually implemented as an *associative memory*, which stores many-to-many associations between words and concepts, where each association is annotated with a confidence weight (usually in the interval $[0, 1]$). Verbalizing a concept then corresponds to picking the word with the highest association to the given concept. The hearer now parses this word by doing the inverse lookup, i.e., selecting the concept with the highest association to the given word. If the hearer selects the correct concept, both agents reinforce their associations by increasing the weights of the selected associations and by decrementing the weights of competing associations. On the other hand, if the hearer selects the wrong concept, the naming game fails and both agents decrement the weights of the associations they used. If the speaker does not have any word associated with the given concept, a new word is created and associated with the given concept. If the hearer receives a novel word, this word is added to the lexicon and a connection to the intended concept is created.

The guessing game

Both the discrimination game and the naming game are by themselves not very interesting, since they are either played only by a single player (discrimination game) or based on the strong assumption that both players share the same conceptual repertoire (naming game). The *guessing game* now integrates the discrimination game and the naming game: Given a context of objects and one focal object, the speaker first plays the discrimination game in order to obtain a concept, which discriminates the focal object from all other context objects. The speaker then initiates a naming game using the selected concept and transmits the best-matching word to the hearer. The hearer now de-references this word in order to obtain a concept, and then uses this concept's description to determine which object in the context was referenced by the speaker. In the case of a success (i.e., the hearer identified the correct object), both the categorization and the lexicon are reinforced. In case of a failure, both the categorization and the lexicon are modified. The guessing game thus combines elements of the discrimination game and the naming game, establishing a connection between the world models of the two agents. In experimental studies, one can either use identical world models [391], or an additional perceptual layer to obtain features from sensory input [67, 389]. The latter approach provides a more realistic setting, but may also cause considerable amounts of additional noise. In the following, we will briefly introduce several studies based on the guessing game.

The *talking heads* experiment by Steels [389] was one of the first empirical studies based on the guessing game. In this experiment, the environment consisted of a whiteboard with several geometric figures (triangles, rectangles, etc.) in various sizes, shapes, and colors. This whiteboard was observed by two cameras, one for each agent playing the game. In order to turn the raw camera image into a feature-based world model, several computing steps were implemented in the perceptual layer, starting with a segmentation of the input image based on colored areas, followed by computing features such as WIDTH, HEIGHT, POSITION, COLOR, OR NUMBER OF EDGES for each of the detected objects, and then re-scaling these feature values based on the current context.

Talking heads

The conceptual layer of the talking heads architecture was implemented with discrimination trees, i.e., binary trees, where each inner node splits up one feature at a specified threshold. This is relatively similar to decision trees as introduced in Section 5.2.4. As Steels notes, the resulting categories based on such a discrimination tree are not based on high within-category similarity, but on pairs of categories, which describe notable differences. The conceptual layer furthermore kept track of the success rate for different categorizers as well as their frequency of use. Whenever the discrimination game failed, the discrimination tree was expanded, considering only nodes, which had been useful in the past. Based on the collected statistics, old categorizers with a low success rate or only infrequent use were pruned from the discrimination tree. This combination of extension and pruning mechanisms can be related to the insertion and merging operators used in COBWEB [159] for growing and shrinking the conceptual hierarchy (cf. Section 7.3.2). The lexical layer of the talking heads was implemented as an associative network as described above. Overall, the talking heads experiment thus used an embodied version of the guessing game, which involved multiple domains (namely, COLOR, SHAPE, SIZE, and LOCATION). As Steels was able to show, a population of agents was able to converge onto a shared vocabulary and a shared conceptualization, if they repeatedly played the guessing game based on the visual input of the two cameras.

Implementation of the conceptual layer

In a related study, Steels and Belpaeme [391] have focused on the COLOR domain, using the psychologically motivated CIE $L^*a^*b^*$ color space. It uses three dimensions, namely LIGHTNESS (L^* dimension), the RED-GREEN channel (a^* dimension), and the YELLOW-BLUE channel (b^* dimension). As stimuli, Steels and Belpaeme used Munsell color chips, which cover the CIE $L^*a^*b^*$ color space in a regular pattern. In contrast to the talking heads setup, Steels and Belpaeme used a prototype-based approach for representing color concepts. More specifically, the activation of each color category k was computed as a weighted sum of J locally reactive units based on a radial basis function (RBF):

Guessing game in the COLOR domain

$$y_k(\vec{x}) = \sum_{j=1}^J w_j \cdot e^{-\frac{1}{2} \sum_{i=1}^n \left(\frac{x_i - p_i^{(j)}}{\sigma} \right)^2}$$

Prototype-based
representation of
concepts

Here, $p_i^{(j)}$ is the i th coordinate of the j th prototype for concept k , and σ specifies the size of the receptive fields. This RBF-based approach can be connected to the conceptual spaces theory, if one assumes $J = 1$, i.e., a single receptive field per concept: In this case, $\vec{p}^{(1)}$ corresponds to the prototype associated with the concept, and the activation function $y_k(\vec{x})$ is convex. In order to update the RBF network associated with each concept, Steels and Belpaeme use the following strategy: In case of a success, the weight of each RBF unit j is increased based on its individual activation. In case of a failure, a new RBF unit is created, which is centered on the stimulus, and either associated to the best matching category or to a new category. After each step, a weight decay mechanism is used to slowly decrease all association weights between RBF units and concepts to simulate forgetting.

Results

In their experiments, Steels and Belpaeme found, that a shared category structure can only emerge, if the agents in their population played the guessing game. If only individual discrimination games were played, the agents ended up with fairly different conceptual boundaries.

Embodied guessing
game for colors

Bleys et al. [67] have provided an additional study on the color domain, using, however, perceptual input from humanoid robots in order to investigate the effects of domain structure and perceptual deviation. Their scenes were based on an office environment with two to four colored objects from an overall set of 20 objects. In the perceptual layer of their architecture, they first segmented the image into regions and then computed each objects' color as the average CIE $L^*a^*b^*$ coordinates of all of its pixels. For each scene, they recorded images from two humanoid robots with different perspectives. The conceptual layer of their setup was based on prototype theory: Each concept was represented by a prototypical point in color space, and a Voronoi tessellation was used for making classifications.

Update strategies

Updates to this conceptual representation were based on three strategies: The *invention strategy* was used to create a new category (with the prototype being identical to the focal object) and to associate this category to a newly created word. The invention strategy was triggered, if the current categorization was not successful in discriminating the focal object from the context. The *adoption strategy*, on the other hand, was used, if the hearer encountered an unknown word, and consisted simply of creating a new concept centered on the focal object and associating this new concept with the novel word. Finally, the *alignment strategy* was used to update existing categories at the end of the guessing game in both agents: In case of a success, the success score of the category was increased and its prototype was moved slightly towards the focal object. In case of a failure, the category score was decreased. Categories with a low score were removed from the lexicon after some time. Unlike previous setups, the model by Bleys et al. did not use an associative memory for mapping concepts to words, but rather assumed a fixed 1:1 mapping.

Experimental results

In their experiments, Bleys et al. considered three embodiment conditions: In *shared simulated perception*, both agents had an identical world

model based on randomly chosen Munsell chips, similar to the setup by Steels and Belpaeme [391]. In *shared grounded perception*, an identical world model was based on the perceptual input to the same humanoid robot. Finally, in *individual grounded perception*, perceptual input from two different humanoid robots with slightly different perspectives on the same scene was used. In their experiments, Bleys et al. investigated the influence of the embodiment conditions and the initial concept inventory on communicative success. They observed, that the added structure from embodiment had a positive impact, while perceptual deviation had a limited negative impact. Agents were able to successfully play the guessing game with the given update strategies, both when using a predefined color lexicon of English color terms, and when starting from a blank slate. In the latter case, a larger number of categories evolved in the population, which allowed for finer-grained distinctions. The resulting category structure was in this case, however, less similar to English color terms. Overall, their experiments showed, that a simple prototype-based model with some fairly straightforward update mechanism is capable of overcoming the difficulties stemming from embodiment.

Eyre and Lawry [149] have explicitly used a representation based on conceptual spaces for language games. Their representation of concepts is based on the formalization by Lewis and Lawry [253] (cf. Sections 2.4.2, 2.5.3, and 7.2.3): Each concept C_k was described by a prototypical point $\vec{p}^{(k)}$ and an uncertain distance threshold $\epsilon_k \geq 0$ (whose probability distribution was given by δ_k). The concept C_k was deemed applicable to a given observation \vec{x} if $d(\vec{p}^{(k)}, \vec{x}) \leq \epsilon_k$. This leads to a fuzzy membership function, which is based on the probability of \vec{x} falling under C_k with the given distribution δ_k over the distance threshold ϵ_k :

$$\mu_k(\vec{x}) = \mathbb{P}(d(\vec{p}^{(k)}, \vec{x}) < \epsilon_k \mid \epsilon_k \sim \delta_k)$$

Eyre and Lawry consider a population of N agents. Each agent has an importance weight w , which reflects their respective "seniority" within the population. Overall, there is a fixed number of concepts, which remains constant during the experiment. At each step, $\frac{N}{2}$ parallel interactions take place, where the speaker asserts that " \vec{x} is C_k " or " \vec{x} is $\neg C_k$ ". The assertion to make is chosen through Bayesian reasoning based on the membership values of \vec{x} to the individual concepts. The hearer then calculates the probability of this assertion being true based on their own definition of the underlying concepts. If this probability is larger than the seniority weight w of the speaker, no action is taken. If this probability is, however, smaller than w , then the hearer updates their representation of the concept C_k : In case of a positive assertion (" \vec{x} is C_k "), this corresponds to moving the prototype of C_k closer to the observation \vec{x} and increasing the concept's threshold ϵ_k (which corresponds to widening the concept). In case of a negative assertion (" \vec{x} is $\neg C_k$ "), the update consists in moving $\vec{p}^{(k)}$ further away from \vec{x} and decreasing the concept's membership threshold ϵ_k (thus making it

*Conceptual spaces
for language games*

Update strategies

more narrow). The size of the update is determined in such a way, that the assertion's degree of truth equals w and that minimal changes are made to the overall conceptual inventory.

Simulations

In their simulations, Eyre and Lawry used 100 agents, a fixed number of 11 concepts, and a three-dimensional artificial feature space. The initial seniority level of the agents was chosen randomly from the interval $[0.1, 0.9]$ and slightly increased after each interaction, simulating the process of aging. As soon as the seniority weight reached a value of 0.9, it was reset to 0.1 to simulate agents leaving and joining the population. Eyre and Lawry observed, that the agents successfully converged on a common conceptual system, as measured both through the overlap of their conceptual regions and through overlap in the generated assertions for novel stimuli. Eyre and Lawry furthermore showed, that negative assertions, which use the negation of a concept, can be a useful communication tool in such scenarios – an aspect, which has been missing from most language game studies.

Co-evolution of domains and compositionality

Also Vogt [421] has considered the combination of language games and conceptual spaces. The focus of his research is, however, on the co-evolution of conceptual domains and compositional language. He notes, that while some domains are probably innate (such as the COLOR domain), other domains may be at least partially determined by culture. Also the compositional nature of language, where the meaning of an overall utterance is based on the meaning of the individual terms, may be attributed to cultural rather than biological evolution. In his simulations, Vogt aims to show, that the compositionality of language and the division of an overall conceptual space into different domains can co-evolve in the course of repeated embodied interactions.

The perceptual and conceptual layer

In his simulations, Vogt uses 120 colored geometrical shapes, which are represented by four features in total. Three features describe the RED, GREEN, and BLUE channel of COLOR perception, while the fourth feature represents the object's SHAPE. Each of these dimensions is discretized into categories based on a set of prototypical values. By mapping each feature value onto its corresponding category, an observation can be described as a vector of four category labels. This vector is then used as a concept in the discrimination game: If the description of the focal object in terms of these category labels differs from all other objects in the current context, the discrimination game was successful. If this is not the case, then the raw values on the dimensions are used to create the prototypes for new feature-wise categories. Vogt notes, that the overall conceptual space has four dimensions, but that individual domains can be constructed by combining a subset of these dimensions. The concepts from the different domains can then be used as basic meanings in the agent's language. Compositionality then corresponds to the combination of meanings from different domains.

The lexical layer

The grammar for mapping between concepts and utterances is based on simple rewrite rules with an associated confidence score. These rules can be either holistic (i.e., using all dimensions) or compositional

(i.e., using different domains for different parts of the utterance, thus introducing one level of indirection in the rewrite process).

In his simulations, Vogt used 50 agents, which played a total number of one million guessing games with a context of eight objects. As Vogt notes, a division of the overall conceptual space into a *COLOR* domain (using the *RED*, *GREEN*, and *BLUE* dimensions) and a *SHAPE* domain (using only the *SHAPE* dimension) is most efficient with respect to the number of rules needed to generate and interpret utterances. However, since the *RED* channel was quite informative for the given dataset, one may also expect agents to use a separate domain for the *RED* dimension, while combining all remaining dimensions into another domain. The simulations showed, that if the pool of agents was kept fixed, communicative success reached a level of 80%. However, agents used the same type of internal rules only in 50% of the interactions. Thus, even though agents tended to produce the same utterance, they did so using different types of rules (i.e., different domain structures). Holistic rules were very frequent in the beginning of the experiments, but were subsequently replaced by compositional rules. The most frequent rule type were rules, which distinguished the *COLOR* domain from the *SHAPE* domain, but other types of rules were also able to survive.

In a second set of simulations, Vogt still kept the overall number of agents fixed, but introduced a life cycle similar to the one used by Eyre and Lawry [149]: After having played a certain number of guessing games, agents were removed with a certain probability from the population and replaced by new agents without any conceptual or grammatical knowledge. As Vogt observed, the results were less stable than in the first experiment (due to the additional fluctuation in the population), but communicative success reached comparable levels. More importantly, the agents used the same rule types in about 75% of the interactions, with a much stronger emphasis on compositional rules using a distinction into the *COLOR* domain and the *SHAPE* domain. It thus seems like the influx of new agents prevented the population from getting trapped in a local optimum by introducing another pressure, which favors more compact language systems.

Let us conclude our discussion with the work by Ohmer et al. [305, 306] as a recent example for combining language games with deep neural networks. Their studies focus on the mutual influence between perceptual biases and emergent languages. In their experiments, Ohmer et al. used a *reference game*, where the speaker was presented only with the focal object and then created a multi-word utterance. This utterance was then used by the hearer to identify the focal object in a context of distractors. Both speaker and hearer were implemented using a convolutional neural network (cf. Section 6.2.2) for perception and a recurrent neural network for translating between the CNN's high-level representation and the multi-word utterance. Due to the immense computational complexity of training large neural networks, Ohmer et al. only simulated a single pair of agents with fixed role assignment instead of using a complete population of agents. Their work can thus

Simulations

Adding a life cycle

*Language games
with neural networks*

be seen as an intermediate case between vertical transmission (using a single pair of agents with fixed role) and horizontal transmission (both agents adapting their strategies).

Adding perceptual bias with relational label smoothing

Ohmer et al. used the 3dshapes dataset, which contains artificial images of colored objects in front of a (differently) colored background, and which has, for instance, also be used for evaluating the SCAN system [197] (cf. Section 6.4.1). Ohmer et al. defined different classes based on combinations of the COLOR, SHAPE, and SIZE of the foreground object, and used these classes to pretrain the CNNs on a classification task. In order to obtain different perceptual biases for the agents, they used a technique called *relational label smoothing* [276]: Instead of using a one-hot vector for encoding the target classification output, they used a target value of $1 - \epsilon$ for the true class and $\frac{\epsilon}{K}$ for the K classes, which shared the respective bias feature with the target class. For instance, in order to induce a perceptual bias towards the SHAPE domain, an input consisting of a SMALL RED CUBE caused also the classes representing a SMALL YELLOW CUBE OR A LARGE GREEN CUBE to receive a nonzero target value. Unlike the general label smoothing approach (cf. Section 6.2.3), which distributes a fraction of the output probabilities not among *all* other classes, relational label smoothing thus only makes such a distribution among semantically related classes. This tends to lead to representations, which respect the similarity relations among classes [276]. Ohmer et al. pretrained different CNN variants either without relational label smoothing (DEFAULT configuration), by using relational label smoothing with respect to a single feature (COLOR, SHAPE, and SIZE), or with respect to all three features at the same time (ALL configuration). By assessing the average Cosine similarities of the CNNs' learned representations for pairs of examples from all possible class combinations, Ohmer et al. were able to show, that the learned representations exhibited a bias towards the respectively targeted feature.

Experiments

In a first set of language game experiments, Ohmer et al. investigated combinations of agents with the same or different biases. If both agents exhibited the same bias, the messages, which evolved as part of the reference game, showed a clear grounding of the preferred feature, as indicated by entropy-based evaluation metrics. An unbiased agent tended to adapt its language to a biased interlocutor, which the effect being stronger for a biased speaker than for a biased hearer. Moreover, a pair of agents with an ALL bias received higher rewards than an unbiased pair of agents, indicating, that the perception of relevant similarity relationships improved communication. In a second set of experiments, Ohmer et al. investigated the influence of language on perception by allowing the agents to fine-tune the weights of their pretrained CNNs. They compared the Cosine similarities for pairs of images between the speaker's CNN and the hearer's CNN, and observed a co-adaptation of their perceptual layers, where the hearer was influenced by the speaker more strongly than the other way around. Finally, Ohmer et al. considered a population of agents with an identical perceptual bias, and analyzed, whether an agent with a different bias

was able to "intrude" this population by obtaining consistently higher rewards than its current members. They found, that a population of agents with an ALL bias could not be invaded, indicating, that a perceptual bias for all relevant perceptual features is an evolutionary stable solution.

7.5 SUMMARY

As we have seen in this chapter, there are many aspects and approaches for making machine learning approaches more cognitively plausible. We have focused on three aspects, which we deemed to be the most relevant to conceptual spaces: In Section 7.2, we considered the incorporation of background knowledge, and we discussed logic tensor networks as a promising way for connecting the symbolic and the conceptual layer. The usage of unlabeled data with a focus on concept formation algorithms was then discussed in Section 7.3, before we focused on learning in the context of communication in Section 7.4 in the form of language games. There are, of course, several other strands of research, which are relevant and may spark interesting research programs, such as active learning [361], learning with incomplete information and insufficient resources [425], or multi-modal embodied learning [301].

Lessons learned

An obvious question, which arises in this context, concerns the relation to our formalization of conceptual spaces from Part I of this dissertation. In Section 7.2.3, we have argued, that logic tensor networks can in principle be used with our proposed membership function. However, this membership function may have serious limitations with respect to the gradients it provides. Nevertheless, this seems to be a promising research direction, since the incorporation of explicit top-down constraints has so far not been addressed in the context of conceptual spaces. In Section 7.3, we have already discussed, that when using our formalization as means of representation, one can certainly take inspiration from existing concept formation algorithms, but would need to adapt many concrete measures and update steps. In Section 7.4, we already noted, that some researchers have already used conceptual spaces as a representation format for language games. One could extend this approach by using our formalization of the framework, which would allow to also introduce operations on concepts: Agents could then for instance also pick a word like AND as referring to the intersection operation from Section 3.1.2, and then use utterances such as "RED AND ROUND" to describe an object, which is both RED and ROUND. Again, such an application of our formalization would, however, require the development of specific update procedures for the conceptual representations based on the results of the interactions.

Open ends: using our formalization

In the end, the final goal is of course to combine all three aspects discussed in this chapter into a single algorithm. This single learning mechanism would then use an incremental procedure, which is able to

Combining multiple aspects

work in an entirely unsupervised way, but which can also incorporate the indirect feedback from language games. Moreover, it would be able to extract structured knowledge (such as "all apples are round") from the geometric representation and to use this background knowledge as constraints for subsequent learning tasks.

First steps

Needless to say, such a tight integration of different aspects in a single algorithm is very complex and thus difficult to achieve. First steps into this direction should thus probably concentrate on individual aspects. For instance, Alomari et al. [12] attempted to learn a grounding of natural language concepts in visual perception. They considered video clips of different activities, which were annotated with natural language commands. After applying object detection to a given scene, they used multiple feature spaces to represent each object, namely COLOR, SHAPE, and LOCATION, as well as RELATIVE DISTANCE and RELATIVE ORIENTATION of two objects to each other. This resembles the domain structure of a conceptual space. In each feature space, they used an incremental Gaussian mixture model to cluster the raw values into Gaussian components, which can be interpreted as properties within the respective domain. On top of these properties, Alomari et al. created a graph-based representation of the overall scene, which can be thought of as a description in the symbolic layer. They then mapped subsequences of the natural language commands to subgraphs of the overall scene graph by counting relative co-occurrences. Overall, the approach by Alomari et al. can be related to research in conceptual spaces and explicitly targets incremental, loosely supervised learning.

Outlook

This chapter marks the end of our background on machine learning algorithms. In the next chapter, we will introduce an optimization technique called *multidimensional scaling*, which obtains similarity spaces based on dissimilarity ratings from psychological studies. We will furthermore argue, that a combination of this psychological approach with the representation learning techniques introduced in Chapter 6 can combine psychological validity with the ability to generalize to novel inputs. This hybrid approach will then be applied to two exemplary datasets in Part III of this dissertation.

8

MULTIDIMENSIONAL SCALING AND A HYBRID PROPOSAL

8.1	Obtaining Dissimilarity Ratings	435
8.1.1	Direct Methods	437
8.1.2	The Spatial Arrangement Method	438
8.1.3	Indirect Methods	441
8.1.4	General Considerations	443
8.1.5	Extracting Dissimilarity Ratings From Datasets .	446
8.2	Multidimensional Scaling as an Optimization Problem	451
8.2.1	The Optimization Problem	451
8.2.2	Metric and Nonmetric MDS	454
8.2.3	General Remarks	456
8.2.4	An Illustrative Example	457
8.3	Analytical Solutions with Classical MDS	460
8.3.1	Mathematical Background	460
8.3.2	The Algorithm	463
8.3.3	An Illustrative Example	466
8.4	Kruskal's Gradient Descent Approach	469
8.4.1	Gradient Descent for Stress	469
8.4.2	Monotone Regression	471
8.4.3	Example for Monotone Regression	473
8.4.4	The Algorithm	475
8.4.5	An Illustrative Example	477
8.5	Iterative Stress Majorization Through SMACOF	480
8.5.1	Mathematical Background	480
8.5.2	A Majorizing Function for Stress	482
8.5.3	The Algorithm	486
8.5.4	An Illustrative Example	487
8.6	Evaluating MDS Solutions	490
8.6.1	Stress-based Evaluation	491
8.6.2	Interpretability-based Evaluation	493
8.7	A Hybrid Proposal	498
8.7.1	Proposed Procedure	498
8.7.2	Possible Network Architectures	501
8.7.3	Related Work	505
8.8	Summary	509

In Chapter 6, we have considered the usage of artificial neural networks (ANNs) for extracting the dimensions of a conceptual space. One great weakness of this approach is its lack of psychological plausibility: The solutions are found by iteratively optimizing a given error measure on a large dataset of examples – a procedure, which is not necessarily cognitively plausible. Insights from psychological studies are typically

*Psychological
implausibility of
ANNs*

not taken into account by this approach. In this chapter, we therefore introduce another way for obtaining conceptual spaces, namely, the technique of *multidimensional scaling* (MDS). This approach is typically followed in psychophysics and cognitive psychology.

*Eliciting
dissimilarity ratings*

Multidimensional scaling is based on pairwise *dissimilarity ratings* for a given set of stimuli. These dissimilarity ratings are typically obtained through psychological studies. We will introduce different ways of eliciting such dissimilarity ratings in Section 8.1. The result of this collection step for a set of N stimuli is usually represented as a so-called *dissimilarity matrix* Δ of size $N \times N$.

*Multidimensional
scaling*

The input to multidimensional scaling then consists of this dissimilarity matrix Δ and the desired number n of dimensions for the similarity space. MDS then represents each stimulus as a point in an n -dimensional space, and tries to arrange these points in such a way, that the distance of two points accurately reflects the dissimilarity rating of the stimuli they represent. The underlying optimization problem of minimizing the so-called *Stress* measure will be treated more formally in Section 8.2. There exists a variety of different algorithms for solving the MDS optimization problem. In Section 8.3, we introduce classical MDS, which finds an analytical solution based on an eigenvalue decomposition. Afterwards, we describe Kruskal's algorithm (which is based on gradient descent) in Section 8.4, before discussing the SMACOF algorithm (which uses iterative function majorization) in Section 8.5.

*Evaluating and
interpreting
similarity spaces*

Choosing the optimal value for the number n of dimensions is often not trivial. Therefore, one typically creates similarity spaces for different values of n , and compares them to each other. In Section 8.6, we introduce two evaluation approaches for the similarity spaces produced by MDS. On the one hand, one can directly measure how well the optimization problem has been solved by quantifying the remaining level of Stress. On the other hand, one can analyze, whether the similarity space is structured in a meaningful way. Since the solutions provided by MDS are invariant under rotation and reflection, the coordinates of the resulting configuration typically do not coincide with interpretable features. We will also show in Section 8.6, how one can find directions in the similarity space, which correspond to such interpretable features.

*A hybrid approach:
combining MDS
with ANNs*

A serious drawback of the MDS approach is, that it is inherently limited to the stimuli from the psychological experiment. If a previously unseen stimulus arrives, it is in general impossible to directly map it onto a point in the conceptual space without eliciting a sufficient amount of dissimilarity judgments between this new stimulus and stimuli from the original study [35]. Such additional dissimilarity judgments could then be used to either triangulate the position of the new stimulus in the conceptual space, or to re-run the MDS algorithm with an enlarged dissimilarity matrix of size $(N + 1) \times (N + 1)$. While such a procedure may be acceptable, if one is only interested in doing psychological research, it is clearly not suitable for applications in artificial intelligence. In Section 8.7, we propose to initialize a similarity space with MDS, and then to harvest the generalization capability of artificial

neural networks for learning a mapping from raw stimuli to points in the conceptual space. We argue, that this hybrid approach combines the psychological grounding of MDS with the generalization capability of ANNs, thus taking the best from both worlds.

Finally, in Section 8.8, we summarize the key insights from this chapter, and give a brief outlook on our experimental studies reported in Part III of this dissertation.

The content of this chapter is partially based on work previously published in [40, 47].¹ Moreover, much of the general background information is based on the textbook by Borg and Groenen [71].

8.1 OBTAINING DISSIMILARITY RATINGS

In order to apply multidimensional scaling, one needs to provide a matrix Δ of pair-wise dissimilarities for all stimuli under consideration. In order to collect dissimilarity ratings from human participants, several different techniques can be used (see e.g., [71, Chapter 6] and [169, 201, 433]). Table 8.1 gives an overview of the different methods, which we will cover in this section. Typically, one distinguishes direct from indirect methods: In *direct methods*, participants are fully aware that they rate, sort, or classify different stimuli according to their pairwise (dis-)similarities. *Indirect methods*, on the other hand, are based on secondary empirical measurements such as correlations, confusion probabilities, or reaction times. An appropriate transformation of these secondary measurements results in so-called *derived dissimilarities*.

Direct and indirect methods

In Table 8.1, we also introduce a second way of classifying the elicitation methods, namely into pair-based and stimulus-based methods: In *pair-based methods*, participants are presented with pairs of stimuli, and their task is directly based on these stimulus pairs. Here, dissimilarity information is collected for each pair of stimuli individually. In *stimulus-based methods*, on the other hand, participants operate mainly on individual stimuli, e.g., by grouping or rating them individually. Dissimilarity ratings for pairs of stimuli can then be obtained by combining the ratings of the individual stimuli in an appropriate way (e.g., by computing their correlation).

Pair-based and stimulus-based methods

Since the classification into direct and indirect methods is much more prevalent in the literature, the remainder of this section will be organized according to this distinction: We will give an overview of several direct methods in Section 8.1.1. In Section 8.1.2, we will look at one of the direct methods, namely the Spatial Arrangement Method (SpAM), in more detail. Afterwards, we will give an overview of several

Overview

¹ The initial workshop paper [40] has been joint work with Elektra Kypridemou. In [40], Elektra Kypridemou has described the background with respect to multidimensional scaling (MDS), and has extracted similarity spaces with MDS from the raw data, while I have mainly developed and described the hybrid approach, and implemented and evaluated the machine learning experiments.

	Direct	Indirect
Pair-based	Pairwise comparisons Ranking Binary ranking Q-sort Anchor stimulus	Perceptual confusion Conversion-based
Stimulus-based	Constrained sorting Free sorting Triad Spatial arrangement	Correlation-based Attribute profiles Co-occurrence-based

Table 8.1: Classification of psychological methods for eliciting dissimilarities.

indirect methods in Section 8.1.3. Some general considerations with respect to the experimental setup apply to both types of elicitation methods. They are discussed in Section 8.1.4, where the distinction into pair-based and stimulus-based techniques will again play a role.

Dissimilarities from other sources

In addition to the elicitation of dissimilarity ratings in psychological studies, it is also possible to derive dissimilarities from other sources such as text corpora or pretrained artificial neural networks. While these dissimilarities can no longer claim to be cognitively grounded, their collection can be automated and thus easily be scaled up to thousands of stimuli. In Section 8.1.5, we therefore summarize an approach for deriving dissimilarities from text corpora as well as a study investigating whether the activations of artificial neural networks can be used to predict human dissimilarity ratings.

Relation to metric learning

At this point, we would also like to refer back to relative distance constraints from metric learning (cf. Section 5.3.3), which can be expressed as follows for a set R of data point triples $(\vec{x}^{(j_1)}, \vec{x}^{(j_2)}, \vec{x}^{(j_3)})$, the desired distance function \hat{d} , and a margin $s > 0$:

$$\forall (\vec{x}^{(j_1)}, \vec{x}^{(j_2)}, \vec{x}^{(j_3)}) \in R : \hat{d}(\vec{x}^{(j_1)}, \vec{x}^{(j_2)}) < \hat{d}(\vec{x}^{(j_1)}, \vec{x}^{(j_3)}) + s \quad (5.26)$$

Relative distance constraints based on dissimilarities

In Section 5.3.3, we noted, that such relative distance constraints can be based on class labels, with $\vec{x}^{(j_1)}$ and $\vec{x}^{(j_2)}$ belonging to the same class, but $\vec{x}^{(j_3)}$ belonging to a different class. One can easily see, that also the entries of a dissimilarity matrix Δ can be used to create such constraints, namely, by choosing stimuli $\vec{x}^{(j_1)}, \vec{x}^{(j_2)}, \vec{x}^{(j_3)}$, for which $\delta_{j_1 j_2} < \delta_{j_1 j_3} + t$ for a margin $t > 0$, i.e., whose pairwise dissimilarities differ at least by t .² This would be one possible way of grounding machine learning techniques in psychological data. A deeper discussion of this idea, is, however, beyond the scope of this dissertation.

² Since dissimilarity ratings and desired distances may be scaled differently, we use two different margins s and t .

8.1.1 Direct Methods

The probably most common way of collecting dissimilarities in a direct way is based on explicit ratings for *pairwise comparisons* (see [71, Chapter 6] and [169, 201, 229, 433]): In this approach, all possible pairs from a set of stimuli are presented to participants (one pair at a time), and participants rate the dissimilarity of each pair on a given dissimilarity scale. Each point of the dissimilarity scale is associated with a numeric value, where large values correspond to large dissimilarities and small values express small dissimilarities. This method is conceptually very straightforward, and the obtained ratings can be directly used for subsequent analysis without further transformations.

Pairwise comparisons

Instead of asking for numerical dissimilarity ratings, *ranking tasks* try to elicit a ranking of the dissimilarities (see [71, Chapter 6] and [433]). Participants are presented with a set of cards, which contain pairs of stimuli. They are then asked to sort these cards according to the dissimilarity of the stimulus pairs printed on them: The card with the most similar stimulus pair should be on top of the resulting stack, while the card with the most dissimilar item pair should be on the bottom of the stack. This full ranking thus implies a dissimilarity scale with $\frac{N \cdot (N-1)}{2}$ points for a set of N stimuli.

Ranking tasks

Creating a complete ordering of all stimulus pairs is, however, quite demanding for participants, especially when the number of stimuli is very large. A *binary ranking task* thus only requires participants to sort the cards into two piles, one for similar and one for dissimilar pairs of stimuli [433]. This is somewhat reminiscent of the absolute constraints for metric learning discussed in Section 5.3.3, where pairs of data points were also defined to be either similar or dissimilar.

Binary ranking

Binary ranking, however, only gives very coarse-grained information. As a compromise between cognitive workload and fine-grained results, one can thus ask the participants to again divide up each pile into two sub-piles by distinguishing more similar from less similar stimulus pairs [71, Chapter 6]. This can be recursively repeated, until the participants are no longer able to make a distinction. The resulting equivalence classes can then be transformed into numeric dissimilarity values.

Hierarchical binary ranking

Another way of reducing the workload for the participants in ranking tasks consists in predefining the number of piles. Each pile represents one degree of dissimilarity and can be assigned a numeric dissimilarity value. Sorting cards with stimulus pairs into these pre-defined piles thus results in dissimilarity ratings similar to the ones from pairwise comparisons. If one furthermore defines, how many cards need to be placed into each pile (typically following a normal distribution), one arrives at the so-called *Q-sort technique* [71, Chapter 6]. This additional constraint on the distribution of the dissimilarity ratings is easily enforced when sorting cards into piles, but is not directly applicable to pairwise comparisons, where the participants only see one stimulus pair at a time.

Q-sort

Anchor stimulus method

Another direct method based on stimulus pairs is called the *anchor stimulus method* [71, Chapter 6], which elicits conditional dissimilarities. Here, one stimulus is picked as an anchor, and participants are asked to judge the respective dissimilarity of all other stimuli to this anchor stimulus on a given dissimilarity scale. In the experimental design, each stimulus needs to be selected as anchor once. The resulting dissimilarity ratings are now conditional in the sense, that they depend on the anchor: One can only make a meaningful comparison between the dissimilarities of two stimulus pairs, if they involve a common anchor. While this can account for asymmetric dissimilarities (which have already played a role in Section 4.4.1), one needs to employ special MDS algorithms for processing these conditional dissimilarities.

Sorting tasks

Sorting tasks again ask participants sort cards into piles. However, in contrast to the methods described above, this time each card contains a single stimulus instead of a stimulus pair. Participants are now asked to build piles of similar items. If the number of piles is pre-defined by the experimenter, one could call this *constrained sorting*. Typically, however, the number of piles is left to the discretion of the participants. The resulting method is then called *free sorting* [169, 229, 433]. Dissimilarity scores for stimulus pairs can be derived based on their co-occurrence: If two stimuli are found in the same pile, they receive a dissimilarity score of zero. If two stimuli have been sorted into different piles, they obtain a dissimilarity score of one. The elicited dissimilarities are thus binary in nature.

Triad method

A more constrained version of the sorting task is called the *triad method* [201]: Here, only a subset of three stimuli is shown at once, and participants are asked to put them into two piles. This corresponds to selecting which stimulus pair is the most similar one. Again, this results in a binary notion of dissimilarity. The triad method obviously needs to be repeated multiple times, until many (or preferably all) triples of stimuli have been presented. It can be related to the relative constraints for metric learning discussed in Section 5.3.3, where one also considers triples of data points. One can moreover view the third stimulus in this comparison as a context, which highlights the commonalities among the two other stimuli [194]. The overall similarity between two stimuli can then for instance be approximated by their probability of being grouped together, irrespective of the context [194].

8.1.2 The Spatial Arrangement Method

SpAM in a nutshell

The *Spatial Arrangement Method (SpAM)* proposed by Goldstone [169] is a direct method of eliciting dissimilarity ratings, which makes use of a geometric approach. In this collection technique, multiple visual stimuli are displayed on a computer screen. In the beginning, the arrangement of these stimuli is randomized, and participants are then asked to arrange them via drag and drop in such a way, that the distances between the stimuli are proportional to their respective dissimilarity.

Once participants are satisfied with their solution, they can store the arrangement. The dissimilarity of two stimuli is then recorded as their Euclidean distance in pixels. Let us assume, that N stimuli are displayed at the same time. If the participant modifies the position of a single stimulus, its distance to all other $N - 1$ stimuli have been changed. Thus, each modification by the participant updates $N - 1$ dissimilarity values at the same time which makes this procedure very efficient. Moreover, SpAM quite naturally incorporates geometric constraints: If A and B are placed close together, and C if is placed far away from A , then it cannot be very close to B .

It is a natural thought to combine the similarity ratings obtained through SpAM with a subsequent application of multidimensional scaling: In both cases, a spatial representation of dissimilarities is used, dissimilarity is assumed to be a symmetric relation, and a full matrix of dissimilarities is expected. As Goldstone put it:

SpAM and MDS

As such, the technique can be thought of as requiring subjects to create their own personal multidimensional scaling (MDS) solutions. [169, p. 381]

In an experiment with 64 uppercase letters, Goldstone [169] demonstrated, that the similarity values obtained through SpAM have a high correlation to similarity values gathered from pairwise comparisons and from confusion tasks. Instead of displaying all 64 stimuli at once on the screen, he conducted multiple runs of SpAM with randomly chosen subsets of 20 stimuli each. By aggregating over these multiple runs, he was able to fill the complete dissimilarity matrix, while keeping the cognitive load for the participants at a manageable level. Moreover, Goldstone found, that with SpAM he could collect similarity ratings for all 64 stimuli within a one-hour session, whereas for the other methods, only a similarity matrix of 15 stimuli could be collected in the same amount of time. This highlights, that SpAM is very efficient, while at the same time producing similarity ratings of a high quality.

Quality of the resulting ratings

Constraining the participants to a two-dimensional space for expressing similarity relations can in theory be a very strong constraint, which may negatively affect the quality of the results: If the given stimuli differ in $n \gg 2$ dimensions, then participants might not be able to make a meaningful arrangement in two dimensions. However, Goldstone [169] found, that the optimal MDS solution for his stimulus set of upper case letters involved five dimensions, even though the collection technique limited participants to two-dimensional arrangements. Thus, it is possible to extract high-dimensional similarity spaces from low-dimensional spatial arrangements. By looking at the results of this experiment, Goldstone also noted, that SpAM seemed to reflect cognitive similarity (e.g., category membership) rather than perceptual similarity (e.g., visual appearance).

The 2D constraint

More recently, Hout et al. [201] have conducted additional experiments to evaluate the SpAM technique. They found, that SpAM provided MDS solutions, whose quality was comparable to that of the

Additional experiments

pairwise comparison method. Moreover, the MDS solutions based on SpAM were quite stable with respect to multiple runs of the MDS algorithm, and with respect to the removal of entries from the dissimilarity matrix. In their experiments, contrary to Goldstone [169], they found that SpAM worked well not only for extracting conceptual similarities, but also for extracting perceptual similarities.

General properties of SpAM

Overall, Hout et al. argue that SpAM provides fine-grained dissimilarity ratings, which are limited only by the screen resolution. Moreover, they classify SpAM as intuitive and user friendly. Another advantage of using SpAM is that participants are automatically "calibrated": By seeing a large subset of the stimulus set at once, they are provided with enough context to make consistent judgments. In contrast to that, if users are presented with one pair of stimuli at a time, their first ratings might be quite unreliable, because they make these ratings without any context. Later ratings might, however, be influenced by the stimulus pairs observed before. Hout et al. also note, that while being a promising technique for collecting similarity judgments, SpAM is inherently limited to the visual domain, because stimuli need to be presented on a computer screen.

SpAM for abstract stimuli

Richie et al. [328] have recently investigated whether SpAM is also applicable for higher-dimensional, conceptual stimuli in the form of words. With their work, they try to address the common skepticism, that the two-dimensional arrangements created with SpAM may implicitly restrict participants to focus their similarity ratings on only two underlying dimensions. It is thus somewhat unclear, whether higher-dimensional structures (such as presumably used by conceptual rather than perceptual stimuli) can be recovered from SpAM ratings. In their experiments, Richie et al. confirmed, that the SpAM ratings had a high correlation with pairwise comparison ratings. Moreover, they successfully reconstructed the five-dimensional BigFive space of personality traits from SpAM-based dissimilarity ratings on personality trait adjectives like "sentimental" or "approachable".

SpAM and contextualized pairwise comparisons

Also Verheyen et al. [418] have compared SpAM to the pairwise comparison method. However, they considered a variant called *total-set pairwise comparison*, where the full set of stimuli is always shown to the participants as a reference while judging the pairwise similarities. Verheyen et al. confirmed, that SpAM was considerably faster than pairwise comparisons, but also found, that the elicited ratings were less reliable, i.e., less comparable across different participants. However, if ratings were aggregated across different participants, the resulting averaged similarities were of a comparable quality for both approaches. Verheyen et al. also noted, that SpAM was biased towards spatial representations of the data (such as obtained via MDS), but that a feature-based representation (where each stimulus is described by the presence or absence of certain features, cf. Sections 1.1.1 and 4.4.1) is difficult to extract from these ratings. Ratings based on total-set pairwise comparisons, on the other hand, were not found to have such a bias in their study. However, Verheyen et al. also point out,

that perceptual stimuli (which are the main focus in this dissertation) usually lend themselves to a spatial representation, while feature-based representations seem to be mainly beneficial for more abstract stimuli.

Kriegeskorte and Mur [229] have proposed a variant of SpAM called the *multi-arrangement method*: Like Goldstone, they propose to ask participants for multiple arrangements of different subsets of the overall set of stimuli. However, instead of using randomly chosen subsets, they initially collect an arrangement of the overall set of stimuli. Then, based on this initial arrangement they present clusters of similar items on subsequent trials – in some sense, they "zoom in" in order to obtain finer-grained distinctions. Kriegeskorte and Mur propose to use an online algorithm for selecting the next subset, which keeps track the amount of evidence collected for each item pair, and which selects a subset of items, such that additional ratings are collected for the stimulus pairs with the currently weakest evidence.

*Multi-arrangement
method*

Due to the "zooming in" effect, the dissimilarities collected in the individual trials no longer reflect the same order of magnitude. In order to nevertheless aggregate them into a global dissimilarity matrix, Kriegeskorte and Mur propose an *inverse MDS* approach: An initial estimate of the dissimilarity matrix is used to obtain a two-dimensional arrangement with metric MDS for all stimulus subsets used in the different trials. The initial dissimilarity matrix is then iteratively adjusted, such that the resulting MDS arrangements are more closely aligned with the actual two-dimensional arrangements provided by participants. The final dissimilarity matrix (which achieves the best agreement with the two-dimensional arrangements) can then be used for subsequent analyses, such as extracting higher-dimensional similarity spaces with MDS. When comparing their proposed multi-arrangement method to pairwise comparisons, Kriegeskorte and Mur found a similar test-retest reliability (i.e., results were consistent across different experimental sessions) and a high correlation of the resulting dissimilarities.

Inverse MDS

8.1.3 Indirect Methods

In indirect methods, participants are typically not aware, that dissimilarity ratings are collected. This also implies, that dissimilarity is measured only indirectly based on other quantities such as reaction times or correlations. The dissimilarity matrix is then derived afterwards by an appropriate transformation of the measured quantities.

General approach

One popular indirect method uses *perceptual confusion tasks*. Typically, participants are asked to report as fast as possible, whether two stimuli (which may be only presented for a very brief amount of time) are the same or different [169]. This is done for all pairs of stimuli. In this case, confusion probabilities and reaction times are measured in order to infer the dissimilarities. The underlying assumptions are the following: On the one hand, incorrectly responding "same", even if the stimuli

*Perceptual confusion
tasks*

are different, is much more likely for pairs of similar stimuli than for pairs of dissimilar stimuli. On the other hand, the decision to correctly respond "different" for pairs of similar stimuli takes more time than for pairs of dissimilar stimuli.

Strengths and weaknesses

This approach has the advantage of being very close to the perceptual level [433], because participants do not have enough time to involve complex cognitive operations. Therefore, data collected through perceptual confusion tasks may be less biased by background knowledge and context. However, the stimulus set must be chosen in such a way, that perceptual confusions are likely enough to take place. Moreover, the dissimilarity data obtained based on perceptual confusion tasks might be asymmetric, which is problematic for many standard MDS algorithms, which require symmetric dissimilarity values.

Conversion-based methods

Conversion-based methods [71, Chapter 6] take another approach: Pairs of stimuli are presented, and participants are asked to choose one of the two stimuli based on a given criterion. If the stimulus set consists of cars, one could for instance ask participants to choose the more attractive one. If both stimuli are very similar, one would expect that they are both selected by about 50% of the participants. Dissimilarity can then be computed as the degree to which the selection probability differs from 50%. Obviously, the resulting dissimilarity ratings are strongly influenced by the decision criterion used for elicitation.

Correlation-based methods

Correlation-based methods [433] start by asking participants to rate individual stimuli on a single given scale (e.g., to rate cars with respect to their attractiveness). The similarity of two stimuli is then computed as the correlation between their ratings across participants. That is, two stimuli are considered to be similar, if a higher rating for the first stimulus is associated with a higher rating for the second stimulus. A matrix of dissimilarities can then for instance be obtained by multiplying the correlation coefficient with minus one in order to convert similarities into dissimilarities.

Attribute profiles

Another indirect method is based on *attribute profiles* [71, Chapter 6]. Here, participants are asked to rate individual stimuli with respect to multiple attributes (e.g., rating cars with respect to performance, economy, and luxury). One can obtain a so-called *stimulus-attribute matrix* by averaging the ratings across all participants. In this stimulus-attribute matrix, each stimulus is represented by a vector of attribute values, which is similar to the design matrix used in machine learning (cf. Section 5.1.2), where each example is represented as a vector of feature values. Dissimilarities between pairs of stimuli can now be derived by computing for example the Manhattan distance between their attribute vectors. One can also compute the correlation of the different stimuli based on their average attribute values [433]. A crucial design decision for the analysis of attribute profiles concerns the choice of an appropriate distance function in the context of on the given application.

A final type of indirect methods is *co-occurrence-based* [71, Chapter 6]: Here, participants are asked, whether or not given binary features apply to a given stimulus or not. Instead of estimating the similarity of the stimuli (as in the attribute profiles approach), we can now determine the similarity of the features by counting how often they co-occur. One possible choice for a distance metric is the Jaccard similarity $Sim_J(A, B) = \frac{|A \cap B|}{|A \cup B|}$, which divides the number of stimuli, that have been labeled with both features A and B , by the number of stimuli, that have been labeled with at least one of the features (cf. Section 4.4.2).

Co-occurrence-based methods

8.1.4 General Considerations

Many of the direct and indirect methods introduced above explicitly consider all pairs of stimuli. Since the number of stimulus pairs grows quadratically with the number of stimuli, these pair-based collection techniques tend to be relatively inefficient. As Borg and Groenen [71, Chapter 6] note, one can drastically reduce the number of item pairs by assuming, that dissimilarities are symmetric, and that the dissimilarity of any given stimulus to itself is always zero. As we will see, this is in line with the assumptions made by typical MDS algorithms. If we plan to analyze the collected dissimilarity ratings with MDS, these assumptions are therefore justified. Instead of collecting N^2 ratings for N stimuli, one then only needs to collect $\frac{N \cdot (N-1)}{2}$ judgments. Nevertheless, the number of stimulus pairs under consideration grows quite quickly with an increasing number of stimuli. If participants are required to rate all $\frac{N \cdot (N-1)}{2}$ pairs, this increases the duration of the experimental sessions. Long-lasting experimental sessions are not only time consuming, but they might also result in both training effects (i.e., participants changing their strategy over time) and fatigue effects (i.e., participants losing concentration and/or interest over time) [169].

Pair-based methods do not scale well

Borg and Groenen [71, Chapter 6] therefore propose to collect ratings only for a subset of the stimulus pairs. According to them, MDS still works reasonably well even with incomplete dissimilarity matrices – provided, that the data quality is high enough. Kruskal [234] for example claims, that a dissimilarity matrix, which is only filled to 25-50%, may still be sufficient for good results, if the entries are properly distributed to cover all stimuli to a sufficient degree. Hout et al. [201], however, note, that most researchers still prefer to collect full dissimilarity matrices, because they tend to provide more robust and precise solutions. A reasonable compromise between these two approaches might consist in collecting an incomplete dissimilarity matrix for each participant, but to make sure, that each stimulus pair is rated by some of the participants through a counter-balanced design. When aggregating the individual dissimilarity ratings across participants, one can then nevertheless obtain a complete global dissimilarity matrix. Borg and Groenen [71, Chapter 6] furthermore propose to use coarse-grained rating scales in order to speed up the rating procedure (in the extreme case using only two classes, namely, similar and dissimilar). Again, they argue, that

Collecting incomplete data

MDS can still yield reasonably good results, if the overall data quality is high otherwise.

Ordering effects

Borg and Groenen [71, Chapter 6] furthermore note, that the order of stimuli within a pair and the order, in which the stimulus pairs are presented to the participants, can have an influence on the resulting rating. They recommend to either randomize the orders or to use a counter-balanced experimental design, such that these ordering effects cancel out, when the dissimilarities are aggregated across participants.

Systematicity effects

Goldstone [169] points out, that many pair-based methods do not make use of systematicities, that are to be expected. For instance, if A and B are considered to be similar to each other, while C is deemed to be quite dissimilar from A , then B and C are not expected to be very similar to each other. This systematicity is, however, not necessarily respected, if stimulus pairs are rated independently of each other. All collection techniques based on individual stimuli incorporate these systematicities more naturally, because the dissimilarities are based on a suitable distance function applied to the individual stimuli. This distance function typically satisfies the above-mentioned constraints.

Stimulus-based methods require post-processing

Elicitation methods based on individual stimuli rather than stimulus pairs have the disadvantage that the dissimilarity information needs to be extracted through an additional post-processing step [433]. This extraction of dissimilarities sometimes requires additional assumptions (such as which exact transformation function should be applied), and might add further noise to the collected data. However, stimulus-based methods can be more intuitive for participants.

Comparison of methods

When comparing different elicitation methods, it seems, that pairwise comparison and free sorting result in the highest data quality and provide engaging tasks for the participants [71, Chapter 6]. Also SpAM has been shown to yield high-quality results, while providing an intuitive interface for participants [169, 201]. On the other hand, the anchor stimulus method is often considered as boring by participants [71, Chapter 6]. Dry and Storms [139] have compared the pairwise comparison method to various attribute profile approaches. They found that similarities based on attribute profiles (converted into similarity ratings by using correlations, distances, or common/distinctive features) were easier to fit by metric MDS than pairwise comparison. Moreover, the resulting similarity spaces gave better predictions for typicality effects. However, especially for perceptual stimuli, for which it is difficult to define a comprehensive list of features, attribute profile approaches may not be easily applicable.

Choosing and appropriate method

Above, we have presented many different approaches for obtaining dissimilarity data. A natural question that may arise is which approach should be taken for a given research problem. As we have seen, all of them have their individual strengths and weaknesses. It is therefore recommended to start with the current research problem in mind and to seek an elicitation method, which fits the current requirements best. Some approaches might be obviously unfit (e.g., perceptual confusion

tasks, if the stimuli are not easily confusable) and can be ruled out in advance. Another example is given by Borg and Groenen [71, Chapter 6]: When eliciting dissimilarities for a psychological study on human perception, many indirect methods (such as attribute profiles) are not applicable, because they impose an experimenter-chosen structure (e.g., the list of attributes deemed relevant) onto the participants' ratings, which might obscure the results. In this case, direct methods (where participants are not constrained with respect to how they judge dissimilarities) are preferable. Often, multiple methods seem appropriate and it is up to the researcher to decide which one is easiest to implement or promises to give higher quality data.

All of the above mentioned methods are limited to a relatively small number of stimuli, because each similarity judgment made by a human requires a certain amount of time. Although SpAM provides a considerable improvement in terms of efficiency, it can hardly be scaled to thousands of stimuli. Moreover, the psychological experiments need to be carefully prepared and conducted in order to minimize the amount of noise in the dissimilarity judgments. Also the variability between participants might introduce considerable amounts of noise into the collected data. As argued by Goldstone [170], one can in general assume, that the dissimilarity ratings from psychological studies are quite noisy. Before applying MDS, one therefore typically averages the similarity ratings across all participants in order to get more stable values, usually by applying the arithmetic mean [139, 248, 433].

Noise in the data

Whenever processing dissimilarities one should explicitly consider the underlying type of measurement scale [392]:

Measurement scales

- If dissimilarities are *nominally* scaled, then we can only say, whether two dissimilarity ratings are the same or different.
- If they are *ordinally* scaled, we can also determine, whether one dissimilarity rating is larger or smaller than another one.
- If dissimilarities lie on an *interval* scale, also differences between two dissimilarity ratings can be computed and are meaningful.
- For *ratio* scaled dissimilarities, we are additionally allowed to compute ratios of two dissimilarities.

In general, one assumes at least an ordinal scale, since we want to distinguish low dissimilarities from high dissimilarities. Assuming an interval or a ratio scale allows for more powerful analysis methods, but is often not justified for ratings elicited in psychological studies. With respect to the SpAM approach, it has, however, been argued, that the collected dissimilarities are ratio scaled [200], because they are based on Euclidean distances on the computer screen. However, as participants are likely to make only a rough arrangement of the stimuli, this assumption might be too strong in practice. One can therefore argue, that it is nevertheless preferable to assume only an ordinal scale.³

Are SpAM ratings ratio scaled?

³ Personal communication with Robert Goldstone, January 2019.

As we will see in Section 8.2, the assumption about the underlying scale type influences the type of MDS algorithm one should use on a given dissimilarity matrix. We will return to this question in Chapter 9, where we analyze a dissimilarity matrix based on SpAM.

*Measurement scale
and aggregation*

Moreover, the scale type plays a role when aggregating dissimilarity ratings across participants. As stated above, it is common practice to take the arithmetic mean of the individual participants' ratings in order to eliminate noise [139, 248, 433]. However, the arithmetic mean is only applicable to interval and ratio scales, because it involves the computation of differences. If we assume that, the dissimilarities are only ordinally scaled, then we are strictly speaking not permitted to use the arithmetic mean for aggregating them – one should instead use the median, which also provides a central tendency, but only requires an ordinal scale. In Chapters 10 and 11, we will provide a comparison of the mean and median aggregation over participants, when analyzing ratings with respect to visual dissimilarity.

8.1.5 Extracting Dissimilarity Ratings From Datasets

*Automating the
elicitation process*

As mentioned before, eliciting similarity ratings from humans can be a very time consuming process. In principle, however, multidimensional scaling can be applied to *any* matrix of dissimilarities. Therefore, one can also define a measure of dissimilarity for a given domain of stimuli and automatically compute the dissimilarity matrix. In this section, we will summarize the approach taken by Derrac and Schockaert [123] (cf. Section 1.2.5, 6.3.2, and 7.1.3), who use natural language processing techniques on tags and review texts to automatically extract dissimilarities for the domains of PLACE TYPE, MOVIE, and WINE. Moreover, we will introduce the work of Peterson et al. [318, 319], who investigated, whether the internal activations of artificial neural networks can be used to predict human dissimilarity ratings on pairs of images.

*Collecting data for
the PLACE TYPE
domain*

Derrac and Schockaert [123] extracted similarity spaces for three domains (namely, PLACE TYPE, MOVIE, and WINE) based on textual data. They used tags from the image sharing website Flickr⁴ to derive dissimilarities between different place types. Starting with a set of candidate place types from given taxonomies (GeoNames⁵, Foursquare⁶, and OpenCyc⁷ [281]), they retrieved 22 million images from Flickr, which were tagged with at least one of these place types. For composite terms like "football stadium", they considered both the whole phrase and individual words (i.e., "football" and "stadium") as matching tags. After removing all place types with less than 1,000 associated images, Derrac and Schockaert represented each place type as a bag of words (cf. Section 6.3.2) containing all of its associated tags.

4 See <https://flickr.com/>.

5 See <https://www.geonames.org/export/codes.html>.

6 See <https://developer.foursquare.com/docs/categories>.

7 OpenCyc was a publicly available subset of the commercial Cyc ontology (see <https://cyc.com/>), which has been discontinued in 2017.

The dissimilarities for the `MOVIE` domain are based on movie reviews from IMDB⁸, Rotten Tomatoes⁹, and other sources. Derrac and Schockaert started with the 50,000 movies, which had the most votes on IMDB, and selected the 15,000 movies, for which the concatenation of all reviews contained the most words. Each movie was then represented by a bag of words containing all words from all reviews converted to lower case, excluding very frequently occurring words like "the" as well as punctuation and accents.

Collecting data for the MOVIE domain

Finally, the dissimilarity matrix for the `WINE` domain is based on the wine review corpus of the SNAP project [282], which includes two million reviews of almost 500,000 wines. Each wine is tagged with its so-called *variant*, e.g., SYRAH. Derrac and Schockaert considered all variants, whose reviews accumulated to at least 1,000 words. Analogous to the movies, each variant was represented as a bag of words based on the concatenation of all reviews.

Collecting data for the WINE domain

In all three cases, each stimulus (i.e., place type, movie, wine variant) was represented by a bag of words. Derrac and Schockaert used positive pointwise mutual information (PPMI) to define the association strength between a word w and a given stimulus s . Please recall from Section 6.3.2, that PPMI assigns high association strengths only to word-stimulus combinations, where the word w occurs rarely in the overall corpus, but frequently in the representation of the stimulus s . In other words, it assigns a large weight to very diagnostic words.

Identifying diagnostic words

Each stimulus was then represented as a vector of PPMI values with one entry for each word in the vocabulary.¹⁰ In order to derive a dissimilarity rating for pairs of stimuli, Derrac and Schockaert use the normalized angular difference between the PPMI vectors of the respective stimuli. This normalized angular difference essentially measures the angle between the two PPMI vectors, and can thus be related to the Cosine similarity discussed in Section 6.3.2. If the angle between the two vectors is small, they point in a similar direction and contain similar diagnostic words. The pair of underlying stimuli thus receives a low dissimilarity value. On the other hand, if the angle between the vectors is large, they contain very different diagnostic words and the underlying stimuli are therefore judged to be very dissimilar.

Obtaining dissimilarities

As Derrac and Schockaert were able to show, the similarity spaces extracted from these dissimilarity matrices give intuitive results with respect to semantic distance and semantic betweenness (cf. Section 7.1.3). We will return to their further analysis of the similarity spaces in Section 8.6. When comparing their approach to the psychological methods discussed above, it seems that Derrac and Schockaert followed the attribute profile approach, where the attributes correspond to the PPMI values of the individual words, and where the normalized angular difference was used as a distance metric.

PPMI-based attribute profiles

⁸ See <https://www.imdb.com/>.

⁹ See <https://www.rottentomatoes.com/>.

¹⁰ All of these PPMI vectors as well as further data from their study are publicly available online at <http://www.cs.cf.ac.uk/semanticspaces/>.

*Predicting
dissimilarity
judgments with
pretrained ANNs*

Peterson et al. [318, 319] have investigated, whether the activation vectors of an artificial neural network can be used to predict human dissimilarity ratings. In order to motivate their research, they argue, that deep learning networks achieve their excellent performance on image datasets, because they are able to effectively extract useful features from high-dimensional inputs – a task that the human brain is faced with as well. Although adversarial examples [401] clearly show, that neural networks do not discover the exact same set of features as humans for classifying visual input, Peterson et al. assume, that there is still a considerable amount of overlap.

*Potential
applications of
ANN-based
dissimilarities*

While most psychological studies are limited to small sets of artificial stimuli (where the features are well-defined and easy to manipulate by the experimenter), artificial neural networks can be applied to large numbers of natural stimuli. Peterson et al. argue, that predicting human similarity judgments from the activations of neural networks can enable researchers to validate psychological theories on large datasets of real world images. Moreover, it is difficult to directly compare the representations extracted by artificial neural networks to the ones used by humans, because human mental representations are not directly accessible by an experimenter. However, an indirect comparison is possible, if the dissimilarity judgments produced by these two types of representations are considered.

*Stimuli and human
ratings*

In their study, they used six datasets of different visual domains (namely, ANIMAL, AUTOMOBILE, FRUIT, FURNITURE, VEGETABLE, and VARIOUS), each containing 120 images with a resolution of 300 by 300 pixels. Peterson et al. conducted a psychological study, which elicited pairwise dissimilarity ratings for all pairs of images using the pairwise comparison approach. When applying multidimensional scaling to the resulting dissimilarity matrix, they were able to identify clear clusters in the resulting similarity space (e.g., all birds being located in a similar region of the ANIMAL space). Also when applying a hierarchical clustering algorithm (cf. Section 7.3.1) on the collected dissimilarity data, a meaningful dendrogram emerged.

*Extracting
dissimilarities from
ANNs*

In order to extract dissimilarity ratings from five different neural networks (four convolutional feedforward networks and one unsupervised GAN variant, all pretrained on ImageNet [120]), Peterson et al. used the following procedure: For each of the images, they computed the activation in the second-to-last layer of the network. Then for each pair of images, they defined their similarity as the inner product of these activation vectors. Here, the inner product of two n -dimensional vectors u and v is defined as $u^T v = \sum_{i=1}^n u_i v_i$. These similarities can be easily converted into dissimilarities by inverting them. When applying MDS to the resulting dissimilarity matrix, no meaningful clusters were observed. Also a hierarchical clustering algorithm did not result in a meaningful dendrogram. When considering the correlation between the dissimilarity ratings obtained from the neural networks and the human dissimilarity matrix, Peterson et al. were able to achieve values of R^2 between 0.19 and 0.58 (depending on the visual domain). Overall, all feedforward convolutional networks achieved comparable perfor-

mance, while the unsupervised GAN variant showed inferior results. Peterson et al. also found, that using the last hidden layer of the neural network led to better results than using either the output layer or any of the earlier hidden layers.

In order to further improve their results, Peterson et al. argued that the activation vectors in the second-to-last layer of a neural network are subject to a linear transformation (by being multiplied with a weight matrix) before being used for classification. This is, however, not taken into account, when their inner product is computed in order to obtain similarity scores. Peterson et al. therefore modified their extraction mechanism by using a weighted version of the inner product with one weight w_i per feature, i.e., $\sum_{i=1}^n w_i u_i v_i$. The weights w_i were optimized by a linear regression, such that the difference between the resulting similarity scores and the human dissimilarity ratings was minimized. In order to avoid overfitting, Peterson et al. used L2 regularization on the weights. The resulting ridge regression was trained in a six-fold cross validation, using a hyperparameter search for the optimal strength of regularization. By applying this procedure, Peterson et al. found, that their results considerably improved: Both the similarity space obtained by MDS and the dendrogram obtained by hierarchical clustering became more human-like. Moreover, the correlation between the predicted dissimilarities and the human dissimilarity ratings increased to values of R^2 between 0.35 and 0.74.

Peterson et al. noted, that this transformation does, however, not generalize well to other domains: For instance, if the weights were estimated on the ANIMAL domain, performance on the AUTOMOBILE domain was worse than for unweighted dissimilarities. Nevertheless, Peterson et al. were able to show, that estimating the weights based on five out of their six domains and applying the results onto the remaining domain helped to mitigate this effect: In this case, values of R^2 between 0.53 and 0.63 were obtainable.

In a final experimental step, Peterson et al. investigated, whether the predicted dissimilarity ratings generalized to other cognitive tasks on unseen data points. In order to do this, they picked 120 additional stimuli from each domain and extracted their ANN-based dissimilarities. They used the k -means clustering algorithm (cf. Section 7.3.1) on the resulting dissimilarity matrix in order to divide the stimuli into k categories. Participants were then trained to categorize the new stimuli into these categories. Peterson et al. found, that participants were more successful in learning the categorization in those cases, where the ANN-based dissimilarities were closer to the human similarity judgments.

Also the approach by Peterson et al. can be interpreted as a variant of the attribute profile method, this time using the ANN activation vectors as attribute vectors and the negated inner product as distance measure. Please note, that Peterson et al. are not the only researchers following this approach: Kubilius et al. [235] have found, that pretrained artificial neural networks can predict human judgments with respect to SHAPE similarity quite well. Their analysis furthermore showed, that the lower layers of the network were better at predicting raw visual

Adding feature weights

Generalization between domains

Generalization to other cognitive tasks

ANN-based attribute profiles

similarity (measured as pixel-wise difference of the images), while higher layers of the network were more closely related to higher-level similarity judgments as elicited from humans. Erdogan et al. [146] have used the triad method to collect human shape similarity ratings for artificial block-based objects. In their results, they found, that pretrained artificial neural networks were able to predict the binary human similarity judgments with an accuracy of about 60%, being only outperformed by a Bayesian inference model explicitly targeted at *SHAPE* similarity. Both studies will be discussed in more detail in Chapter 10 in the context of the *SHAPE* domain. For now, it suffices to notice, that their results are in line with the ones provided by Peterson et al.

Advantages of the data-driven approach

Both studies presented above are similar in nature and can be seen as two different examples of a general data-driven approach. One advantage of this overall approach is that it is more capable of dealing with new stimuli: Dissimilarity is not based on human ratings, which are expensive to collect, but on a mathematical function, that can be automatically evaluated. Therefore, it is relatively easy to compute dissimilarity ratings between a new stimulus and all previously known stimuli, making it possible to enlarge the dissimilarity matrix and to re-run the MDS algorithm if necessary. Moreover, the data-driven approach can process a much larger stimulus set than the psychological approach. Even though the computation of the $N \times N$ dissimilarity matrix becomes computationally expensive for large numbers of N , the runtime of a computer program is negligible to the amount of time necessary for conducting a psychological experiment.

Disadvantages of the data-driven approach

However, this data-driven approach also comes with some drawbacks: First of all, the psychological plausibility of the resulting dissimilarities is unclear, because we have no direct grounding in a psychological experiment. This criticism mainly applies to the work by Derrac and Schockaert, while Peterson et al. explicitly compare their ANN-based dissimilarities to dissimilarities from a psychological experiment. Moreover, the mathematical definition of the dissimilarity function is a very critical design decision in this approach. If the dissimilarity function is flawed in some way, the resulting dissimilarity matrix may not be very useful for further processing. Derrac and Schockaert use text documents and tag collections as data source and can build upon dissimilarity measures already established in the natural language processing community. The results by Peterson et al. indicate, that their choice of the negated inner product seems to work well on the ANN activation vectors. However, despite these limitations, it seems, that the data-driven approach to collecting dissimilarities is a worthwhile avenue for further research.

8.2 MULTIDIMENSIONAL SCALING AS AN OPTIMIZATION PROBLEM

Given a set of stimuli and their pairwise dissimilarities, multidimensional scaling (MDS) tries to represent each stimulus as a point in a similarity space, such that the distances between these points are an adequate representation of the dissimilarities of the respective stimuli. As argued by Goldstone [170], applying MDS can be seen as a form of information compression: Instead of a dissimilarity matrix Δ of size $N \times N$, we get an $N \times n$ matrix X representing the coordinates of N stimuli in an n -dimensional space. If n is considerably smaller than N (which is usually the case), then the information from Δ needs to be compressed in order to fit into X . We can thus interpret MDS as a form of dimensionality reduction, more specifically as a feature extraction approach (cf. Section 5.3.2). The distance of two points according to the configuration X is not only influenced by the dissimilarity rating of the corresponding stimulus pair from Δ , but also by the dissimilarities of other stimulus pairs. These additional constraints help to cancel out noise contained in the original ratings. Thus, the similarities derived based on the distance within the resulting similarity space can be thought of as more robust than the original similarity ratings.

MDS as dimensionality reduction

In Section 8.2.1, we will provide a mathematical definition of the optimization problem underlying MDS, mainly based on [71, Chapters 3, 8, and 9] and [233]. We then introduce the distinction into metric and nonmetric MDS in Section 8.2.2. Section 8.2.3 contains some general remarks, and Section 8.2.4 concludes our general discussion with an illustrative example.

Overview

8.2.1 The Optimization Problem

Multidimensional scaling takes as input an $N \times N$ matrix Δ of pairwise dissimilarities $\delta_{j_1 j_2}$ for all pairs of stimuli j_1 and j_2 . Usually, one assumes, that $\delta_{j_1 j_2} = \delta_{j_2 j_1}$, and that $\delta_{j j} = 0$, i.e., that dissimilarities are symmetric, and that self-dissimilarity is zero [233]. Moreover, one needs to specify the dimensionality n of the target similarity space. Multidimensional scaling then tries to find an arrangement of N points in an n -dimensional space, whose pairwise distances reflect the dissimilarities as closely as possible. Typically, the solution is represented by an $N \times n$ matrix X , whose rows correspond to the points, and whose columns correspond to the coordinate axes of the similarity space [71, Chapter 3]. Based on X , one can compute an $N \times N$ distance matrix $D(X)$, which contains all pairwise distances of these points. While one can in theory use any kind of distance metric (e.g., a general Minkowski metric [233]), one usually employs the Euclidean distance.

MDS in general

In its most simple variant, MDS tries to find an exact match between distances and dissimilarities. A perfect match may, however, not be achievable in practice, for example due to noise in the dissimilarity

Quantifying solution quality with Stress

matrix Δ , or because the dimensionality n of the similarity space was set too small. If a perfect solution is not obtainable, we are nevertheless interested in a configuration X , which is as good as possible. In order to quantify the quality of a configuration, one uses a loss function called *Stress*, which is the sum of squared representation errors:

$$\sigma_r(X) = \sum_{j_1=1}^N \sum_{j_2=j_1+1}^N (d_{j_1 j_2}(X) - \delta_{j_1 j_2})^2 \quad (8.1)$$

Properties of Stress

One can easily see, that by minimizing raw Stress as defined in Equation 8.1, we obtain a configuration of points X , whose distances match the dissimilarities as closely as possible. Here, we only sum over all $j_1 < j_2$, i.e., all matrix entries above the diagonal, because we assume, that Δ is symmetric, and that the entries on the diagonal are not informative. Please note, that Stress is invariant to rotations, reflections, and translations of the configuration X , because these operations do not affect the distances between the points.

MDS and machine learning

One may note that raw Stress as defined in Equation 8.1 corresponds to the mean squared error (cf. Section 5.1.2) of the distance matrix $D(X)$ with respect to the dissimilarity matrix Δ . However, we would like to emphasize, that Stress minimization should be interpreted as an optimization rather than a machine learning problem: While we minimize the mean squared error of the model's predicted distances with respect to the psychological ground truth, this model does not provide an input-output mapping like typical machine learning models. Therefore, the resulting model cannot be applied to novel inputs, and is incapable of generalizing – which is, however, the key aspect of machine learning [172, Chapter 5]. We will address this lack of generalization capabilities again in Section 8.7.

Normalizing Stress

Raw Stress as defined in Equation 8.1 depends on the scale of both the distances and the dissimilarities [71, Chapter 3]. Therefore, one typically normalizes Equation 8.1 by dividing through the sum of squared distances and taking the square root [233]. The resulting measure is called *Stress-1* and can be computed as follows:

$$\sigma_1(X) = \sqrt{\frac{\sum_{j_1=1}^N \sum_{j_2=j_1+1}^N (d_{j_1 j_2}(X) - \delta_{j_1 j_2})^2}{\sum_{j_1=1}^N \sum_{j_2=j_1+1}^N d_{j_1 j_2}^2(X)}} \quad (8.2)$$

Since it is normalized, Stress-1 is more suitable for comparing the representational quality of different configurations to each other.

Transforming dissimilarities into disparities

The optimization problem introduced so far is actually a special case of MDS, because it assumes, that the dissimilarities from the dissimilarity matrix Δ can directly be interpreted as geometric distances. Based on the elicitation method used for collecting these dissimilarities, this assumption might, however, be too strong. One can generalize raw Stress as defined in Equation 8.1 by introducing a so-called *representation function* f , which maps the dissimilarities $\delta_{j_1 j_2}$ to so-called *disparities*

$\hat{d}_{j_1 j_2}$ [71, Chapter 3], which represent the distances, that are expected for the given dissimilarities. MDS then tries to minimize the difference between the distances $d_{j_1 j_2}(X)$ and the disparities $\hat{d}_{j_1 j_2} = f(\delta_{j_1 j_2})$. The representation function f thus determines, how dissimilarities and distances should be related to each other. A generalized version of raw Stress can thus be written as follows:

$$\begin{aligned} \sigma_r(X) &= \sum_{j_1=1}^N \sum_{j_2=j_1+1}^N w_{j_1 j_2} \cdot (d_{j_1 j_2}(X) - f(\delta_{j_1 j_2}))^2 \\ &= \sum_{j_1=1}^N \sum_{j_2=j_1+1}^N w_{j_1 j_2} \cdot (d_{j_1 j_2}(X) - \hat{d}_{j_1 j_2})^2 \end{aligned} \tag{8.3}$$

In Equation 8.3, we have also introduced weights $w_{j_1 j_2}$, which assign an importance to each pair of stimuli. In most cases, all weights are set to one, but they can also be used to indicate the reliability of a given dissimilarity rating. As one can see, the choice of f heavily influences the value of raw Stress. Since MDS attempts to minimize raw Stress, different choices of f may lead to different optimal configurations X . Also Stress-1 can be adapted accordingly to consider disparities instead of dissimilarities:

Adding weights

$$\begin{aligned} \sigma_1(X) &= \sqrt{\frac{\sum_{j_1=1}^N \sum_{j_2=j_1+1}^N w_{j_1 j_2} \cdot (d_{j_1 j_2}(X) - f(\delta_{j_1 j_2}))^2}{\sum_{j_1=1}^N \sum_{j_2=j_1+1}^N w_{j_1 j_2} \cdot d_{j_1 j_2}^2(X)}} \\ &= \sqrt{\frac{\sum_{j_1=1}^N \sum_{j_2=j_1+1}^N w_{j_1 j_2} \cdot (d_{j_1 j_2}(X) - \hat{d}_{j_1 j_2})^2}{\sum_{j_1=1}^N \sum_{j_2=j_1+1}^N w_{j_1 j_2} \cdot d_{j_1 j_2}^2(X)}} \end{aligned} \tag{8.4}$$

It is important to note, that the representation function f maps from dissimilarities to distances, and not the other way around. Stress is therefore computed as a difference between expected and actual distances in the MDS space, not as a difference between expected and actual dissimilarity ratings. The reasons for this are twofold: Firstly, we try to optimize the configuration X of points in the MDS space. Since the coordinates of these points are our variables and the dissimilarity ratings are our constants, it makes sense to measure Stress based on the distances. If Stress depends on the configuration of points, we can use it as a feedback signal to improve those configurations. Secondly, the dissimilarities may only be ordinal scaled, which means that differences and ratios are not meaningful. We are therefore not allowed to compute terms like $\delta_{j_1 j_2} - \hat{\delta}_{j_1 j_2}$ [233]. Distances in the similarity space are, however, ratio scaled, and do therefore not pose such a problem.

Comparing distances rather than dissimilarities

Usually, the representation function f is not exactly determined, but it is only constrained to belong to a specific family of functions. For instance, one can require that f is a linear function of the dissimilarities, i.e., $f(\delta_{j_1 j_2}) = a \cdot \delta_{j_1 j_2} + b$, while leaving the values of the parameters a and b unspecified. Minimizing Stress as defined in Equations 8.3 or

Optimizing disparities

8.4 then involves both optimizing the coordinates in X and estimating the parameters of the representation function f . Most MDS algorithms minimize Stress in an iterative way, i.e., by repeatedly applying small updates to the coordinates in X . In this case, the introduction of the representation function f results in an iteratively alternating update of the configuration X and the optimal disparities $\hat{d}_{j_1 j_2}$. This alternating way of optimizing can be linked to the expectation maximization (EM) algorithm [119]. Please recall from Section 7.3.1, that the EM algorithm estimates parameters of a probabilistic model by iteratively alternating between the computation of probability values for the observed data points (expectation step) and the optimization of the model parameters based on these probabilities (maximization step). In the case of our MDS problem, computing the disparities $\hat{d}_{j_1 j_2}$ corresponds to the expectation step, while updating the coordinates X is equivalent to the maximization step.

Avoiding trivial solutions

As Borg and Groenen [71, Chapter 9] point out, a trivial solution when optimizing over both X and f is given by $X = 0$ and $f(\delta_{j_1 j_2}) = 0$ which guarantees minimal raw Stress. In order to avoid this degenerate solution, they propose to ensure that the $\hat{d}_{j_1 j_2}$ are normalized, e.g., by enforcing that $\sum_{j_1=1}^N \sum_{j_2=j_1+1}^N w_{j_1 j_2} \cdot \hat{d}_{j_1 j_2}^2 = \frac{N \cdot (N-1)}{2}$. This ensures, that $f(\delta_{j_1 j_2}) \neq 0$. Other authors such as Kruskal [233] avoid the degenerate solution by optimizing Stress-1 instead of raw Stress. In this case, dividing raw Stress by the sum of squared distances ensures, that the distances $d_{j_1 j_2}(X)$ do not become zero, which prevents $X = 0$.

8.2.2 Metric and Nonmetric MDS

Types of representation functions

Depending on the type of representation function f being used, one can distinguish different types of MDS. A broad distinction can be made into *metric MDS* (where f is continuous) and *nonmetric MDS* (where f is not continuous) [71, Chapter 9]. Typically, metric MDS assumes, that the dissimilarities are on a ratio or interval scale, while nonmetric MDS only assumes an ordinal scale.

Variants of metric MDS

Metric MDS includes the following three linear models, which constitute the most important metric MDS models [71, Chapter 9]:

Absolute MDS: dissimilarities are distance

- *Absolute MDS*, where $f(\delta_{j_1 j_2}) = \delta_{j_1 j_2}$. This is the simplified case considered in Equations 8.1 and 8.2. In absolute MDS, the dissimilarities are therefore treated as distances, and are assumed to be ratio scaled. Since there are no free parameters in f , we only need to optimize over X for minimizing Stress.

Ratio MDS: re-scaling allowed

- *Ratio MDS*, where $f(\delta_{j_1 j_2}) = a \cdot \delta_{j_1 j_2}$. Adding a free parameter in the form of a scaling factor a allows us to rescale the solution as needed. This means, that the order of magnitude of the dissimilarities is not important, but that we still assume a ratio scale: The ratio of any two dissimilarities should be equal to the ratio of the

corresponding distances. Optimizing over f involves finding an optimal scaling factor a , which minimizes Stress.

- *Interval MDS*, where $f(\delta_{j_1j_2}) = a \cdot \delta_{j_1j_2} + b$. By also allowing for an additive constant b , interval MDS is a further generalization of ratio MDS. Now, we expect only the ratios of differences ("intervals") on the dissimilarities to correspond to the ratio of differences on the distances.

*Interval MDS:
adding an intercept*

In all three cases, the optimization over f can be done by using a linear least-squares regression [71, Chapter 3]. In theory, also nonlinear representation functions are possible, e.g., logarithmic, exponential, or polynomial functions. This may, however, cause mathematical problems during the optimization procedures, because the optimal disparities might become negative, whereas distances are nonnegative by definition [71, Chapter 9]. One should therefore only use nonlinear representation functions, if there is a strong justification for this, e.g., based on the way, in which the dissimilarities were collected.

*Nonlinear metric
MDS*

Nonmetric MDS goes back to the pioneering work of Shepard [366] and Kruskal [233] and does not assume a continuous representation function. Typically, the representation function is thus not defined based on an equation (as for metric MDS), but based on constraints, which need to be fulfilled. Borg and Groenen [71, Chapter 9] make a distinction into further subtypes of nonmetric MDS:

*Variants of
nonmetric MDS*

- *Ordinal MDS*, where $\delta_{j_1j_2} < \delta_{j_3j_4} \Rightarrow \hat{d}_{j_1j_2} \leq \hat{d}_{j_3j_4}$. In other words, f is required to be a monotone function, but is left unspecified otherwise. The underlying assumption is that the recorded dissimilarities are based on a monotone distortion of the true semantic distances of item pairs [234]. The optimal disparities $\hat{d}_{j_1j_2}$ can be found by using a monotone regression from the dissimilarities to the distances. We will introduce Kruskal's monotone regression algorithm in Section 8.4. Please note, that also the linear models discussed in metric MDS are monotone functions, and can thus be considered a special case of ordinal MDS.
- *Nominal MDS*, where $\delta_{j_1j_2} = \delta_{j_3j_4} \Rightarrow \hat{d}_{j_1j_2} = \hat{d}_{j_3j_4}$. This type of MDS is only applicable, if the number of different values in the dissimilarity matrix is small, and if these different dissimilarities are only on a nominal scale, i.e., cannot be put in a meaningful order. In practice, nominal MDS is rarely used, because the resulting similarity spaces are typically not easy to interpret.
- *Ad hoc MDS* models, where f consists of different component functions. Typically, the exact constraints on f are based on theoretical considerations with respect to the collected dissimilarities and the respective domain. One example is an ordinal MDS approach, where f is also required to be negatively accelerated (i.e., to show a saturation tendency).

*Ordinal MDS: using
a monotone
representation
function*

*Nominal MDS: only
using a nominal scale*

*Ad hoc MDS:
application-specific
constraints*

Breaking ties in
ordinal MDS

In practice, the terms "nonmetric MDS" and "ordinal MDS" are often used interchangeably, because nominal MDS and ad hoc MDS models are rarely used. In ordinal MDS, a problem arises for ties in the dissimilarity matrix, i.e., for pairs of entries with $\delta_{j_1 j_2} = \delta_{j_3 j_4}$. The so-called *primary approach* to ties does not impose any constraints on the corresponding disparities, while the *secondary approach* requires, that then also $\hat{d}_{j_1 j_2} = \hat{d}_{j_3 j_4}$ (cf. [71, Chapter 9] and [233]). The primary approach is often preferable in practice, because ties in the dissimilarity matrix may be rather an artifact of the data collection process than a useful piece of information: Depending on the elicitation method and the number of decimal places used, ties may not indicate that the dissimilarity of the pairs is exactly identical. Moreover, an exact equality of distances in the similarity space may be quite a strong constraint in practice, which is much harder to fulfill than an ordering constraint. Most MDS algorithms therefore use the primary approach by default.

8.2.3 General Remarks

Incomplete
dissimilarities

Let us now turn to some general considerations with respect to multi-dimensional scaling. First of all, the matrix Δ of dissimilarities might be incomplete. In order to account for these missing entries, we can simply remove the corresponding dissimilarities $\delta_{j_1 j_2}$ and distances $d_{j_1 j_2}$ from the equation for computing Stress [233]. This can be achieved by setting the corresponding weight $w_{j_1 j_2}$ to zero in Equations 8.3 and 8.4. The optimization then only takes place over the given dissimilarities.

Processing multiple
ratings

If for each pair j_1, j_2 of stimuli, K different dissimilarity values are given (e.g., the ratings obtained from K different participants), MDS can be applied in different ways [71, Chapter 3]:

MDS first, data
merging second

- *Scale-then-Merge*: One can first apply MDS to each of the K matrices individually, and then compare the resulting similarity spaces, and merge them into a global space. Using this approach enables the researcher to investigate differences between the similarity spaces of different individuals. Since the ratings of a single participant may, however, not be reliable, the quality of the individual spaces might be relatively low. Moreover, one needs to specify, how exactly the different similarity spaces can be merged, i.e., how to obtain the final coordinates of the stimuli.

Data merging first,
MDS second

- *Merge-then-Scale*: One can also first merge the K matrices (e.g., by simply computing the average over all participants for each matrix entry [115, 248, 433]), and then apply MDS once on this global dissimilarity matrix. By creating a global dissimilarity matrix, the noise of the individual participants cancels out at least partially, which potentially leads to a solution with a higher quality. However, comparisons between individual participants are no longer possible. Moreover, if participants differed considerably with

respect to their rating behavior, the global dissimilarity matrix and thus also the resulting space may not be very informative.

- *Merge-while-Scaling*: A third option consists in redefining the representation function f as a mapping from $\{\delta_{j_1 j_2 k}\}_{k=1}^K$ to $\hat{d}_{j_1 j_2}$. In this case, the merging takes place as part of the optimization itself. While being an elegant and powerful way to incorporate additional constraints into the merging procedure, this approach requires a good understanding of the desired mapping and might come with higher computational costs when computing the optimal disparities.

An integrated approach

In practical applications, especially if one is interested in a *general* similarity space for a given domain, the merge-then-scale approach is the most straightforward one and thus quite popular. Also for the studies in Chapters 9, 10, and 11, we will follow this approach by using a single global dissimilarity matrix.

Merge-then-scale for general similarity spaces

Borg and Groenen [71, Chapter 9] also note, that instead of specifying a representation function f , which is included in the optimization procedure, one can also first apply a transformation to the dissimilarity matrix Δ , and then use absolute MDS on the transformed dissimilarities. This makes the optimization problem easier, because the parameters of f do not need to be estimated. However, this approach requires good domain knowledge for identifying the optimal transformation of the dissimilarities. In practice, one might thus need to tune this transformation in an external rather than internal way: Instead of finding the optimal transformation inside the MDS algorithm, one then needs to systematically vary the transformation and repeatedly apply MDS on the transformed data to find the transformation, which yields lowest Stress. Please note, that this external approach to transforming the dissimilarities is only easily applicable to metric MDS, where the transformation function can be specified by an explicit equation.

Transforming dissimilarities

8.2.4 An Illustrative Example

We will illustrate the three MDS algorithms presented in the following sections by using a simple example consisting of four fruit stimuli. In Table 8.2, we have provided a dissimilarity matrix Δ for the stimuli APPLE, BANANA, LEMON, and PEAR. Please note, that the dissimilarity ratings in Table 8.2 are not based on a psychological experiment, but are entirely artificial. They have been inspired by our fruit space example from Part I of this dissertation. Although MDS is typically applied to much larger dissimilarity matrices, limiting ourselves to four stimuli allows us to explicitly carry out computations, which illustrate the inner workings of the algorithms.

Stimuli and dissimilarities

	APPLE	BANANA	LEMON	PEAR
APPLE	0.0	2.5	3.5	1.4
BANANA	2.5	0.0	3.5	2.0
LEMON	3.5	3.5	0.0	3.7
PEAR	1.4	2.0	3.7	0.0

Table 8.2: Example dissimilarities for four fruit stimuli.

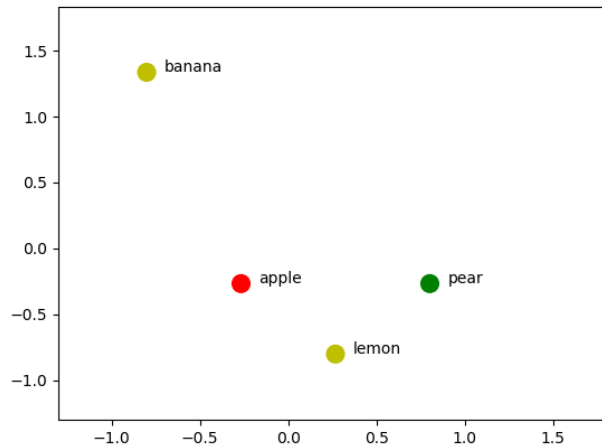


Figure 8.1: The initial configuration X_0 for our fruit example.

Initial configuration

For the two iterative MDS algorithms considered in Sections 8.4 and 8.5, we will compute an update for the initial configuration X_0 shown in Figure 8.1 and defined as follows:

$$X_0 = \begin{pmatrix} -0.2673 & -0.2673 \\ -0.8018 & 1.3363 \\ 0.2673 & -0.8018 \\ 0.8018 & -0.2673 \end{pmatrix}$$

*Distances based on
the initial
configuration*

As one can easily see by comparing Figure 8.1 to Table 8.2, X_0 does not accurately reflect the dissimilarities. For instance, APPLE should be closer to BANANA than to LEMON, but the opposite is the case. As discussed before, Stress is being used as a mathematical way of quantifying this representational error. We will thus use X_0 to illustrate the way in which Stress is computed. We first need to compute the Euclidean distances between all points in X_0 :

$$D(X_0) \approx \begin{pmatrix} 0 & 1.6903 & 0.7560 & 1.0691 \\ 1.6903 & 0 & 2.3905 & 2.2678 \\ 0.7560 & 2.3905 & 0 & 0.7559 \\ 1.0691 & 2.2678 & 0.7559 & 0 \end{pmatrix}$$

Let us first consider absolute MDS, where $\hat{d}_{j_1 j_2} = f(\delta_{j_1 j_2}) = \delta_{j_1 j_2}$, i.e., where no transformation of the dissimilarities takes place. We can use Equation 8.1 to compute the raw Stress of configuration X_0 :

Computing absolute raw Stress

$$\begin{aligned}\sigma_r(X_0) &= \sum_{j_1=1}^N \sum_{j_2=j_1+1}^N (d_{j_1 j_2}(X_0) - \delta_{j_1 j_2})^2 \\ &\approx (1.6903 - 2.5000)^2 + (0.7560 - 3.5000)^2 \\ &\quad + (1.0691 - 1.4000)^2 + (2.3905 - 3.5000)^2 \\ &\quad + (2.2678 - 2.0000)^2 + (0.7559 - 3.7000)^2 \\ &\approx 18.2653\end{aligned}$$

By normalizing $\sigma_r(X_0)$, we can get Stress-1 according to Equation 8.2:

Computing absolute Stress-1

$$\begin{aligned}\sigma_1(X_0) &= \sqrt{\frac{\sum_{j_1=1}^N \sum_{j_2=j_1+1}^N (d_{j_1 j_2}(X_0) - \delta_{j_1 j_2})^2}{\sum_{j_1=1}^N \sum_{j_2=j_1+1}^N d_{j_1 j_2}(X_0)^2}} \\ &= \sqrt{\frac{\sigma_r(X_0)}{\sum_{j_1=1}^N \sum_{j_2=j_1+1}^N d_{j_1 j_2}(X_0)^2}} \\ &\approx \sqrt{\frac{18.2653}{16.0001}} \approx 1.0684\end{aligned}$$

As noted before, it makes more sense to use ordinal MDS with psychological data. Let us therefore also consider raw Stress and Stress-1 for ordinal MDS. In this case, we have $\hat{d}_{j_1 j_2} = f(\delta_{j_1 j_2})$, where f represents an optimal monotone transformation of the dissimilarities, given the distances of the current configuration. In our case, the optimal disparities are given as follows:

Disparities for ordinal MDS

$$\hat{D} = \begin{pmatrix} 0 & 1.5714 & 1.5714 & 1.0691 \\ 1.5714 & 0 & 1.5732 & 1.5714 \\ 1.5714 & 1.5732 & 0 & 1.5732 \\ 1.0691 & 1.5714 & 1.5732 & 0 \end{pmatrix}$$

We will show in Section 8.4, how \hat{D} can be computed for this example, when we introduce Kruskal's algorithm for monotone regression. For now, let us just take \hat{D} as given. We can now compute raw Stress based on Equation 8.3, where we assume, that all weights are set to one:

Computing ordinal raw Stress

$$\begin{aligned}\sigma_r(X_0) &= \sum_{j_1=1}^N \sum_{j_2=j_1+1}^N w_{j_1 j_2} \cdot (d_{j_1 j_2}(X_0) - \hat{d}_{j_1 j_2})^2 \\ &\approx (1.6903 - 1.5714)^2 + (0.7560 - 0.5714)^2 \\ &\quad + (1.0691 - 1.0691)^2 + (2.3905 - 1.5732)^2 \\ &\quad + (2.2678 - 1.5714)^2 + (0.7559 - 1.5732)^2 \\ &\approx 2.5000\end{aligned}$$

Computing ordinal
Stress-1

By normalizing $\sigma_r(X_0)$, we can get again obtain Stress-1, this time according to Equation 8.4:

$$\begin{aligned}\sigma_1(X_0) &= \sqrt{\frac{\sum_{j_1=1}^N \sum_{j_2=j_1+1}^N w_{j_1 j_2} \cdot (d_{j_1 j_2}(X_0) - \hat{d}_{j_1 j_2})^2}{\sum_{j_1=1}^N \sum_{j_2=j_1+1}^N w_{j_1 j_2} \cdot d_{j_1 j_2}(X_0)^2}} \\ &= \sqrt{\frac{\sigma_r(X_0)}{\sum_{j_1=1}^N \sum_{j_2=j_1+1}^N w_{j_1 j_2} \cdot d_{j_1 j_2}(X_0)^2}} \\ &\approx \sqrt{\frac{2.5000}{16.0001}} \approx 0.3953\end{aligned}$$

Comparing the Stress
values

For both metric and nonmetric MDS, the normalization procedure causes Stress-1 to be considerably smaller than raw Stress. Moreover, we can observe, that both raw Stress and Stress-1 are smaller for the ordinal MDS problem than for the absolute MDS problem. This can be easily explained by the fact, that in ordinal MDS, we are allowed to transform the dissimilarities into disparities, while in absolute MDS no such transformation is permitted. Since we always use the optimal transformation for computing Stress, ordinal MDS has the clear advantage of being able to tune free parameters, which lowers the resulting value. These differences observed in our example highlight, that it is always important to only compare Stress values, which have been computed in the same way.

8.3 ANALYTICAL SOLUTIONS WITH CLASSICAL MDS

General properties of
classical MDS

In this section, we will describe *classical MDS*, which was developed by Torgerson [408] and Gower [174]. Classical MDS assumes, that the dissimilarities can be interpreted as distances (cf. absolute MDS, Section 8.2.2), and uses matrix transformations and an eigenvalue decomposition to find an analytical solution. In contrast to most other MDS algorithms, classical MDS thus does not require an iterative procedure, but is able to find a solution to the MDS problem in an analytical way. The remainder of this section is based on the treatment of classical MDS by Borg and Groenen [71, Chapters 7, 12]. We start with some mathematical background in Section 8.3.1, before deriving the algorithm itself in Section 8.3.2. Finally, in Section 8.3.3, we apply classical MDS to our fruit example from Section 8.2.4.

8.3.1 Mathematical Background

Overview

Before we can introduce the algorithm behind classical MDS, we need to cover some mathematical background about the underlying matrix transformations based on [71, Chapter 7]. This involves three ingredients, namely, the computation of distances through matrices, how to center a vector, and the eigenvalue decomposition.

Computing distances

Let us assume, that the coordinates of N points in an n -dimensional space are given in the form of an $N \times n$ matrix X . That means, that each row of the matrix contains the coordinates of one point, while each column of the matrix represents one coordinate axis of the underlying space. This is essentially identical to the design matrix considered in machine learning (cf. Section 5.1.2). The entry $x_i^{(j)}$ represents the i th coordinate of the j th point. We would like to compute an $N \times N$ distance matrix $D(X)$, where each entry $d_{j_1 j_2}(X)$ corresponds to the Euclidean distance between the points \vec{x}^{j_1} and \vec{x}^{j_2} from the coordinate matrix X :

$$d_{j_1 j_2}(X) = \sqrt{\sum_{i=1}^n \left(x_i^{(j_1)} - x_i^{(j_2)}\right)^2}$$

We can transform $D(X)$ into a matrix of squared distance $D^{(2)}(X)$ by taking the element-wise square:

Squared distances

$$\begin{aligned} d_{j_1 j_2}^2(X) &= \sum_{i=1}^n \left(x_i^{(j_1)} - x_i^{(j_2)}\right)^2 \\ &= \sum_{i=1}^n \left(\left(x_i^{(j_1)}\right)^2 + \left(x_i^{(j_2)}\right)^2 - 2x_i^{(j_1)}x_i^{(j_2)} \right) \end{aligned} \quad (8.5)$$

Let $B = XX^T$ (where X^T denotes the transpose of X) and \vec{c} be a vector containing the elements on the main diagonal of B (i.e., $c_j = b_{jj} = \sum_{i=1}^n (x_i^{(j)})^2$). B is called the *scalar product matrix* and will play an important role later on. Please note, that B is guaranteed to be symmetric (i.e., $b_{j_1 j_2} = b_{j_2 j_1}$). One can show, that $D^{(2)}(X)$ can be expressed in terms of B and \vec{c} :

Scalar product matrix

$$D^{(2)}(X) = \vec{c}\vec{1}^T + \vec{1}\vec{c}^T - 2B \quad (8.6)$$

Here, $\vec{1}$ is an $N \times 1$ vector consisting only of ones. To see, that the equivalence from Equation 8.6 holds, we spell out the right hand side:

Spelling out the equivalence

$$\begin{aligned} A &= \vec{c}\vec{1}^T + \vec{1}\vec{c}^T - 2B \\ &= \begin{pmatrix} b_{11} \\ b_{22} \\ \vdots \\ b_{NN} \end{pmatrix} \begin{pmatrix} 1 & 1 & \dots & 1 \end{pmatrix} \\ &\quad + \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} \begin{pmatrix} b_{11} & b_{22} & \dots & b_{NN} \end{pmatrix} \\ &\quad - 2 \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1N} \\ b_{21} & b_{22} & \dots & b_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ b_{N1} & b_{N2} & \dots & b_{NN} \end{pmatrix} \end{aligned}$$

$$\begin{aligned}
 &= \begin{pmatrix} b_{11} & b_{11} & \dots & b_{11} \\ b_{22} & b_{22} & \dots & b_{22} \\ \vdots & \vdots & \ddots & \vdots \\ b_{NN} & b_{NN} & \dots & b_{NN} \end{pmatrix} + \begin{pmatrix} b_{11} & b_{22} & \dots & b_{NN} \\ b_{11} & b_{22} & \dots & b_{NN} \\ \vdots & \vdots & \ddots & \vdots \\ b_{11} & b_{22} & \dots & b_{NN} \end{pmatrix} \\
 &\quad - 2 \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1N} \\ b_{21} & b_{22} & \dots & b_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ b_{N1} & b_{N2} & \dots & b_{NN} \end{pmatrix} \\
 &= \begin{pmatrix} 0 & b_{11} + b_{22} - 2b_{12} & \dots & b_{11} + b_{NN} - 2b_{1N} \\ b_{22} + b_{11} - 2b_{21} & 0 & \dots & b_{22} + b_{NN} - 2b_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ b_{NN} + b_{11} - 2b_{N1} & b_{NN} + b_{22} - 2b_{N2} & \dots & 0 \end{pmatrix}
 \end{aligned}$$

Computing distances with matrix transformations

Since $b_{j_1 j_2} = \sum_{i=1}^n x_i^{(j_1)} x_i^{(j_2)}$, we can easily see, that $a_{j_1 j_2} = \sum_{i=1}^n (x_i^{(j_1)})^2 + (x_i^{(j_2)})^2 - 2x_i^{(j_1)} x_i^{(j_2)}$, which is identical to Equation 8.5. We can thus use Equation 8.6 to transform the coordinate matrix X into the matrix of squared Euclidean distances $D^{(2)}(X)$ based only on matrix multiplications and matrix additions. If we afterwards apply the element-wise square root to $D^{(2)}(X)$, we get the matrix of Euclidean distances $D(X)$. Classical MDS provides us with a way of inverting this procedure, such that we can go from Euclidean distances to coordinates. Equation 8.6 will play a crucial role for this.

Centering a vector

We will later also need to center a vector \vec{x} around its mean value $\bar{x} = \frac{1}{N} \sum_{j=1}^N x_j$, i.e., to compute $\vec{x} - \vec{x}^{(avg)}$ with $x_j^{(avg)} = \bar{x}$. In order to do this, we first need to compute $\vec{x}^{(avg)}$ based on a given vector \vec{x} :

$$\vec{x}^{(avg)} = \frac{1}{N} \vec{1}\vec{1}^T \vec{x}$$

Computing the mean

In order to see, why this equivalence is true, consider that $\vec{1}\vec{1}^T$ results in an $N \times N$ matrix filled with ones. Multiplying \vec{x} with this matrix of ones results in a column vector, where each entry corresponds to $\sum_{j=1}^N x_j$. Dividing this vector by N gives us a column vector where each entry is equal to \bar{x} , i.e., the arithmetic mean of \vec{x} .

Subtraction as matrix multiplication

Now in order to compute $\vec{x} - \vec{x}^{(avg)}$, we simply need to do an element-wise subtraction. This can also be rewritten in the following way, where I is the identity matrix of size $N \times N$:

$$\vec{x} - \vec{x}^{(avg)} = \vec{x} - \frac{1}{N} \vec{1}\vec{1}^T \vec{x} = \underbrace{\left(I - \frac{1}{N} \vec{1}\vec{1}^T \right)}_{=J} \vec{x} \tag{8.7}$$

The centering matrix

The matrix $J = I - \frac{1}{N} \vec{1}\vec{1}^T$ is called the *centering matrix* and will play a role when transforming the distance matrix D into the scalar product matrix B . Please note, that J is symmetric, i.e., $J^T = J$.

A final ingredient needed for classical MDS is the eigenvalue decomposition of an $N \times N$ matrix A , which has the following form (cf. Section 5.3.2):

Eigenvalue decomposition

$$A = Q \Lambda Q^T \tag{5.19}$$

Please recall from Section 5.3.2, that Λ is a diagonal matrix of *eigenvalues* and Q is a matrix of *eigenvectors*, both being of size $N \times N$. Each column of Q corresponds to one eigenvector $\vec{q}^{(j)}$, and the corresponding diagonal entry λ_j of Λ to its associated eigenvalue, such that $A\vec{q}^{(j)} = \lambda_j\vec{q}^{(j)}$. If the matrix A is symmetric and positive semi-definite (e.g., because $A = Y Y^T$), then all eigenvalues λ_j are guaranteed to be non-negative. In Section 5.3.2, we have also seen, that based on Equation 5.19, we can rewrite A as a sum of individual matrices, which are based on the different eigenvectors and eigenvalues:

Eigenvectors and eigenvalues

$$\begin{aligned} A = Q \Lambda Q^T &= \begin{pmatrix} \lambda_1 \vec{q}^{(1)} & \lambda_2 \vec{q}^{(2)} & \dots & \lambda_N \vec{q}^{(N)} \end{pmatrix} \begin{pmatrix} (\vec{q}^{(1)})^T \\ (\vec{q}^{(2)})^T \\ \vdots \\ (\vec{q}^{(N)})^T \end{pmatrix} \\ &= \lambda_1 \vec{q}^{(1)} (\vec{q}^{(1)})^T + \lambda_2 \vec{q}^{(2)} (\vec{q}^{(2)})^T + \dots + \lambda_N \vec{q}^{(N)} (\vec{q}^{(N)})^T \end{aligned} \tag{5.20}$$

If the eigenvalues are sorted in descending order, then the individual matrices in Equation 5.20 are ordered in such a way, that the first matrices have a larger impact on the overall result. In Section 5.3.2, we therefore argued, that we can approximate A based on the first n eigenvalues and their associated eigenvectors. The eigenvalue decomposition will play an important role when extracting the coordinate matrix from the scalar product matrix.

Approximation with eigenvalue decomposition

8.3.2 The Algorithm

In order to understand the algorithm for classical MDS, let us describe, how to transform a matrix $D^{(2)}(X)$ of squared Euclidean distances into a matrix X of coordinates [71, Chapter 12]. We will assume, that the points are centered around the origin, i.e., that X has column means equal to zero. This is not a restriction in practice, because a centering around the origin can be easily obtained through a translation, which leaves the distances intact.

From distances to coordinates

First of all, we apply Equation 8.6 to translate $D^{(2)}(X)$ into the scalar product matrix B . Moreover, we multiply with the centering matrix J from both sides:

Applying Equation 8.6

$$\begin{aligned} J D^{(2)}(X) J &= J \left(\vec{c}\vec{1}^T + \vec{1}\vec{c}^T - 2B \right) J \\ &= \left(J \vec{c}\vec{1}^T J \right) + \left(J \vec{1}\vec{c}^T J \right) + (2 J B J) \end{aligned}$$

Since $J\vec{x} = \vec{x} - \vec{x}^{(avg)}$ (cf. Equation 8.7), it is easy to see, that $J\vec{1} =$

Centering a vector of ones

$\bar{1} - \bar{1}^{(avg)} = 0$. Moreover, J is symmetric, so we analogously get $\bar{1}^T J = 0$. Therefore, the first two terms on the right hand side of the equation become zero. We thus get the following intermediate result:

$$J D^{(2)}(X) J = -2 J B J$$

Rescaling We can now multiply both sides of the equation with $-\frac{1}{2}$:

$$-\frac{1}{2} J D^{(2)}(X) J = J B J$$

Centering the scalar product matrix

We assumed earlier, that X is column-centered, therefore applying the centering matrix J to X does not change anything: $JX = X$. Moreover, J is symmetric (i.e., $J = J^T$) and $B = XX^T$. We thus get the following result:

$$J B J = J X X^T J = (J X) (J^T X)^T = (J X) (J X)^T = X X^T = B$$

From distances to the scalar product matrix

Overall, we thus obtain the following transformation for translating the matrix $D^{(2)}(X)$ of squared Euclidean distances into the scalar product matrix B :

$$B = -\frac{1}{2} J D^{(2)}(X) J \tag{8.8}$$

Double centering

Since we apply the centering matrix J twice to $D^{(2)}(X)$ in order to arrive at B , applying Equation 8.8 is also called *double centering*.

From the scalar product matrix to coordinates

After having successfully extracted the scalar product matrix $B = XX^T$, we now need to recover the coordinate matrix X . This can be done using an eigenvalue decomposition of B [71, Chapter 7]:

$$B = Q \Lambda Q^T$$

Taking the square root of the eigenvalues

Any matrix of the form $A = YY^T$ is symmetric and positive semi-definite (i.e., has only non-negative eigenvalues). We therefore know, that all λ_j are non-negative. This allows us to define $\Lambda^{0.5}$ as the diagonal matrix obtained by taking the element-wise square root of Λ . It is easy to see that $\Lambda = \Lambda^{0.5} \Lambda^{0.5}$. Applying this to our eigenvalue decomposition results in the following observation:

$$\begin{aligned} B &= Q \Lambda Q^T = Q \Lambda^{0.5} \Lambda^{0.5} Q^T \\ &= \underbrace{(Q \Lambda^{0.5})}_{=U} \underbrace{(Q \Lambda^{0.5})^T}_{=U} \end{aligned}$$

Obtaining rotated coordinates

This means, that the matrix $U = Q\Lambda^{0.5}$ contains coordinates, which give rise to the scalar product matrix B , i.e., which respect the squared Euclidean distances given in $D^{(2)}(X)$. This also means, that the distances between the points in U and the distances between the points in X are identical. However, one should note that in general $U \neq X$, because the coordinates are only determined up to a rotation around

Algorithm 8.1: Classical MDS algorithm as described by Borg and Groenen [71, Chapter 12].

Input: Dissimilarity matrix Δ , number of dimensions n

Output: Coordinate matrix X

- 1 Apply the element-wise square to Δ in order to obtain $\Delta^{(2)}$
 - 2 Apply double centering: $B_\Delta = -\frac{1}{2}J\Delta^{(2)}J$
 - 3 Compute eigenvalue decomposition $B_\Delta = Q\Lambda Q^T$
 - 4 Take the n largest positive eigenvalues and their corresponding eigenvectors to construct Λ_+ and Q_+
 - 5 Estimate coordinate matrix $X = Q_+\Lambda_+^{0.5}$
 - 6 **return** X
-

the origin. If necessary, one can find a rotation matrix R to transform U into X , but this is rarely done in practice, since we are mostly interested in the relative configuration of the points.

So if we have a matrix $D(X)$ of Euclidean distances, we can recover the relative coordinates of the original points through double-centering and an eigenvalue decomposition. The dissimilarity matrix Δ used for MDS does, however, in general not contain Euclidean distances. Nevertheless, one can apply the procedure from above to estimate a configuration of points, whose distances reflect the dissimilarities as closely as possible. The corresponding procedure for applying classical MDS is shown in Algorithm 8.1 [71, Chapter 12].

The main difference between Algorithm 8.1 and our prior mathematical derivation concerns line 4: Since the dissimilarities in Δ are not necessarily Euclidean distances, the eigenvalue decomposition of B_Δ may contain some eigenvalues, which are zero or negative. Negative eigenvalues are, however, problematic, because taking their square root is not permitted. In classical MDS, these non-positive eigenvalues are therefore ignored as error, and their corresponding eigenvectors are not taken into account when constructing the coordinate matrix X .

One should furthermore note, that taking into consideration only the n largest eigenvalues can be seen as a form of approximating B_Δ . Moreover, one can show, that classical MDS optimizes the so-called *Strain* $\|XX^T - B_\Delta\|^2$, i.e., the reconstruction error of X with respect to B_Δ . Thus, the solution obtained by classical MDS tries to mimic the scalar products of the coordinates as closely as possible. This is, however, not equivalent to minimizing absolute *Stress*, where the coordinates try to approximate the distances as closely as possible.

Classical MDS differs from most other MDS approaches by providing an analytical solution, which can be computed in a deterministic, non-iterative way. Moreover, the solutions provided by classical MDS are "nested" in the sense, that the first $n' < n$ dimensions of the n -dimensional solution are identical to the n' -dimensional solution. This nesting of dimensions is based on the fact, that the eigenvalue

Classical MDS: assume, that Δ contains distances

Ignoring non-positive eigenvalues

Taking n eigenvalues and optimizing Strain

Nested solutions

decomposition is always the same, independent of the dimensionality of the target space (cf. line 3 in Algorithm 8.1). Such a nesting of dimensions is not expected to happen for other MDS algorithms, which iteratively minimize Stress – there, the dimensionality of the space is defined in the beginning, and a minimum of Stress is searched in the given n -dimensional space.

Relation to PCA

As you may have noted, both the principal components analysis (PCA) discussed in Section 5.3.2 and classical MDS make use of an eigenvalue decomposition of the scalar product matrix $B = XX^T$, and are thus tightly related. However, they start from different inputs and have slightly different goals: A PCA starts with a design matrix $X \in \mathbb{R}^{N \times n}$ and tries to project it into a lower-dimensional space by computing an eigenvalue decomposition of the scalar product matrix XX^T . In the case of a PCA, the original coordinates are thus already given. In classical MDS, on the other hand, we start with a dissimilarity matrix Δ , which we interpret as containing Euclidean distances. We then apply double centering to obtain (an approximation of) XX^T , and an eigenvalue decomposition to recover the underlying coordinate matrix X . Here, obtaining coordinates from distances is our main focus, and finding a low-dimensional solution is only a secondary issue.

Properties of the configuration

Finally, let us comment on the configuration obtained by classical MDS: Since we assume, that X is column-centered, the solutions provided by classical MDS are always centered around the origin. They are thus in some sense normalized by removing one degree of freedom (namely, the translation with respect to the origin). The scale of the solution (i.e., the average distance of the points to the origin) is determined by the values of the dissimilarities: Classical MDS provides a solution, where the distances between the points try to reflect the numerical values of the dissimilarity matrix as closely as possible. A normalization with respect to the scale can thus be achieved either by normalizing the dissimilarity matrix or by normalizing the resulting space afterwards. Finally, the resulting configurations are not determined with respect to their rotation around the origin. This means, that semantically meaningful directions do not necessarily coincide with the axes of the coordinate system.

8.3.3 An Illustrative Example

Squaring dissimilarities

Let us now finally take a look at our fruit example from Section 8.2.4 in order to illustrate how classical MDS works. We start in line 1 of Algorithm 8.1 by computing the element-wise square of the dissimilarities:

$$\Delta = \begin{pmatrix} 0 & 2.5 & 3.5 & 1.4 \\ 2.5 & 0 & 3.5 & 2.0 \\ 3.5 & 3.5 & 0 & 3.7 \\ 1.4 & 2.0 & 3.7 & 0 \end{pmatrix}$$

$$\Delta^{(2)} = \begin{pmatrix} 0 & 6.25 & 12.25 & 1.96 \\ 6.25 & 0 & 12.25 & 4.0 \\ 12.25 & 12.25 & 0 & 13.69 \\ 1.96 & 4.0 & 13.69 & 0 \end{pmatrix}$$

Next, we need to apply double-centering to $\Delta^{(2)}$ in order to obtain the estimate of the scalar product matrix (cf. line 2 in Algorithm 8.1 and Equation 8.8):

Applying double-centering

$$\begin{aligned} B_{\Delta} &= -\frac{1}{2}J\Delta^{(2)}J \\ &= -\frac{1}{2} \begin{pmatrix} 0.75 & -0.25 & -0.25 & -0.25 \\ -0.25 & 0.75 & -0.25 & -0.25 \\ -0.25 & -0.25 & 0.75 & -0.25 \\ -0.25 & -0.25 & -0.25 & 0.75 \end{pmatrix} \\ &\quad \begin{pmatrix} 0 & 6.25 & 12.25 & 1.96 \\ 6.25 & 0 & 12.25 & 4.0 \\ 12.25 & 12.25 & 0 & 13.69 \\ 1.96 & 4.0 & 13.69 & 0 \end{pmatrix} \\ &\quad \begin{pmatrix} 0.75 & -0.25 & -0.25 & -0.25 \\ -0.25 & 0.75 & -0.25 & -0.25 \\ -0.25 & -0.25 & 0.75 & -0.25 \\ -0.25 & -0.25 & -0.25 & 0.75 \end{pmatrix} \\ &\approx \begin{pmatrix} 1.9650 & -0.9050 & -1.9438 & 0.8838 \\ -0.9050 & 2.4750 & -1.6888 & 0.1188 \\ -1.9438 & -1.6888 & 6.3975 & -2.7650 \\ 0.8838 & 0.1188 & -2.7650 & 1.7625 \end{pmatrix} \end{aligned}$$

We are now in line 3 of Algorithm 8.1, where we now compute the eigenvalue decomposition of B_{Δ} into Q and Λ :

Eigenvalue decomposition

$$\begin{aligned} B_{\Delta} = Q\Lambda Q^T \text{ with } \Lambda &\approx \begin{pmatrix} 8.6478 & 0 & 0 & 0 \\ 0 & 3.2420 & 0 & 0 \\ 0 & 0 & 0.7102 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \\ \text{and } Q &\approx \begin{pmatrix} 0.2731 & -0.5761 & -0.5861 & -0.5 \\ 0.2023 & 0.7993 & -0.2649 & -0.5 \\ -0.8588 & -0.0654 & 0.0907 & -0.5 \\ 0.3834 & -0.1578 & 0.7603 & -0.5 \end{pmatrix} \end{aligned}$$

As we can see, there are three positive eigenvalues, thus we can maximally construct a three-dimensional solution. For visualization purposes, we confine ourselves to $n = 2$, i.e., a two-dimensional similarity space. We thus obtain Λ_+ and Q_+ as follows:

Taking the two largest eigenvalues

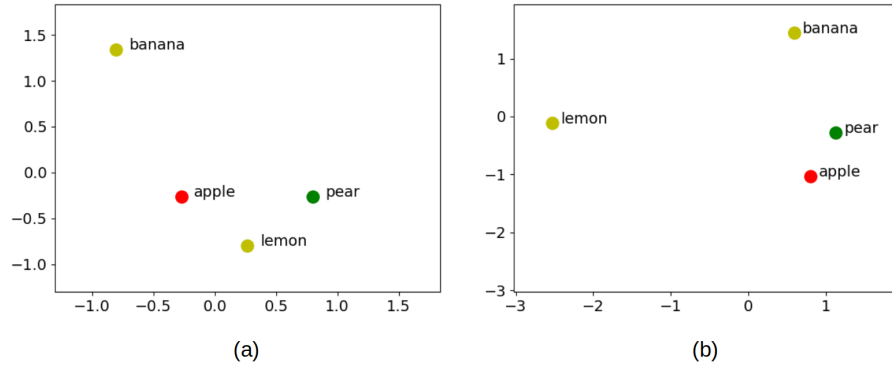


Figure 8.2: (a) Initial configuration from Section 8.2.4. (b) Two-dimensional similarity space obtained by classical MDS for our fruit example.

	Absolute MDS		Ordinal MDS	
	$\sigma_r(X)$	$\sigma_1(X)$	$\sigma_r(X)$	$\sigma_1(X)$
Initial configuration	18.2653	1.0684	2.5000	0.3953
Classical MDS	0.3792	0.08929	0.0000	0.0000

Table 8.3: Stress values for the initial configuration from Section 8.2.4 and for the solution provided by classical MDS.

$$\Lambda_+ \approx \begin{pmatrix} 8.6478 & 0 \\ 0 & 3.2420 \end{pmatrix} \text{ and } Q_+ \approx \begin{pmatrix} 0.2731 & -0.5761 \\ 0.2023 & 0.7993 \\ -0.8588 & -0.0654 \\ 0.3834 & -0.1578 \end{pmatrix}$$

Estimating coordinates

According to line 5 of Algorithm 8.1, we can now estimate our coordinate matrix X as follows:

$$X = Q_+ \Lambda_+^{0.5} \approx \begin{pmatrix} 0.2731 & -0.5761 \\ 0.2023 & 0.7993 \\ -0.8588 & -0.0654 \\ 0.3834 & -0.1578 \end{pmatrix} \begin{pmatrix} 2.9407 & 0 \\ 0 & 1.8006 \end{pmatrix} \approx \begin{pmatrix} 0.8031 & -1.0374 \\ 0.5949 & 1.4392 \\ -2.5254 & -0.1177 \\ 1.1275 & -0.2841 \end{pmatrix}$$

Interpreting the results

This configuration of points is illustrated in Figure 8.2. When comparing Figure 8.2 to the original dissimilarities from Table 8.2, we note, that the overall arrangement seems to capture the dissimilarities quite well. We can check this intuition by computing raw Stress and Stress-1, both based on an absolute MDS model and an ordinal MDS model. The results can be seen in Table 8.3 and are compared to the respective Stress values of the initial configuration from Section 8.2.4. As we can

Algorithm 8.2: General gradient descent procedure.**Input:** Function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, learning rate α , threshold ϵ **Output:** Point $\vec{x} \in \mathbb{R}^n$ which approximately minimizes f

- 1 Set \vec{x} to an arbitrary starting value
- 2 **repeat**
- 3 Compute gradient $\nabla_{\vec{x}} f(\vec{x}) = \left(\frac{\partial f(\vec{x})}{\partial x_1}, \dots, \frac{\partial f(\vec{x})}{\partial x_n} \right)$
- 4 Update $\vec{x} = \vec{x} - \alpha \cdot \nabla_{\vec{x}} f(\vec{x})$
- 5 **until** $|\nabla_{\vec{x}} f(\vec{x})| < \epsilon$
- 6 **return** \vec{x}

see, absolute Stress is very small for the solution obtained through classical MDS and considerably better than for the initial configuration. Moreover, classical MDS is able to push ordinal Stress to zero, i.e., the distances are indeed a monotone transformation of the dissimilarities. This is a nice result, especially given, that classical MDS does not explicitly optimize any of the Stress variants shown in Table 8.3. One should, however, keep in mind, that the results obtained on our very small example do not necessarily carry over to larger and more complex dissimilarity matrices.

8.4 KRUSKAL'S GRADIENT DESCENT APPROACH

The first thoroughly formalized algorithm for nonmetric MDS was developed by Kruskal [233, 234]. It optimizes Stress-1 (i.e., raw Stress normalized by the squared distances of the points) by using a form of gradient descent. This section is based almost entirely on [234], using, however, the notation introduced in Section 8.2.1.

*Nonmetric MDS
with gradient descent*

In Section 8.4.1, we give a short rehash of gradient descent, which has already been introduced in Section 5.2.2. In Section 8.4.2, we then introduce Kruskal's algorithm for monotone regression. Section 8.4.3 applies this monotone regression to our fruit example. We then explain Kruskal's overall MDS algorithm in Section 8.4.4 and illustrate its inner workings with our fruit example in Section 8.4.5.

8.4.1 Gradient Descent for Stress

Please recall from Section 5.2.2, that gradient descent is an iterative optimization method. It is applied mainly, if analytical solutions are either not possible or too costly to compute [71, Chapter 8]. The general procedure of gradient descent is re-printed in Algorithm 8.2 and illustrated in Figure 8.3.

*Iterative
optimization with
gradient descent*

The gradient $\nabla_{\vec{x}} f(\vec{x})$ (which corresponds in the one-dimensional case to the derivative $f'(x)$) of a function $f(\vec{x})$ is zero in all of its so-called *stationary points* (i.e., all minima, maxima, and saddle points). The function $f(x)$ in Figure 8.3 has two local minima (x_1 and the global minimum

*Gradient descent in a
nutshell*

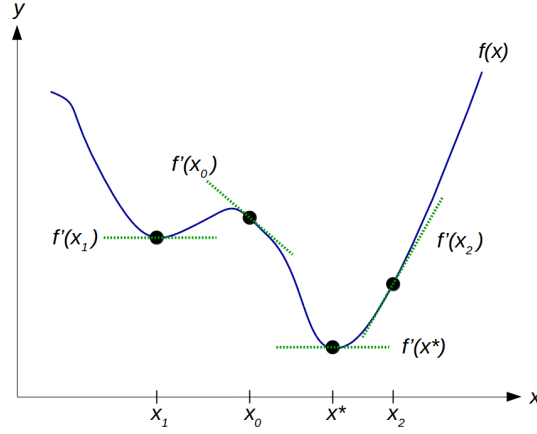


Figure 8.3: Illustration of gradient descent for a function f .

x^*). In both of these points, the derivative of f is zero as indicated by the tangents. Gradient descent aims at finding the global minimum by iteratively improving its current estimate, namely by moving in the opposite direction of the gradient. For instance, if our current estimate of the global minimum in Figure 8.3 is x_0 , gradient descent slightly increases the current estimate to $x + \Delta x$, because $f'(x_0) < 0$. In order to define the step size used for updating x , one typically multiplies the gradient with a constant α , which is called the *learning rate*. Thus, our update rule is $x_{t+1} = x_t - \alpha \cdot f'(x)$ (cf. line 4 in Algorithm 8.2). Gradient descent iteratively makes such small updates, until the gradient becomes close to zero, indicating, that a local minimum has been found.

Minimizing Stress-1

In order to apply gradient descent to the minimization of Stress, we need to compute the respective gradient of the Stress function. Kruskal's algorithm minimizes Stress-1, which was defined in Equation 8.4 as follows, assuming that the disparities $\hat{d}_{j_1 j_2} = f(\delta_{j_1 j_2})$ are given:

$$\sigma_1(X) = \sqrt{\frac{\sum_{j_1=1}^N \sum_{j_2=j_1+1}^N w_{j_1 j_2} \cdot (d_{j_1 j_2}(X) - \hat{d}_{j_1 j_2})^2}{\sum_{j_1=1}^N \sum_{j_2=j_1+1}^N w_{j_1 j_2} \cdot d_{j_1 j_2}^2(X)}} = \sqrt{\frac{\sigma_r(X)}{T(X)}}$$

Gradient for Stress-1

According to Kruskal [234], the gradient $g_i^{(j)} = \frac{\partial \sigma_1(X)}{\partial x_i^{(j)}}$, i.e., with respect to the i th coordinate of the j th point in the configuration, can be computed as follows:

$$g_i^{(j)} = \sigma_1(X) \sum_{j_1=1}^N \sum_{j_2=1}^N \left[(\delta^{(j j_1)} - \delta^{(j j_2)}) \left(\frac{d_{j_1 j_2}(X) - \hat{d}_{j_1 j_2}}{\sigma_r(X)} - \frac{d_{j_1 j_2}(X)}{T(X)} \right) \frac{x_i^{(j_1)} - x_i^{(j_2)}}{d_{j_1 j_2}(X)} \right] \quad (8.9)$$

Here, $\delta^{(j_1 j_2)}$ is the Kronecker delta with $\delta^{(j_1 j_2)} = 1$ if $j_1 = j_2$ and $\delta^{j_1 j_2} = 0$ if $j_1 \neq j_2$. We can thus compute the $N \times n$ matrix G corresponding to the overall gradient $\nabla_X \sigma_1(X)$ by using Equation 8.9 to

populate its entries $g_i^{(j)}$. Our update rule for the coordinate matrix then becomes $X = X - \alpha \cdot G$, i.e., an element-wise update based on the negated gradient.

8.4.2 Monotone Regression

As mentioned in Section 8.2, all iterative optimization techniques applied to the MDS problem need to make an alternating update of the configuration X and the optimal disparities \hat{D} . Thus, we also need to specify how the optimal disparities for a given configuration can be derived. As mentioned before, this can be done with a monotone regression. Kruskal [234] has provided Algorithm 8.3 for computing such a monotone regression. Its basic idea is, that the dissimilarities are partitioned into consecutive blocks. The disparity value \hat{d} assigned to each block corresponds to the average across the respective distances $d_{j_1, j_2}(X)$ in this block. The algorithm ensures, that these disparities are non-decreasing by iteratively merging neighboring blocks, which violate the monotonicity requirement.

*Block-based
monotone regression*

Let us now take a closer look at Algorithm 8.3. We define a *block* $b = (\delta(b), d(b))$ as a tuple of two sets, namely the set of dissimilarities $\delta(b)$ and the set of distances $d(b)$. We furthermore define the disparity of a block b as $\hat{d}(b) = \frac{1}{|d(b)|} \sum_{d \in d(b)} d$. In the beginning, each block contains only a single dissimilarity value along with its corresponding distance (see lines 1-5). All these blocks are kept in a sorted list B (line 6). Please note, that if for two blocks $b \neq b'$ we have $\delta(b) = \delta(b')$ (i.e., there is a tie in the dissimilarity matrix), we use $d(b)$ as a secondary sorting key. This corresponds to using the primary approach to ties (cf. Section 8.2.2).

*Initializing the list of
blocks*

A block is called *up-satisfied*, if its disparity is less than the disparity of the next block (see line 10). It is called *down-satisfied*, if its disparity is greater than the disparity of the previous block (see line 11). A block can be either *inactive*, *up-active*, or *down-active*.

*Satisfaction and
activity*

If a block is up-active, we try to ensure, that it is also up-satisfied by merging it with the next block if necessary (lines 14-19). On the other hand, if a block is down-active, we try to ensure, that it is down-satisfied, again if necessary by merging it with the previous block (lines 20-26). Two blocks are merged by merging their sets of dissimilarities and their sets of distances, respectively. This results in an updated disparity value for the resulting block, which lies between the original disparities of the original blocks, and removes the violation of the monotonicity requirement. By repeatedly merging blocks, we ensure that all violations are resolved and in the end, dissimilarities are mapped to a monotonically non-decreasing sequence of disparities.

Merging blocks

If such a merge occurs, the direction of activity is reversed (cf. lines 18 and 24). The algorithm then checks in the next step, whether the updated disparity value of the merged block has "broken" the satisfaction status with respect to the other direction. If an active block is both up-satisfied

Changing directions

Algorithm 8.3: Monotone regression as described in [234].

Input: Dissimilarity matrix Δ , distance matrix $D(X)$
Output: Disparity matrix \hat{D}

```

1  $B = \langle \rangle$ 
2 forall  $j_1 \leq j_2$  do
3    $b = (\{\delta_{j_1 j_2}\}, \{d_{j_1 j_2}(X)\})$ 
4   Append  $b$  to  $B$ 
5 end
6 Sort  $B$  in ascending order based on  $\delta(b)$ 
7 Mark first block of  $B$  as up-active
8 while there is an active block  $b \in B$  do
9   Denote by  $b^-$  and  $b^+$  the previous and the next block,
   respectively
10   $s_{up} \Leftrightarrow (\hat{d}(b) < \hat{d}(b^+)) \vee b$  is the last block
11   $s_{down} \Leftrightarrow (\hat{d}(b^-) < \hat{d}(b)) \vee b$  is the first block
12  if  $s_{up} \wedge s_{down}$  then
13    | Mark  $b$  as inactive, mark  $b^+$  as up-active
14  else if  $b$  is up-active then
15    | if  $s_{up}$  then
16      | Mark  $b$  as down-active
17    | else
18      | Merge  $b$  with  $b^+$ , mark resulting block as down-active
19    | end
20  else //  $b$  is down-active
21    | if  $s_{down}$  then
22      | Mark  $b$  as up-active
23    | else
24      | Merge  $b$  with  $b^-$ , mark resulting block as up-active
25    | end
26  end
27 end
28 Initialize  $N \times N$  matrix  $\hat{D}$  with entries  $\hat{d}_{j_1 j_2} = 0$ 
29 forall  $j_1 \leq j_2$  do
30   Find block  $b \in B$ , such that  $\delta_{j_1 j_2} \in \delta(b)$ 
31    $\hat{d}_{j_1 j_2} = \hat{d}_{j_1 j_2} = \hat{d}(b)$ 
32 end
33 return  $\hat{D}$ 

```

and down-satisfied, no further merging is required at this point. Thus, it becomes inactive and we consider the next block (lines 12-13).

The algorithm starts by marking the first block as up-active (line 7). It terminates when all blocks are both up-satisfied and down-satisfied and thus no block is active any more (line 8). In the end, we copy the disparities from the blocks into the disparity matrix (lines 28-33).

The main loop

8.4.3 Example for Monotone Regression

In order to illustrate, how Algorithm 8.3 works, let us apply it to the initial configuration X_0 from Section 8.2.4. We therefore assume the following variables to be given:

Starting point

$$D(X_0) \approx \begin{pmatrix} 0 & 1.6903 & 0.7560 & 1.0691 \\ 1.6903 & 0 & 2.3905 & 2.2678 \\ 0.7560 & 2.3905 & 0 & 0.7559 \\ 1.0691 & 2.2678 & 0.7559 & 0 \end{pmatrix}$$

$$\Delta = \begin{pmatrix} 0 & 2.5 & 3.5 & 1.4 \\ 2.5 & 0 & 3.5 & 2.0 \\ 3.5 & 3.5 & 0 & 3.7 \\ 1.4 & 2.0 & 3.7 & 0 \end{pmatrix}$$

First, we have to construct the sorted list B of blocks (lines 1-6) and mark the first block as up-active (line 7). In our example, this results in the following state of B , where each column represents one block b , whose entries $\delta(b)$ and $\tilde{d}(b)$ are shown in separate rows:

A sorted list of blocks

$\delta_{14} = 1.4$	$\delta_{24} = 2.0$	$\delta_{12} = 2.5$	$\delta_{13} = 3.5$	$\delta_{23} = 3.5$	$\delta_{34} = 3.7$
1.0691	2.2678	1.6903	0.7560	2.3905	0.7559
up-active					

The first block is both up-satisfied and down-satisfied (lines 10-11), therefore, we mark the second block as up-active (lines 12-13):

Processing the first block

$\delta_{14} = 1.4$	$\delta_{24} = 2.0$	$\delta_{12} = 2.5$	$\delta_{13} = 3.5$	$\delta_{23} = 3.5$	$\delta_{34} = 3.7$
1.0691	2.2678	1.6903	0.7560	2.3905	0.7559
	up-active				

The second block is up-active and down-satisfied, but not up-satisfied, because its right neighbor has a lower disparity of $1.6903 < 2.2678$. We therefore execute line 18, merge the blocks, and mark the result as down-active:

Merging upwards

$\delta_{14} = 1.4$	$\delta_{24} = 2.0, \delta_{12} = 2.5$	$\delta_{13} = 3.5$	$\delta_{23} = 3.5$	$\delta_{34} = 3.7$
1.0691	1.9791	0.7560	2.3905	0.7559
	down-active			

Changing directions

The current block is now down-active and down-satisfied, hence we change it to being up-active (line 24):

$\delta_{14} = 1.4$	$\delta_{24} = 2.0, \delta_{12} = 2.5$	$\delta_{13} = 3.5$	$\delta_{23} = 3.5$	$\delta_{34} = 3.7$
1.0691	1.9791	0.7560	2.3905	0.7559
	up-active			

Another upward merge

Again, we have the situation that, the current block is up-active, but not up-satisfied, causing us again to merge it with the next block and marking the result as down-active (line 18):

$\delta_{14} = 1.4$	$\delta_{24} = 2.0, \delta_{12} = 2.5, \delta_{13} = 3.5$	$\delta_{23} = 3.5$	$\delta_{34} = 3.7$
1.0691	1.5714	2.3905	0.7559
	down-active		

Another change of directions

The resulting block is still down-satisfied, so we can mark it again as up-active (line 24):

$\delta_{14} = 1.4$	$\delta_{24} = 2.0, \delta_{12} = 2.5, \delta_{13} = 3.5$	$\delta_{23} = 3.5$	$\delta_{34} = 3.7$
1.0691	1.5714	2.3905	0.7559
	up-active		

Moving on

Now, our current block is both up-satisfied and down-satisfied. We can therefore leave it as is and move on to the next block (line 13):

$\delta_{14} = 1.4$	$\delta_{24} = 2.0, \delta_{12} = 2.5, \delta_{13} = 3.5$	$\delta_{23} = 3.5$	$\delta_{34} = 3.7$
1.0691	1.5714	2.3905	0.7559
		up-active	

Merging the last two blocks

A third time, we observe, that the current block is up-active, but not up-satisfied. We therefore merge it with its successor and mark the resulting block as down-active (line 18):

$\delta_{14} = 1.4$	$\delta_{24} = 2.0, \delta_{12} = 2.5, \delta_{13} = 3.5$	$\delta_{23} = 3.5, \delta_{34} = 3.7$
1.0691	1.5714	1.5732
		down-active

Post-processing

The current block is both up-satisfied and down-satisfied. Since this is the last block in our list, the regression loop (lines 8-27) terminates. Now all that is left to do is to transfer the disparities into the disparity matrix \hat{D} (lines 28-33). We therefore get the following result:

$$\hat{D} = \begin{pmatrix} 0 & 1.5714 & 1.5714 & 1.0691 \\ 1.5714 & 0 & 1.5732 & 1.5714 \\ 1.5714 & 1.5732 & 0 & 1.5732 \\ 1.0691 & 1.5714 & 1.5732 & 0 \end{pmatrix}$$

8.4.4 The Algorithm

Having established both a way of computing the gradient of Stress-1 in Section 8.4.1 and a method for computing a monotone regression in Section 8.4.2, we can now specialize the general gradient descent procedure from Algorithm 8.2 into Kruskal's nonmetric MDS algorithm.

Putting everything together

Algorithm 8.4 starts by initializing the configuration of points (line 1), e.g., by using a random number generator, and by setting the learning rate α to 0.2 (line 2). After initializing X , we ensure that it is normalized, i.e., that the centroid of the configuration lies in the origin, and that root-mean-square distance of the points from the origin equals one. This can be done by centering X and by afterwards dividing all its coordinates by $\sqrt{\frac{1}{N} \sum_{j=1}^N \sum_{i=1}^n (x_i^{(j)})^2}$.

Initialization

In each step of the optimization loop (lines 4-15), a number of computations take place: First of all, we compute the optimal disparities \hat{D} for the given configuration X by using the monotone regression from Algorithm 8.3 (line 4). The resulting disparities are then used to compute Stress-1 in line 5 (based on Equation 8.4) as well as the gradient of Stress-1 in line 6 (using Equation 8.9). In line 7, we then compute the magnitude $|\nabla_X \sigma_1(X)|$ of the gradient, which will be used later both as a normalization factor in the update step and as part of the stopping criterion. Afterwards, we compute three factors, which are used to update the learning rate α :

Main computational steps

The *angle factor* α_a (line 8) measures the angle θ between the current and the previous gradient, interpreting these gradients as $N \cdot n$ -dimensional vectors. If θ is close to 0° , then both gradients point in the same direction and make similar adjustments to the configuration. In this case, $\cos(\theta)$ is close to 1, yielding a large angle factor α_a . If θ is close to 180° , then the gradients point into opposite directions, which indicates conflicting updates. This results in $\cos(\theta)$ being close to -1 and α_a being small. The angle factor can thus be seen as a variant of momentum (cf. Section 6.2.5): It tries to increase the step size as long as the direction of the updates stays the same, but attempts to make updates smaller, when the direction of the updates changes. Especially a sharp turn indicated by $\theta \approx 180^\circ$ indicates, that the last update may have corresponded to a jump over a valley (e.g., from x_0 to x_2 in Figure 8.3). In order to reach the bottom of this valley, it makes sense to reduce the learning rate.

Angle factor as a variant of momentum

The *relaxation factor* α_r (line 9) compares the current value of Stress-1 $\sigma_1^{(c)}$ to the Stress value $\sigma_1^{(c-5)}$ from five iterations ago. If $\sigma_1^{(c)} > \sigma_1^{(c-5)}$, then Stress-1 has increased in the recent past. In this case, $\alpha_r = 0.65$, which corresponds to slowing down the updates with the hope of decreasing Stress again. However, if $\sigma_1^{(c)} < 0.786 \cdot \sigma_1^{(c-5)}$, then $\alpha_r > 1$, i.e., the step size increases. Implicitly, we thus expect a relative improvement of about 21.4% within five iterations – if the improvement is larger, then we increase our step size, otherwise we decrease it. The relaxation factor thus looks at the mid-term improvements of Stress-1.

Relaxation factor based on mid-term improvements

Algorithm 8.4: Kruskal's MDS algorithm as described in [234].

Input: Dissimilarity matrix Δ , number of dimensions n , threshold ϵ

Output: Coordinate matrix X

- 1 Choose an arbitrary $N \times n$ matrix X as initial configuration and normalize it
 - 2 Set $c = 1$ and $\alpha = 0.2$
 - 3 **repeat**
 - 4 Compute disparities \hat{D} with Algorithm 8.3, using Δ and $D(X)$
 - 5 Compute current Stress-1 $\sigma_1^{(c)} = \sigma_1(X)$
 - 6 Compute gradient $\nabla_X \sigma_1(X) = \frac{\partial \sigma_1(X)}{\partial X}$
 - 7 Compute magnitude of the gradient $|\nabla_X \sigma_1(X)|$
 - 8 Compute angle factor $\alpha_a = 4^{\cos(\theta)^3}$
 - 9 Compute relaxation factor $\alpha_r = 1.3 / \left(1 + \min \left(1, \frac{\sigma_1^{(c)}}{\sigma_1^{(c-5)}} \right)^5 \right)$
 - 10 Compute good luck factor $\alpha_{gl} = \min \left(1, \frac{\sigma_1^{(c)}}{\sigma_1^{(c-1)}} \right)$
 - 11 Update learning rate $\alpha = \alpha \cdot \alpha_a \cdot \alpha_r \cdot \alpha_{gl}$
 - 12 Update $X = X - \frac{\alpha}{|\nabla_X \sigma_1(X)|} \cdot \nabla_X \sigma_1(X)$
 - 13 Normalize X
 - 14 Increase c
 - 15 **until** $|\nabla_X \sigma_1(X)| < \epsilon$
 - 16 **return** X
-

Good luck factor based on short-term improvements

The good luck factor α_{gl} (line 10) considers the direct improvement of Stress-1 with respect to the previous iteration: If $\sigma_1^{(c)} < \sigma_1^{(c-1)}$, then the step size is reduced, while for $\sigma_1^{(c)} > \sigma_1^{(c-1)}$, the step size remains constant. The good luck factor therefore reduces the step size as long as Stress is improving. The relaxation factor α_r and the good luck factor α_{gl} are thus somewhat conflicting with α_r increasing the step size for decreasing Stress values and α_{gl} increasing it. While α_r can thus be seen as rewarding recent improvements, α_{gl} tries to slow down the search for sudden jumps in order to avoid overstepping close-by minima.

Overall update

The overall learning rate α is then updated based on these three factors (line 11). The updated α is normalized by the magnitude of the gradient when applying the update to the configuration X in line 12. Afterwards, we ensure, that the updated configuration is normalized by following the procedure described above (line 13). The algorithm terminates, once the magnitude of the gradient becomes smaller than a given threshold ϵ (line 15), which indicates, that X is close enough to a local minimum. Based on his experimentations, Kruskal proposes to set ϵ to about 2% of the expected Stress-1 for an arbitrary configuration.

As Kruskal notes, his algorithm is not guaranteed to find the *global* minimum of Stress-1 – it may as well converge to an (arbitrarily bad) *local* minimum. This is a general property of gradient descent as already discussed in Sections 5.2.2 and 6.2.5. Kruskal proposes to deal with this problem by running the algorithm multiple times with different randomly chosen initial configurations and by keeping the best result. Moreover, he argues, that the interpretability of the resulting configuration can indicate, whether a global minimum has been reached.

Local optima

His update of the learning rate α is based on the intuition, that one should start with a relatively large learning rate, which decreases over time: In the beginning, larger updates are needed in order to reach a region of low Stress values. Once this target region has been reached, smaller updates are used to find the actual minimum. His three update factors α_a , α_r , and α_{gl} try to capture this intuition mathematically, and have emerged from extensive experimentation. However, one can criticize them for being quite arbitrarily chosen. Other subsequent developments in the area of artificial neural networks, such as momentum or automatic learning rate adaption (cf. Section 6.2.5) may be superior to Kruskal's handcrafted learning rate updates, especially, since they stand on a stronger theoretical foundation.

Learning rate modifications

De Leeuw [115] pointed out, that Kruskal's gradient descent method has two major shortcomings: Firstly, if the points for two stimuli coincide (i.e., $\vec{x}^{(j_1)} = \vec{x}^{(j_2)}$), then the distance function of these two points is not differentiable. Since Kruskal's algorithm uses gradient descent for the optimization, this can lead to problems. Secondly, Kruskal was not able to give a proof of convergence for his algorithm – we are therefore not guaranteed, that the algorithm terminates at all. In order to overcome these limitations, de Leeuw derived the SMACOF algorithm, which will be introduced in Section 8.5.

Shortcomings of Kruskal's algorithm

8.4.5 An Illustrative Example

Let us again use our fruit example from Section 8.2.4 to illustrate the inner workings of Kruskal's MDS algorithm. We will consider a single update to the initial configuration X_0 . We have already shown, how to compute the disparities \hat{D} in Section 8.4.3. We therefore assume the following variables to be given:

Starting points

$$X_0 = \begin{pmatrix} -0.2673 & -0.2673 \\ -0.8018 & 1.3363 \\ 0.2673 & -0.8018 \\ 0.8018 & -0.2673 \end{pmatrix}$$

$$D(X_0) \approx \begin{pmatrix} 0 & 1.6903 & 0.7560 & 1.0691 \\ 1.6903 & 0 & 2.3905 & 2.2678 \\ 0.7560 & 2.3905 & 0 & 0.7559 \\ 1.0691 & 2.2678 & 0.7559 & 0 \end{pmatrix}$$

$$\Delta = \begin{pmatrix} 0 & 2.5 & 3.5 & 1.4 \\ 2.5 & 0 & 3.5 & 2.0 \\ 3.5 & 3.5 & 0 & 3.7 \\ 1.4 & 2.0 & 3.7 & 0 \end{pmatrix}$$

$$\hat{D} \approx \begin{pmatrix} 0 & 1.5714 & 1.5714 & 1.0691 \\ 1.5714 & 0 & 1.5732 & 1.5714 \\ 1.5714 & 1.5732 & 0 & 1.5732 \\ 1.0691 & 1.5714 & 1.5732 & 0 \end{pmatrix}$$

Computing Stress-1

Please note, that X_0 is already properly normalized, i.e., centered around the origin with a root mean squared distance of one. We can now look at the optimization loop of Algorithm 8.4. The next step after initializing X_0 and α (lines 1-2) and after computing \hat{D} (line 4) consists in computing the value of Stress-1 for the current configuration (line 5). This has already been done in Section 8.2.4. From there, we know that $\sigma_1^{(1)} = \sigma_1(X_0) \approx 0.3953$.

Computing the gradient

Now we compute the gradient $\nabla_X \sigma_1(X_0)$ of Stress-1 with respect to the coordinates in X_0 by using Equation 8.9 (see line 6). This results in the following gradient:

$$\nabla_X \sigma_1(X_0) \approx \begin{pmatrix} 0.1235 & -0.0826 \\ -0.0624 & 0.0793 \\ 0.0316 & 0.1462 \\ -0.0927 & -0.1428 \end{pmatrix}$$

Updating the configuration

Afterwards, in line 7, we compute the magnitude of this gradient, which results in $|\nabla_X \sigma_1(X_0)| \approx 0.0209$. Since we are in the first iteration of the algorithm, there is no prior gradient or prior Stress value. In these cases, we simply use the first value, which has been computed [234], i.e., the one from the current iteration. We therefore get $\theta = 0^\circ$, $\alpha_a = 4$, $\alpha_r = 0.65$, and $\alpha_{gl} = 1$ (lines 8-10). Our initial learning rate of $\alpha = 0.2$ is thus updated to $\alpha = 0.2 \cdot 4 \cdot 0.65 \cdot 1 = 0.52$ (line 11). Now we can perform our update of X according to line 12:

$$X = X_0 - \frac{\alpha}{|\nabla_X \sigma_1(X_0)|} \cdot \nabla_X \sigma_1(X_0)$$

$$\approx \begin{pmatrix} -0.2673 & -0.2673 \\ -0.8018 & 1.3363 \\ 0.2673 & -0.8018 \\ 0.8018 & -0.2673 \end{pmatrix} - \frac{0.5200}{0.0209} \cdot \begin{pmatrix} 0.1235 & -0.0826 \\ -0.0624 & 0.0793 \\ 0.0316 & 0.1462 \\ -0.0927 & -0.1428 \end{pmatrix}$$

$$\approx \begin{pmatrix} -3.3403 & 1.7871 \\ 0.7498 & -0.6347 \\ -0.5183 & -4.4381 \\ 3.1088 & 3.2856 \end{pmatrix}$$

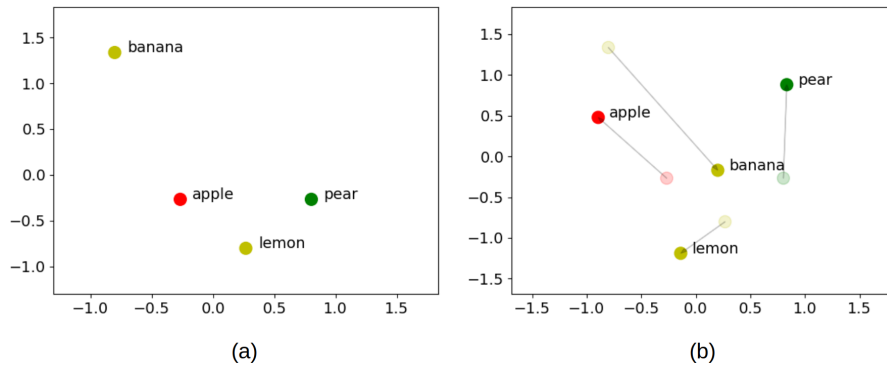


Figure 8.4: Illustration of initial configuration X_0 (a) and the update for Kruskal's algorithm (b).

	Absolute MDS		Ordinal MDS	
	$\sigma_r(X)$	$\sigma_1(X)$	$\sigma_r(X)$	$\sigma_1(X)$
Initial configuration	18.2653	1.0684	2.5000	0.3953
Updated configuration	12.9152	0.8984	0.2763	0.1314

Table 8.4: Stress values for the initial configuration and the first update provided by Kruskal's algorithm.

Now we normalize X by subtracting the mean and dividing by the root mean squared distance (line 13). This results in our updated configuration, which is illustrated in Figure 8.4:

Normalizing X

$$X \approx \begin{pmatrix} -0.8948 & 0.4787 \\ 0.2008 & -0.1700 \\ -0.1388 & -1.1889 \\ 0.8328 & 0.8802 \end{pmatrix}$$

As we can see in Figure 8.4, the configuration of points has changed considerably in this first optimization step, which is in line with the relatively high learning rate used for the update. In Table 8.4, we have compared the Stress values of the updated configuration to the ones of the initial configuration. We can see a clear improvement with respect to ordinal Stress (which is explicitly being optimized by Kruskal's algorithm). Also the values for absolute Stress have decreased slightly. However, looking at Figure 8.4 reveals, that the updated configuration is not yet optimal. For instance, the pair APPLE-PEAR has the lowest dissimilarity, but the corresponding points are quite far away from each other. So while the update seems to actually have improved the configuration, further updates in subsequent iterations are necessary before reaching convergence. Since the goal of this section was only to illustrate the general processing steps of Algorithm 8.4, we do not consider any further updates and leave the discussion of Kruskal's algorithm at this stage.

Interpreting the results

8.5 ITERATIVE STRESS MAJORIZATION THROUGH SMACOF

The *SMACOF algorithm* (an acronym for "Scaling by Majorizing a Complicated Function") was developed by de Leeuw [115] in order to overcome certain shortcomings of gradient-based approaches. We base our treatment mainly on the derivation of the algorithm provided by Borg and Groenen [71, Chapters 8,9].

In Section 8.5.1, we introduce the necessary mathematical background, namely, iterative majorization and matrix traces. We then derive a majorizing function for raw Stress in Section 8.5.2, which will be used in the SMACOF algorithm presented in Section 8.5.3. In Section 8.5.4, we apply SMACOF to our fruit example from Section 8.2.4.

8.5.1 Mathematical Background

Iterative majorization

The SMACOF algorithm is a special case of a general optimization approach called *iterative majorization* [71, Chapter 8]. Iterative majorization is used to find the minimum of a given function in an iterative way, if solving the problem analytically (i.e., by computing the function's derivative and setting it to zero) is not feasible. It has thus the same goal as gradient descent, but follows a different approach: In order to minimize a given function $f(\vec{x})$, one first constructs a so-called *majorizing function* $g(\vec{x}, \vec{z})$ (where \vec{z} is treated as a constant), which fulfills the following three properties:

1. $g(\vec{x}, \vec{z})$ is easier to minimize than $f(\vec{x})$
2. $\forall \vec{x}, \vec{z} : f(\vec{x}) \leq g(\vec{x}, \vec{z})$
3. $\forall \vec{z} : f(\vec{z}) = g(\vec{z}, \vec{z})$

Important property of the majorizing function

Let \vec{z} be arbitrary but fixed, and let $g(\vec{x}, \vec{z})$ obtain its minimum over \vec{x} for this fixed value of \vec{z} in \vec{x}^* . Based on properties 2 and 3, one can easily see, that the following inequality holds:

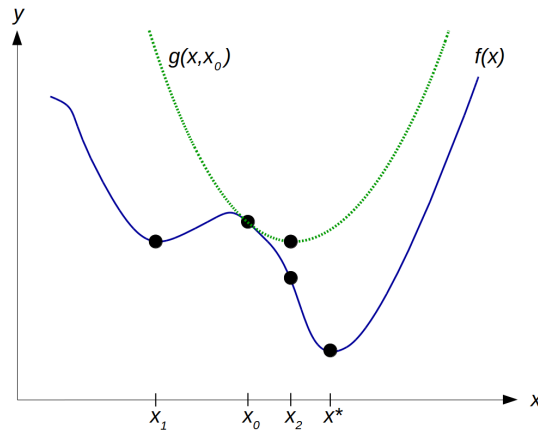
$$f(\vec{x}^*) \leq g(\vec{x}^*, \vec{z}) \leq g(\vec{z}, \vec{z}) = f(\vec{z}) \quad (8.10)$$

The general algorithm

Equation 8.10 is the basis for the general procedure of iterative majorization shown in Algorithm 8.5 and illustrated in Figure 8.5. We start by picking an arbitrary first estimate \vec{z} of the minimum (line 1). Then, we repeatedly compute the current value of the function f at \vec{z} (line 3), and try to find a point \vec{x} , which decreases the value of the majorizing function g (line 4). If g is simple enough (e.g., a quadratic function), we can directly jump to its global minimum. In the more general case, we are, however, already satisfied if we can find a point \vec{x} , which decreases the function value of g . Since $g(\vec{x}, \vec{z}) \leq g(\vec{z}, \vec{z})$, we know based on Equation 8.10, that also $f(\vec{x}) \leq f(\vec{z})$. That is, by repeatedly decreasing the function value of g , we iteratively minimize f . This is achieved by using the new estimate \vec{x} of the minimum as

Algorithm 8.5: General iterative majorization algorithm.**Input:** Function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, majorizing function g , threshold ϵ **Output:** Point $\vec{x} \in \mathbb{R}^n$, which approximately minimizes f

- 1 Set \vec{z} to an arbitrary starting value
- 2 **repeat**
- 3 Compute $y = f(\vec{z})$
- 4 Find \vec{x} , such that $g(\vec{x}, \vec{z}) \leq g(\vec{z}, \vec{z})$
- 5 Set $\vec{z} = \vec{x}$
- 6 **until** $y - f(\vec{x}) < \epsilon$
- 7 **return** \vec{x}

**Figure 8.5:** Illustration of iterative majorization for a function f .

starting point \vec{z} in the next iteration (line 5). If $f(\vec{z})$ and $f(\vec{x})$ are almost indistinguishable (line 6), a stationary point has been found.

Let us consider the example from Figure 8.5, assuming, that the current estimate of the minimum is x_0 . In this example, $g(x, z)$ is a quadratic function, which can be easily minimized. In Figure 8.5, the point x_2 minimizes $g(x, x_0)$, thus our next estimate for the minimum of $f(x)$ is x_2 . We can see in Figure 8.5 that $f(x_2) < f(x_0)$, i.e., we have improved our estimate. In the next step, we will then need to minimize $g(x, x_2)$ in order to get closer to the global minimum x^* .

Overall, iterative majorization generates a monotonically nonincreasing sequence of function values. In general, the iterative majorization algorithm can stop at any stationary point (i.e., any point, where the derivative is zero, which includes for example also saddle points), but typically finds a local minimum. In our example from Figure 8.5, we therefore expect to converge to either x^* or x_1 . In comparison to other optimization algorithms such as gradient descent, iterative majorization is not very fast, but it is guaranteed to converge, if the function f is bounded from below. A key challenge when applying iterative majorization is the definition of a suitable majorizing function g , which fulfills all three required properties. We will derive such a majorizing function for raw Stress in Section 8.5.2.

*A simple example**Properties of iterative majorization*

Matrix trace

Before we moving on, we need to introduce the *matrix trace*. The trace of an $N \times N$ matrix A is defined as $\text{tr } A = \sum_{j=1}^N a_{jj}$, i.e., the sum over the entries on the main diagonal. The trace is useful for expressing linear and quadratic functions. Consider, for instance, a linear function, which takes a square matrix $X \in \mathbb{R}^{N \times N}$ as an argument:

$$f(X) = f(x_{11}, \dots, x_{j_1 j_2}, \dots, x_{NN}) = \sum_{j_1=1}^N \sum_{j_2=1}^N a_{j_1 j_2} x_{j_2 j_1}$$

Matrix trace and linear functions

Using the matrix trace, we can write f in a more compact way as $f(X) = \text{tr } AX$. In order to show this equivalence, let us define $B = AX$. The diagonal entries of B can then be written as $b_{j_1 j_1} = \sum_{j_2=1}^N a_{j_1 j_2} x_{j_2 j_1}$. We then get:

$$\text{tr } AX = \text{tr } B = \sum_{j_1=1}^N b_{j_1 j_1} = \sum_{j_1=1}^N \sum_{j_2=1}^N a_{j_1 j_2} x_{j_2 j_1} = f(X) \quad (8.11)$$

Matrix trace and quadratic functions

In a similar way, one can easily show the following equivalence for quadratic functions of X :

$$\forall X \in \mathbb{R}^{N \times N} : \text{tr } X^T X = \sum_{j_1=1}^N \sum_{j_2=1}^N x_{j_1 j_2}^2 \quad (8.12)$$

8.5.2 A Majorizing Function for Stress

Rewriting raw Stress

SMACOF applies iterative majorization to raw Stress. Let us assume that \hat{D} is a matrix of optimal disparities $\hat{d}_{j_1 j_2} = f(\delta_{j_1 j_2})$, which are based on the dissimilarities $\delta_{j_1 j_2}$. Based on the binomial formula, we can then rewrite raw Stress from Equation 8.3 as a sum of three components (cf. [71, Chapter 9] and [115]):

$$\begin{aligned} \sigma_r(X) &= \sum_{j_1=1}^N \sum_{j_2=j_1+1}^N w_{j_1 j_2} (d_{j_1 j_2}(X) - \hat{d}_{j_1 j_2})^2 \\ &= \sum_{j_1=1}^N \sum_{j_2=j_1+1}^N w_{j_1 j_2} \left(d_{j_1 j_2}^2(X) - 2 \cdot d_{j_1 j_2}(X) \cdot \hat{d}_{j_1 j_2} + \hat{d}_{j_1 j_2}^2 \right) \\ &= \underbrace{\sum_{j_1=1}^N \sum_{j_2=j_1+1}^N w_{j_1 j_2} \cdot \hat{d}_{j_1 j_2}^2}_{\eta^2(\hat{D})} + \underbrace{\sum_{j_1=1}^N \sum_{j_2=j_1+1}^N w_{j_1 j_2} \cdot d_{j_1 j_2}^2(X)}_{\eta^2(X)} \\ &\quad - 2 \underbrace{\sum_{j_1=1}^N \sum_{j_2=j_1+1}^N w_{j_1 j_2} \cdot \hat{d}_{j_1 j_2} d_{j_1 j_2}(X)}_{\rho(X, \hat{D})} \end{aligned}$$

Three components of raw Stress

The first term, $\eta^2(\hat{D})$, depends only on the weights $w_{j_1 j_2}$ and the disparities $\hat{d}_{j_1 j_2}$, but not on the coordinates X . It can thus be treated as

an additive constant, which is irrelevant when minimizing $\sigma_r(X)$ over X . The second component, $\eta^2(X)$ is a weighted sum of the squared distances $d_{j_1 j_2}^2(X)$. Also the third component, $\rho(X, \hat{D})$, can be seen as a weighted sum, where the weights are given by $w_{j_1 j_2} \hat{d}_{j_1 j_2}$, and where the sum runs over the distances $d_{j_1 j_2}(X)$. Both $\eta^2(X)$ and $\rho(X, \hat{D})$ depend on X and therefore play an important role when minimizing $\sigma_r(X)$ over X . In order to find a suitable majorizing function, we therefore consider these two components in more detail, based on their treatment in [71, Chapters 8 and 9] and [115].

Let us first consider $\eta^2(X) = \sum_{j_1=1}^N \sum_{j_2=j_1+1}^N w_{j_1 j_2} \cdot d_{j_1 j_2}^2(X)$. We would like to express $\eta^2(X)$ in the form of a matrix multiplication of the coordinate matrix X with some helper matrix V . For now, let us focus only on computing $d_{j_1 j_2}^2(X)$. Let us denote by X_i the column i of the coordinate matrix X (i.e., the i th coordinate of all points) and by e_j the j th unit vector (i.e., a vector of length N , where the j th entry equals one and all other entries are zero). It is easy to see, that $e_{j_1} - e_{j_2}$ contains the entries 1 at index j_1 and -1 at index j_2 , while being filled with zeros at all other positions. Therefore, $x_i^{(j_1)} - x_i^{(j_2)} = (e_{j_1} - e_{j_2})^T X_i$. We can then express $d_{j_1 j_2}^2(X)$ as follows:

Computing squared distances

$$\begin{aligned} d_{j_1 j_2}^2(X) &= \sum_{i=1}^n \left(x_i^{(j_1)} - x_i^{(j_2)} \right)^2 \\ &= \sum_{i=1}^n X_i^T \underbrace{(e_{j_1} - e_{j_2})(e_{j_1} - e_{j_2})^T}_{=A_{j_1 j_2}} X_i \end{aligned} \quad (8.13)$$

One can easily see, that $A_{j_1 j_2}$ is a matrix filled with zeros except for $a_{j_1 j_1} = a_{j_2 j_2} = 1$ and $a_{j_1 j_2} = a_{j_2 j_1} = -1$. Using $A_{j_1 j_2}$ and the matrix trace (cf. Equations 8.11 and 8.12 from Section 8.5.1), we can rewrite $d_{j_1 j_2}^2(X)$ as follows:

A helper matrix

$$\begin{aligned} d_{j_1 j_2}^2(X) &= \sum_{i=1}^n X_i^T (e_{j_1} - e_{j_2})(e_{j_1} - e_{j_2})^T X_i \\ &= \sum_{i=1}^n X_i^T A_{j_1 j_2} X_i = \text{tr } X^T A_{j_1 j_2} X \end{aligned}$$

The validity of the last transformation can be easily seen by remembering, that the trace sums over the elements of the main diagonal. If $X^T A_{j_1 j_2} X = B$, then we sum over all $b_{ii} = X_i^T A_{j_1 j_2} X_i$.

Applying the matrix trace

Now that we have expressed the individual $d_{j_1 j_2}^2(X)$ in the form of matrix multiplications and the trace function, we can construct a similar formula for $\eta^2(X)$, which is just a weighted sum over these $d_{j_1 j_2}^2(X)$:

Computing $\eta^2(X)$

$$\eta^2(X) = \sum_{j_1=1}^N \sum_{j_2=j_1+1}^N w_{j_1 j_2} \cdot d_{j_1 j_2}^2(X)$$

$$\begin{aligned}
 &= \sum_{j_1=1}^N \sum_{j_2=j_1+1}^N w_{j_1 j_2} \cdot \text{tr } X^T A_{j_1 j_2} X \\
 &= \sum_{j_1=1}^N \sum_{j_2=j_1+1}^N \text{tr } X^T (w_{j_1 j_2} \cdot A_{j_1 j_2}) X \\
 &= \text{tr } X^T \left(\sum_{j_1=1}^N \sum_{j_2=j_1+1}^N w_{j_1 j_2} \cdot A_{j_1 j_2} \right) X
 \end{aligned}$$

A quadratic function

Defining $V = \sum_{j_1=1}^N \sum_{j_2=j_1+1}^N w_{j_1 j_2} \cdot A_{j_1 j_2}$, we can thus express $\eta^2(X)$ as a quadratic function in X :

$$\eta^2(X) = \text{tr } X^T V X$$

Optimizing $\eta^2(X)$

A quadratic function is easy to minimize, because it has exactly one extremum. The matrix V can also be defined as $\forall j_1 \neq j_2 : v_{j_1 j_2} = -w_{j_1 j_2}$ and $v_{j_1 j_1} = \sum_{j_2 \neq j_1} w_{j_1 j_2}$.

$\rho(X, \hat{D})$ and the Cauchy-Schwarz inequality

Let us now take a look at $\rho(X, \hat{D}) = \sum_{j_1=1}^N \sum_{j_2=j_1+1}^N w_{j_1 j_2} \cdot \hat{d}_{j_1 j_2} \cdot d_{j_1 j_2}(X)$. Again, we will first focus on expressing the individual $d_{j_1 j_2}(X)$. For this, we will use the Cauchy-Schwarz inequality:

$$\sum_{i=1}^n p_i q_i \leq \sqrt{\sum_{i=1}^n p_i^2} \sqrt{\sum_{i=1}^n q_i^2}$$

Inserting coordinate-wise differences

Please note, that the two terms are identical, if $q_i = c \cdot p_i$ for any constant $c \neq 0$. By replacing p_i with $(x_i^{(j_1)} - x_i^{(j_2)})$ and q_i with $(z_i^{(j_1)} - z_i^{(j_2)})$, we get the following result, where equality happens for $Z = X$:

$$\sum_{i=1}^n (x_i^{(j_1)} - x_i^{(j_2)})(z_i^{(j_1)} - z_i^{(j_2)}) \leq \underbrace{\sqrt{\sum_{i=1}^n (x_i^{(j_1)} - x_i^{(j_2)})^2}}_{=d_{j_1 j_2}(X)} \underbrace{\sqrt{\sum_{i=1}^n (z_i^{(j_1)} - z_i^{(j_2)})^2}}_{d_{j_1 j_2}(Z)}$$

A lower bound for distances

As we can see, the factors on the right hand side of the inequality are identical to the definition of the Euclidean distance, and can hence be replaced by $d_{j_1 j_2}$. Solving the inequality for $d_{j_1 j_2}(X)$ results in the following expression:

$$d_{j_1 j_2}(X) \geq \frac{\sum_{i=1}^n (x_i^{(j_1)} - x_i^{(j_2)})(z_i^{(j_1)} - z_i^{(j_2)})}{d_{j_1 j_2}(Z)} \tag{8.14}$$

Further observations

If $d_{j_1 j_2}(Z) = 0$, i.e., if the coordinates of the points j_1 and j_2 are identical in the matrix Z , then the right hand side of the inequality is not defined. However, we know that in any case $d_{j_1 j_2}(X) \geq 0$, because distances cannot be negative. Thus, if $d_{j_1 j_2}(Z) = 0$, we can simply replace the complete right hand side of the formula with zero and still

have a valid lower bound. Based on our considerations for $\eta^2(X)$, we furthermore know that the following equality holds (cf. Equation 8.13):

$$\sum_{i=1}^n (x_i^{(j_1)} - x_i^{(j_2)})(z_i^{(j_1)} - z_i^{(j_2)}) = \text{tr } X^T A_{j_1 j_2} Z \quad (8.15)$$

We can now derive the following expression for $\rho(X, \hat{D})$ by combining Equations 8.14 and 8.15 to bound $d_{j_1 j_2}(X)$ from below:

A lower bound for $\rho(X, \hat{D})$

$$\begin{aligned} \rho(X, \hat{D}) &= \sum_{j_1=1}^N \sum_{j_2=j_1+1}^N w_{j_1 j_2} \cdot \hat{d}_{j_1 j_2} \cdot d_{j_1 j_2}(X) \\ &\geq \sum_{j_1=1}^N \sum_{j_2=j_1+1}^N (w_{j_1 j_2} \cdot \hat{d}_{j_1 j_2}) \frac{\text{tr } X^T A_{j_1 j_2} Z}{d_{j_1 j_2}(Z)} \\ &= \sum_{j_1=1}^N \sum_{j_2=j_1+1}^N \text{tr } X^T \left(\frac{w_{j_1 j_2} \cdot \hat{d}_{j_1 j_2} \cdot A_{j_1 j_2}}{d_{j_1 j_2}(Z)} \right) Z \\ &= \text{tr } X^T \underbrace{\left(\sum_{j_1=1}^N \sum_{j_2=j_1+1}^N \frac{w_{j_1 j_2} \cdot \hat{d}_{j_1 j_2}}{d_{j_1 j_2}(Z)} A_{j_1 j_2} \right)}_{=B(Z)} Z \end{aligned} \quad (8.16)$$

Analogously to V , the matrix $B(Z)$ is defined as having the elements $\forall j_1 \neq j_2 : b_{j_1 j_2} = -\frac{w_{j_1 j_2} \hat{d}_{j_1 j_2}}{d_{j_1 j_2}(Z)}$, if $d_{j_1 j_2}(Z) \neq 0$, and $b_{j_1 j_2} = 0$ otherwise. Moreover, $b_{j_1 j_1} = -\sum_{j_2 \neq j_1} b_{j_1 j_2}$. Since the inequality in Equation 8.16 is still based on the Cauchy-Schwarz inequality, we still get an equality for $Z = X$. Therefore, we can write:

Properties of the helper matrix

$$\rho(X, \hat{D}) = \text{tr } X^T B(X) X \geq \text{tr } X^T B(Z) Z$$

In our overall formula for Stress, $\rho(X, \hat{D})$ appears with a negative sign, thus it is convenient to multiply this formula with -1:

Changing the sign

$$-\rho(X, \hat{D}) = -\text{tr } X^T B(X) X \leq -\text{tr } X^T B(Z) Z \quad (8.17)$$

In other words, we found a way to majorize the component $-\rho(X, \hat{D})$ of raw Stress with $-\text{tr } X^T B(Z) Z$, which is a linear function in X , and which is thus much easier to minimize than $-\rho(X, \hat{D})$.

Majorizing $\rho(X, \hat{D})$

After having considered the individual components of raw Stress, it is now time to put the pieces together in order to derive a majorizing function for raw Stress. We simply replace $\eta^2(X)$ and $\rho(X, \hat{D})$ by $\text{tr } X^T V X$ and $\text{tr } X^T B(X) X$, respectively, and apply Equation 8.17:

Majorizing raw Stress

$$\begin{aligned} \sigma_r(X) &= \eta^2(\hat{D}) + \eta^2(X) - 2\rho(X, \hat{D}) \\ &= \eta^2(\hat{D}) + \text{tr } X^T V X - 2\text{tr } X^T B(X) X \\ &\leq \eta^2(\hat{D}) + \text{tr } X^T V X - 2\text{tr } X^T B(Z) Z = \tau(X, Z) \end{aligned}$$

One can easily see, that $\tau(X, Z)$ is a majorizing function of $\sigma_r(X)$.

Minimizing $\tau(X, Z)$

Moreover, $\tau(X, Z)$ is quadratic in X , which means, that it can be easily minimized by setting its derivative $\nabla_X \tau(X, Z)$ to zero. In analogy to the derivative of quadratic functions of a scalar $x \in \mathbb{R}$, the constant term $\eta^2(\hat{D})$ disappears, the quadratic term $\text{tr } X^T V X$ is translated into a linear term $2VX$, and the linear term $\text{tr } X^T B(Z)Z$ is translated to a constant $2B(Z)Z$:

$$\nabla_X \tau(X, Z) = 2VX - 2B(Z)Z = 0$$

Solving for X

In order to find a configuration X minimizing $\tau(X, Z)$, we thus need to solve the following equality for X :

$$VX = B(Z)Z$$

Inverting V

Normally, one could achieve this by premultiplying both sides of the equation with V^{-1} , i.e., the inverse of V . However, V can unfortunately not be inverted, because it does not have full rank. Instead, we use the Moore-Pensore inverse, which is defined as follows:

$$V^+ = (V + \bar{\mathbf{1}}\bar{\mathbf{1}}^T)^{-1} - N^{-2}\bar{\mathbf{1}}\bar{\mathbf{1}}^T$$

The update step

By premultiplying with V^+ , we obtain the following rule for minimizing $\tau(X, Z)$:

$$X = V^+ B(Z)Z \quad (8.18)$$

A special case

If all weights $w_{j_1 j_2}$ are equal to one, then $V^+ = N^{-1}(I - N^{-1}\bar{\mathbf{1}}\bar{\mathbf{1}}^T)$ and Equation 8.18 can be simplified to $X = N^{-1}B(Z)Z$ [71, Chapter 8].

8.5.3 The Algorithm

Deriving SMACOF

Having defined both the majorizing function $\tau(X, Z)$ and a formula for finding its minimum, we can now use the general iterative majorization procedure from Algorithm 8.5 and apply it to the majorization of raw Stress. The resulting SMACOF algorithm is shown in Algorithm 8.6.

Variations of the algorithm

Algorithm 8.6 assumes, that all weights are set to one. If this is not the case, the update of the configuration in line 6 needs to be replaced by the more general $X = V^+ B(Z)Z$ from Equation 8.18, where V^+ is based on the weights $w_{j_1 j_2}$ as described in Section 8.5.2. Moreover, we would like to point out, that we had to insert line 4 into the algorithm to ensure, that also the disparities $\hat{d}_{j_1 j_2}$ are updated in each step. This alternating update of disparities and configuration is not part of the general iterative majorization algorithm. If the representation function f is linear (e.g., interval MDS), then the optimization of \hat{D} in line 4 can be carried out through a linear least squares regression. If f is constrained to a monotone function (i.e., ordinal MDS), then Kruskal's algorithm for monotone regression from Section 8.4.2 can be used.

Properties of SMACOF

SMACOF guarantees a monotone convergence of Stress, because it is based on iterative majorization. Moreover, one can show, that X

Algorithm 8.6: The SMACOF algorithm.

Input: Dissimilarity matrix Δ , number of dimensions n ,
representation function f , threshold ϵ

Output: Coordinate matrix X

- 1 Choose an arbitrary $N \times n$ matrix X as initial configuration
- 2 Set $Z = X$
- 3 **repeat**
- 4 Compute disparities $\hat{D} \neq 0$ using Δ , f , and $D(X)$
- 5 Compute $y = \sigma_r(X)$
- 6 Compute $X = N^{-1}B(Z)Z$
- 7 Set $Z = X$
- 8 **until** $y - \sigma_r(X) < \epsilon$
- 9 **return** X

converges linearly to a stationary point [71, Chapter 8]. Furthermore, edge cases, where two points $\bar{x}^{(j_1)}$ and $\bar{x}^{(j_2)}$ coincide, and thus $d_{j_1 j_2} = 0$, are not problematic, because we defined the respective matrix entry $b_{j_1 j_2}$ to be zero in such cases. These two properties (guaranteed convergence and ability to handle zero distances) set SMACOF apart from Kruskal's gradient descent algorithm. While our treatment has only covered the Euclidean distance, de Leeuw has also provided a generalization of the SMACOF algorithm to arbitrary Minkowski metrics [115].

However, one should note, that also SMACOF may terminate in (arbitrarily bad) local minima, because there is no guarantee to converge in a global minimum. Thus, also SMACOF is typically started multiple times with different random initializations in order to improve the chances of finding the global minimum in any of the runs. Moreover, since iterative majorization is in general relatively slow, SMACOF might take more time to converge than Kruskal's gradient descent algorithm.

*Limitations of
SMACOF*

8.5.4 An Illustrative Example

Let us now take again a look at our fruit example. Since SMACOF was explicitly designed to deal with an arbitrary representation function f , we will look at two variants: absolute SMACOF and ordinal SMACOF. For absolute SMACOF, we simply use the original dissimilarities as disparities, while for ordinal SMACOF, we employ Kruskal's monotone regression for obtaining disparities. Again, we consider a single update step to the initial configuration X_0 from Section 8.2.4. We therefore assume the following variables to be given:

Starting point

$$X_0 = \begin{pmatrix} -0.2673 & -0.2673 \\ -0.8018 & 1.3363 \\ 0.2673 & -0.8018 \\ 0.8018 & -0.2673 \end{pmatrix}$$

$$D(X_0) \approx \begin{pmatrix} 0 & 1.6903 & 0.7560 & 1.0691 \\ 1.6903 & 0 & 2.3905 & 2.2678 \\ 0.7560 & 2.3905 & 0 & 0.7559 \\ 1.0691 & 2.2678 & 0.7559 & 0 \end{pmatrix}$$

$$\Delta = \begin{pmatrix} 0 & 2.5 & 3.5 & 1.4 \\ 2.5 & 0 & 3.5 & 2.0 \\ 3.5 & 3.5 & 0 & 3.7 \\ 1.4 & 2.0 & 3.7 & 0 \end{pmatrix}$$

$$\hat{D} \approx \begin{pmatrix} 0 & 1.5714 & 1.5714 & 1.0691 \\ 1.5714 & 0 & 1.5732 & 1.5714 \\ 1.5714 & 1.5732 & 0 & 1.5732 \\ 1.0691 & 1.5714 & 1.5732 & 0 \end{pmatrix}$$

Computing $B(X_0)$

In the first iteration of SMACOF, we thus only need to execute the update in line 6 of Algorithm 8.6. In order to do so, we need to define the matrix $B(X_0)$. Recall that $\forall j_1 \neq j_2 : b_{j_1 j_2} = -\frac{w_{j_1 j_2} \hat{d}_{j_1 j_2}}{d_{j_1 j_2}(Z)}$, if $d_{j_1 j_2}(Z) \neq 0$, and $b_{j_1 j_2} = 0$ otherwise, and that $\forall j_1 : b_{j_1 j_1} = -\sum_{j_2 \neq j_1} b_{j_1 j_2}$. In our case, we assume that $\forall j_1, j_2 : w_{j_1 j_2} = 1$. Depending on which disparities we use, we thus get two different versions of $B(X_0)$:

$$B_{abs}(X_0) \approx \begin{pmatrix} 7.4183 & -1.4790 & -4.6298 & -1.3095 \\ -1.4790 & 3.8250 & -1.4641 & -0.8819 \\ -4.6298 & -1.4641 & 10.9888 & -4.8948 \\ -1.3095 & -0.8819 & -4.8948 & 7.0863 \end{pmatrix}$$

$$B_{ord}(X_0) \approx \begin{pmatrix} 4.0893 & -0.9296 & -2.0786 & -1.0000 \\ -0.9296 & 2.2806 & -0.6581 & -0.6929 \\ -2.0786 & -0.6581 & 4.8180 & -2.0812 \\ -1.0000 & -0.6929 & -2.0812 & 3.7741 \end{pmatrix}$$

Computing the update

We therefore get the following updated configurations, which are illustrated in Figure 8.6:

$$X_{abs} = \frac{1}{4} B_{abs}(X_0) X_0 \approx \begin{pmatrix} 0.7711 & -0.0257 \\ 0.9425 & -1.7291 \\ -0.3560 & 2.0554 \\ -1.3576 & -0.3005 \end{pmatrix}$$

$$X_{ord} = \frac{1}{4} B_{ord}(X_0) X_0 \approx \begin{pmatrix} 0.4209 & 0.0949 \\ 0.5779 & -1.0022 \\ -0.176 & 0.9076 \\ -0.8232 & -0.0003 \end{pmatrix}$$

Comparing configurations

When comparing the two configurations X_{abs} and X_{ord} in Figure 8.6, one immediately notes, that the overall arrangement of points is

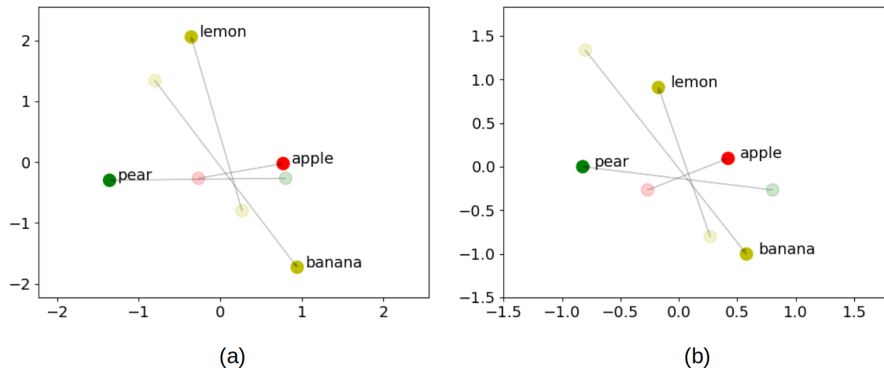


Figure 8.6: Illustration of the update to the initial configuration X_0 for absolute (a) and ordinal (b) SMACOF.

	Absolute MDS		Ordinal MDS	
	$\sigma_r(X)$	$\sigma_1(X)$	$\sigma_r(X)$	$\sigma_1(X)$
Initial configuration	18.2653	1.0684	2.5000	0.3953
Updated configuration (absolute)	4.5140	0.3239	1.5341	0.1881
Updated configuration (ordinal)	17.0210	1.1798	0.7390	0.2458

Table 8.5: Stress values for the initial configuration and the first update provided by absolute and ordinal SMACOF, respectively.

quite similar. Only LEMON is placed further away from the other points in absolute SMACOF than in ordinal SMACOF. However, the scale of the configuration differs with absolute SMACOF resulting in larger distances than ordinal SMACOF. This can be explained by the fact, that absolute SMACOF tries to align $D(X)$ with Δ , while ordinal SMACOF tries to align $D(X)$ with \hat{D} . Since the values in Δ are larger than the ones in \hat{D} , absolute SMACOF results in larger distance values than ordinal SMACOF.

When comparing the Stress values in Table 8.5, a more fine-grained image emerges: Absolute SMACOF explicitly optimizes raw Stress of absolute MDS, and thus is able to achieve a considerable improvement on this metric. Improving absolute Stress also comes with reasonable improvements with respect to ordinal Stress. Ordinal SMACOF, on the other hand, optimizes raw ordinal Stress. It is clearly able to outperform absolute SMACOF with respect to this metric. However, since it does not pay attention to interval and ratio information, it is considerably worse on absolute Stress. It is also interesting to note, that even though absolute SMACOF is worse than ordinal SMACOF on raw ordinal Stress, it is better with respect to ordinal Stress-1. The reason for this is, that the distances produced by absolute SMACOF are larger than the ones resulting from ordinal SMACOF, thus the denominator for computing Stress-1 (cf. Equation 8.4) becomes larger.

As noted in prior examples, one should take the numbers from Table 8.5 as well as the configurations from Figure 8.6 with a grain of salt: Firstly, they are intermediate results after a single iteration of the

Comparing Stress

A word of caution

optimization loop and do thus not represent the final solution. Secondly, we only considered a very small toy example, which is certainly not representative of larger dissimilarity matrices.

8.6 EVALUATING MDS SOLUTIONS

Motivation

As we have seen in Sections 8.3, 8.4, and 8.5, there are different algorithms for solving the optimization MDS problem. In general, these different algorithms will return at least slightly different similarity spaces. If we have multiple MDS algorithms to choose from, we might therefore be interested in using the one which gives the "best" results. Moreover, every MDS algorithm expects that the dimensionality n of the resulting similarity space is known in advance. This is, however, hardly the case, if we apply MDS to dissimilarity ratings from a novel domain. Thus, we are also interested in finding the optimal value of n by comparing similarity spaces of different dimensionality according to their quality.

Choosing the optimal number of dimensions

Before we go into further detail with respect to the evaluation approaches, let us comment on the general trade-off involved when choosing the number of dimensions. This issue is strongly related to finding the optimal number of features in dimensionality reduction (cf. Section 5.3) and to tuning the capacity of a machine learning model (cf. Sections 5.1.1 and 5.1.4). While it is usually unclear, which value of n will prove to be optimal, one can expect certain general trends in the context of MDS: If n is set too low (e.g., $n = 1$ for a very complex dataset, where stimuli are expected to differ in multiple ways), then not all pieces of information from the dissimilarity matrix Δ can be represented in the similarity space [71, Chapter 3]. On the other hand, if n is set too large, then also the noise components from Δ are included in the similarity space [71, Chapter 3]. In the extreme case, one can perfectly represent any dissimilarity matrix in ordinal MDS when using $n = N - 2$ dimensions [71, Chapter 3]. In general, we thus face a trade-off between a good compression (for small n) and more expressive spaces (for large n) [123].

Stress-based evaluation

One can distinguish two approaches for evaluating the quality of a given similarity space: Firstly, one compares different similarity spaces based on their Stress values, which measure the difference between the distances in the similarity space and the dissimilarity ratings from the original dissimilarity matrix. We will give an overview of different Stress-based evaluation approaches in Section 8.6.1.

Interpretability-based evaluation

Secondly, one can inspect the interpretability of a given similarity space. This evaluation approach typically tries to identify meaningful regions and directions in a given similarity space, which lend themselves to an intuitive interpretation. We will give more detail on how to measure the interpretability of similarity spaces in Section 8.6.2.

8.6.1 Stress-based Evaluation

Since Stress is being explicitly optimized in MDS, it makes sense to use it also for making a direct judgment about the quality of the generated similarity space. For instance, Kruskal [233] proposes to interpret nonmetric Stress-1 as follows: A value of 0.2 corresponds to a poor reflection of the dissimilarities, 0.1 is deemed fair, 0.05 good, 0.025 excellent, and 0.0 perfect. Borg and Groenen [71, Chapter 3] have, however, pointed out, that one should not trust this interpretation too much, because the raw Stress values are influenced by various circumstances:

Interpreting Stress directly

- *Representation function:* Stress is generally higher for metric MDS than for nonmetric MDS, because the latter has more degrees of freedom in its representation function.
- *Size of the dataset:* A larger number N of stimuli generally tends to increase Stress, because Stress is not normalized with respect to the number of stimulus pairs.
- *Number of ties:* If the dissimilarity matrix Δ contains many ties, and if ordinal MDS with the primary tie-breaking approach is used, then Stress is expected to be lower, because the tied stimulus pairs are not considered when computing Stress.
- *Missing data:* Larger fractions of missing entries in Δ generally reduce Stress, because less pairs of stimuli are being considered.
- *Noisy data:* A higher error in the dissimilarity data (i.e., larger amounts of noise) contributes to elevated Stress levels, because it makes the data harder to fit by an MDS model.
- *Dimensionality of the similarity space:* Increasing the dimensionality n of the similarity space typically leads to a decrease of Stress, because the model has more free parameters (i.e., more coordinates per point) in order to fit the information from the dissimilarity matrix.

Due to these external influences, it is in general preferable to rather compare the resulting Stress values different MDS solutions, which were all based on the same dissimilarity matrix Δ . One should note, that in this case Stress-1 is preferable to raw Stress, because it is normalized with respect to the distances in the similarity space (cf. Section 8.2.1).

Comparing Stress values

Another way of inspecting the quality of a given similarity space is given by the so-called *Shepard diagram* [71, Chapter 3]. The Shepard diagram is a scatter plot, where the dissimilarities $\delta_{j_1 j_2}$ are plotted on the x-axis and their corresponding distances $d_{j_1 j_2}$ from the similarity space are plotted on the y-axis. Moreover, the regression line relating dissimilarities to distances according to the representation function f can be shown. The Shepard diagram therefore visualizes, how well

Shepard diagrams

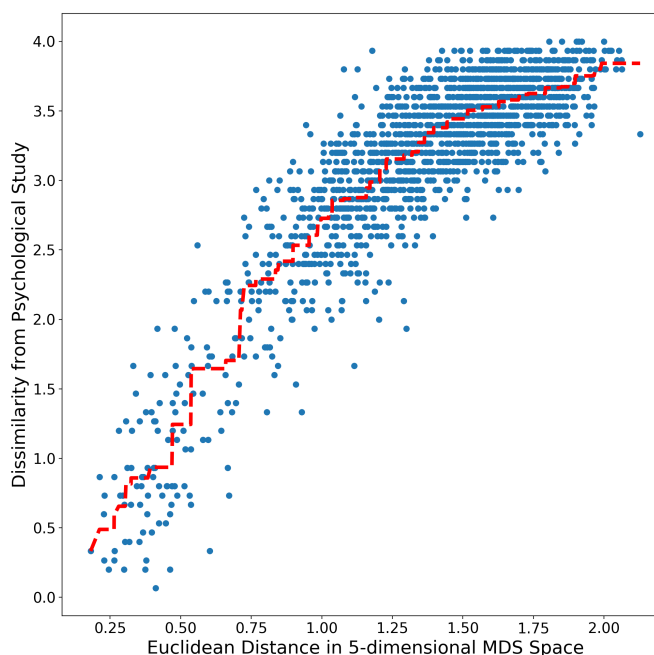


Figure 8.7: Shepard diagram illustrating the relation between dissimilarities and distances (based on the study presented in Chapter 10).

the representation function is capable of mapping dissimilarities to distances. It is especially useful, if one uses ordinal MDS and is interested in the exact shape of the monotone mapping function. Figure 8.7 shows such a Shepard diagram from a study described in Chapter 10.

Stress per point

In the Shepard diagram, one can also easily spot outliers, i.e., the individual pairs of stimuli, for which the mapping is not very successful. A more mathematical way of capturing these outliers is *Stress per point* [71, Chapter 3]. Stress per point is defined as the average squared error which is not computed over all pairs (as in raw Stress), but only over those pairs containing the given stimulus. Stress per point thus measures, to what extent the dissimilarities for this given stimulus are violated in the MDS solution.

Random baseline

In order to compare the Stress value of a given configuration to a meaningful baseline, one can compute the expected Stress value for a randomly generated matrix of size $N \times N$ when using an n -dimensional space [71, Chapter 3]. The Stress value achieved by MDS on the real dissimilarities should then be considerably lower than the expected value for a random dissimilarity matrix. If this is not the case, then the given dissimilarities do not lend themselves towards a spatial representation.

Scree plots

In order to optimize the number n of dimensions for the similarity space, one can take a look at the so-called *Scree plot*, which visualizes Stress as a function of n . If one can identify an "elbow" in this diagram (i.e., a point, after which Stress decreases much slower than before), this can point towards a useful value of n [233]. The rationale behind this approach is as follows: If going from $n - 1$ to n dimensions leads to a relatively large decrease in Stress, this means that the n th dimension

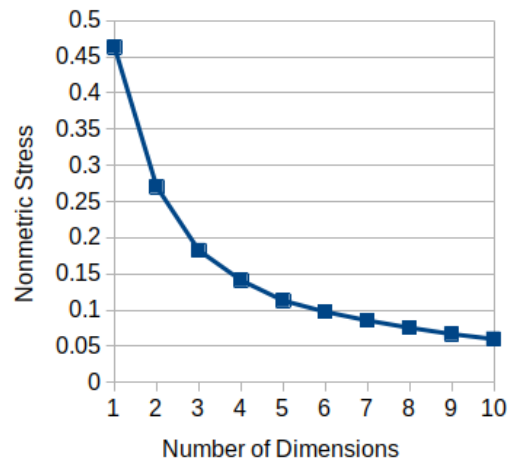


Figure 8.8: Scree plot illustrating Stress as a function of n (based on the study presented in Chapter 10).

represents additional meaningful information from the dissimilarity matrix Δ . If at the same time going from n to $n + 1$ dimensions only causes minor improvements in the Stress value, then the additional dimension mainly captures noise [71, Chapter 3]. Therefore, an elbow in the Scree plot indicates, that all pieces of meaningful information are represented in the similarity space, while largely ignoring noise. Figure 8.8 shows a Scree plot from a study described in Chapter 10.

If information about the reliability of the dissimilarity matrix Δ is available (e.g., an independent estimate of its statistical error), then one can choose n in such a way, that the remaining Stress corresponds to the random component of the data (see [71, Chapter 3] and [233]). This means, that for reliable dissimilarities with a small error, higher-dimensional similarity spaces are chosen, whereas a lower-dimensional space is preferred for noisy dissimilarities. However, in practice, such reliability information is often not available.

Using reliability information

8.6.2 Interpretability-based Evaluation

The Stress-based evaluation approach analyzes, to which extent the dissimilarities are reflected correctly by the distances. One can, however, also analyze a similarity space by trying to interpret the overall configuration of points. This can also be helpful for deciding how many dimensions to use: If the optimal configuration in an n -dimensional space has a sufficient degree of interpretability, and if the optimal configuration in an $n + 1$ -dimensional space does not add more structure, then an n -dimensional space might be sufficient [233].

Motivation

The theory of conceptual spaces states, that a conceptual space is spanned by cognitively meaningful dimensions, and that this space can be partitioned into convex regions, which represent categories of objects [179]. We therefore focus our attention on finding meaningful regions and interpretable directions in a given similarity space. Please

Interpretability based on conceptual spaces

note, that a manual analysis is limited to two- and three-dimensional similarity spaces, which can be easily visualized. Moreover, it relies heavily on the subjective impression of the experimenter. For these reasons, we will focus on automated procedures, which can be also applied to high-dimensional spaces, and which yield quantifiable (and thus more objective) results.

Regions as clusters

Meaningful regions in a similarity space correspond to clusters of stimuli, which belong to the same semantic category. In order to automate the analysis of regions, it is thus necessary, that the stimuli are labeled with their respective categories. In general, we expect points from the same category to have small distances, while points from different categories are expected to have large distances. Since the distances in an MDS solution are based on the dissimilarities, this pattern should already be evident in the dissimilarity matrix. Nevertheless, an additional analysis of the similarity space can ensure, that this general expectation is met.

Measuring cluster quality

This can for instance be operationalized with a k nearest neighbor classifier (cf. Section 5.2.5), whose performance can indicate, whether the closest neighbors of any given stimulus belong to the same category. Alternatively, a clustering algorithm such as k -means (cf. Section 7.3.1) can be used to identify clusters in the similarity space, whose distribution of class labels can then be analyzed in a second step.

Conceptual overlap

In addition to the general existence of such clusters, the conceptual spaces theory assumes, that the similarity space can be *partitioned* into these clusters (given, that all classes under consideration lie on the same level of the conceptual hierarchy). This means, that the individual clusters should not overlap. We can verify, whether this additional expectation is being met by constructing the convex hull for each of the categories, and by investigating to which degree these convex hulls overlap. Alternatively, one can investigate the classification performance of Gärdenfors' proposed Voronoi tessellations of the similarity space (cf. Section 7.1.1).

Finding interpretable directions

Meaningful directions in a similarity space are expected to represent interpretable features from the domain under analysis. In order to identify such meaningful directions, the stimuli therefore need to be annotated with values for a given set of candidate features. For each of these features, one can then try to identify a direction in the similarity space, which orders the stimuli according to their respective values on that feature. This overall approach is sometimes called *property vector fitting* [179, Section 1.7]. We will now describe the procedure followed by Derrac and Schockaert [123], which uses linear support vector machines.

Support vector machines

Please recall from Section 5.2.3, that a linear support vector machine (SVM) [73, 110] tries to separate two classes with a hyperplane, which maximizes the margin size M between the two classes. While other machine learning algorithms based on a linear model (e.g., a logistic regression) may also find a suitable hyperplane for separating positive

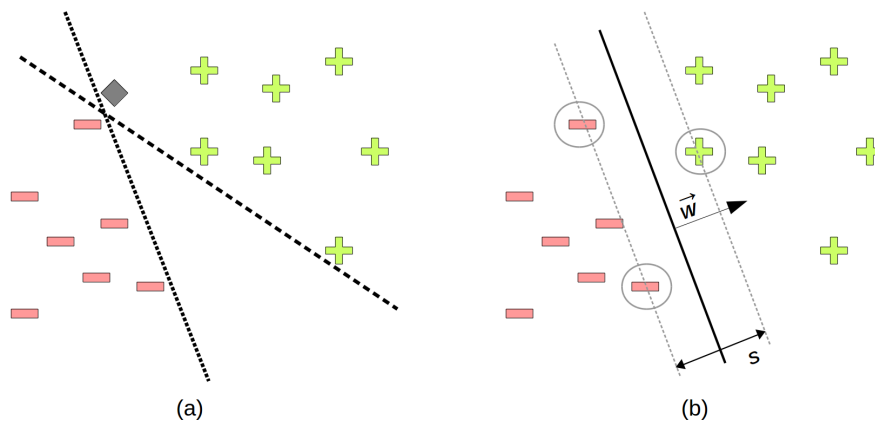


Figure 8.9: (a) Non-optimal decision boundaries in a two-dimensional feature space. (b) Maximum margin separating hyperplane.

from negative examples, these solutions may lie too close to the training examples, and thus carry the risk of misclassifications (see Figure 8.9a). Moreover, there may be multiple possible solutions, such that the result depends also on the initialization of the algorithm. The maximum margin hyperplane found by a SVM, on the other hand, is uniquely defined and minimizes the expected generalization error.

The approach by Derrac and Schockaert [123] for finding interpretable directions in their text-based similarity spaces (cf. Sections 6.3.2 and 8.1.5) works as follows: They started with a set of candidate features, which in their case are terms and short phrases, that occurred frequently in the underlying text corpus. For instance, in their MOVIE domain, such candidate features may include FUNNY or ORGANIZED CRIME. For each of these features, they trained a linear support vector machine in order to separate positive examples (i.e., entities, which were associated with the given term) from negative examples (i.e., entities not associated with the given term). If the linear SVM is successful in separating these two classes, then the weight vector \vec{w} , which is perpendicular to the separating hyperplane, corresponds to the direction pointing from negative to positive examples (cf. Figure 8.9b). Thus, Derrac and Schockaert use \vec{w} as a direction in the similarity space which represents the given candidate feature. Since \vec{w} is uniquely determined, the direction associated to a candidate feature such as FUNNY can be easily reproduced.

In order to evaluate, whether the SVM successfully solves the classification problem, Derrac and Schockaert used Cohen's κ , which is an evaluation metric, that is robust with respect to imbalanced classes (cf. Section 5.1.2). They only considered directions for candidate terms, where κ was above a given threshold, which indicates, that the classes can be reasonably well separated.

Derrac and Schockaert assumed, that the information about the candidate features is given by binary classification labels. However, if we have continuous feature values, we can also make use of regression techniques. One can for instance use so-called *support vector regression*

*Interpretable
directions with
SVMs*

*Quantifying the
quality of a direction*

*Linear regression for
continuous data*

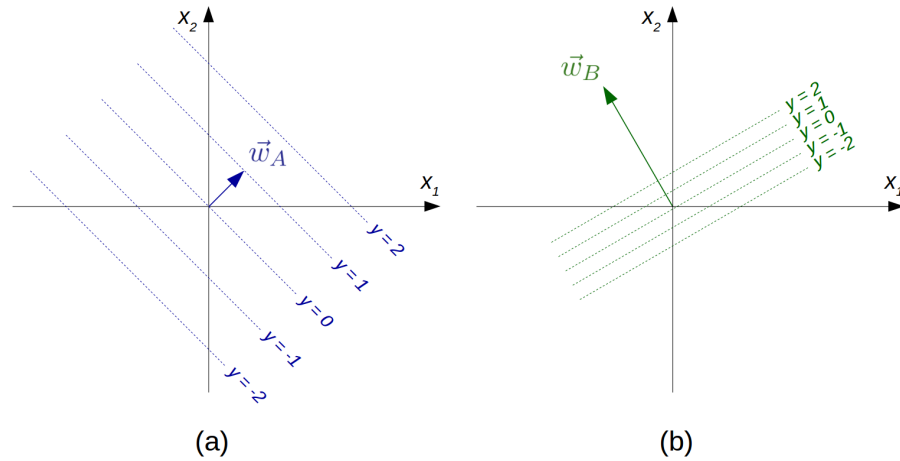


Figure 8.10: Weight vector of a linear regression as direction in a two-dimensional feature space.

machines, which extend SVMs from classification to regression tasks [138]. However, also a simple least-squares linear regression may be sufficient: As argued in Section 5.2.1, the weight vector learned by such a linear regression can also be interpreted as a direction in the feature space, pointing from examples with low target values to examples with high target values (cf. Figure 8.10). Since the solution of an unregularized linear regression is unique, also the directions extracted with a linear regression can be easily reproduced.

From directions to dimensions

As we have noted before, the Stress value of an MDS solution is invariant to rotation, reflection and translation. This means, that the coordinate axes used by the MDS solutions do not necessarily coincide with the underlying interpretable features. As Derrac and Schockaert [123] have shown, one can now use the identified meaningful directions as a new basis of the space, allowing to transform the coordinates retrieved by MDS into values on interpretable dimensions.

Finding a new basis

In order to find a new basis for the conceptual space, Derrac and Schockaert proceeded as follows: They first eliminated all candidate features (and their associated directions) with $\kappa < 0.5$ from their considerations. They then started constructing the new basis by taking the feature with the highest value of κ . Afterwards, they repeatedly added the feature i to the basis, which minimized $\max_{i' < i} \cos(\vec{w}_{i'}, \vec{w}_i)$, where \vec{w}_i and $\vec{w}_{i'}$ refer to the directions associated with features i and i' , respectively. In other words, they iteratively added the direction, which is least similar to the already selected ones. Since those directions are not necessarily orthogonal or linearly independent, Derrac and Schockaert collected $2n$ directions for an original similarity space with n dimensions in order to ensure, that the full space can be regenerated from the selected directions. Their approach can thus be criticized for yielding a redundant space – it would be much more desirable to ensure, that the dimensionality of the space remains constant.

Ager et al. [6] have made a proposal for improving the interpretability of the extracted dimensions. They argue, that some stimuli might be very dissimilar to most other stimuli and are thus located far away from all other stimuli in the similarity space. They may therefore have extreme values when being projected onto the interpretable directions. In order to deal with this potential weakness, Ager et al. propose to train a simple feedforward network for recalibrating both the points and the directions of the similarity space, such that a projection of the points onto the directions respect the ranking with respect to the corresponding feature. Ager et al. were able to show on four different datasets (the PLACE TYPE and MOVIE datasets of Derrac and Schockaert [123] plus two standard NLP datasets), that this post-processing step considerably improves the ranking of objects with respect to the interpretable directions.

*Improving
interpretability*

However, one should keep in mind, that modifying the coordinates of the points probably increases Stress, because distances and dissimilarities may not be so well-aligned any more after the transformation. One interesting avenue for future research could consist in devising an MDS algorithm, which optimizes both Stress and the quality of the interpretable directions at the same time. Such a joint optimization could let the experimenter specify, how much weight to put on the dissimilarities, and how much weight to put onto the interpretable directions, thus making the trade-off mentioned above more explicit.

A trade-off

Also Schockaert and Jameel [351] extended the work of Derrac and Schockaert: They proposed to build a qualitative representation from the interpretable directions. Schockaert and Jameel suggest to define an ordering of entities for each of the interpretable dimensions, and to use only these orderings for further processing. In order to represent a concept, one could then define an interval on each of these orderings. For instance, all ADVENTURE movies could be described as lying between two known entities e_{min} and e_{max} with respect to the EXCITING dimension. If one interval per interpretable direction is used, concepts are represented as cuboids. This is quite similar in spirit to our own formalization developed in Part I of this dissertation with the crucial difference, that we allow concepts to consist of *multiple* cuboids, which allows us to represent correlations. Schockaert and Jameel argue, that the important notion of conceptual betweenness can still be approximated based on these orderings, and that this is computationally much cheaper than computing a convex hull in high-dimensional spaces. Moreover, using only the qualitative orderings and not the underlying space makes the representation easier to interpret.

*A qualitative
representation of
concepts*

Ager et al. [5] proposed to apply stacked denoising autoencoders (cf. Section 6.3.1) to the spaces obtained by Derrac and Schockaert [123] in order to obtain more abstract representations. The hidden neurons in the bottleneck layer of this autoencoder were then interpreted as dimensions of a higher-level space, and Ager et al. applied the same method as Derrac and Schockaert in this space for finding interpretable dimensions. They repeated this procedure several times in order to find

*Obtaining more
abstract spaces*

more and more abstract representations. Moreover, they used an off-the-shelf rule learner to infer rules, which connect the directions found in the k th space to those existing in the preceding $(k - 1)$ th space. An exemplary rule extracted by this approach looks as follows:

$$IF \text{ SCARES}_2 \text{ AND } \text{ BLOOD}_2 \text{ AND } \text{ NOT } \text{ FUNNIEST}_2 \text{ THEN } \text{ GORE}_3$$

Experimental results

This can be interpreted as follows: If a movie received high scores with respect to the SCARES and BLOOD directions of the second level space, and if that movie also has a low score with respect to the FUNNIEST direction in that space, then this movie is likely to have a high score on the GORE direction in the third level space. One can interpret this as a kind of definition: Movies are likely to contain acts related to goring, if they are scary, contain blood and are not funny. In their experiments, Ager et al. observed, that higher-level spaces tended to contain more general properties like SOCIETY and RELATIONSHIP, and that the induced rules established meaningful links between the different levels of abstraction.

8.7 A HYBRID PROPOSAL

*Combining MDS
with ANNs*

Multidimensional scaling provides us with psychologically grounded similarity spaces, but is unable to generalize to unseen stimuli without collecting additional dissimilarity ratings [35]. Artificial neural networks, on the other hand, can hardly claim a high degree of psychological plausibility, but they are able to generalize to unseen examples. In this section, we outline a general proposal for combining multidimensional scaling with artificial neural networks in order combine their strengths while eliminating their individual weaknesses. This hybrid approach can result in a psychologically plausible and generalizable mapping from raw stimuli into a conceptual similarity space.

In Section 8.7.1, we introduce the overall procedure, before discussing possible network architectures in Section 8.7.2. We furthermore present related studies and proposals from the literature in Section 8.7.3.

8.7.1 Proposed Procedure

Overview

In a nutshell, we propose to use MDS for initializing a similarity space and to use the resulting coordinates as training targets for an ANN. The overall procedure is illustrated in Figure 8.11 and will now be explained in more detail. Please note, that we confine our discussion to stimuli in the form of images, but that the general approach can also be applied to other types of stimuli, e.g., sounds.

Obtaining a dataset

After having determined the domain of interest (e.g., the domain of animals), one first needs to acquire a dataset D of stimuli from this domain. This dataset should cover a wide variety of stimuli, and it should be large enough for applying machine learning algorithms.

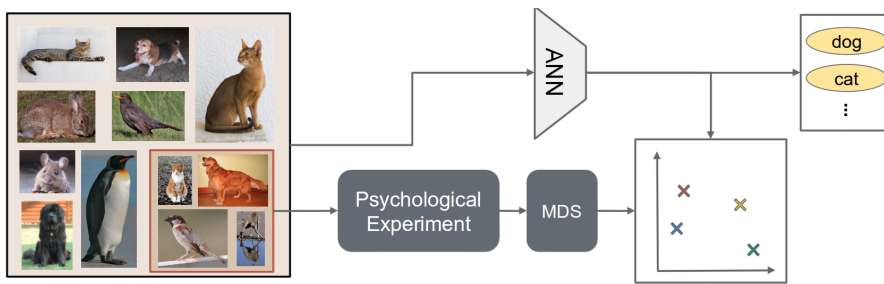


Figure 8.11: Illustration of the proposed hybrid procedure.

Using the whole dataset with potentially thousands of stimuli in a psychological experiment is, however, infeasible in practice (cf. Sections 8.1.4 and 8.1.5): The number of entries in the dissimilarity matrix grows quadratically with the number of stimuli. As each of these matrix entries needs to be based on several dissimilarity judgments, the number of participants and trials needed in a psychological study quickly becomes prohibitively large. Therefore, a relatively small, but still sufficiently representative subset $D_{Psy} \subseteq D$ of the overall dataset needs to be selected for the elicitation of human similarity ratings.

This subset of stimuli is then used in a psychological experiment, where dissimilarity judgments by humans are obtained, using one of the techniques described in Section 8.1. Please recall from Section 8.1.4, that the choice of the data collection method usually depends on the type of stimuli, the size of the dataset, and the aims of the study. Choosing an adequate elicitation method is crucial for the quality of the resulting space [397]. If for instance a large number of visual stimuli is used, for which mainly conceptual similarity is relevant, then SpAM might be a promising method. On the other hand, if low-level perceptual similarity information about auditory stimuli is desired, then a perceptual confusion task might be more appropriate.

*Eliciting
dissimilarity ratings*

In the next step, one can apply MDS to the collected similarity judgments in order to extract a spatial representation of the underlying domain. As stated earlier, one needs to manually select the desired number of dimensions – either based on prior knowledge or by manually optimizing the trade-off between high representational accuracy and compactness. The resulting similarity space should ideally be analyzed for meaningful structures and a good reflection of the original dissimilarity ratings as discussed in Section 8.6.

Applying MDS

Once this mapping from stimuli (e.g., images of animals) to points in a similarity space has been established, we can use it in order to derive a ground truth for a machine learning problem: We can simply treat the stimulus-point mappings as labeled training instances, where the stimulus is identified with the input vector, and where the point in the similarity space is used as its label. We can therefore set up a regression task from the stimulus space to the similarity space.

A regression problem

Artificial neural networks (ANNs) are powerful regressors, which are capable of discovering highly non-linear relationships between

Applying ANNs

raw low-level stimuli (such as images) and desired output variables. We therefore propose to train an ANN to map from stimuli (e.g., images) to points in the similarity space. ANNs are typically a very data-hungry machine learning method – they need large amounts of training examples and many training iterations in order to achieve good performance. However, the available number of stimulus-point pairs in our proposed procedure is quite low for a machine learning problem – as argued before, we can only look at a small number of stimuli in a psychological experiment. We propose to resolve this dilemma not only through data augmentation (cf. Sections 6.2.3 and 7.2.1), but also by introducing an additional training objective (e.g., correctly classifying the given images into their respective classes such as CAT and DOG). This additional training objective can also be optimized on all stimuli from $D \setminus D_{P_{sy}}$, i.e., the part of the overall dataset, which has not been used in the psychological experiment. Essentially, we thus propose to follow a multi-task approach (cf. Section 6.2.3) with a regression task on stimuli from $D_{P_{sy}}$ and a secondary task for all stimuli from D .

Motivation for multi-task learning

The underlying intuition is as follows: If the network is forced to learn a classification task, then it will likely develop an internal representation, where all members of the same class are represented in a similar way. For learning this internal representation, also data points without known coordinates in the similarity space are useful. The network then "only" needs to learn a mapping from this internal representation (which presumably already encodes at least some aspects of similarity between stimuli) into the target similarity space. Therefore, the secondary training objective should help the network to learn the correct mapping even from a relatively small number of data points.

Promises of our approach

If training the ANN is successful (as can be measured e.g., by the mean squared error, when comparing its predictions to the ground truth points), then it has learned a mapping from stimuli to points in the similarity space. While MDS only maps from a fixed set of stimuli to points in the similarity space, the ANN is expected to generalize this mapping to previously unseen stimuli: The ANN's internal representation of a novel CAT image is hopefully reasonably similar to the internal representation of the CAT stimulus from $D_{P_{sy}}$. Then, we can expect, that also the predicted point in the similarity space will be reasonably close to the ground truth point of the known CAT stimulus. Thus, the network can be queried for previously unseen stimuli and return a corresponding point in the similarity space. In addition to this generalization capability, the similarity space is psychologically grounded, because it is based on various "anchor points", which have been derived with MDS from the results of a psychological experiment. The ANN's predictions for novel stimuli can therefore also claim some indirect psychological grounding.

Limitations of our approach

At this point, we need to acknowledge, that while being a powerful machine learning tool, also ANNs are prone to erroneous predictions (as e.g., illustrated by adversarial examples [401]). The mapping learned by the ANN may thus occasionally return points in the similarity space,

which are clearly incorrect. This is, however, not a specific weakness of our proposal, but rather a general problem of machine learning models. We think, that in our application scenario, even an imperfect mapping can be useful, since it provides us with coordinates in the similarity space, which can then be used in downstream tasks (such as classification). This is in our opinion an important step forward, because an imperfect mapping for novel stimuli is better than no mapping at all. Moreover, erroneous points in the similarity space can potentially be discovered and properly handled as exceptions or outliers in downstream tasks. Moreover, our approach is limited to perceptual domains, since neural networks may have difficulties in learning more abstract domains based for instance on affordances or functional properties: The four legs of a horse are similar to the four wheels of a car in the sense, that they support the main body of the overall object, and that their local movement enables the overall movement of the entire object.¹¹ It is somewhat unclear, whether ANNs are capable of learning such non-perceptual notions of similarity.

8.7.2 Possible Network Architectures

Figure 8.12 illustrates different potential network architectures for our proposed procedure, where the dashed gray lines illustrate the constraints used in the network's loss function. Figure 8.12a shows a standard feedforward network, where the mapping task is supplemented by a secondary classification task. This approach is only applicable, if the dataset contains class labels. As argued in Section 8.7.1, we expect, that the classification task incites the network to use similar internal representations (i.e., activation patterns) for similar inputs (i.e., stimuli from the same class). This similarity-based internal representation should facilitate learning the mapping into the similarity space. A transfer-learning variant of this approach (with two separate training phases for the two training objectives) will be investigated in our experimental studies in Chapter 9 and 12.

If we use the representation of the similarity space as second-to-last layer in a classification network (see Figure 8.12b), we can use RBF units (cf. Sections 5.2.3 and 7.4.3) in the output layer: Their Gaussian-shaped activation function can be interpreted as a convex region in the similarity space, which is in line with the conceptual spaces theory. By using RBF neurons in the output layer for classification, the network is implicitly steered towards encoding stimuli from the same class with points, that have a small Euclidean distance. Moreover, one can interpret the activation functions of the output neurons as describing a membership function of the corresponding concepts in the similarity space. Just as deep neural networks learn both the feature space and the decision boundaries at the same time (cf. Section 6.2), this network would learn both the psychological similarity space and the conceptual regions inside it in a single training setup. Please note that this proposal

*Feedforward
classification network*

*Similarity space as
internal
representation*

¹¹ Thanks to Michael Marino for this criticism and this example.

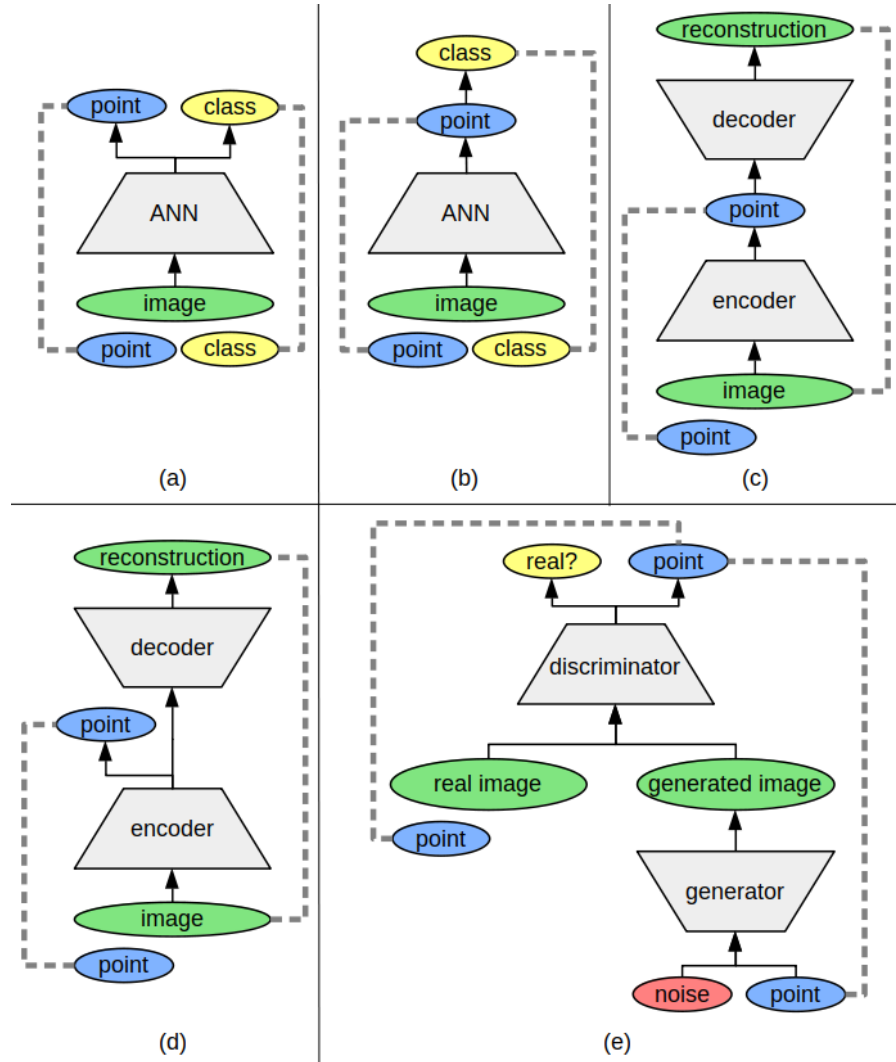


Figure 8.12: Potential network architectures for our proposed hybrid approach. Constraints are shown as dashed gray lines. (a) A standard feed-forward network with two separate outputs for coordinates and class. (b) A standard feedforward network using the coordinates as intermediate step for predicting the class. (c) An autoencoder with a constrained bottleneck layer. (d) An autoencoder with an unconstrained bottleneck layer and a separate output for the coordinates. (e) An InfoGAN network [101] with an additional reconstruction constraint for labeled images.

is highly reminiscent of ProtoNet [381] from the area of metric learning (cf. Section 5.3.3), where a neural network is trained to map from images into a low-dimensional embedding space, and where then each class is represented by a single prototype. However, ProtoNet does not consider psychologically grounded similarity ratings when constructing the embedding space, which we implicitly do by using the similarity spaces obtained through MDS as a learning target. The general approach illustrated in Figure 8.12b will play a role in our experiments in Chapter 12

Figure 8.12c shows the network structure of an autoencoder: The network is trained to minimize the difference between the input images and their reconstruction, while being encouraged to use the similarity space in its bottleneck layer. A variant of this approach does not use the similarity space as the bottleneck layer of the autoencoder, but rather uses an additional output, which predicts MDS coordinates based on the unconstrained bottleneck layer (see Figure 8.12d). Since the computation of the reconstruction error does not need any class labels, this variant is applicable also to unlabeled datasets, which are in general larger and easier obtain than labeled datasets.

In the case of an autoencoder, the underlying intuition of the secondary task is the following one: The network needs to accurately reconstruct the given stimuli, while using only information from a small bottleneck layer. On the one hand, the network needs to encode all perceptually relevant information in this bottleneck layer. On the other hand, the small size of the bottleneck layer creates an incentive to encode similar input stimuli in such a way that the created reconstructions are also similar to each other (cf. Sections 6.3.1 and 6.3.4). The network is therefore expected to implicitly learn a similarity relation of its inputs, which should be useful for learning the mapping into the similarity space. Of course, instead of using a regular autoencoder, one can also employ VAE [224] or β -VAE [196] as discussed in Section 6.3.4. The autoencoder structure has the additional advantage, that one can use the decoder network to generate an image based on a point in the similarity space. This can be a useful tool for visualization and further analysis. We will investigate such autoencoder-based networks in our experiments in Chapter 12.

Figure 8.12e shows an extended version of the InfoGAN framework [101] (cf. Section 6.3.3), where the discriminator network is also constrained to correctly extract the coordinates in the similarity space. This is in some sense reminiscent of the AC-GAN network [304], which, however, predicts categorical class labels instead of continuous points in a semantic space. Also this network architecture is capable of generating images based on a given point in the conceptual space and does not need a labeled dataset.

Of course, it is also possible to combine the proposed architectures, e.g., by using both classification and reconstruction as secondary tasks.

Autoencoders for semi-supervised learning

Intuition behind the reconstruction objective

InfoGAN

Combining multiple approaches

Moreover, there might be additional secondary tasks one might want to consider, which may lead to other network architectures.

*Predicting
dissimilarities
instead of coordinates*

Instead of learning a direct mapping from images to points in a conceptual space, one can also train a network to predict the dissimilarity ratings for pairs of images. In this case, the neural network receives two images as input and predicts their similarity. This modification causes the input size to double, but leads to a smaller output size and a larger number of different training inputs. Again, an additional training objective might be needed in order to prevent overfitting. In principle, the approaches discussed above can be adapted also to this learning problem. One can use such a network to indirectly map an image onto a point in the similarity space as follows: The network predicts the dissimilarity of the given image to a fixed set of "anchor images", for which a mapping into the similarity space is known. One can then use the points representing these anchor images together with the predicted dissimilarity ratings to triangulate the point representing the new image. This modified approach is highly reminiscent of the work by Peterson et al. [318, 319] introduced in Section 8.1.5, who used the internal representations of pretrained classification networks to predict dissimilarity scores.

*An ANN-based
MDS algorithm*

If the general approach described in Section 8.7.1 yields promising and useful results, one could use the observation, that InfoGAN [101] (cf. Section 6.3.3) and β -VAE [196] (cf. Section 6.3.4) tend to discover meaningful dimensions, in order to devise a new MDS algorithm based on these networks. This algorithm would train a standard β -VAE or InfoGAN network, while ensuring through an additional term in the loss function, that the hidden space extracted by the network accurately reflects the psychological dissimilarity ratings. This overall algorithm would thus result in a spatial representation of psychological similarities, which generalizes to unseen images, uses interpretable dimensions, and can be used to generate new images based on points in the conceptual space.

Modified β -VAE

If we for instance would like to extend the β -VAE framework from Section 6.3.4 (Equation 6.4), we could simply add raw Stress as defined in Section 8.2.1 (Equation 8.3) to the loss function:

$$\mathbb{E}_{\vec{z} \sim \mathbb{Q}_\phi(\vec{z}|\vec{x})} [\log \mathbb{P}_\theta(\vec{x} | \vec{z})] - \beta \cdot D_{KL}(\mathbb{Q}_\phi(\vec{z} | \vec{x}) || \mathbb{P}(\vec{z})) \\ - \gamma \cdot \sum_{j_1=1}^N \sum_{j_2=j_1+1}^N w_{j_1 j_2} \cdot \mathbb{E}_{\substack{\vec{z}^{(j_1)} \sim \mathbb{Q}_\phi(\vec{z}^{(j_1)}|\vec{x}^{(j_1)}) \\ \vec{z}^{(j_2)} \sim \mathbb{Q}_\phi(\vec{z}^{(j_2)}|\vec{x}^{(j_2)})}} \left[\left(d_{j_1 j_2}(Z) - \hat{d}_{j_1 j_2} \right)^2 \right]$$

*Interpreting the loss
function*

This introduces another hyperparameter γ , which represents the relative importance of correctly representing the disparities $\hat{d}_{j_1 j_2}$. The resulting network thus tries to achieve a compromise between reconstruction (first term), interpretability (second term, weighted by β), and psychological plausibility (third term, weighted by γ). In this case, β and γ are two important hyperparameters of the model, which regulate the trade-off between the different training objectives.

On a final note, we want to comment on the difference between perceptual and conceptual similarity (cf. Section 8.1.2) in the context of our proposal. *Perceptual similarity* focuses on the similarity of the raw stimuli, e.g., with respect to their *SHAPE*, *SIZE*, and *COLOR*. *Conceptual similarity*, on the other hand, works on a more abstract level and involves conceptual information such as the typical *USAGE* of an object or the typical *CONTEXT*, where a given object might be found. For instance, a *VIOLIN* and a *PIANO* are perceptually not very similar, because they have different *SIZE* and *SHAPE*. Conceptually, they might, however, be quite similar, because they are both musical instruments, which can be found in an orchestra.

*Perceptual and
conceptual similarity*

While class labels can be assigned on both the perceptual (*ROUND* vs. *ELONGATED*) and the conceptual level (*MUSICAL INSTRUMENT* vs. *FRUIT*), the reconstruction objective always operates on the perceptual level. If the dissimilarity data collected in the psychological experiment is of perceptual nature, then all proposed networks architectures seem promising. If we, however, target conceptual similarity, then a feed-forward network with target classes on the conceptual level seems to be preferable, because both the (variational) autoencoder and the InfoGAN might be led astray by their reconstruction objective.

*Suitable network
architectures*

8.7.3 Related Work

As we have already discussed in Section 8.1.5, Peterson et al. [318, 319], Erdogan et al. [146], and Kubilius et al. [235] have investigated the correlation between the internal representations used by ANNs and human dissimilarity ratings. While their respective results illustrate, that there is a connection between the features learned by neural networks and human dissimilarity ratings, their work differs from our proposed approach in one important aspect: All of the above cited researchers focus on the abstraction level of dissimilarity ratings, whereas we work with similarity spaces. While they do apply MDS to investigate the quality of the ANN's predictions, their primary goal is to find a way to predict the dissimilarity ratings directly. Our research, on the other hand, is not that much interested in the actual dissimilarity ratings, but in the underlying similarity space. Therefore, we use MDS in order to generate a similarity space, whose structure we try to learn with a neural network. Nevertheless, the results cited above provide support for our general proposal by establishing a link between artificial neural networks and human dissimilarity judgments.

*ANNs and
dissimilarity ratings*

Recently, Sanders and Nosofsky [346, 347] have independently made a very similar hybrid proposal for using artificial neural networks to map input images to coordinates in MDS-based similarity spaces. In their exemplary study, they trained an ensemble of convolutional neural networks (cf. Section 6.2.2) to predict the correct coordinates in an MDS-based similarity space for dataset of rocks. They used the dataset

*Predicting
coordinates in a
similarity space of
rocks*

provided by Nosofsky et al. [303]¹², which consists of 360 pictures of rocks along with an 8-dimensional similarity space, which has been extracted by using MDS on human dissimilarity ratings. Sanders and Nosofsky split up this dataset into 180 images for training, 90 images for validation, and 90 images for testing. Since this dataset is considerably too small for training an ANN from scratch, they used a pretrained network (ResNet50 [193], which was trained on ImageNet [120]) as a starting point. They removed the topmost layer and replaced it by two untrained fully connected layers with an output of eight linear units, one per dimension of the similarity space. In order to increase the size of their dataset, they applied data augmentation methods by flipping, rotating, cropping, stretching and shrinking the original images (cf. Sections 6.2.3 and 7.2.1). First, only the new layers were trained (using Adam [223], cf. Section 6.2.5), before all weights of the whole network were fine-tuned (using stochastic gradient descent with a low learning rate and high momentum).

Experimental results

Their ensemble of ten networks (each trained using the same procedure, but with a different random initialization) achieved a coefficient of determination of $R^2 = 0.767$ on the test set, which means, that almost 77% of the variance was accounted for by the neural network (cf. Section 5.1.2). As an additional means of evaluation, Sanders and Nosofsky also conducted a categorization experiment with humans on a second dataset of rocks. They then applied a simple exemplar-based classification model on the original MDS space, the predictions of the trained ANNs, and the activation vectors of a pretrained neural network. Their results showed, that the original MDS space could account for 89.7% of the variance in the human classifications, while the MDS coordinates predicted by the ensemble of ANNs reached a slightly lower, but still comparable value of $R^2 = 0.882$. Using the hidden activations of a pretrained neural network and optimized weights for the individual dimensions (following the approach by Peterson et al. [319], cf. Section 8.1.5) yielded a considerably lower value of $R^2 = 0.797$. This highlights, that off-the-shelf networks are clearly inferior to a hybrid approach.

Extending the similarity space

In a final analysis step, Sanders and Nosofsky [347] manually extended the MDS-based similarity space with five additional dimensions. These additional dimensions corresponded to domain-specific features, which are very discriminatory in classification tasks, but not very salient when making general dissimilarity judgments. By extending the eight-dimensional similarity space with these additional five features (scaled in a suitable way to have a similar range of values as the MDS dimensions), the exemplar model was able to explain 93.6% of the variance observed in human classification performance. Sanders and Nosofsky also re-trained their ensemble of ANNs in order to predict also the values on the additional five dimensions with a test set performance of $R^2 = 0.707$. The difference to the value of $R^2 = 0.767$ obtained on the test set, when considering only the eight dimensions of the MDS space, can be explained by the larger number of output dimensions, which

¹² Publicly available online at <https://osf.io/w64fv/>.

makes the regression problem more difficult. An exemplar model on the ANN-based predictions was able to predict 90.6% of the variance observed in the psychological experiment, which indicates, that also a hybrid approach is capable of taking into account additional features in order to improve psychological validity.

The results by Sanders and Nosofsky clearly show that our proposed hybrid procedure is feasible in practice. However, in contrast to our own work, they do not work with the framework of conceptual spaces, and therefore do not consider a distinction of different cognitive domains (such as *SHAPE*, *SIZE*, and *COLOR*). Moreover, all their experiments only involve a single eight-dimensional target space. We will extend their work in two studies reported in Chapters 9 and 12 by taking exactly these two aspects into consideration.

Relation to our work

We have already mentioned in Section 8.7.2, that one of our proposed network architectures can be related to ProtoNet [381]. In general, our hybrid proposal can be seen as a variant of nonlinear metric learning [236] as introduced in Section 5.3.3: We seek to map raw images into a low-dimensional similarity space by training an artificial neural network. The training objective of most metric learning approaches leverages classification information by requiring, that points belonging to the same class should have a small distance in the similarity space, while points belonging to different classes should have a large distance. In contrast to that, our own approach tries to minimize the difference between the network's output and the coordinates in a similarity space obtained by MDS. Our training objective thus depends (although indirectly) on psychological similarity ratings. However, classification information also indirectly plays a role, if the secondary training objective of the network is related to classification.

Our hybrid approach and metric learning

Similar to what we discussed for the extended β -VAE in Section 8.7.2, one can easily imagine to train a network from scratch with a training objective explicitly based on the dissimilarity ratings and classification information. In this case, the dissimilarity ratings (or disparities) give rise to much finer-grained constraints than usually considered in metric learning. The resulting system could then be interpreted as a novel MDS algorithm, which makes use of additional information in the form of class labels. Moreover, if values of candidate features are given, one can interpret these as spanning a feature space. One could then try to apply metric learning to this feature space, using the psychological dissimilarities as additional relative constraints. By investigating the learned mapping, one may then be able to gain information about the relation of the interpretable features to the overall dissimilarity ratings.

Ideas for extensions

In Section 7.4.3, we have already introduced *relational label smoothing* [276] as a way of injecting background knowledge into artificial neural networks: The target outputs for the final softmax layer of a classification network are modified in such a way, that a fraction of the probability mass is shifted from the correct class to semantically related classes. This incites an internal representation, which respects

Our hybrid approach and relational label smoothing

the similarity relation between classes. Relational label smoothing can thus be related to metric learning, with the key difference of defining similarity constraints not on the level of individual data points, but on the level of classes. Again, one can envision an overall system, which takes into account human dissimilarity ratings for pairs of stimuli (in the form of Stress), classification information (in the form of metric learning constraints), and information about the relations between classes (in the form of relational label smoothing).

*Our hybrid approach
and few shot learning*

In the context of few shot learning, Lampert et al. [241] have proposed to use attributes as an intermediate layer in a classifier cascade (cf. Section 7.2.1). Essentially, they derive binary attributes for a given set of example classes and train one classifier on each these attributes. The classes are then defined based on the presence and absence of attributes. Classification of a new stimulus then takes place by applying the classifiers to retrieve attribute values, and the class definitions to determine class membership. In a zero shot learning context, new classes can then be added simply based on their definition in terms of the already trained attributes. This overall proposal is somewhat similar to our hybrid approach by introducing an intermediate representational layer (binary attributes in their case, a continuous MDS-based similarity space in our case). While Lampert et al. use multiple nonlinear SVMs on traditional computer vision features to learn a mapping into the space of attributes, we use a single convolutional neural network for this mapping task. If we assume, that attribute values for our stimuli are given, one could potentially augment our network architectures with additional output units for predicting these attribute values. This corresponds to finding interpretable directions in the similarity space.

*Combining computer
vision, MDS, and
ANNs*

Finally, we would also like to mention the work by Morgenstern et al. [297], which will be introduced in more detail in Chapter 10: They started from a large number of computer vision features for describing shapes and obtained a global dissimilarity matrix following a data-driven way similar to the ones described in Section 8.1.5. By using classical MDS, they obtained a 22-dimensional similarity space, which aligned quite well with human dissimilarity judgments. Morgenstern et al. then trained a convolutional neural network to map from raw inputs (either low-resolution pixel information, or the coordinates for points on the surface of the object) into this similarity space. Their work thus essentially follows our proposed hybrid procedure, but uses computer vision features rather than human dissimilarity ratings as a starting point. Nevertheless, since their resulting similarity space seems to correlate well with human ratings, it is to some extent psychologically grounded. We will take a more detailed look at their work in Chapter 10 in the context of the SHAPE domain.

8.8 SUMMARY

In this chapter, we have introduced the technique of multidimensional scaling, which can be used for constructing similarity spaces from dissimilarity ratings collected in psychological studies. As we have seen in Section 8.1, there are many different ways of collecting such dissimilarity ratings. Moreover, there exists a wide variety of algorithms for solving the MDS optimization problem (cf. Sections 8.2 to 8.5). The resulting similarity spaces can be evaluated by using Stress, which measures, how well the dissimilarities are reflected in the distances, and which is explicitly being minimized by the algorithms. Moreover, one can also take a look at the interpretability of the configurations which includes the search for meaningful clusters and the identification of directions corresponding to meaningful features (cf. Section 8.6).

Lessons learned

The main advantage of constructing conceptual spaces through MDS lies in its psychological grounding: Since the resulting similarity spaces are directly based on ratings elicited in psychological studies, they can be deemed to be psychologically plausible. However, MDS also has its shortcomings: On the one hand, the axes of the coordinate system returned by MDS do in general not coincide with interpretable features from the given domain. However, the conceptual spaces framework assumes, that the similarity spaces are spanned by such meaningful features. Semi-automated ways of discovering such features as directions in the similarity spaces can alleviate this weakness and help to identify a more meaningful basis for the similarity space (cf. Section 8.6.2).

Merits and limitations of MDS

Moreover, MDS is not able to generalize to unseen examples – we can in general not predict the coordinates for novel stimuli, which were not part of the psychological study. Our hybrid proposal from Section 8.7 aims at also alleviating this shortcoming by using neural networks for this mapping task.

Our hybrid approach

In the following chapters, we will present two studies based on MDS and our hybrid proposal: Chapter 9 reports the results of a first feasibility study. There, we investigate, whether the dissimilarities obtained with SpAM should be interpreted as ratio scaled or as ordinally scaled (cf. Section 8.1.4). This is done by comparing the results of metric MDS (which assumes ratio scaled data) to nonmetric MDS (which only assumes an ordinal scale). Moreover, we use a linear regression on top of a pretrained convolutional neural network to provide a first proof of concept for our hybrid approach.

Outlook: a first feasibility study...

Afterwards, we conduct a deeper analysis of the SHAPE domain in Chapters 10 and 11, where we also investigate the effect of different aggregators for averaging dissimilarities across participants. Moreover, we compare different ways of automatically extracting directions in the similarity space, which correspond to psychologically meaningful shape features. Finally, in Chapter 12 we apply the second step of our hybrid approach to the SHAPE domain, comparing the transfer learning approach from Chapter 9 to alternative architectures and training schemes (cf. Section 8.7.2).

...and an investigation of the SHAPE domain

Part III

EXPERIMENTAL STUDIES

In the third part of this dissertation, we report the results of two empirical studies based on our hybrid proposal from Chapter 8: In Chapter 9, we conduct a first feasibility study on a data set of novel and unknown objects, investigating also the difference between metric and nonmetric multidimensional scaling (MDS) on a dissimilarity data collected with the spatial arrangement method. We afterwards shift our focus to the `SHAPE` domain: In Chapter 10, we summarize related work on shape perception and representation from both psychology and computer vision, before introducing and analyzing a novel dataset of line drawings. This dataset is then used in Chapter 11 to extract similarity spaces of the `SHAPE` domain using MDS. In this context, we also investigate, whether central predictions of the conceptual spaces theory are met by the resulting spaces, and whether averaging the individual ratings with the mean or the median makes a difference. Finally, in Chapter 12, we investigate transfer learning and multi-task learning, based on both classification networks and autoencoders for learning a mapping from raw images into our `SHAPE` similarity spaces.

9

FEASIBILITY STUDY ON NOVEL OBJECTS

9.1	Comparing Metric and Nonmetric MDS	514
9.1.1	The NOUN Dataset	514
9.1.2	Methods	516
9.1.3	Results	523
9.1.4	Discussion	532
9.2	Machine Learning Experiments	533
9.2.1	General Methods	534
9.2.2	Analyzing Feature Spaces and Baselines	539
9.2.3	Comparing Feature Spaces and Regressors	542
9.2.4	Comparing MDS Algorithms	548
9.2.5	Generalization to Other Target Spaces	553
9.2.6	Discussion	557
9.3	Summary	559

In this chapter, we use existing dissimilarity ratings for a dataset of novel, but realistic stimuli as a first test bed for the hybrid approach proposed in Section 8.7. Our analysis consists of two main steps: We first obtain psychological similarity spaces with MDS, before using regression algorithms to learn a mapping from raw images into these similarity spaces.

A feasibility study

The dissimilarities we use have been collected with the spatial arrangement method (SpAM, cf. Section 8.1.2) [169], where participants are repeatedly asked to arrange a subset of stimuli on their computer screen based on their pairwise dissimilarity. The dissimilarity ratings are then collected as Euclidean distances between the items, which has led some researchers to interpret them as ratio scaled [200] (cf. Section 8.1.4). In Section 9.1, we investigate, whether this assumption is justified, by comparing the performance of four different MDS variants (two metric and two nonmetric) based on the algorithms presented in Chapter 8. If the dissimilarity values are indeed ratio scaled, then metric MDS algorithms should have an advantage over nonmetric MDS, since they explicitly take into account interval and ratio information. Since the stimuli are novel, we furthermore assume, that the dissimilarity judgments are mainly based on perceptual information. We therefore also measure, to which degree two simple image-based approaches can predict the dissimilarity ratings. These approaches are based on raw pixel information and on the hidden activations of a pretrained neural network, respectively (replicating the setup of Peterson et al. [318, 319], cf. Section 8.1.5).

Comparing metric and nonmetric MDS

After having extracted and analyzed various similarity spaces, we consider the task of mapping raw images onto points in these similarity spaces in Section 9.2. Our experiments in this context are strongly

Machine learning experiments

related to the study by Sanders and Nosofsky [346, 347] (cf. Section 8.7.3), exploring, however, a wider variety of setups: We compare a simple linear regression, a random forest regression, and a lasso regression on two types of feature spaces, namely, the pixels of downscaled images and the hidden activations of a pretrained neural network. These two feature spaces correspond to the two image-based prediction approaches used in Section 9.1. Moreover, we investigate the influence of the type of MDS algorithm as well as the number of dimensions in the similarity space on the regression results. Like Sanders and Nosofsky, we limit ourselves to a classification-based transfer learning setup.

Section 9.3 then summarizes the main takeaways of our investigations.

The content of this chapter is partially based on work previously published in [40, 47].¹ All raw results of our study and the source code for reproducing them are publicly available on GitHub [37].²

9.1 COMPARING METRIC AND NONMETRIC MDS

*The measurement
scale of SpAM
ratings*

As already discussed in Section 8.1.4, it is debatable, whether metric or nonmetric MDS should be used with data collected through SpAM: On the one hand, dissimilarity ratings are recorded as Euclidean distances between items on the computer screen, and may thus be interpreted as ratio scaled. If this is the case, then metric MDS should be able to include the information about differences and ratios as useful constraints into the optimization procedure. On the other hand, if the dissimilarities are only ordinally scaled, then the main assumption of metric MDS is violated. Nonmetric MDS makes less assumptions about the underlying measurement scale and therefore seems to be the "safer" choice.

Overview

In this section, we use the NOUN dataset by Horst and Hout [199] in order to compare the spaces obtained with metric MDS to the ones obtained with nonmetric MDS. This dataset is "freely available to the scientific community from the authors for noncommercial use" [199]. In Section 9.1.1, we present this dataset in more detail, and we motivate our hypotheses for this study. We afterwards describe the methods used in our analysis in Section 9.1.2, before presenting the obtained results in Section 9.1.3. Finally, in Section 9.1.4, we discuss the overall findings and their implications.

9.1.1 The NOUN Dataset

The stimuli

For our study, we used existing dissimilarity ratings reported for the *Novel Object and Unusual Name (NOUN)* dataset [199], a set of 64 images

¹ The initial workshop paper [40] is based on a collaboration with Elektra Kypridemou. In [40], Elektra Kypridemou has described the general background with respect to multidimensional scaling (MDS), and has extracted similarity spaces with MDS from the raw data, while I have mainly developed and described the hybrid approach, and implemented and evaluated the machine learning experiments.

² See <https://github.com/lbechberger/LearningPsychologicalSpaces/>.

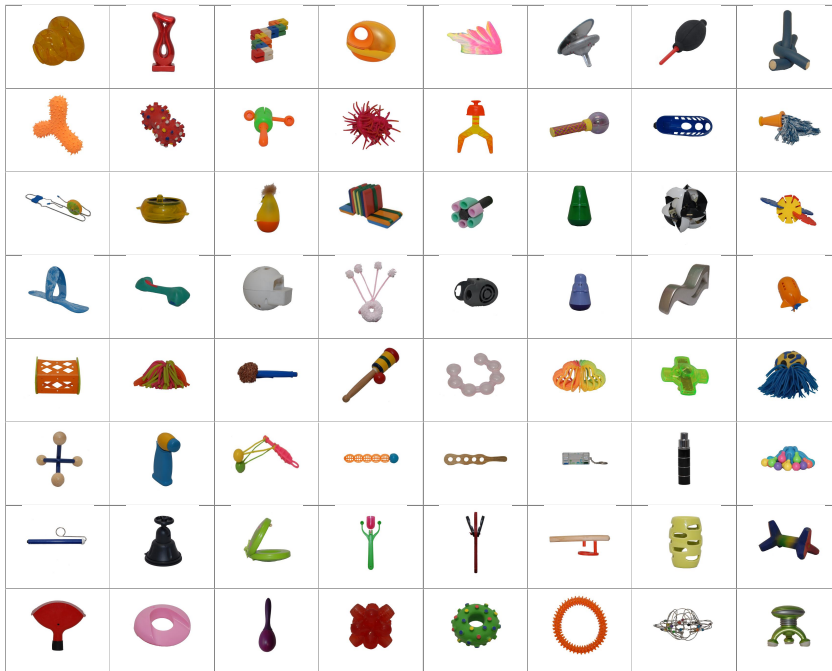


Table 9.1: Overview of the stimuli in the NOUN dataset [199].

of three-dimensional objects, which were designed to be novel, but also to look naturalistic. Table 9.1 visualizes all stimuli from this dataset.

Since the objects from the NOUN dataset are designed to be novel, they have only little resemblance with everyday objects. Hence, prior knowledge about their typical usage or their relations to other objects does not exist and therefore cannot influence the dissimilarity ratings. We can thus expect, that the dissimilarities between pairs of these objects are mainly based on perceptual information such as COLOR, SHAPE, SIZE, and TEXTURE. Since conceptual spaces are considered in this dissertation mainly as a spatial representation of *perceptual* information, this dataset seems to be a good starting point for a first feasibility study. One should, however, note, that while being limited to visual perception, the dissimilarity ratings from the NOUN dataset may still combine information from multiple domains (e.g., SHAPE and COLOR).

Horst and Hout [199] adopted the SpAM approach [169] (cf. Section 8.1.2) in order to elicit pairwise dissimilarity ratings for these stimuli. Participants were presented with 13 trials of 20 objects each, randomly assigned in such a way, that all pairwise comparisons among the 64 images were evaluated. At the beginning of each trial, the 20 stimuli were arranged in four rows and five columns on a computer screen. Participants were then asked to re-arrange these stimuli in such a way, that the distance between each pair of stimuli in the final configuration reflected the dissimilarity of the two stimuli. The pixel-based Euclidean distance of the stimuli in the final configuration was used in order to calculate an overall dissimilarity matrix for each participant, which

*Assuming perceptual
dissimilarities*

*Eliciting
dissimilarities*

were then averaged into a global dissimilarity matrix.³

Applying MDS

In their analysis of the collected dissimilarity data, Horst and Hout [199] applied *metric* MDS in order to obtain a four-dimensional psychological similarity space.⁴ As argued by Hout et al. [200], dissimilarities were treated as ratio-level data, because they were computed based on the Euclidean distances on the computer screen.

Ratio or ordinal scale?

However, as discussed in Section 8.1.4, it is debatable, whether the dissimilarity ratings obtained through SpAM should be considered as ratio scaled, since participants may produce only very rough arrangements of the stimuli. In our current study, we therefore want to investigate this issue by applying both metric MDS (which assumes a ratio scale) and nonmetric MDS (which only assumes an ordinal scale) to the data collected by Horst and Hout [199].⁵

Expectations

If the dissimilarities obtained with SpAM are ratio scaled, then the differences and ratios of dissimilarities should contain considerable amounts of additional information. Since metric MDS is able to use this additional source of information, it should have a clear advantage over nonmetric MDS and create spaces of a higher quality. If the dissimilarities obtained with SpAM are, however, only ordinally scaled, then the main assumption of metric MDS is violated. The differences and ratios of dissimilarities are in this case not meaningful, and may thus be misleading. In this case, we expect, that metric MDS produces solutions of lower quality than nonmetric MDS. Hence, by comparing the results obtained with metric MDS to the ones obtained with nonmetric MDS, we can make some inferences about the underlying scale of measurement of the dissimilarity ratings.

9.1.2 Methods

MDS algorithms

In order to investigate the differences between metric and nonmetric MDS in the context of dissimilarity ratings obtained with SpAM, we compared the three MDS algorithms introduced in Chapter 8:

Classical MDS as metric approach

- *Classical multidimensional scaling* [174, 408] is a variant of *metric* MDS, which uses an Eigenvalue decomposition in order to derive the coordinates in the similarity space (cf. Section 8.3). We used the `cmdscale()` function from the `stats` library in R.⁶

Kruskal's nonmetric algorithm

- *Kruskal's nonmetric MDS algorithm* [233, 234] is based on gradient descent (cf. Section 8.4). We used the implementation available in the `MASS` library of R.⁷ We ran the algorithm with 256 random

³ We would like to thank Michael Hout for providing us with all participant-based dissimilarity matrices.

⁴ The type of MDS algorithm being used is not stated explicitly in their paper, but was confirmed through a private communication with Michael Hout in January 2019.

⁵ Please note, that we use the terms "metric MDS" and "nonmetric MDS" to refer to absolute and ordinal MDS, respectively (cf. Section 8.2.2).

⁶ See <https://stat.ethz.ch/R-manual/R-devel/library/stats/html/cmdscale.html>.

⁷ See <https://stat.ethz.ch/R-manual/R-devel/library/MASS/html/isoMDS.html>.

starts, where the maximum number of iterations per random start was set to 1,000. The overall best result over these 256 random starts was kept as final result.

- The *SMACOF algorithm* [115] uses iterative function majorization for minimizing Stress, and can be used both in a *metric* and in a *nonmetric* variant (cf. Section 8.5). The underlying algorithm stays the same, only the definition of Stress and thus the optimization target differs. Both variants were explored in our study, using the original implementation from R's `smacof` library [116].⁸ We used 256 random starts with the maximum number of iterations per random start set to 1,000. The overall best result over these 256 random starts was kept as final result.

Metric and nonmetric SMACOF

For each of the four MDS variants, we constructed MDS spaces of different dimensionality (ranging from one to ten dimensions). Most MDS algorithms explicitly minimize Stress, i.e., the difference between the dissimilarities predicted by the similarity space and the actual dissimilarities. Remember from Section 8.2.1, that in its most general form, Stress-1 of a configuration X is defined as follows (where $d_{j_1 j_2}(X)$ is the distance between the points j_1 and j_2 in the configuration X , the weight $w_{j_1 j_2}$ indicates the importance of the item pair (j_1, j_2) , and the disparities $\hat{d}_{j_1 j_2}$ are based on the original dissimilarities):

The definition of Stress

$$\sigma_1(X) = \sqrt{\frac{\sum_{j_1=1}^N \sum_{j_2=j_1+1}^N w_{j_1 j_2} \cdot (d_{j_1 j_2}(X) - \hat{d}_{j_1 j_2})^2}{\sum_{j_1=1}^N \sum_{j_2=j_1+1}^N w_{j_1 j_2} \cdot d_{j_1 j_2}^2(X)}} \quad (8.4)$$

For metric Stress, the disparities $\hat{d}_{j_1 j_2}$ are identical to the original dissimilarities $\delta_{j_1 j_2}$, while for nonmetric Stress, they are obtained through a monotone regression from dissimilarities to distances (cf. Sections 8.2.2 and 8.4.2). We use the `stress0` function from R's `smacof` package to compute both metric and nonmetric Stress-1 for all similarity spaces, using the settings "ratio" and "ordinal", respectively. We expect Stress to decrease as the number of dimensions increases.

Computing Stress

Nonmetric Stress quantifies the difference between the *ordering* of dissimilarities and distances, while metric Stress measures the difference between their actual numeric values. If the data obtained through SpAM is ratio scaled, then we would expect, that metric MDS achieves better values on metric Stress (and potentially also on nonmetric Stress) than nonmetric MDS. If the SpAM dissimilarities are not ratio scaled, then metric MDS should not have any advantage over nonmetric MDS.

Expectations

As an additional way of evaluation, we measured the correlation between the distances in the MDS space and the original dissimilarity ratings. Since a high correlation is expected to correspond to low values of Stress and vice versa, this seems to be a valid approach of evaluation. It has the advantage, that correlation coefficients are more well-known

Considering correlations

⁸ See <https://cran.r-project.org/web/packages/smacof/smacof.pdf>

and potentially easier to interpret than raw Stress values.⁹ Since we assume, that the dissimilarity ratings are mainly based on perceptual information, we also investigated, to which extent they can be predicted from the original images. In order to do so, we additionally included two baselines into our correlation analysis, leading to the overall setup depicted in Figure 9.1:

Pixel baseline

The *pixel baseline* uses the pixel-based distance of downscaled images: For each original image (with both a width and height of 300 pixels), we created lower-resolution variants by aggregating all the pixels in a $k \times k$ block into a single pixel (with $k \in [2, 300]$). We compared different aggregation functions, namely, minimum, mean, median, and maximum. The pixels of the resulting downscaled image were then interpreted as a point in a $\lceil \frac{300}{k} \rceil \times \lceil \frac{300}{k} \rceil$ dimensional space. If the dissimilarity ratings are based on the overall shape of the stimuli, then a more coarse-grained representation of the images (corresponding to larger block sizes) should yield better results.

ANN baseline

For the *ANN baseline*, we extracted the activation vectors from the second-to-last layer of the inception-v3 network [400] (cf. Section 6.2.2) for each of the images from the NOUN dataset. Each stimulus was thus represented by its corresponding activation pattern. While the downscaled images can be seen as surface level features, the activation patterns of the neural network can be interpreted as a more abstract representation of the image.

Distances

For each of the three representation variants (downscaled images, ANN activations, and points in an MDS-based similarity space), we computed three types of distances between all pairs of stimuli, namely the Euclidean distance d_E , the Manhattan distance d_M , and the negated inner product d_{IP} :

$$d_E(\vec{x}^{(j_1)}, \vec{x}^{(j_2)}) = \sqrt{\sum_{i=1}^N w_i^2 \cdot (x_i^{(j_1)} - x_i^{(j_2)})^2}$$

$$d_M(\vec{x}^{(j_1)}, \vec{x}^{(j_2)}) = \sum_{i=1}^N w_i \cdot |x_i^{(j_1)} - x_i^{(j_2)}|$$

$$d_{IP}(\vec{x}^{(j_1)}, \vec{x}^{(j_2)}) = -1 \cdot \sum_{i=1}^N w_i^2 \cdot x_i^{(j_1)} \cdot x_i^{(j_2)}$$

Motivation for the inner product

While d_E and d_M measure distance in the usual sense (cf. Section 2.1.1), d_{IP} is based on the inner product of two vectors. The inner product is often used as a measure of *similarity* for activation vectors of artificial neural networks (cf. the study by Peterson et al. [318, 319], Section 8.1.5, and the usage of the Cosine similarity in word embeddings [291], Section 6.3.2). Since we are, however, interested in a measure of *dissimilarity*, we simply take the negative value of the inner product.

Dimension weights

In all of the distances listed above, we have included weights w_i for

⁹ Comparing dissimilarity matrices of different systems with correlation coefficients is also known as *relational similarity analysis* (RSA) in neuroscience [230].

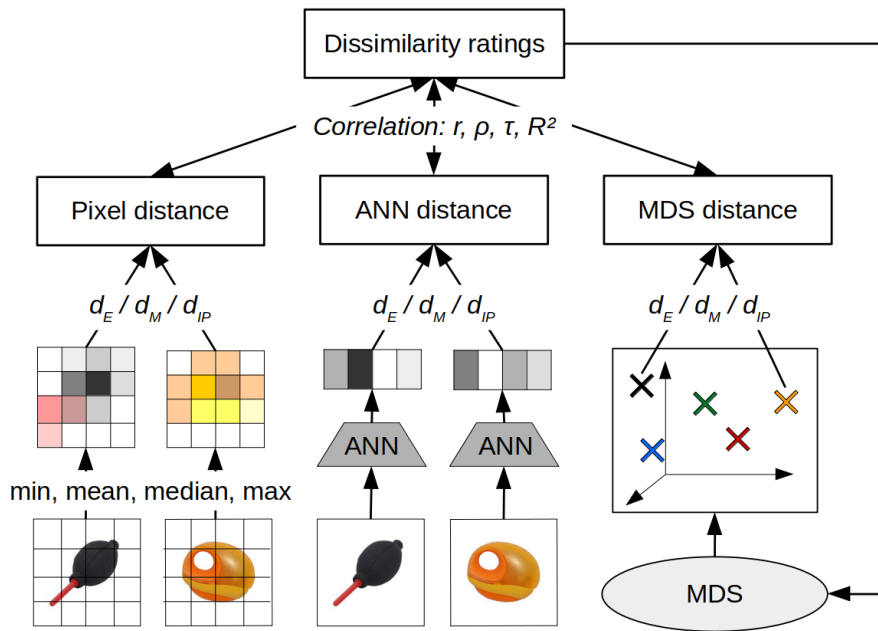


Figure 9.1: Illustration of our analysis setup, which considers distances based on raw pixels (left), activations of an artificial neural network (middle), and similarity spaces obtained with MDS (right).

each of the dimensions i . By default, all of these weights are set to one, i.e., all dimensions are considered to be of equal importance. However, as Peterson et al. [318, 319] have shown, the quality of the dissimilarities predicted by the activation vectors of a neural network can be greatly improved by optimizing the dimension weights w_i (cf. Section 8.1.5). Similar to their approach, we use a non-negative least squares regression in a five-fold cross validation to estimate these weights. Especially for the high-dimensional feature vectors produced by the ANN baseline and the pixel baseline, this cross-validation scheme is necessary in order to avoid overfitting: Since the number of dimensions is considerably larger than the number of data points, it might otherwise be too easy to find an optimal set of weights, which yields a very high correlation on the training data, but does not generalize well. In our analysis, we will consider both uniform and optimized weights.

For each of the resulting distance matrices, we compute four correlation coefficients with respect to the target dissimilarity ratings, namely, Pearson's r [315], Spearman's ρ [384], Kendall's τ [219], and the coefficient of determination R^2 [439] for a monotone regression. When computing these correlation coefficients, we only consider matrix entries above the diagonal, because the matrices are symmetric, and because all distances on the diagonal are by definition zero. In the following, we describe the four correlation coefficients in more detail.

Pearson's r [315] measures the linear correlation of two random variables by dividing their covariance by the product of their individual

Computing correlations

Pearson's r

variances. Given two vectors \vec{a} and \vec{b} (each containing k samples from the random variables A and B , respectively), Pearson's r can be estimated as follows, where \bar{a} and \bar{b} are the average values of the two vectors:

$$r_{ab} = \frac{\sum_{l=1}^k (a_l - \bar{a})(b_l - \bar{b})}{\sqrt{\sum_{l=1}^k (a_l - \bar{a})^2} \sqrt{\sum_{l=1}^k (b_l - \bar{b})^2}}$$

Interpreting the numerator

Pearson's r computes the covariance of the two variables and normalizes it with their respective standard deviations [155, Section 6.3]. It is greater than zero for positively correlated random variables and less than zero for negatively correlated ones. In order to understand this property of Pearson's r , consider for now only a single term from the numerator, i.e., $(a_l - \bar{a})(b_l - \bar{b})$ for a fixed value of l . It is clear, that this term can only become positive, if both of its constituents have the same sign. This, however, can only happen, if a_l and b_l are either both greater than their corresponding mean, or both smaller than their corresponding mean. This means, that the term $(a_l - \bar{a})(b_l - \bar{b})$ can only be positive if both a_l and b_l lie on the same side of their respective mean. If they lie on different sides (i.e., one being greater and the other one being smaller than its respective mean), then the term $(a_l - \bar{a})(b_l - \bar{b})$ becomes negative. Overall, the sum over all l can only reach a large absolute value, if there is a consistent trend in the observed sample, i.e., if either most of the terms are greater than zero, or most of them are less than zero.

Properties of Pearson's r

The denominator of the overall formula can be seen as a normalization term, which is always nonnegative, and which ensures, that $r \in [-1, 1]$. One can easily see, that Pearson's r can only reach its maximum and minimum value for linear dependencies. It is important to note, that Pearson's r implicitly assumes, that the data is at least interval scaled, and that the two random variables A and B follow a normal distribution [155, Section 6.5]: If this assumption is not fulfilled, then computing their mean, variance, and covariance is not meaningful.

Spearman's ρ

Spearman's ρ [384] generalizes Pearson's r by allowing also for non-linear monotone relationships between the two variables. It can be computed by replacing each observation a_l and b_l with its corresponding rank, i.e., its index in a sorted list, and by then computing Pearson's r on these ranks. By replacing the actual values with their ranks, the actual numeric distances between the sample values lose their importance – what counts, is now only the correct *ordering* of the samples. Spearman's ρ can therefore also be applied to non-normally distributed data [155, Section 6.5]. Like Pearson's r , Spearman's ρ is confined to the interval $[-1, 1]$ with positive values indicating a monotonically increasing relationship.

Kendall's τ

Kendall's τ [219] also measures the monotonicity of the relationship between two random variables A and B . Instead of computing mean, variance, and covariance of the given sample, Kendall's τ is based on frequency counts. Let us again assume, that \vec{a} and \vec{b} are vectors

sampled from A and B . Each tuple (a_l, b_l) is treated as one observation. We now look at all possible pairs of observations $(a_{l_1}, b_{l_1}), (a_{l_2}, b_{l_2})$. A pair of observations is called *concordant*, if $a_{l_1} > a_{l_2} \wedge b_{l_1} > b_{l_2}$ or if $a_{l_1} < a_{l_2} \wedge b_{l_1} < b_{l_2}$. In other words, a pair of observations is concordant, if it does not violate the assumption of a monotonically increasing relation between the two variables. Pairs with $a_{l_1} = a_{l_2}$ or $b_{l_1} = b_{l_2}$ are ignored, and all other pairs of observations (which support a monotonically decreasing relation) are called *discordant*. Let N_c be the number of concordant pairs, N_d the number of discordant pairs, and $N = \frac{k \cdot (k-1)}{2}$ be the number of all possible pairs of k observations. Kendall's τ is then defined as follows:

$$\tau_{ab} = \frac{N_c - N_d}{N}$$

It is easy to see, that also Kendall's τ is confined to the interval $[-1, 1]$ with positive values corresponding to a monotonically increasing relationship. In contrast to Spearman's ρ , Kendall's τ explicitly considers the correct ordering of pairs and is thus more similar to the nonmetric Stress function used to obtain an MDS solution. Kendall's τ is especially recommended for small datasets with a large number of tied ranks [155, Section 6.5].

Properties of Kendall's τ

While all of the previous metrics directly measure the relation between two random variables, the *coefficient of determination* R^2 [439] quantifies the performance of a regressor with respect to a given ground truth (cf. Section 5.1.2). Here, we assume, that the vector \vec{y} contains the ground truth values, and that the function \hat{f} represents the predictions made by the regressor. Please recall from Section 5.1.2, that R^2 is then defined as follows [155, Section 7.2.3]:

Coefficient of determination R^2

$$R^2 = 1 - \frac{S_{residual}}{S_{total}} = \frac{S_{total} - S_{residual}}{S_{total}}$$

with $S_{residual} = \sum_{j=1}^N (y^{(j)} - \hat{f}(\vec{x}^{(j)}))^2$

and $S_{total} = \sum_{j=1}^N (y^{(j)} - \bar{y})^2$

Here, S_{total} is the *total sum of squares*, which measures the overall variance in the ground truth data by quantifying how much the target varies around its mean. $S_{residual}$ is called the *residual sum of squares* and measures, how far the regressor's predictions depart from the ground truth. Please also recall, that the coefficient of determination can be interpreted as the percentage of variance in the ground truth explained by the regressor [155, Section 7.2.3]. The highest possible value of R^2 is one and can only be achieved if $S_{residual}$ becomes zero. For a regression error larger than the variance of the ground truth, R^2 becomes negative. Since the regression error can become arbitrarily large, the possible values of R^2 are not bounded from below, hence $R^2 \in (-\infty, 1]$.

Interpreting R^2

Considering a monotone regression

In principle, the formulation of R^2 is agnostic about the type of regression being used. One can, however, show, that for a linear least-squares regression, R^2 is identical the squared value of Pearson's r . Since we already consider Pearson's r as evaluation metric, computing R^2 for a linear regression would not provide novel information. Instead, we compute R^2 for a monotone regression from distances to dissimilarities. Also the coefficient of determination can be related to nonmetric Stress, since it considers a monotone regression and the sum over the squared differences to the ground truth.

Expectations for similarity spaces

Overall, we thus consider one metric based on a linear relationship (namely, Pearson's r) and three metrics based on a monotone relationship (Spearman's ρ , Kendall's τ , and R^2 with a monotone regression). We assume, that for the MDS-based similarity spaces all of these metrics improve as the number of dimensions increases. A comparison between the three monotone correlation metrics can reveal, whether any of them is more informative than the others about differences between the similarity spaces.

Metric vs. nonmetric MDS

All MDS variants can be expected to find a configuration, such that there is a monotone relationship between the distances in the similarity space and the original dissimilarity matrix. That is, smaller dissimilarities should correspond to smaller distances and larger dissimilarities correspond to larger distances. We therefore do not expect notable differences for the monotone correlation coefficients. Metric MDS also assumes a *linear* relationship between dissimilarities and distances. Therefore, if the dissimilarities obtained by SpAM are ratio scaled, then metric MDS should give better results with respect to Pearson's r than nonmetric MDS.

Expectations for baselines

For our two baselines, we expect overall lower correlation values than for the MDS-based similarity spaces, since they do not have access to the original dissimilarity ratings. Due to its higher complexity, we furthermore expect the ANN baseline to outperform the pixel baseline. Since we have no strong reason to believe otherwise, we assume, that all of these effects can be observed for all of the correlation metrics.

Interpretability-based evaluation

Another possible way of judging the quality of an MDS solution is to look for interpretable directions and meaningful clusters in the resulting space (cf. Section 8.6.2). However, Horst and Hout [199] have argued, that since their NOUN dataset consists of novel stimuli, there are no obvious directions or clusters, which one would expect. Without a list of candidate directions and categories, an efficient and objective evaluation based on interpretable directions and meaningful clusters is, however, difficult to achieve. We therefore limit ourselves to a simple visual inspection of all two-dimensional spaces: If a visualization of a given similarity space shows meaningful structures and clusters, this indicates a high quality of the semantic space. We do not consider spaces with more dimensions, because they are much harder to visualize.

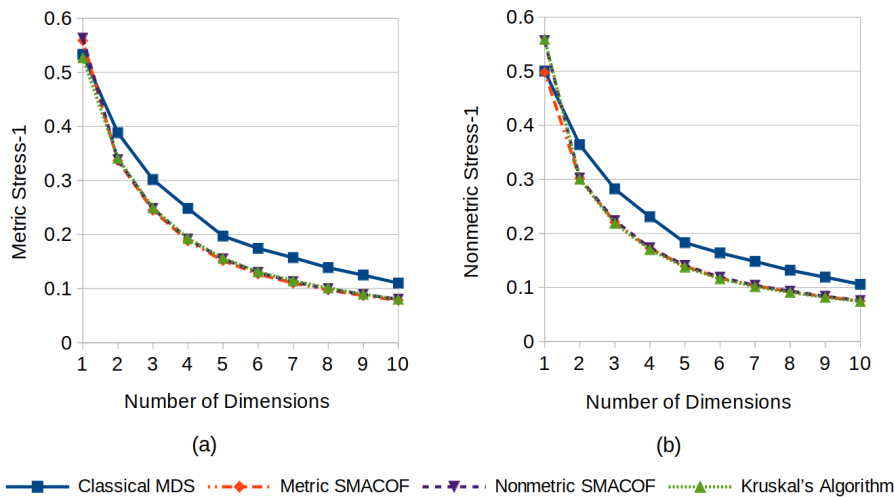


Figure 9.2: Scree plots showing the remaining metric (a) and nonmetric (b) Stress, respectively, as a function of the number of dimensions for the four MDS algorithms under consideration.

9.1.3 Results

Figure 9.2 shows the Scree plots (cf. Section 8.6.1) of the four MDS algorithms for both metric and nonmetric Stress. As one would expect, Stress decreases for all four MDS variants with an increasing number of dimensions: More dimensions help to represent the dissimilarity ratings more accurately.

In both cases, classical MDS consistently yields higher Stress values than the other MDS algorithms. We can explain this observation by the fact, that (as discussed in Section 8.3.2) classical MDS is the only algorithm in our study, which does not explicitly optimize Stress (i.e., the difference between the distances and suitably transformed dissimilarities). It rather optimizes the related, but different notion of Stain (i.e., the difference between the inner product matrix of the configuration and a suitable transformation of the dissimilarity matrix). Due to its different optimization target, classical MDS can therefore not be expected to find a solution minimizing Stress. All other MDS algorithms yield almost identical performance with respect to both metric and nonmetric Stress. This suggests, that interpreting the SpAM dissimilarity ratings as ratio scaled is neither helpful nor harmful.

In order to identify the optimal number of dimensions for a conceptual similarity space, one usually tries to identify an "elbow" in the Scree plot (cf. Section 8.6.1). In both graphs, it seems, that two or three dimensions might be a candidate for such an elbow, but the effect is not very pronounced. Moreover, we can make the observation, that the Stress curves start to saturate after five or six dimensions. This is in line with the results by Horst and Hout [199], who selected a four-dimensional space as a compact but faithful representation of the dissimilarity ratings.

Scree plots

Comparing the algorithms

The optimal number of dimensions

Baseline	Weights	Pearson's r	Spearman's ρ	Kendall's τ	R^2 (Monotone)
Pixel	uniform	0.2402	0.2472	0.1675	0.0939
Pixel	optimized	0.3920	0.3820	0.2608	0.2018
ANN	uniform	0.2580	0.2479	0.1660	0.0955
ANN	optimized	0.5417	0.5145	0.3594	0.3562

Table 9.2: Performance comparison of pixel baseline and ANN baseline for uniform and optimized weights.

Analyzing correlations

Let us now turn to the results of our correlation analysis. In addition to the four MDS algorithms, we also considered the pixel baseline and the ANN baseline, as well as the four-dimensional space as provided by Horst and Hout [199]. Moreover, we generated 100 random configurations of points for spaces from one to ten dimensions and computed their average correlation coefficients. As one would expect, the expected correlation for randomly generated configurations is zero and does not depend on the number of dimensions. Any correlation value considerably larger can thus be interpreted as a (partial) success with respect to predicting human dissimilarity ratings.

Performance of the baselines

Let us first focus on our two baselines. As we can see in Table 9.2, the pixel baseline and the ANN baseline yield almost identical performance if used with uniform weights. For both baselines, the Manhattan distance gave the highest correlation values. In both cases, we see a considerable improvement, if optimized weights are used. This improvement is much more pronounced for the ANN baseline than for the pixel baseline – it thus seems, that raw pixel information is less useful in our scenario than the more high-level features extracted by the ANN. In the case of optimized weights, the Euclidean distance yielded the best results for both baselines. With respect to the ANN baseline, this observation is somewhat surprising, because it conflicts with the usage of the inner product in the study by Peterson et al. [318, 319]. It is furthermore interesting to observe, that for each of the baselines, the same configuration (i.e., distance function, and in case of the pixel baseline also block size and aggregator) was optimal for all evaluation metrics. For the pixel baseline, we furthermore observed, that the minimum aggregator yielded the best results.

The pixel baseline with uniform weights

Figure 9.3 illustrates Pearson's r for the pixel baseline (using the Manhattan metric and uniform weights) with respect to different aggregators and different block sizes. We can make the following observations: The maximum does not seem to be a very useful aggregator, because its performance is considerably worse than that of the other options. Overall, the minimum seems to be the best choice, having a slight advantage over the mean and the median. Only for large block sizes (corresponding to small resulting images), we can observe, that the minimum performs slightly worse than the mean and the median. The performance of all aggregators drops for very large block sizes (i.e.,

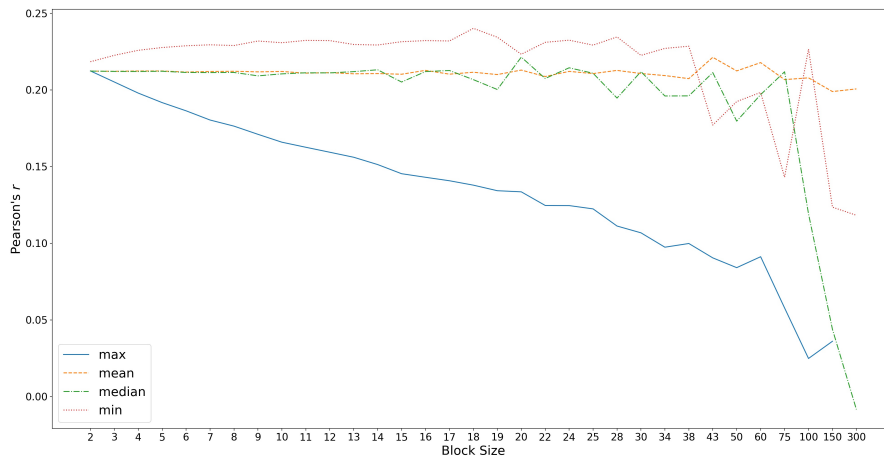


Figure 9.3: Pearson's r for pixel-based distances obtained through the Manhattan metric (using uniform weights) with respect to different aggregators and different block sizes.

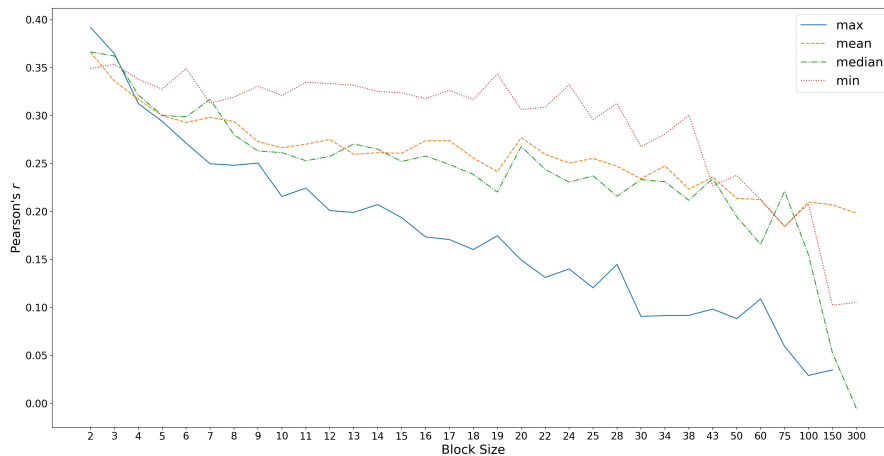


Figure 9.4: Pearson's r for pixel-based distances obtained through the Euclidean metric (using optimized weights) with respect to different aggregators and different block sizes.

very small image sizes), indicating that crucial information is lost after a certain point. Only the mean aggregator seems to be able to compensate this effect quite well. Other than that, we do not observe a clear tendency, which would imply a certain optimal image size, because the curves for minimum, mean, and median are almost constant for most of the block sizes. Plots for all other evaluation metrics and other distance functions (using uniform weights) can be found in Appendix D.1 and do in general conform with the observations described above.

Figure 9.4 shows a similar plot, but now using optimized weights and the Euclidean distance. Again, we observe that the maximum yields relatively poor performance. In general, best performance is reached for all aggregators when very small block sizes are used. This corresponds to large image sizes and hence very high-dimensional feature vectors. Since this implies, that a very large number of weights

*The pixel baseline
with optimized
weights*

needs to be optimized, the performance advantage of small block sizes may thus be explained by the larger degrees of freedom involved when optimizing weights. However, due to the five-fold cross-validation used for optimizing the weights (cf. Section 9.1.2), we know, that the numbers shown in Figure 9.4 correspond to the generalization performance and not to the (overly optimistic) training performance (cf. Section 5.1.2). Thus, a higher model capacity (i.e., a larger number of weights to optimize) does indeed yield a more accurate model, which is not subject to extreme overfitting. Again, the minimum seems to be the best choice overall, yielding relatively constant performance for a large variety of block sizes. Appendix D.2 contains analogous plots for other evaluation metrics and other distance functions, which follow the same pattern.

The role of the image size

Overall, we observed a performance drop for very small images, but our prediction, that smaller images more accurately reflect dissimilarities than high-resolution images did not match our practical observations. The information about the overall rough shape of the object can be better represented in low-resolution images than in high-resolution ones (in the sense, that distracting noise is being removed), whereas details are only available in high-resolution images. Since the resolution did not play a major role, neither of these aspects seems to be the main driver for the dissimilarity ratings. Probably both aspects play a role, and no image size can capture them equally well.

Interpreting the aggregators

The minimum has been identified as the best aggregation function for the pixel baseline. Since the images are encoded using the RGB color space, the white background of the images has values of 255 on all three color channels. The object itself, on the other hand, is generally not white, and has thus lower values on at least one of the color channels. Using the minimum aggregator (which is applied to each color channel individually) thus corresponds to detecting, whether the respective block contains at least one pixel of the object. The maximum aggregator, on the other hand, would correspond to a focus on the background, while both the mean and the median provide some continuous notion of the degree, to which a given block is filled. Since the minimum yields the highest correlation values, it seems, that the aforementioned focus on object detection ("is there at least one pixel of the object in this block?") seems to be more useful in predicting dissimilarities.

Performance of the ANN baseline

The overall performance level of the ANN baseline is slightly below the observations by Peterson et al. [318, 319]: They reported, that R^2 (based on a linear regression) improved from a range of [0.19, 0.58] (uniform weights) to a range of [0.35, 0.74] (optimized weights), depending on the dataset. In our case, we observed values of Pearson's $r \approx 0.26$ for uniform and $r \approx 0.54$ for optimized weights, which correspond to $R^2 = r^2 \approx 0.07$ for uniform and $R^2 \approx 0.29$ for optimized weights. One should, however, note, that the values reported by Peterson et al. differ greatly based on the dataset being used. Our dataset consists of novel objects, while the ANN was pretrained on images of real-life objects. Also the datasets of Peterson et al. consisted of photographs of real-life objects. Our dataset might thus pose a more difficult generalization

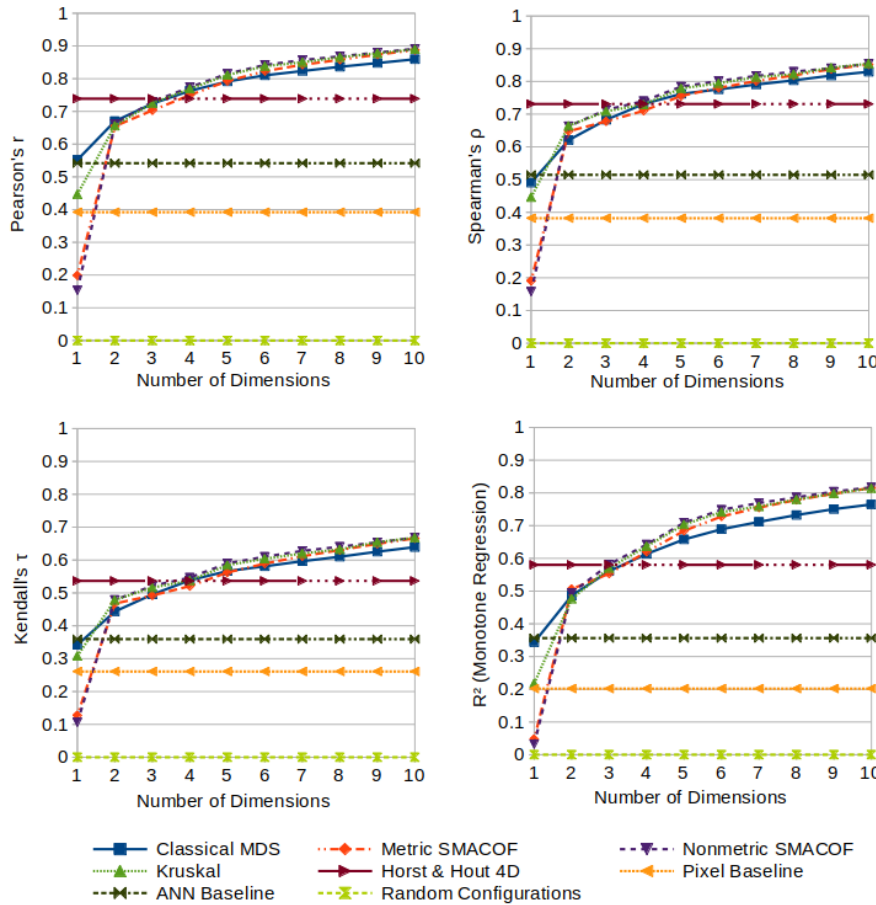


Figure 9.5: Correlation analysis for the different MDS solutions and the baselines, visualized as function of the number of dimensions.

problem for the ANN than the natural categories used by Peterson et al., which might cause the observed performance difference. Moreover, it seems, that Peterson et al. computed their numbers based on the complete distance matrix, whereas we only look at the part above the diagonal. Since the diagonal entries are always guaranteed to be zero for both the dissimilarity matrix and the distance matrix, this naturally increases the correlation between the two matrices. This difference might also contribute to the lower numbers observed on our dataset.

Let us now turn to the similarity spaces obtained by MDS. For almost all similarity spaces, the Euclidean distance yielded higher correlations than both the Manhattan distance and the negated inner product. This is to be expected, because MDS constructs these spaces in such a way, that the unweighted Euclidean distances reflect the dissimilarities as closely as possible. The only exception to this observation were the spaces created by classical MDS, where the negated inner product yielded considerably higher correlations to the dissimilarities. This exception may be explained by classical MDS trying to optimize Stain (i.e., the inner product matrix of the configuration) instead of Stress (i.e., the distances). Using optimized weights did in general not improve

Optimal distances for MDS spaces

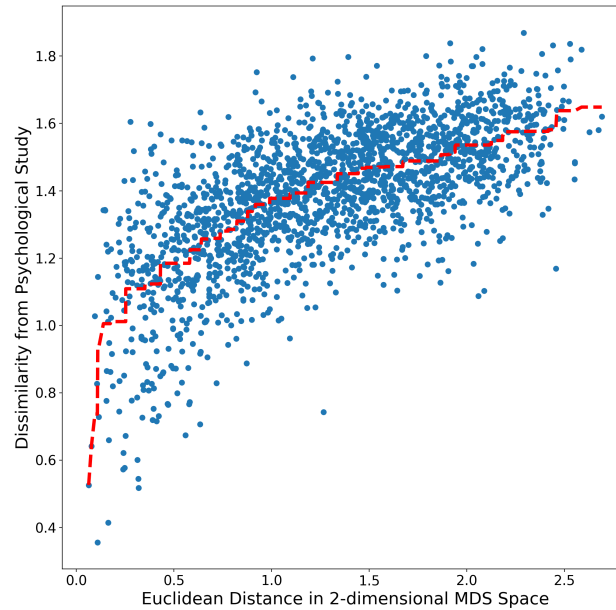


Figure 9.6: Shepard diagram for the two-dimensional MDS space, based on nonmetric SMACOF and the Euclidean distance.

results for the MDS spaces. In most cases, performance was equivalent or slightly worse than for uniform weights. The latter case is probably an artifact of the cross-validation procedure, where the weights are optimized on 80% of the entries in the distance matrix and evaluated on the remaining 20%. Only for the coefficient of determination R^2 , we were able to observe a slight, but consistent improvement based on the weight optimization. Since this improvement was, however, in general quite small (usually around 0.02 absolute), we conclude, that weight optimization is not worthwhile on the MDS-based spaces. Again, the fact that MDS chooses configurations of points, such that their unweighted Euclidean distances reflect the dissimilarities as closely as possible, explains this observation. Therefore, we only consider uniform weights for the MDS-based spaces in the following analysis.

Visualizing correlations

Figure 9.5 illustrates the correlation coefficients for the solutions provided by the different MDS variants as a function of the number of dimensions. For classical MDS, we use the correlations computed based on the negated inner product, while for all other MDS algorithms, the Euclidean distance was used. Figure 9.5 also shows the expected correlation of random configurations of the same dimensionality as the MDS solutions.¹⁰ Moreover, it shows the best results of the two baselines as well as the space provided by Horst and Hout [199]. All of these three baselines are shown as horizontal lines, since the dimensionality of their underlying space is fixed.

MDS spaces and baselines

We can observe in Figure 9.5, that the MDS solutions provide us with a considerably better reflection of the dissimilarity ratings than both pixel-

¹⁰ The numbers shown in Figure 9.5 are based on normally distributed points using the unweighted Euclidean distance. Using a uniform distribution or a different distance metric resulted in almost indistinguishable results.

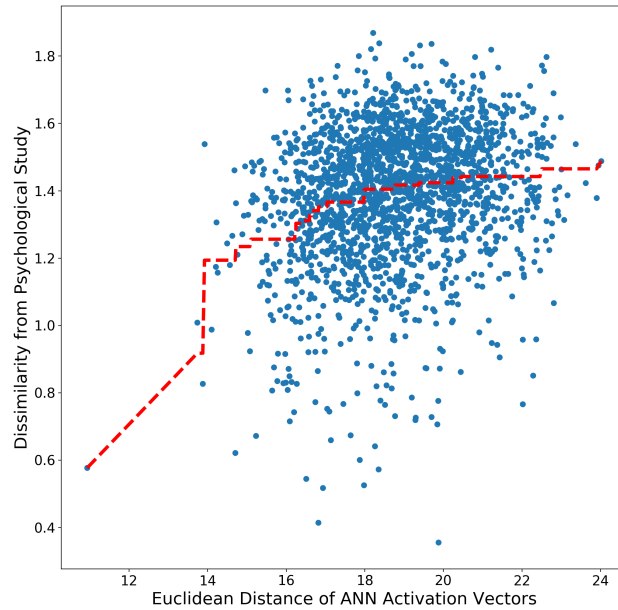


Figure 9.7: Shepard diagram for the ANN baseline, based on the Euclidean distance and optimized weights).

based and ANN-based distances, if we use at least a two-dimensional similarity space. Figures 9.6 and 9.7 show two Shepard diagrams (cf. Section 8.6.1) based on the distances of the two-dimensional similarity space found by nonmetric SMACOF using uniform weights, and based on the distances used by the ANN baseline with optimized weights, respectively. In both Shepard diagrams, we can observe a monotone trend (illustrated by the dashed red regression line), which is, however, much clearer in the MDS space than for the ANN baseline. The visualizations from Figures 9.6 and 9.7 thus support and confirm the observations from Figure 9.5, where the two-dimensional MDS spaces are judged to be superior to the ANN baseline with respect to all correlation metrics.

When comparing the different MDS variants, Figure 9.5 confirms our overall observations from the Scree plots in Figure 9.2: Classical MDS performs slightly worse than all other MDS variants, which yield very similar performance on all the metrics. This again supports the view, that the assumption of ratio scaled dissimilarity ratings is neither beneficial nor harmful for our dataset. Again, classical MDS seems to be inferior to iterative Stress-minimization methods. We think, that the poorer performance of classical MDS is linked to the observation, that although Euclidean distances are targeted when constructing the space, the negated inner product allows us to make better predictions of the original dissimilarities.

For one-dimensional spaces, it seems that both Kruskal's algorithm and classical MDS are considerably better than both SMACOF variants. This gap, however, disappears as soon as two dimensions are used, indicating that SMACOF is somewhat incapable of extracting one-dimensional similarity spaces. The performance of the four-dimensional space by Horst and Hout [199] coincides with the performance of the

Comparing MDS algorithms

Considering the number of dimensions

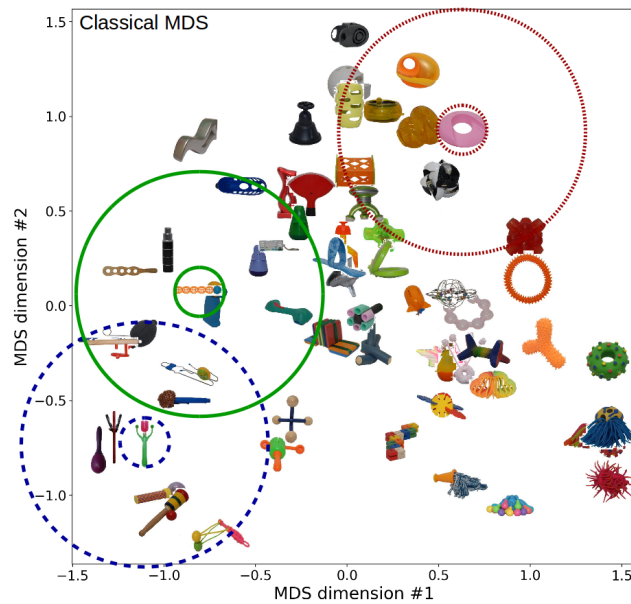


Figure 9.8: Two-dimensional similarity space based on classical MDS.

four-dimensional spaces as constructed by the algorithms used in our study, showing again, that the choice of the MDS algorithm seems to be largely irrelevant. Overall, we find the tendency of improved performance with an increasing number of dimensions. This again illustrates, that MDS is able to fit more information into the space, if this space has a larger dimensionality. Just as with the Scree plots, performance seems to level off after five or six dimensions.

Comparing correlation coefficients

When comparing the different correlation coefficients, we can note, that the absolute numbers seem to be somewhat lower for Kendall's τ and the coefficient of determination R^2 than for Pearson's r and Spearman's ρ . Other than that, we observe the same qualitative effects on all four plots. When comparing Person's r to the three other correlation coefficients, this supports again the view, that metric MDS does not have any advantage on data collected with SpAM. When focusing on the three rank-based correlation coefficients, we can interpret this observation as none of them being in principle preferable over the others.

Visual inspection

Finally, let us look at the two-dimensional spaces generated by the different MDS algorithms in order to get an intuitive feeling for their semantic structure. Figures 9.8 to 9.11 visualize these spaces along with the local neighborhood of three selected items. These neighborhoods illustrate, whether stimuli are grouped in a meaningful way. The item in the red dotted circle is a roundish object with a hole in the middle and little structure otherwise. One can see, that in most spaces, it is being grouped with other roundish objects with holes. The item in the blue dashed circle is a vertically elongated object with a "head" on one end. Again, it is grouped with other objects, which fit this description in most of the spaces. Finally, the item in the green solid circle is a horizontally elongated object, which is again mostly grouped with

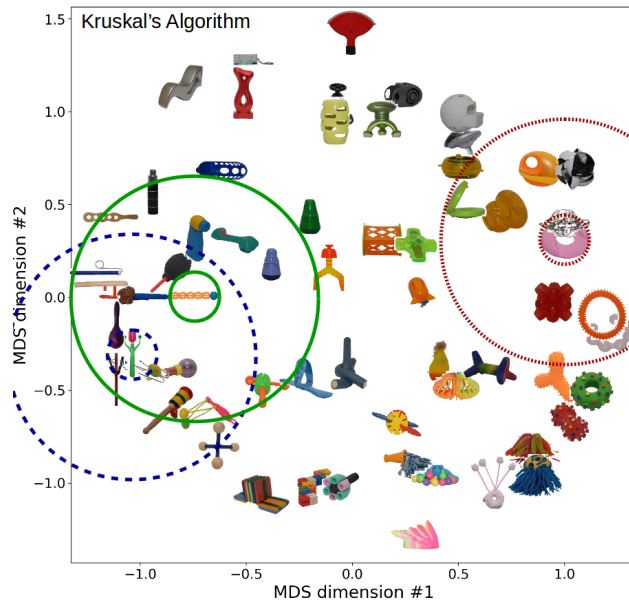


Figure 9.9: Two-dimensional similarity space based on Kruskal's algorithm.

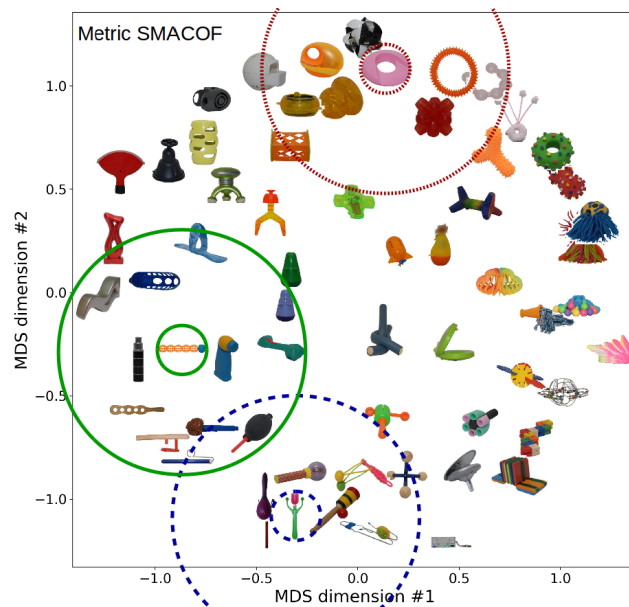


Figure 9.10: Two-dimensional similarity space based on metric SMACOF.

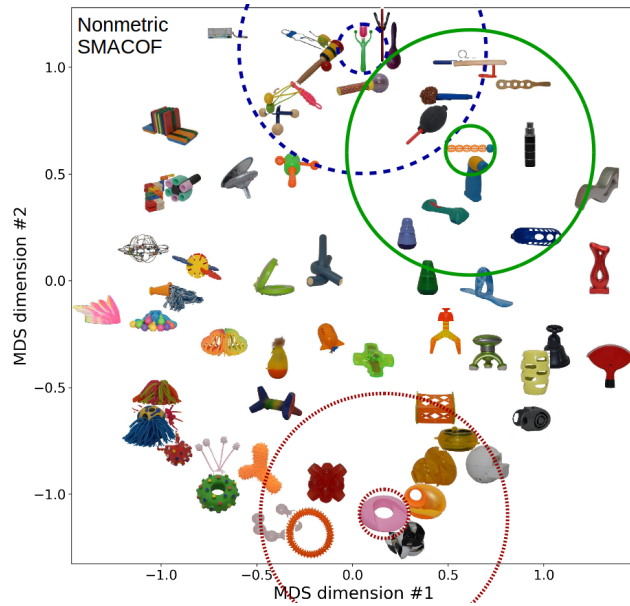


Figure 9.11: Two-dimensional similarity space based on nonmetric SMACOF.

other elongated objects. In almost all of the spaces, the blue dashed neighborhood and the green solid neighborhood are relatively close, making up one larger neighborhood of elongated objects. From our visual inspection, it thus seems, that all MDS variants create reasonable semantic similarity spaces.

9.1.4 Discussion

*Metric vs. nonmetric
MDS*

Based on the results of our analysis, we can make the following conclusions: Overall, we do not find any systematic difference between metric and nonmetric MDS on the given dataset. It thus seems, that assuming a ratio scale is neither beneficial nor harmful when trying to extract a similarity space from dissimilarity ratings collected with SpAM. On the one hand, we cannot exclude, that the dissimilarities obtained through SpAM are ratio scaled. On the other hand, the additional information conveyed by differences and ratios of dissimilarities does not seem to improve the overall results. We therefore advocate to use nonmetric MDS due to the smaller amount of assumptions made about the dissimilarity ratings.

Baselines

The similarity spaces obtained by MDS are clearly superior to both the pixel baseline and the ANN baseline. This might indicate, that the dissimilarity ratings are not directly based on raw perceptual similarity (which is targeted by our baselines), but on more abstract and complex conceptual similarity. This interpretation receives further support from the observation, that the ANN baseline has a higher correlation to the dissimilarities than the pixel baseline, since the features extracted by the ANN can be viewed as more abstract and semantically meaningful than raw pixel values [235]. Eliciting dissimilarity judgments from humans therefore seems to be necessary even on a dataset of novel

objects, for which background knowledge is not expected to play a role. With respect to the pixel baseline, we found no clear effect of the block size used for downscaling the images. It thus seems, that the dissimilarity ratings focus neither on the overall *SHAPE* of the objects (which would correspond to large block sizes being optimal), nor on the details of their *TEXTURE* (which would correspond to small block sizes being clearly superior). We conclude, that probably both domains play a role for the dissimilarities of the NOUN dataset.

The three monotone correlation metrics (namely, Spearman's ρ , Kendall's τ , and the coefficient of determination R^2 based on a monotone regression) did not show fundamentally different behaviors in our investigations. It thus seems, that all three of them are equally well equipped for this kind of analysis. Kendall's τ and the coefficient of determination R^2 are closer in spirit to the formulation of Stress than Spearman's ρ . However, R^2 is not confined to the nicely interpretable interval $[-1, 1]$. We therefore recommend to use Kendall's τ when computing the correlation between distances and dissimilarities.

Correlation metrics

9.2 MACHINE LEARNING EXPERIMENTS

After having extracted and analyzed a variety of similarity spaces for the NOUN dataset, we now consider the task of mapping images onto points in these similarity spaces. Instead of training a neural network from scratch, we limit ourselves to simple off-the-shelf regressors (both linear and nonlinear models) on top of features extracted either by a pretrained image classification network or from raw pixels. With the experiments in this study, we address the following research questions:

Task and research questions

1. Can we learn a useful mapping from colored images into a low-dimensional psychological similarity space from a small dataset of novel images, for which no background knowledge is available? *Our prediction: The learned mapping is able to clearly beat simple baselines. However, it does not reach the level of generalization observed in the study of Sanders and Nosofsky [346] due to the smaller amount of data available.*
2. To what extent do the type of feature space, the type of regressor, and the amount of regularization influence the machine learning results? *Our prediction: ANN-based feature spaces are constructed through a nonlinear transformation of the input pixels, therefore more complex than simple pixel-based features, and hence more likely to give good results. This is especially to be expected, if the target similarity spaces are not exclusively based on the *SHAPE* domain. Nonlinear regressors are more powerful than linear models, but also more prone to overfitting, especially in the context of small datasets. We therefore have no clear expectation with respect to the type of regression model. Since overfitting can be expected to be an issue in our experiments, we also expect regularization to have a positive effect on the regression results.*

Feasibility

Machine learning setup

- MDS algorithms* 3. How does the MDS algorithm being used to construct the target similarity space influence the machine learning results?
Our prediction: There are no considerable differences between the different spaces. If any, we expect more "well-behaved" spaces (i.e., spaces with a more faithful reflection of the dissimilarity ratings) to lead to slightly superior performance.
- Dimensionality* 4. How does the dimensionality of the target similarity space influence the machine learning results?
Our prediction: Very small target spaces are not able to reflect the similarity ratings very well and do not contain much meaningful structure. Very large target spaces on the other hand increase the number of parameters in the regressor, which makes overfitting more likely. By this reasoning, medium-sized target spaces should provide a good trade-off between these tendencies, and should therefore also achieve the best regression performance.
- Experimental steps* Each of these overarching research questions will be addressed in one experiment on our dataset. In Section 9.2.1, we first describe our overall experimental setup, before analyzing our feature spaces and baselines in Section 9.2.2. In Section 9.2.3, we then conduct first regression experiments on a single target space, but with different feature spaces, regressors, and regularization strengths (research questions 1 and 2). The third research question is then considered in Section 9.2.4, where we compare results obtained with respect to different target spaces of the same dimensionality, but created with different MDS algorithms. In Section 9.2.5, we finally investigate the influence of the target space's dimensionality on the regression results (research question 4). The overall observations from these experiments are then further discussed in Section 9.2.6.
- Relation to Sanders and Nosofsky* One can interpret our study as an extension of the work by Sanders and Nosofsky [346, 347], which has been summarized in Section 8.7.3: They used a dataset of 360 images of rocks and an eight-dimensional target similarity space. In contrast to that, we confine ourselves to a much smaller dataset of 64 images, which furthermore represent novel and unfamiliar objects. Moreover, we investigate the effects of MDS algorithm, regularization, and dimensionality of the similarity space. These three external influences have not been investigated by Sanders and Nosofsky, but may have a considerable influence on the results.

9.2.1 General Methods

- Normalizing MDS spaces* In order to make the regression results in the different similarity spaces comparable, it is crucial to normalize them. Kruskal [234] has noted, that Stress is invariant under translation and uniform stretching and shrinking of the similarity space. He proposed to normalize MDS solutions by translating and stretching them in such a way, that the centroid of the configuration resides in the origin, and that the mean

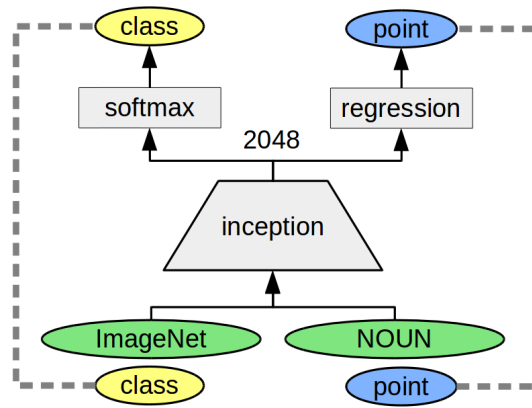


Figure 9.12: Illustration of the network structure used in our experiments.

squared distance of the points from the origin equals one. More formally, he proposes to ensure the following (where we use $y_i^{(j)}$ to denote the i th coordinate of the j th point in the similarity space):

$$\left(\forall i \in \{1, \dots, n\} : \frac{1}{N} \cdot \sum_{j=1}^N y_i^{(j)} = 0 \right) \wedge \left(\frac{1}{N} \sum_{j=1}^N \sum_{i=1}^n \left(y_i^{(j)} \right)^2 = 1 \right)$$

We have applied this procedure in order to normalize all similarity spaces under consideration. This was achieved by a subtraction of the centroid and a division through the mean squared distance of the original configuration.

Applying the normalization

Please recall from Section 9.1.1, that the NOUN data base contains only 64 images with an image size of 300 by 300 pixels. Since this number of training examples is too low for using machine learning techniques, we augmented the dataset (cf. Sections 6.2.3 and 7.2.1) by applying random crops, a Gaussian blur, additive Gaussian noise, affine transformations (i.e., rotations, shears, translations, and scaling), and by manipulating the image's contrast and brightness. These augmentation steps were executed in random order and with randomized parameter settings. We did not consider horizontal and vertical flips for our experiments, because they might influence the perception of an image quite strongly and thus carry the risk of introducing too much noise into the dataset. For each of the original 64 images, we created 1,000 augmented versions, resulting in a dataset of 64,000 images in total. We assigned the target coordinates of the original image to each of the 1,000 augmented versions.

Dataset augmentation

Given the small number of original images, training a regressor on raw pixel input carries a very large risk of overfitting. We therefore employed two types of lower-dimensional feature spaces: The pixels of downscaled images and activation vectors of a pretrained ANN.

Feature spaces

For the ANN-based features, we again used the pretrained inception-v3 network [400] (cf. Section 6.2.2), which has already been employed for

ANN-based features

the ANN baseline in Section 9.1.2. It has been trained on ImageNet [120], where it has achieved a state of the art top-5 error rate of 3.46% when classifying images into one of 1,000 classes. For each of the augmented images, we used the activations of the second-to-last layer as a 2048-dimensional feature vector. Our network architecture can be considered a special variant of the feedforward network proposed in Section 8.7.2: Instead of training both the mapping and the classification task simultaneously, we use an already pretrained network, keep its weights fixed, and augment it by an additional output layer for regression, which is trained separately. This overall structure is visualized in Figure 9.12 and corresponds to a transfer learning setup (cf. Section 6.2.3).

Pixel-based features

As a comparison to the ANN-based features, we used an approach similar to the pixel baseline from Section 9.1.2: We downscaled each of the augmented images by dividing it into equally sized blocks and by computing the minimum (which has shown the best correlation to the dissimilarity ratings in Section 9.1.3) across all values in each of these blocks as one entry of the feature vector. When choosing the block size, we only considered factors of the original image size in order to avoid introducing noise through zero-padding at the borders.

Different dimensionality

We created two pixel-based feature spaces of different dimensionality. More specifically, we used block sizes of 12 and 24, resulting in feature vectors of size 1875 and 507, respectively (based on three color channels for downsampled images of size 25 x 25 and 13 x 13, respectively). By comparing these two pixel-based feature spaces, we can analyze differences between low-dimensional and high-dimensional feature spaces. Since the high-dimensional feature space is in the same order of magnitude as the ANN-based feature space, we can also make a meaningful comparison between pixel-based features and ANN-based features. Moreover, the two block sizes achieved a comparable correlation to the dissimilarity ratings in Section 9.1.3, which makes a comparison between them meaningful.

Regressors

With this study, we aim to deliver a proof of concept for the overall approach. Achieving outstanding performance is therefore not a concern. While training a deep neural network from scratch might yield superior performance, it involves many degrees of freedom and hence requires a more thorough hyperparameter optimization. Hence, we limit ourselves to three simple off-the-shelf regressors from Python's `scikit-learn` library [316].¹¹

Linear regression

Please recall from Section 5.2.1, that a *linear least-squares regression* makes use of a simple linear model for each of the dimensions in the target space. A linear regression is one of the simplest regression approaches available and comes without any hyperparameters.

Lasso regression

Since the number of features is quite high in our context, even a linear regression needs to estimate a large number of weights: If we use a feature vector of size 2048 and a target space with four dimensions, a linear regression needs to estimate more than 8,000 weights. In

¹¹ See <https://scikit-learn.org/>.

order to prevent overfitting, we also consider a *lasso regression*, which learns a linear model, but additionally incorporates the L_1 norm of the weight matrix as regularization term (cf. Section 5.2.1). In our experiments, we investigated the following values for the regularization strength β (where $\beta = 0$ corresponds to an ordinary linear least squares regression):

$$\beta \in \{0, 0.001, 0.002, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10\}$$

Before training both the linear and the lasso regression, we normalized the underlying feature space by subtracting the mean and dividing by the standard deviation for each feature. This ensures, that the regression weights (and thus also the regularization term) are in the same order of magnitude for different feature spaces.

Normalizing feature spaces

In order to incorporate also nonlinear regression techniques, we furthermore considered a *random forest regression* (cf. Section 5.2.4) with 100 trees. We used `scikit-learn`'s default hyperparameters and did not perform any hyperparameter optimization. A linear regression can easily be integrated into an existing neural network by adding linear output units and using the mean squared error as the network's loss function. A random forest regressor on the other hand follows a different internal logic and cannot be easily formulated in terms of artificial neurons. However, in contrast to a linear regression, a random forest regression can also approximate nonlinear functions. By also evaluating the performance of a random forest regression, we can understand, whether the overall system would benefit from a nonlinear mapping from features to target MDS coordinates. If this is the case, such a nonlinear mapping could be implemented by a simple feedforward network (cf. Section 6.2.1) in the final system. For our feasibility study, a random forest regression is, however, much faster and easier to work with than a feedforward neural network.

Random forest regression

In order to define a lower bound for regression performance, we considered the following four baselines: The *zero baseline* always predicts the origin of the coordinate system. The *mean baseline*, on the other hand, computes the mean of all target points from the training set and always uses this point as prediction. The *normal distribution baseline* estimates the mean and covariance matrix of the target points from the training set. It draws a random sample from the corresponding multivariate normal distribution when making predictions. Finally, the *random draw baseline* uses a randomly selected target point from the training set as prediction.

Baselines

As part of our experiments, we would also like to investigate, whether learning a mapping into a psychological similarity space is easier than learning a mapping into an arbitrary space of the same dimensionality. In addition to the real regression targets (which are the coordinates from the similarity space obtained by MDS), we created another set of regression targets by randomly shuffling the assignment from images to target points. We ensured, that all augmented images created from the same original image were still mapped onto the same target point. With this shuffling procedure, we aimed to destroy any semantic

Shuffled targets

structure inherent in the target space. By using a shuffled mapping rather than a mapping to randomly chosen points in the space, we ensure, that the distribution of the target points remains constant. This is necessary in order to make a meaningful comparison between the two regression configurations. We expect that the regression works better for the original targets than for the shuffled targets.

Evaluation metrics

In order to evaluate our system, we used the following three different evaluation metrics:

- MSE*
- The *mean squared error (MSE)* as introduced in Section 5.1.2 is the loss function being minimized in a linear regression. It sums over the average squared difference between the prediction and the ground truth for each output dimension.

$$MSE = \sum_{i=1}^n \frac{1}{N} \sum_{j=1}^N \left(y_i^{(j)} - \hat{f}(\vec{x}^{(j)})_i \right)^2$$

- MED*
- The *mean euclidean distance (MED)* is the average of the Euclidean distance between the predicted point and the corresponding ground truth point. It provides us with a way of quantifying the average distance between the prediction and the target in the similarity space.

$$MED = \frac{1}{N} \cdot \sum_{j=1}^N \sqrt{\sum_{i=1}^n \left(y_i^{(j)} - \hat{f}(\vec{x}^{(j)})_i \right)^2}$$

- R²*
- The *coefficient of determination R²* as introduced in Section 5.1.2 can be interpreted as the amount of variance in the targets, which is explained by the regressor's predictions.

$$R^2 = 1 - \frac{S_{residual}}{S_{total}} = \frac{S_{total} - S_{residual}}{S_{total}}$$

$$\text{with } S_{residual} = \sum_{j=1}^N \left(y_i^{(j)} - \hat{f}(\vec{x}^{(j)})_i \right)^2$$

$$\text{and } S_{total} = \sum_{j=1}^N \left(y_i^{(j)} - \bar{y}_i \right)^2$$

Evaluation scheme

We evaluated all regressors using an eight-fold cross validation approach (cf. Section 5.1.2). Two augmented images, which were generated based on the same original image, might still be relatively similar to one another. If one of them is shown during training and the other one during testing, this might constitute a serious information leak, giving the system the opportunity to simply memorize the training examples and showing good generalization to the test set without having learned a general rule. We therefore made sure, that all augmented images

based on the same original image belonged to the same fold. In each iteration, one of these folds was used as test set, whereas all other folds were used as training set. We aggregated all predictions over these eight iterations (providing exactly one prediction per data point) and computed the evaluation metrics on this set of aggregated predictions.

9.2.2 Analyzing Feature Spaces and Baselines

Before conducting our regression experiments, we first analyzed the structure of the different feature spaces and the performance of four simple baselines. This preliminary analysis is used to obtain some first intuitions about the regression task and to form expectations for our subsequent experiments.

Motivation

In order to analyze, how structured the different feature spaces are, we investigated, whether the augmented images form strong clusters in the feature space. If all augmented images, which are based on the same original image, form a clear cluster in a given feature space, then learning a mapping from this feature space to the target similarity space might be easier than if there is no clear structure in the feature space.

Clusters in feature space

We conducted a simple cluster analysis by assigning all feature vectors to the original image they are based on. This gives us a total amount of 64 clusters. We then quantified the quality of this cluster assignment by using the *Silhouette coefficient* [339]. The Silhouette coefficient for a single data point j is defined as follows:

The Silhouette coefficient

$$s(j) = \frac{b(j) - a(j)}{\max\{a(j), b(j)\}}$$

Here, $a(j)$ is the average distance of j to all other data points lying in the same cluster C as j . The value $b(j)$ represents the average distance of j to the nearest cluster C' , to which j has not been assigned. Overall, the Silhouette coefficient of a given point therefore becomes large, if the distance to other points in the same cluster is small, while the distance to points from other clusters is large. By taking the average across all data points, the Silhouette coefficient can also be computed for a whole dataset. We can therefore use the Silhouette coefficient as an indication for the overall structure of the feature space.

Interpretation

Table 9.3 shows the Silhouette coefficient for the different feature spaces, both for the correct assignment of feature vectors to original images and for a shuffled assignment. In each of the feature spaces, we computed the Silhouette coefficient based on the Euclidean, Manhattan, and Cosine distance.

Analysis results

Let us first consider the ANN-based feature space. We can see from Table 9.3, that the Silhouette coefficient is largest when considering the Cosine distance, which corresponds to the inner product of the *normalized* feature vectors. This is in line with Peterson et al. [318, 319], who used the inner product for assessing the similarity of ANN activation vectors (cf. Section 8.1.5). However, this observation contradicts

ANN-based features

	Silhouette coefficient		
	Euclidean distance	Manhattan distance	Cosine distance
ANN (2048)	0.3699 (-0.0233)	0.3824 (-0.0229)	0.5966 (-0.0354)
Pixel (1875)	0.0054 (-0.0360)	0.0359 (-0.0505)	-0.0224 (-0.0664)
Pixel (507)	0.0203 (-0.0376)	0.0546 (-0.0510)	0.0062 (-0.0684)

Table 9.3: Silhouette coefficient for the different feature spaces. Values achieved on a shuffled assignment of images to feature vectors are shown in parentheses for comparison. Best values for each of the feature spaces are highlighted in boldface.

the results from Section 9.1.3, where we found, that the Euclidean and Manhattan distances were more useful in predicting the dissimilarities of images in the context of the ANN baseline.

Pixel-based features

For both pixel-based feature spaces, the Silhouette coefficient hovers around zero, indicating, that there are no well-separated clusters in these spaces. Especially when comparing the results to the ones obtained on a shuffled cluster assignment (where we do not expect any meaningful clustering), one can clearly see, that pixel-based feature spaces are not very structured: While the Silhouette coefficient is often higher on the correct cluster assignment, the difference to the random assignment is very small, especially when compared to the ANN-based features.

Interpreting the results

We can explain the observed difference between the ANN-based features and the pixel-based features as follows: The pixel-based features are based on a simple surface-level aggregation of pixel values (namely, taking the minimum value over a block of fixed size), whereas the ANN-based features involve a cascade of nonlinear transformations. In the data augmentation step, different versions of the same original image were created by applying various surface-level transformations (such as translations, rotations, and noise). The different variants of the same original image thus differ on the surface level. Since the pixel-based features also only operate on the surface level, these surface-level differences are not filtered out. The ANN-based features, on the other hand, operate on a "deeper" level: The cascade of nonlinear operations on the raw pixel information seems to be able to filter out a large part of the noise introduced in the augmentation step (e.g., by ignoring translations though the use of convolutional processing steps, cf. Section 6.2.2). Therefore, the resulting feature vectors differ not as much as the augmented images, leading to a better structure in the feature space.

Expectations

Based on its superior structure, we therefore expect all regression results (independent of the actual regressor used) to be better for the ANN-based feature space than for the pixel-based feature spaces.

Baseline	Training			Test		
	MSE	MED	R^2	MSE	MED	R^2
Zero	1.0000	0.9962	0.0000	1.0000	0.9962	0.0000
Mean	0.9974	0.9946	0.0026	1.0389	1.0149	-0.0389
Normal Distribution	1.9945	1.3432	-0.9947	2.0433	1.3635	-1.0439
Random Draw	1.9958	1.3469	-0.9960	2.0377	1.3735	-1.0377

Table 9.4: Performance of different baselines on the four-dimensional space by Horst and Hout [199]. Best values in each column are highlighted in boldface.

Let us now consider our four simple baselines, which operate independent of the feature space, and which therefore yield identical results for the correct and the shuffled label assignment. Table 9.4 shows the performance of the four baselines both on the training and the test set for the four-dimensional similarity space provided by Horst and Hout [199], which will be used in our first experiment. We can observe, that the zero baseline (i.e., always predicting the origin) yields the best results with respect to all evaluation metrics on the test set. The mean baseline performs slightly better on the training set, but slightly worse on the test set. Both nondeterministic baselines perform considerably worse than the mean and the zero baseline. Again, we observe better performance on the training set than on the test set.

The observed differences between training and test set performance can be explained by the fact, that the latter three baselines use information about the points from the training set for making their predictions. Since the test set differs from the training set, these predictions are not able to perfectly generalize. Consider, for instance, the difference between the zero baseline and the mean baseline: In each iteration of our cross-validation procedure, one fold based on eight original images is used as test set. Since the coordinates of these points are not part of the training set, and since the centroid of the overall dataset resides in the origin, the centroid of the training set lies outside of the origin. The mean baseline uses this centroid for making predictions, which is more successful on the training set than predicting the origin, since this centroid lies on average closer to the data point from the training set. Since the centroid of the overall dataset lies in the origin, the centroid of the test set, however, lies in the opposite direction from the origin than the centroid of the training set. Using the centroid of the training set as a prediction on the test set is thus in general worse than using the origin of the coordinate system. Similar considerations also apply to the other two baselines, and are again based on the normalization of the similarity space.

The performance difference between the normal distribution baseline and the random draw baseline on the one hand, and the mean and the

Baseline performance

Generalization behavior

Constant vs. sampled predictions

zero baseline on the other hand is based on their prediction behavior: While mean and zero baseline make the same constant prediction all the time, the other two baselines sample a point from a distribution (a multivariate normal distribution in the similarity space, and a uniform distribution over the points in the training set, respectively). This predicted point might be closer to the ground truth than a constant prediction, but it might also be further away. Based on the worse performance of these two baselines, it seems, that these predictions are in general worse than if one had simply predicted a constant point somewhere in the center of the target space.

Outlook For our further considerations, we only take the zero baseline into account, because it outperformed all other baselines. Since it completely disregards all input, its evaluation results are identical for training set, test set, correct targets, and shuffled targets.

MSE, MED, and normalization One might note, that even though the space has been normalized, the MED achieved by the zero baseline does not equal one. However, it is easy to see, that the normalization procedure targets the MSE rather than the MED of the zero baseline:

$$\underbrace{\sum_{i=1}^n \frac{1}{N} \sum_{j=1}^N \left(y_i^{(j)} - 0 \right)^2}_{\text{MSE of zero baseline}} = \underbrace{\frac{1}{N} \sum_{j=1}^N \sum_{i=1}^n \left(y_i^{(j)} \right)^2}_{\text{Normalized to 1}} \neq \underbrace{\frac{1}{N} \cdot \sum_{j=1}^N \sqrt{\sum_{i=1}^n \left(y_i^{(j)} \right)^2}}_{\text{MED of zero baseline}}$$

9.2.3 Comparing Feature Spaces and Regressors

Hypotheses With our first machine learning experiment, we would like to test the following fundamental hypotheses:

Feasibility 1. Our overall approach is able to clearly beat simple baselines. However, it does not reach the level of generalization observed in the study of Sanders and Nosofsky [346] due to the smaller amount of data available.

Feature spaces 2. A regression from the ANN-based features is more successful than a regression from the pixel-based features.

Correct vs. shuffled targets 3. Since the similarity spaces created by MDS encode semantic similarity by geometric distance, we expect, that learning the correct mapping generalizes better to the test set than learning a shuffled mapping.

Overfitting 4. Due to the small amount of training data available, a nonlinear regression (which has more free parameters) suffers more from overfitting than a simple linear regression. Introducing regularization can help to reduce overfitting tendencies.

Number of features 5. For smaller feature vectors, we expect less overfitting than for larger feature vectors. Therefore, less regularization should be needed to achieve optimal performance.

Feature Space	Regressor	Targets	Training			Test		
			MSE	MED	R^2	MED	MED	R^2
Any	Zero Baseline	Any	1.0000	0.9962	0.0000	1.0000	0.9962	0.0000
ANN (2048)	Linear	Correct	0.0142	0.1089	0.8956	0.6076	0.7498	0.3766
		Shuffled	0.0202	0.1296	0.9798	1.1440	1.0505	-0.1390
	Random Forest	Correct	0.0019	0.0175	0.9980	0.7946	0.8667	0.1944
		Shuffled	0.0027	0.0262	0.9973	1.1338	1.0543	-0.1267
Pixel (1875)	Linear	Correct	0.5029	0.6649	0.4941	1.3172	1.0845	-0.3251
		Shuffled	0.6037	0.7323	0.3990	1.6075	1.2099	-0.5955
	Random Forest	Correct	0.0059	0.0473	0.9941	0.8891	0.9089	0.1037
		Shuffled	0.0073	0.0553	0.9927	1.2061	1.0708	-0.1915
Pixel (507)	Linear	Correct	0.5168	0.6757	0.4806	1.2073	1.0428	-0.2120
		Shuffled	0.6329	0.7528	0.3698	1.5853	1.2072	-0.5727
	Random Forest	Correct	0.0047	0.0368	0.9953	0.9035	0.9153	0.0886
		Shuffled	0.0059	0.0435	0.9941	1.2538	1.0925	-0.2367

Table 9.5: Performance of different regressors for different feature spaces and correct vs. shuffled targets, with respect to the four-dimensional target space by Horst and Hout [199]. The best results for each combination of column and feature space are highlighted in boldface.

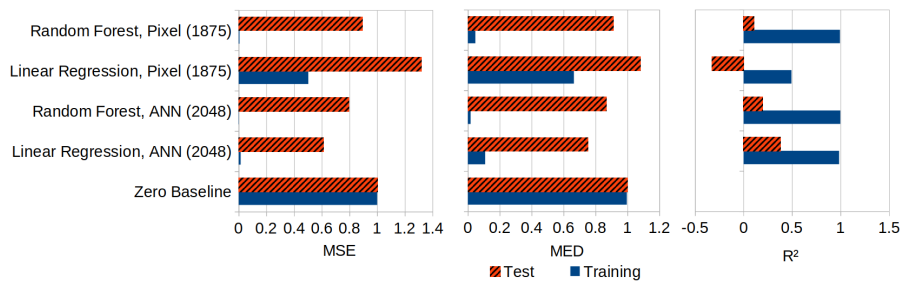


Figure 9.13: Illustration of selected regression results from Table 9.5, considering only correct targets.

In this experiment, we limit ourselves to a single target space, namely the four-dimensional MDS space by Horst and Hout [199] obtained through metric MDS.

Target space

Let us first compare the results based on the different feature spaces, considering only standard regression approaches, which do not make use of explicit regularization. Table 9.5 shows the results obtained in our experiments, grouped by the feature space, regression algorithm, and target mapping used. Figure 9.13 illustrates selected results with respect to the correct targets.

Unregularized regression

We first focus on the results obtained on the ANN-based feature space. As we can see, both the linear regression and the random forest are able to beat the zero baseline when trained on the correct targets. While the random forest is able to achieve the best performance on the training set, the linear regression performs better on the test set. Both regressors show strong overfitting tendencies, observable as very large

ANN-based features

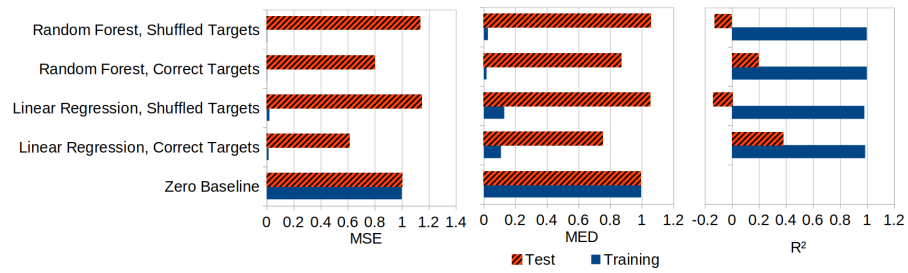


Figure 9.14: Illustration of the effect of correct vs. shuffled mappings on the ANN-based feature space from Table 9.5.

differences between training and test performance. When trained on the shuffled targets, both regressors obtain good results on the training set, but completely fail to generalize to the test set (cf. Figure 9.14).

Interpretation

Since the linear regression is easily able to beat the baseline on the correct labels, our overall approach seems to be sound – it is indeed possible to learn a mapping from images into the psychological similarity space. Moreover, we do not observe any benefits from using a more powerful random forest, indicating that a nonlinear mapping is not necessarily required in this case. Due to its additional degrees of freedom, the random forest is able to achieve better performance on the training set, but it also suffers more heavily from overfitting. The observation, that test set results based on the shuffled targets are not able to exceed baseline performance, shows, that the correct mapping (which has a semantic meaning) is easier to learn than an unstructured mapping. In other words, the semantic structure of the similarity space makes generalization possible.

Pixel-based features

Let us now consider the pixel-based feature spaces. For both of these spaces, we make similar observations: The linear regression performs worse than the baseline, while the random forest is able to outperform the baseline, and to achieve considerably better results on both the training and the test set than the linear regression. However, its performance is worse than on the ANN-based features. Moreover, we can again see, that learning the shuffled mapping results in poorer performance than learning the correct mapping. Overall, performance on the pixel-based feature spaces is lower than on the ANN-based feature space. Just as with the ANN-based feature space, we again observe strong overfitting tendencies. Finally, when comparing the two pixel-based feature spaces, we note, that the linear regression performs slightly better on the smaller feature space, while the random forest performs slightly better on the larger space. Overall, the differences between the two pixel-based feature spaces are, however, quite small.

Interpretation

The observation, that a linear regression is not sufficient to beat a very simple baseline on the pixel-based feature spaces, indicates, that these feature spaces are not as useful as the ANN-based one. Since a random forest is capable of achieving better performance than

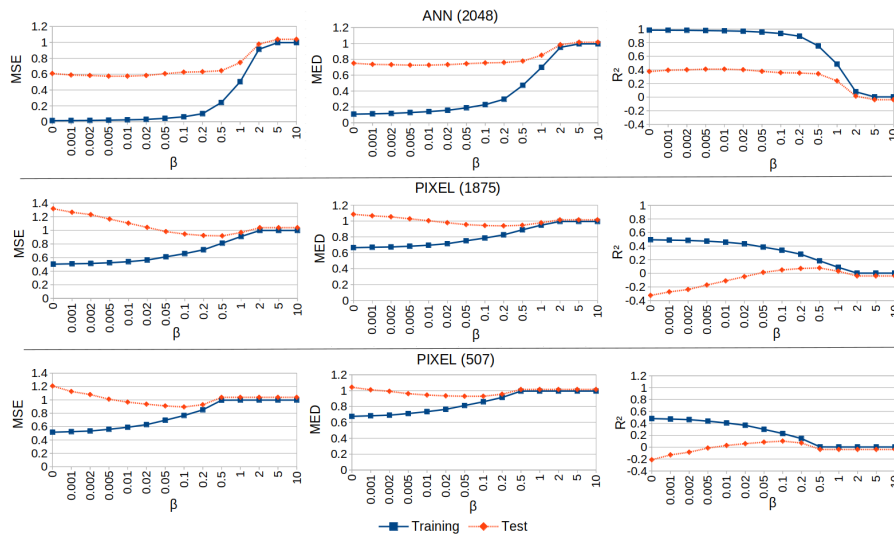


Figure 9.15: Visualization of the effects of regularization based on Table 9.6.

both the zero baseline and the linear regression, one might argue, that mapping the pixel-based feature vectors onto the target points in the psychological similarity space requires a nonlinear mapping. Overall, ANN-based features seem to be much more useful for our mapping task than the simple pixel-based features. This is in line with their higher correlation to the dissimilarity ratings observed in Section 9.1.3 and their higher Silhouette values reported in Section 9.2.2.

In order to further improve the results from Table 9.5, we also considered a lasso regressor with various values for the regularization factor β . The results can be seen in Table 9.6 and are visualized in Figure 9.15. Here, a value of $\beta = 0$ corresponds to a vanilla linear regression.

For the ANN-based feature space, we are able to achieve a slight, but consistent improvement by introducing a regularization term: Increasing β causes poorer performance on the training set, while yielding improvements on the test set. The best results on the test set are achieved for $\beta \in \{0.005, 0.01\}$. However, if β becomes too large, then performance on the test set starts to decrease again – for $\beta = 0.05$ we do not see any considerable improvements over the vanilla linear regression any more. For $\beta \geq 5$, the lasso regression collapses and becomes identical to the mean baseline.

Although we are able to improve our performance slightly, the gap between training set performance and test set performance still remains quite high. It seems, that the overfitting problem can be somewhat mitigated, but not solved on our dataset with the introduction of a simple regularization term.

When comparing our best results to the ones obtained by Sanders and Nosofsky [346, 347], who achieved values of $R^2 \approx 0.77$, we have to recognize, that our approach performs considerably worse with $R^2 \approx 0.41$. However, the much smaller number of data points in our experiment (please recall, that their dataset contained about six times

Adding regularization

ANN-based features

Overfitting issues

Comparison to Sanders and Nosofsky

Feature Space	β	Training			Test		
		MSE	MED	R^2	MSE	MED	R^2
Any	Zero Baseline	1.0000	0.9962	0.0000	1.0000	0.9962	0.0000
	Mean Baseline	0.9974	0.9946	0.0026	1.0389	1.0149	-0.0389
ANN (2048)	0	0.0142	0.1089	0.8956	0.6076	0.7498	0.3766
	0.001	0.0154	0.1129	0.9844	0.5892	0.7362	0.3958
	0.002	0.0168	0.1180	0.9829	0.5833	0.7322	0.4017
	0.005	0.0202	0.1292	0.9795	0.5740	0.7264	0.4107
	0.01	0.0243	0.1414	0.9753	0.5742	0.7264	0.4111
	0.02	0.0302	0.1575	0.9694	0.5829	0.7324	0.4026
	0.05	0.0431	0.1886	0.9562	0.6056	0.7444	0.3790
	0.1	0.0625	0.2287	0.9364	0.6249	0.7547	0.3590
	0.2	0.1020	0.2966	0.8960	0.6302	0.7592	0.3539
	0.5	0.2428	0.4725	0.7522	0.6432	0.7773	0.3424
	1	0.5047	0.6984	0.4848	0.7456	0.8512	0.2377
	2	0.9126	0.9509	0.0777	0.9779	0.9840	0.0144
≥ 5	0.9974	0.9946	0.0026	1.0389	1.0149	-0.0389	
Pixel (1875)	0	0.5029	0.6649	0.4941	1.3172	1.0845	-0.3251
	0.001	0.5082	0.6694	0.4888	1.2655	1.0657	-0.2730
	0.002	0.5127	0.6732	0.4842	1.2308	1.0528	-0.2380
	0.005	0.5237	0.6826	0.4732	1.1653	1.0274	-0.1719
	0.01	0.5390	0.6952	0.4578	1.1051	1.0034	-0.1112
	0.02	0.5632	0.7144	0.4337	1.0429	0.9784	-0.0483
	0.05	0.6106	0.7511	0.3864	0.9814	0.9548	0.0135
	0.1	0.6574	0.7857	0.3391	0.9449	0.9431	0.0497
	0.2	0.7143	0.8259	0.2808	0.9233	0.9391	0.0714
	0.5	0.8112	0.8894	0.1837	0.9183	0.9465	0.0788
	1	0.9095	0.9480	0.0882	0.9680	0.9783	0.0308
	≥ 2	0.9974	0.9946	0.0026	1.0389	1.0149	-0.0389
Pixel	0	0.5168	0.6757	0.4806	1.2073	1.0428	-0.2120
	0.001	0.5251	0.6832	0.4723	1.1258	1.0103	-0.1302
	0.002	0.5348	0.6912	0.4625	1.0801	0.9913	-0.0845
	0.005	0.5603	0.7114	0.4371	1.0103	0.9618	-0.0149
	0.01	0.5912	0.7352	0.4063	0.9670	0.9445	0.0282
	0.02	0.6300	0.7645	0.3670 (507)	0.9362	0.9342	0.0584
	0.05	0.6962	0.8125	0.2993	0.9099	0.9292	0.0843
	0.1	0.7661	0.8598	0.2287	0.8946	0.9293	0.1015
	0.2	0.8517	0.9144	0.1445	0.9283	0.9551	0.0701
	0.5	0.9959	0.9939	0.0043	1.0382	1.0146	-0.0381
	≥ 1	0.9974	0.9946	0.0026	1.0389	1.0149	-0.0389

Table 9.6: Performance of the lasso regressor for different values of β with respect to the four-dimensional target space by Horst and Hout [199]. The best results for each combination of column and feature space are highlighted in boldface.

as many stimuli) makes our learning problem much harder than theirs. Even though we use data augmentation, the small number of target data points might put a hard limit on the quality of the results obtainable in this setting. Moreover, Sanders and Nosofsky retrained the whole neural network in their experiments, whereas we limited ourselves to a linear regression on top of the features extracted by the pretrained network. Since we are able to clearly beat our baselines, we nevertheless interpret our results as supporting the general approach.

For the pixel-based feature spaces, we can also observe slight positive effects of regularization. For the large space, the best results on the test set are achieved for larger values of $\beta \in \{0.2, 0.5\}$. These results are better than the zero baseline, but still slightly worse than the random forest regressor (cf. Table 9.5). However, the gap between training and test set performance is smaller for the lasso regressor than for the random forest regressor. Again, strong regularization causes test set performance to decrease, leading to the mean baseline for $\beta \geq 2$. It is interesting to observe, that the mean baseline performs better with respect to all evaluation metrics than a vanilla linear regression ($\beta = 0$).

*Large pixel-based
feature space*

For the small pixel-based feature space, the optimal value of β lies in $\{0.05, 0.1\}$, leading to a test set performance slightly superior to the one obtained by the random forest on this feature space. In case of the small pixel-based feature space, the model already collapses for $\beta \geq 1$.

*Small pixel-based
feature space*

Comparing the regularization results on the three feature spaces, we can conclude, that regularization is indeed helpful, but only to a small degree. On the ANN-based feature space, we still observe a large amount of overfitting, and performance on the pixel-based feature spaces is still far worse than on the ANN-based features. Looking at the optimal values of β , it seems, that the lower-dimensional pixel-based feature space needs less regularization than its higher-dimensional counterpart. Presumably, this is caused by the smaller possibility for overfitting in the lower-dimensional feature space. Even though the larger pixel-based feature space and the ANN-based feature space have a similar dimensionality, the pixel-based feature space requires a larger degree of regularization for obtaining optimal performance, indicating that it is more prone to overfitting than the ANN-based feature space.

*Comparison across
feature spaces*

Overall, our first experiment has shown, that learning a mapping from images to a psychological feature space is possible on our dataset. We found, that for all regressors ANN-based features are much more useful than pixel-based features. Poor performance on the shuffled mapping in contrast to good performance on the correct mapping showed, that the psychological similarity spaces contain semantic structures, which help machine learning algorithms to generalize to unseen inputs. While pixel-based feature spaces profit from the usage of random forests, a linear regression seems to be sufficient for the ANN-based feature space. Additional regularization was helpful to some extent, but only

Summary

Target Space	MSE	MED	R^2
Horst and Hout [199]	1.0000	0.9962	0.0000
Classical MDS	1.0000	0.9636	0.0000
Kruskal's Algorithm	1.0000	0.9957	0.0000
Metric SMACOF	1.0000	0.9981	0.0000
Nonmetric SMACOF	1.0000	0.9956	0.0000

Table 9.7: Performance of the zero baseline on the four-dimensional target spaces derived with different MDS algorithms.

improved performance slightly – overfitting still remains a crucial issue in our machine learning setup.

9.2.4 Comparing MDS Algorithms

Motivation

After having analyzed the soundness of our overall approach in Section 9.2.3, we will now compare target spaces of the same dimensionality, but obtained with different MDS algorithms. More specifically, we look at all four-dimensional spaces produced by the four MDS algorithms investigated in Section 9.1.

The zero baseline

In a first step, we can compare the different target spaces by taking a look at the behavior of the zero baseline in each of them (see Table 9.7). As we can see, the values for MSE and R^2 are identical for all of the different spaces due to the normalization of the configurations and the usage of a constant prediction (cf. Section 9.2.2). Only for the MED we can observe some variations, which can be explained by the slightly different arrangements of points in the different similarity spaces. Consider for example a similarity space with only two stimuli A and B . This space is normalized under the following condition:

$$\frac{1}{2} \left(\underbrace{\sum_{i=1}^n (y_i^{(A)})^2}_{z_A} + \underbrace{\sum_{i=1}^n (y_i^{(B)})^2}_{z_B} \right) = 1$$

MED and the arrangement of points

If $z_A = z_B = 1$ (i.e., both points lie on the unit circle around the origin), then the space is obviously normalized, and $MED = \frac{1}{2} (\sqrt{z_A} + \sqrt{z_B}) = \frac{1}{2} (1 + 1) = 1$. Now let $z_A = 0.5$ and $z_B = 1.5$. The space is still normalized, but $MED = \frac{1}{2} (\sqrt{z_A} + \sqrt{z_B}) = \frac{1}{2} (\sqrt{0.5} + \sqrt{1.5}) \approx 0.9659$. Finally, if $z_A = 0.25$ and $z_B = 1.75$, then the space is still normalized, but $MED = \frac{1}{2} (\sqrt{z_A} + \sqrt{z_B}) = \frac{1}{2} (\sqrt{0.25} + \sqrt{1.75}) \approx 0.9114$. As one can see based on this small toy example, the MED of the zero baseline becomes smaller as the distribution of the points in the similarity space becomes less homogeneous.

Comparing the target spaces

The MED on the similarity space constructed by classical MDS is considerably smaller than for the other spaces, indicating a more uneven distribution of the points in the similarity space. Apart from this

Target Space	Test Set Performance			Degree of Overfitting		
	MSE	MED	R^2	MSE	MED	R^2
Horst and Hout [199]	0.6076	0.7498	0.3766	42.7093	6.8876	2.6171
Classical MDS	0.5537	0.7063	0.4029	41.8626	6.7370	2.4475
Kruskal's Algorithm	0.6097	0.7460	0.3903	42.3550	6.8155	2.5250
Metric SMACOF	0.6172	0.7560	0.3766	42.2885	6.8583	2.6162
Nonmetric SMACOF	0.6086	0.7461	0.3706	42.4380	6.8305	2.6585

Table 9.8: Test set performance of the linear regression on ANN-based features with respect to the different four-dimensional target similarity spaces, along with the relative degree of overfitting. Best results in each of the columns are highlighted in boldface.

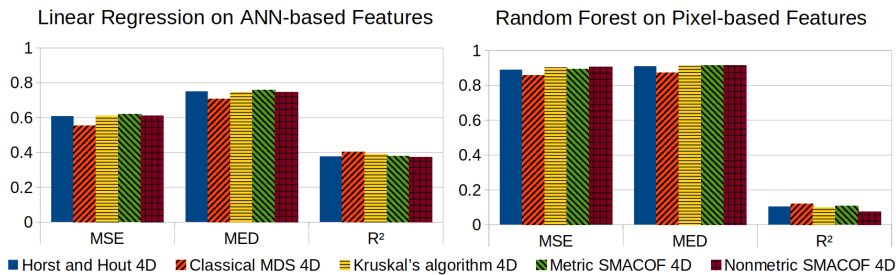


Figure 9.16: Visualization of the unregularized regression results for the different four-dimensional target spaces.

observation, we can, however, conclude, that based on the performance of the zero baseline, the different spaces are comparable.

Let us now turn to the performance of a linear regression and a random forest regression. We consider both ANN-based features and the large pixel-based feature space. For the sake of simplicity, the small pixel-based feature space was excluded from our considerations. On all of the target spaces, we observed the same general effects already described in Section 9.2.3 for our first experiment: Results on the correct targets were considerably better than results on the shuffled targets and the ANN-based feature space led to better performance than the pixel-based feature space. On the ANN-based feature space, a linear regression led to the best results, whereas on the pixel-based feature space, the random forest yielded superior performance. Overall, large degrees of overfitting could be witnessed. Detailed results can be found in Appendix D.3.

Table 9.8 summarizes the test set results for the linear regression on the ANN-based feature space for each of the different target spaces. Ta-

*Unregularized
regression*

Experimental results

Target Space	Test Set Performance			Degree of Overfitting		
	MSE	MED	R ²	MSE	MED	R ²
Horst and Hout [199]	0.8891	0.9089	0.1037	151.2546	19.2047	9.5849
Classical MDS	0.8556	0.8721	0.1174	146.7819	18.8177	8.4655
Kruskal's Algorithm	0.9002	0.9111	0.0972	149.6741	19.2624	10.2279
Metric SMACOF	0.8913	0.9118	0.1063	150.5864	19.2871	9.3561
Nonmetric SMACOF	0.9033	0.9118	0.0732	146.1260	19.1303	13.5712

Table 9.9: Test set performance of the random forest regression on the pixel-based features with respect to the different four-dimensional target similarity spaces, along with the relative degree of overfitting. Best results in each of the columns are highlighted in boldface.

ble 9.9 shows analogous results for the random forest on the pixel-based feature space. In both tables, we also report the degree of overfitting, which was computed by dividing the test set error by the training set error. A degree of overfitting of 10 for the MSE thus means, that the MSE on the test set was ten times higher than the MSE on the training set. Figure 9.16 illustrates these numbers with bar charts. As we can see, the results on the different target spaces are in general comparable. However, it seems, that the similarity space obtained by classical MDS leads to slightly better results than the other spaces.

Outperformance of classical MDS

Based on our observations, it thus seems, that the target space obtained by classical MDS is somewhat easier to work with in our regression task than the spaces produced by other MDS variants. For the mean Euclidean distance, this may not be surprising, since already the zero baseline achieves lower values on this space than on the others. For classical MDS, the MED is reduced by a linear regression on the ANN-based feature space to 91.09 % of the baseline performance. This is comparable to the other target spaces, where we observe reduction to levels between 91.02 % and 92.55 % of the baseline performance. However, for both mean squared error and R^2 , the baseline performance is identical across target spaces (cf. Table 9.7), but we are nevertheless able to observe better regression performance for the target space obtained by classical MDS.

Contradicting our expectations

These observations contradict our expectations: After all, classical MDS yielded the poorest correlations to the original similarity ratings in Section 9.1.3 (and preferred the inner product rather than the Euclidean distance), whereas Kruskal's algorithm and nonmetric SMACOF created similarity spaces with a better reflection of the similarity ratings.

However, in the machine learning results, classical MDS seems to produce an "easier" target space than any of the other MDS algorithms. So while we expected that similarity spaces, which reflect the original similarity ratings more faithfully, will lead to improved regression performance, we were not able to find such an effect in our experiment. Our conclusion from these observations is, that a better reflection of human similarity ratings does not necessarily make the machine learning problem any easier or harder. It is unclear whether the superior results on the space obtained by classical MDS can be explained by the different optimization target (Stain vs. Stress). Our observations therefore clearly urge for further investigations.

One could hypothesize, that the space obtained by classical MDS is easier to learn, *because* it is worse in representing the dissimilarities than the spaces obtained by the other MDS variants. It may thus be closer to the reflection of the dissimilarity ratings in the ANN-based feature space, which was found in Section 9.1.3 to be inferior to all MDS variants. In other words, the learning problem might be easier because the "gap" between the feature space and the target space, when it comes to representing dissimilarities, is smaller.

A possible explanation

From a more technical perspective, there are two possible reasons for the superior test set performance on the target space created by classical MDS: Either this target space makes generalization to unseen data points easier (which would be reflected in a reduced amount of overfitting), or it is simply easier to learn (which would be reflected in better performance also on the training set, when compared to other target spaces). In Tables 9.8 and 9.9, we have also reported the observed degree of overfitting by dividing training set error by test set error. Perfect generalization would thus result in an degree of overfitting of one, while larger values reflect the factor to which the regression is more successful on the training set than on the test set.

The technical perspective

As we can see from Tables 9.8 and 9.9, the four-dimensional space created by classical MDS does indeed seem to have a slight advantage with respect to overfitting when compared to the other target spaces, indicating that the target space provided by classical MDS might make generalization a bit easier. However, the difference to the other target spaces is quite small and not sufficient to explain the considerable performance difference. It thus seems that this target space is also in some sense "easier" to learn than the other target spaces.

Similar overfitting behavior

Since we only observed slight advantages for classical MDS, but none for metric SMACOF or the space by Horst and Hout [199], we cannot assign this observation to the difference between metric and nonmetric MDS. This adds further support to our results from Section 9.1: Also when considering the usage as target space for machine learning, metric MDS does not seem to have any clear advantage over nonmetric MDS.

Metric vs. nonmetric MDS

Let us finally take a look at the effects of regularization by considering a lasso regression on the ANN-based features. We observed similar effects for all of the target spaces: A certain amount of regularization is helpful to improve test set performance, while too much emphasis on

Adding regularization

Target Space	Linear Regression			Lasso Regression			
	MSE	MED	R^2	MSE	MED	R^2	β
Horst and Hout [199]	0.6076	0.7498	0.3766	0.5740	0.7264	0.4111	0.005, 0.01
Classical MDS	0.5537	0.7063	0.4029	0.5271	0.6855	0.4317	0.002
Kruskal's Algorithm	0.6097	0.7460	0.3903	0.6070	0.7420	0.3932	0.001
Metric SMACOF	0.6172	0.7560	0.3766	0.6052	0.7458	0.3880	0.002
Nonmetric SMACOF	0.6086	0.7461	0.3706	0.5937	0.7316	0.3853	0.005

Table 9.10: Test set performance of the lasso regression on the ANN-based features with respect to the different four-dimensional target spaces. Best results in each column are highlighted in boldface.

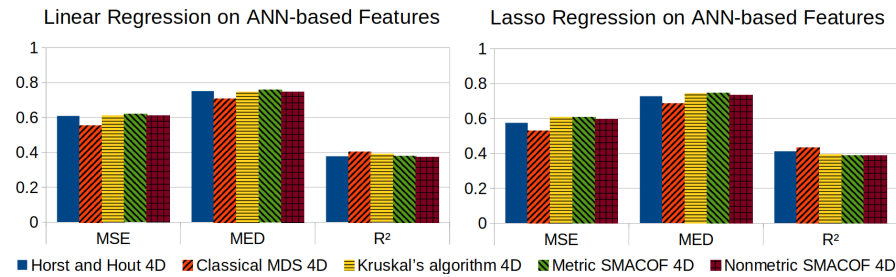


Figure 9.17: Visualization of the regularization effects on the ANN-based feature space for the different four-dimensional target spaces, based on Table 9.10.

the regularization term causes both training and test set performance to collapse. Again, we still observe a large amount of overfitting even after using regularization. Table 9.10 shows the optimal results of the lasso regression (along with the optimal values for β) in comparison to the results obtained by linear regression. Figure 9.17 illustrates these results. Tables with all results for all values of β can be found in Appendix D.4.

Results

We can observe, that again the target space obtained with classical MDS yields the best results with respect to all evaluation metrics. All other target spaces show slightly worse performance. The optimal values for the regularization parameter β are overall in a similar region. When looking at the relative improvement obtained by the regularization term, it seems that the space from Horst and Hout and the space obtained through classical MDS benefit the most from the additional regularization, while we can barely make improvements on the similarity space provided by Kruskal's algorithm.

Interpretation

Overall, the observations with respect to the lasso regressor thus support our conclusion, that there are no crucial differences between the different target spaces, with classical MDS producing for some reason slightly "easier" regression tasks.

9.2.5 Generalization to Other Target Spaces

With our third and final experiment in this study, we investigated, to what extent the number of dimensions in the target space influences the regression results. We compared similarity spaces with one to ten dimensions, which had been created by the same MDS algorithm, namely nonmetric SMACOF. We only considered the ANN-based feature spaces and compute the results for the zero baseline, the linear regression, and the lasso regression.

Overview

Larger target spaces lead to a larger number of regression weights, which need to be estimated, increasing the risk for overfitting. On the other hand, higher-dimensional similarity spaces reflect the similarity ratings more accurately than lower-dimensional spaces (cf. Section 9.1.3). Based on these two observations, we expect that performance suffers for both very low-dimensional spaces (because they are not able to encode the similarity structure very well) and very high-dimensional spaces (since the machine learning problem becomes harder). As a consequence, we expect to observe a "performance peak" for medium-sized target spaces. Moreover, given the larger risk for overfitting in higher-dimensional target spaces, we expect that they require larger amounts of regularization.

Expectations

Table 9.11 summarizes the results of our third experiment. Let us first consider the zero baseline on each of these spaces. We observe, that the MED tends to grow with an increasing number of dimensions, with an asymptote of one. This indicates, that in higher-dimensional spaces, the points seem to lie closer to the surface of a unit hypersphere around the origin (cf. our argumentation in Section 9.2.2). For both MSE and R^2 , we do not observe any differences between the target spaces. For the MSE, this is based on the normalization of the target spaces, whereas for R^2 , this is a result of predicting a constant point.

The zero baseline

Let us now consider the linear regression on the ANN-based feature space for each of the different target spaces. It seems, that for all the evaluation metrics, a two-dimensional target space yields the best results. With an increasing number of dimensions in the target space, performance tends to decrease. We can also observe, that the degree of overfitting (again computed by dividing the test set error by the training set error, cf. Section 9.2.4) is optimal for a two-dimensional space, and that it tends to increase with an increasing number of dimensions. A notable exception is the one-dimensional space, which suffers strongly from overfitting, and whose performance with respect to all three evaluation metrics is clearly worse than the zero baseline.

Unregularized regression

Let us now consider the optimal performance of a lasso regressor on the different target spaces when trained on the ANN-based features, along with the optimal choices for β to achieve these results. Tables with complete results for all target spaces and all values of β can be found in Appendix D.5. For all target spaces, we made again the observation, that the regularization can help to improve performance, but that too much regularization decreases performance. Again, we can only counteract a

Adding regularization

Regressor	n	Test Set Performance			Degree of Overfitting			β
		MSE	MED	R^2	MSE	MED	R^2	
Baseline	1	1.0000	0.8664	0.0000	1.0000	1.0000	1.0000	-
	2	1.0000	0.9580	0.0000	1.0000	1.0000	1.0000	-
	3	1.0000	0.9848	0.0000	1.0000	1.0000	1.0000	-
	4	1.0000	0.9956	0.0000	1.0000	1.0000	1.0000	-
	5	1.0000	0.9966	0.0000	1.0000	1.0000	1.0000	-
	6	1.0000	0.9973	0.0000	1.0000	1.0000	1.0000	-
	7	1.0000	0.9978	0.0000	1.0000	1.0000	1.0000	-
	8	1.0000	0.9980	0.0000	1.0000	1.0000	1.0000	-
	9	1.0000	0.9982	0.0000	1.0000	1.0000	1.0000	-
	10	1.0000	0.9984	0.0000	1.0000	1.0000	1.0000	-
Linear	1	1.1499	0.9046	-0.1499	59.0040	8.3419	-6.5413	-
	2	0.4995	0.6370	0.5002	38.9046	6.5291	1.9734	-
	3	0.5554	0.6979	0.4435	41.4309	6.7360	2.2243	-
	4	0.6086	0.7461	0.3706	42.4380	6.8305	2.6585	-
	5	0.6333	0.7692	0.3595	43.4577	6.9023	2.7405	-
	6	0.6359	0.7734	0.3469	43.4900	6.8770	2.8397	-
	7	0.6675	0.7956	0.3204	44.7364	6.9621	3.0741	-
	8	0.6846	0.8094	0.3033	45.1247	6.9876	3.2459	-
	9	0.6810	0.8078	0.2983	44.8367	6.9591	3.3004	-
	10	0.7107	0.8259	0.2807	46.0530	7.0432	3.5076	-
Lasso	1	0.9912	0.8368	0.0088	1.3656	1.7043	30.9878	1, 2
	2	0.4728	0.6052	0.5271	19.1298	4.5081	1.8504	0.02
	3	0.5322	0.6720	0.4722	19.4148	4.5725	2.0593	0.02
	4	0.5938	0.7316	0.3853	29.1237	5.6497	2.5413	0.005
	5	0.6180	0.7576	0.3755	35.1383	6.2167	2.6160	0.002
	6	0.6274	0.7651	0.3548	35.0732	6.1797	2.7724	0.001, 0.002
	7	0.6589	0.7839	0.3280	39.6352	5.0619	2.9979	0.001, 0.01
	8	0.6752	0.8022	0.3117	39.5496	6.5669	3.1527	0.001
	9	0.6680	0.7980	0.3108	38.8777	6.1359	3.1615	0.001, 0.002
	10	0.6993	0.8166	0.2924	35.3561	5.5563	3.3513	0.002, 0.005

Table 9.11: Performance of the zero baseline, the linear regression, and the lasso regression on the ANN-based features and with respect to target spaces of different dimensionality n derived with nonmetric SMACOF, along with the relative degree of overfitting. Best values for each column are highlighted for each of the regressors.

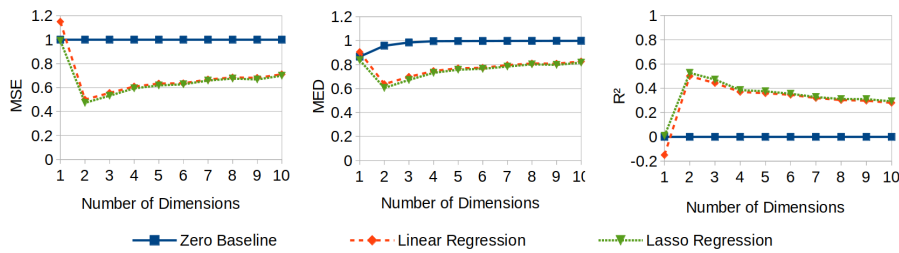


Figure 9.18: Visualization of the regression results for MSE, MED, and R^2 as a function of the number of dimensions.

relatively small amount of the observed overfitting. As we can see in Table 9.11, again a two-dimensional space yields the best results. By comparing the results to the ones from the linear regression, we can observe that the strongest relative improvements can be achieved for relatively low-dimensional target spaces. For instance, the strong regularization with $\beta \in \{1, 2\}$ applied to the one-dimensional space suffices to achieve performance levels slightly better than the baseline, while an unregularized linear regression performs worse than the baseline. Moreover, regularization tends to be less helpful in high-dimensional target spaces, especially when considering MSE and MED. Here, the degree of overfitting remains quite high, and we observe only modest improvements in performance. For the optimal values of β , we can observe the following rough general trend: For a one-dimensional space (where a linear regression is not able to beat the baseline), a large degree of regularization is needed. For the two-dimensional and three-dimensional spaces, we found an intermediate value of $\beta = 0.02$ to yield the best performance, while for all higher-dimensional spaces, only very small degrees of regularization are applied.

Figure 9.18 illustrates the results discussed before by showing the performance of the regressors as a function of the target space's dimensionality for MSE, MED, and R^2 . For the linear regression and the lasso regression, we can see clear performance "dents" for the two-dimensional target space. Overall, Figure 9.18 also illustrates, that performance decreases with an increasing number of dimensions. Moreover, we can see, that a lasso regression performs slightly better than linear regression in all cases. However, the performance gain obtained by introducing a regularization term is almost negligible in most cases.

Taken together, the results of our third experiment show that a higher-dimensional target space makes the regression problem more difficult, but that a one-dimensional target space does not contain enough semantic structure for a successful mapping. It seems that a two-dimensional space is in our case the optimal trade-off.

Finally, Figure 9.19 illustrates the two-dimensional target space along with the performance with respect to the MED achieved by the different regressors. In this visualization, we assume, that the predicted point lies in the origin. The colored circles around the origin illustrate all

Visualizing the results

Interpretation

Visualizing the prediction error

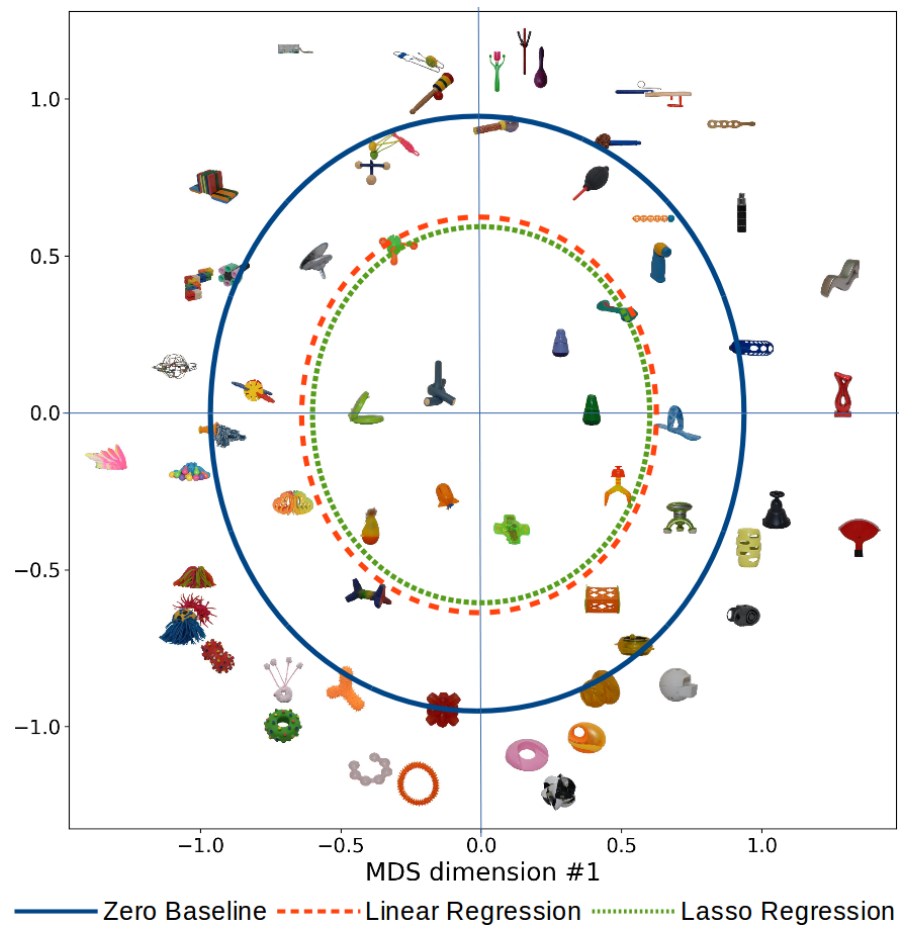


Figure 9.19: Visualization of the two-dimensional target similarity space based on nonmetric SMACOF, and the performance of the individual regressors as measured with the MED.

points, which have a distance of the given regressor's MED to the origin, i.e., they show the average location of the ground truth if the origin was predicted. The blue solid circle shows the results for the zero baseline. It reflects the overall distribution of the stimuli in the similarity space. The orange dashed circle represents the performance of the linear regression on the ANN-based feature space. It is considerably better than the baseline. Finally, the green dotted circle illustrates the MED of the lasso regression on the ANN-based feature space. As we can see, performance is only slightly better than for the linear regression.

Overall, Figure 9.19 illustrates, that we are clearly able to beat the best baseline. However, it also highlights, that the performance level achieved by our best regression is still far from satisfactory: While our regression might be capable of identifying the overall region of the similarity space correctly, the average distance between the prediction and the ground truth is still too large for any practical applications.

Interpretation

9.2.6 Discussion

As we have seen in Section 9.2.3, our system is able to perform better than the baselines, indicating, that a mapping from images to points in a psychological similarity space can be learned. Moreover, the observed performance difference on the test set between learning the correct mapping and learning the shuffled mapping indicates, that the semantic structure of a similarity space facilitates generalization to unseen stimuli. The ANN-based feature space turned out to be much more useful than the pixel-based feature space, indicating that a nonlinear, "deep" feature extraction is necessary for this task. Adding a regularization term to the linear regression helped to improve performance, but the observed strong overfitting tendencies persisted.

Feature spaces and baselines

Target spaces of same dimensionality, but based on different MDS algorithms tended to yield similar performance (cf. Section 9.2.4). One exception was the target space obtained through classical MDS, which seems to pose an easier learning problem – however, at the cost of a less faithful representation of the semantic similarity (cf. Section 9.1.3). As argued before, we assume, that the observed differences are based on a different distribution of the data points in the target space, which may in turn be rooted in the different optimization target when constructing the similarity space. In general, we did, however, not observe systematic differences between metric and nonmetric MDS.

MDS algorithms

Finally, based on our analyses in Section 9.2.5, there seems to be a sweet spot for regression performance when considering the dimensionality of the target space: A regression to a one-dimensional target space was not very successful (probably due to its poor semantic structure, cf. Section 9.1.3). Also for high-dimensional target spaces (consisting e.g., of ten dimensions), we can only observe poor regression performance – probably due to the larger number of model parameters, which leads to a very high risk of overfitting. A two-dimensional target space seems to be a good trade-off between these two tendencies, because it results

Number of dimensions

in the best regression performance. It has the additional advantage of being easy to visualize.

Soundness of our approach

These overall results are in line with our expectations and indicate, that our overall approach is sound and promising. The achieved performance is still far from perfect and considerably worse than the one reported by Sanders and Nosofsky [346], probably because of the observed overfitting tendencies. However, the goal of this feasibility study was not to aim for optimal performance, but to evaluate the underlying idea of our approach.

Reasons for overfitting

We suspect, that the relatively large amount of overfitting is caused by the small set of original images and the large size of the extracted feature vectors. Although we applied a variety of augmentation techniques to the original images, the resulting feature vectors might still be too similar to each other. This is for instance indicated by the relatively high Silhouette values observed in Section 9.2.2. Moreover, since the ANN-based feature vector contains 2048 entries, the linear regression has to estimate a large number of parameters. Furthermore, being limited to 64 different target points severely limits the variability of the dataset. Finally, since the ANN-based features were extracted from a pretrained network, which had been optimized on a different dataset with different characteristics and a different task, we cannot expect a perfect generalization.

Difficulty of the task

From a different perspective, one could argue, that even humans might fail to achieve optimal performance on this dataset, because the given problem is inherently difficult: The similarity spaces were created based on the dissimilarity ratings obtained in Horst and Hout's [199] study on a set of novel and unknown objects, for which the dissimilarity judgments might differ to a large amount between different individuals. This might be another aspect in which our dataset poses a more difficult problem than the one used by Sanders and Nosofsky [346], who used rocks as their domain of interest.

Contributions of our study

Although not reaching the performance level reported by Sanders and Nosofsky [346], our three experiments make valuable contributions to the overall research area: In our first experiment, we were able to confirm, that artificial neural networks seem to be an adequate machine learning technique for this task. In our second experiment, we showed, that the type of MDS algorithm has no significant impact on the regression results (if we exclude classical MDS from our considerations). In our third experiment, we demonstrated, that the dimensionality of the target space has a large influence on the overall results, highlighting, that this is an important hyperparameter, which needs to be considered in future studies.

Open ends

While our overall approach seems to be sound and worth pursuing, the observed overfitting tendencies and the clearly suboptimal performance found in our experiments highlight the need to focus on better performing approaches. Moreover, the performance advantage of the spaces obtained with classical MDS requires further investigations.

9.3 SUMMARY

In this chapter, we presented a first proof of concept application of our proposed hybrid approach for grounding conceptual spaces from Section 8.7. In Section 9.1, we investigated, whether the dissimilarity ratings obtained through SpAM are ratio scaled. This was analyzed by applying both metric MDS (which assumes a ratio scale) and nonmetric MDS (which only assumes an ordinal scale). Both MDS variants produced comparable results – it thus seems, that assuming a ratio scale is neither beneficial nor harmful. We therefore recommend to use nonmetric MDS, because its underlying assumptions are weaker. Future studies on other datasets obtained through SpAM should seek to confirm or contradict our results. Moreover, we noticed, that neither pixel-based nor ANN-based distances are sufficient for a good prediction of human dissimilarities, but that ANN-based distances are clearly better than pixel-based distances. In Section 9.1, we only considered Euclidean similarity spaces, i.e., we tacitly assumed, that the resulting similarity space represents a single cognitive domain. One should re-run our analyses with similarity spaces based on the Manhattan metric, which would reflect the assumption, that multiple cognitive domains (such as SHAPE, COLOR, and TEXTURE) contributed to the dissimilarity ratings.¹²

*Metric vs. nonmetric
MDS*

In Section 9.2, we then showed, that using the activations of a pre-trained ANN as features for a regression task is superior to pixel-based feature spaces. Moreover, we demonstrated, that the choice of the MDS algorithm used to construct the target similarity space is largely irrelevant, while the dimensionality of the target space is of crucial importance to regression performance. However, we observed very strong overfitting tendencies in all of our experiments. Furthermore, the overall performance level we were able to achieve is still far from satisfactory. The results by Sanders and Nosofsky [346], however, show, that larger amounts of training data can alleviate these problems. Future work in this area should focus on improvements in performance and robustness of our approach.

Machine learning

In the context of conceptual spaces, the results from our first empirical study have the obvious limitation of incorporating information from various domains such as SHAPE, COLOR, and TEXTURE. Since the dissimilarity ratings obtained by Horst and Hout [199] were not collected with a focus on one specific cognitive domain, it is not possible to use their dataset for domain-specific investigations. Moreover, due to the novel nature of the stimuli, we have no obvious candidates for meaningful directions or regions in the similarity space, limiting the types of analyses we are able to apply. Nevertheless, their dataset has served as a useful testbed for our ideas. In the following three chapters, we describe further experiments carried out on a dataset targeting explicitly the SHAPE domain. There, we will re-use and extend the analysis methods developed in this chapter.

Outlook

¹² Thanks to Corina Strößner for bringing up this point.

10 | A DATASET ON SHAPE PERCEPTION

10.1	Related Work	562
10.1.1	Shape Perception in (Neuro-)Psychology	562
10.1.2	The Shape Domain in Conceptual Spaces	571
10.1.3	Shape Features in Computer Vision	577
10.1.4	Shape Sensitivity of CNNs	582
10.2	Our New Dataset	591
10.2.1	Stimuli	591
10.2.2	Elicitation of Dissimilarity Ratings	593
10.2.3	Elicitation of Feature Ratings	595
10.3	Analysis of the Dataset	600
10.3.1	Statistical Tools	600
10.3.2	Comparing Visual and Conceptual Similarity	607
10.3.3	Comparing Pre-Attentive and Attentive Feature Ratings	615
10.4	Summary	620

It is well known, that the shape of an object is an important cue for category membership. One example for this is the so-called *shape bias* in young children [212]: Given a novel name such as "dax" for an unfamiliar object, infants tend to generalize this name to other objects with similar shapes, while disregarding other visual cues such as size or texture. However, when asked to select other objects, which are similar to the presented object, no such preference for shape information can be observed. It has therefore been argued, that the *SHAPE* domain serves as a bootstrapping device for constructing concepts [127, 242]. The shape bias is especially prominent in young children, but continues to play an important role also in adults [212, 242].¹

The shape bias

Learning concepts in a cognitive AI system should therefore be at least partially based on the *SHAPE* domain. It is, however, unclear, whether the dissimilarity ratings of the *NOUN* dataset [199] used in Chapter 9 can be solely explained by *SHAPE* similarity, or whether other additional domains (such as *COLOR*, *SIZE*, and *TEXTURE*) were also involved. In this chapter, we therefore present a new dataset, which focuses explicitly on shape similarity.

Importance of the SHAPE domain

Despite its rather obvious importance to human cognition, the internal structure of the *SHAPE* domain is still only poorly understood. In Section 10.1, we review various proposals for representing the shape

Related work

¹ While the shape bias is a relatively robust effect, some studies such as the one by León-Villagrà et al. [254] find a preference for the *COLOR* rather than the *SHAPE* domain, which may, however, be an artifact of their stimulus set and experimental setup.

domain from different points of view, including psychological theories, conceptual spaces, classical computer vision approaches, and convolutional neural networks.

Our dataset Afterwards, we present our dataset in Section 10.2. It consists of 60 line drawings of common objects from six visually coherent and six visually variable categories. We have elicited both visual and conceptual dissimilarity ratings for all pairs of these stimuli. Moreover, each stimulus has been rated with respect to three psychological shape features using two different experimental paradigms.

Data analysis Section 10.3 then provides an analysis of the raw information contained in our dataset. This includes a comparison between visual and conceptual similarity, an analysis of the ratings obtained with respect to the psychological shape features, and a comparison of visually coherent and visually heterogeneous categories.

Summary Finally, Section 10.4 summarizes the main results from this chapter and gives an outlook on future work.

The research described in this chapter has been conducted in cooperation with Margit Scheibel.² Preliminary results have been published in [52, 53], and a publication of our final results is currently in preparation [54]. Both the elicited dissimilarity and feature ratings from the psychological experiments, and the source code necessary for reproducing our results can be found on GitHub [39].³ Due to copyright issues, we are unfortunately not allowed to publicly share the full set of line drawings used as stimuli in our dataset.⁴ They are, however, available upon request.

10.1 RELATED WORK

Different perspectives on shape perception

Before presenting our own approach for grounding the SHAPE domain, we review related work about shape perception from different research areas. This includes psychological theories of shape perception (Section 10.1.1), prior accounts of the SHAPE domain in the context of conceptual spaces (Section 10.1.2), shape features used in classical computer vision setups (Section 10.1.3), and a discussion of the implicit shape sensitivity of convolutional neural networks (Section 10.1.4).

10.1.1 Shape Perception in (Neuro-)Psychology

Overview As argued above, the SHAPE domain is of crucial importance for the

² Margit Scheibel has designed and conducted the psychological experiments (Section 10.2). The dataset was then analyzed jointly (Section 10.3): I have preprocessed the raw ratings (e.g., aggregated them into dissimilarity matrices) and created several visualizations, while Margit Scheibel has focused on the statistical analysis.

³ See <https://github.com/lbechberger/LearningPsychologicalSpaces/>.

⁴ In this and the following chapters, we use a PARROT image with CC BY-NC 4.0 license (source: <http://clipartmag.com/cockatiel-drawing>) and a manual redrawing of the TELEVISION stimulus as visual examples.

recognition and classification of objects. The scientific understanding of human shape perception is, however, still quite limited, and no general consensus has emerged among researchers. In the following, we will give a brief summary of selected relevant studies in the areas of psychology and neuroscience. A comprehensive literature review is, however, beyond the scope of this dissertation.

One of the most well-known early neurological studies on visual perception has been conducted by Hubel and Wiesel [203]. They studied the receptive fields of individual neurons in the visual cortex of anesthetized cats by presenting different light stimuli (either on a screen or directly projected onto the retina) and measuring the respective firing rate of the neurons. For most neurons, they were able to identify a clear receptive field consisting of excitatory and inhibitory regions. If the light stimulus covered the excitatory regions, Hubel and Wiesel observed an increased firing rate of the respective neuron, while a light stimulus covering the inhibitory regions led to a decrease in neural firing rate and an activity burst on the disappearance of the stimulus.

The visual cortex of cats

The strongest responses could be elicited by light stimuli covering the entire excitatory regions, but no part of the inhibitory regions. In other words, the shape of the light stimulus had to match the structure of the receptive field. Hubel and Wiesel found, that the receptive fields typically had a central region of one type (excitatory or inhibitory) flanked by regions of the opposite type, leading to preferred stimuli of a slit-like form. However, they also reported other forms of receptive fields, for instance, with asymmetrical regions. Hubel and Wiesel observed, that by rotating a rectangular light spot, one can find the preferred stimulus orientation for neurons with slit-like receptive fields. In many cases, moving this slit-like stimulus across the visual field was necessary to elicit neural responses. Motion-sensitive neurons also had a preferred stimulus orientation and movement direction, and tended to respond asymmetrically to "forward" and "backward" movements.

Receptive fields

Overall, the study by Hubel and Wiesel illustrates the importance of oriented edges for visual perception with respect to both stationary and moving objects.

Edges as building blocks

Neuroscientific research on shape perception has of course continued. For instance, Op de Beeck et al. [308] have conducted an fMRI study⁵ with human participants using artificial stimuli from nine classes. These classes had some overlap with respect to their aspect ratio and overall region (called the *shape envelope* by Op de Beeck et al.), but differed with respect to their local surface (smooth, spiky, or edgy; referred to as *shape features*). Op de Beeck et al. found in their study, that pixel-based similarity captures mostly the shape envelope, while human dissimilarity ratings mostly referred to the local shape features. Moreover, shapes judged as "similar" in a psychological rating experiment also resulted

Human fMRI results

⁵ *Functional magnetic resonance imaging (fMRI)* is a noninvasive technique for measuring brain activity based on changes associated with blood flow – brain areas with increased blood flow are assumed to be more active than brain areas with reduced blood flow.

in similar brain activity in corresponding fMRI scans, highlighting, that shape similarity has a neural grounding.

The HMAX model

An example for a neuroscientifically motivated model of the human visual system is the *HMAX model* as proposed by Riesenhuber and Poggio [331]. Their motivation stems from the observation, that there are cells in the inferotemporal cortex (a brain area associated with high-level vision) of macaque monkeys, which respond selectively to individual faces: They recognize their preferred stimulus independent of its size and location in the visual field, and are at the same time able to distinguish it from other faces in similar poses. The HMAX model is a hierarchical feedforward architecture, which aims to obtain viewpoint independence for higher-level neurons by pooling over a set of viewpoint-sensitive neurons. Riesenhuber and Poggio note, that this pooling can in principle be achieved through a linear summation of activation values, or by considering only the maximal activation among the inputs. They argue, that the maximum is expected to provide more robust responses in the case of cluttered scenes, where additional objects in the background can be expected to increase the activity of most input cells: While the linear summation results in a considerably increased output activity (since it sums the additional noise components from all inputs), the nonlinear maximum does not consider the increased activity of non-winning inputs. Riesenhuber and Poggio also discuss the softmax function (cf. Section 6.2.1) as an anatomically more plausible variant of maximum pooling: They note, that an approximation of softmax can be explained for instance by lateral inhibition in the cortex.

Relation to ANNs

The resulting model uses different layers of neurons, which compute either a weighted sum of their inputs (in order to obtain more complex features), or which apply the maximum over their inputs (in order to obtain a more invariant representation). Riesenhuber and Poggio successfully applied their model for a dataset of paper-clip-like three-dimensional objects from different viewpoints. Their model used handcrafted receptive fields (e.g., simple Gaussians of varying rotation for the first layer, mimicking to some extent the slit-like receptive fields observed by Hubel and Wiesel [203]) and fixed mappings between the layers. The HMAX model should thus not be confused with ANN-based machine learning approaches as introduced in Chapter 6. However, the usage of a maximum operator for increasing feature stability can be related to the usage of max-pooling layers in convolutional neural networks (cf. Section 6.2.2). Also the emphasis on using nonlinear operations can be related to the discussions from Section 6.2.

The role of top-down expectations

The HMAX model is an example of the bottom-up feature construction process associated with the visual ventral pathway, which uses simple basic features such as ORIENTATION and LOCATION to construct a hierarchy of more and more complex features based on vertices, multiple parts, and different viewpoints. Bar [29] notices, that the existence of a large number of top-down connections in the brain is often largely unacknowledged in such approaches. He argues, that top-down processes

are nevertheless of crucial importance: The expectations about possible interpretations of the input can serve as additional constraints for the object recognition process. Only if the bottom-up and the top-down processes meet and match, successful recognition can take place.⁶ It is, however, unclear, how such a top-down process could be initiated at the same time as the bottom-up process, i.e., before the visual input has been completely analyzed.

Bar notes, that spatial high-frequency information represents abrupt spatial changes (such as edges or other fine details), while low-frequency information corresponds to the global object shape (including its orientation and proportions). The low-frequency information can thus be interpreted as a blurred version of the original image, which can be used to restrict the set of candidate categories for the given object. Bar therefore speculates, that low spatial frequencies of the image are transmitted to the prefrontal cortex and are there converted into expectations about possible interpretations. These expectations are then projected to the highest level of the visual hierarchy, where the respective representations of the candidate categories are activated and then integrated with the bottom-up analysis of the image. In Bar's account, a rough perception of the overall object shape therefore enables a "first guess" about object identity, before the perception of finer details disambiguates between different possible interpretations.

Spatial frequencies

An important psychological study on relevant features in shape perception has been conducted by Treisman and Gormican [409], who have focused on differences between early and later stages of visual processing. They assume, that early visual processes are responsible for extracting a description of the three-dimensional object based on visual primitives, while later stages of visual processing then use this initial representation to perform object detection. Treisman and Gormican furthermore assume, that attentional mechanisms can be used to distinguish between earlier and later processing stages, since they may change the perception of a given scene. For instance, an upright T is harder to spot in a context of Ls than in a context of tilted Ts, where it immediately "pops out" due to the different orientation of its vertical line. Interestingly, this *pop-out effect* is relatively independent from the number of objects in the current context. However, when explicitly asking study participants about the respective visual similarity relations, they typically insist that an upright T is more similar to a tilted T than to an L. In this conscious comparison, the arrangement of the lines seems to receive more attention than their exact orientation.

Two steps of visual processing

Treisman and Gormican now propose to use the aforementioned visual pop-out effect as an indicator for the availability of features in early visual processes: If a target stimulus differs from a context of distractors only with respect to one visual property, and if the presence or absence of the target stimulus can be quickly determined based

Using the pop-out effect

⁶ The general idea of matching bottom-up activation with top-down expectations has already been discussed in Section 7.3.3 in the context of adaptive resonance theory [90, 175].

on the pop-out effect, then this feature is likely to be used in early visual processes. Treisman and Gormican argue, that the recognition of low-level features can be modeled through feature detectors, which are applied to all areas of the visual scene simultaneously, and which yield high activity, if the desired feature is detected, and low activity otherwise. According to their reasoning, a spot of high activity on a background of low activity is easier to detect than a spot of low activity in a context of high activity. Treisman and Gormican therefore expect asymmetries in the observed behavior: If the target differs from the distractors through the presence of an additional property, it should be easier to detect than if it differs from the distractors through the absence of a given property.

Experimental results

In their study, Treisman and Gormican investigated twelve different candidate features for shape. They employed a visual search task, where they measured participants' reaction times for simple stimuli consisting of different types of lines. Features, which seemed to relate to early vision (as indicated by the pop-out effect), include COLOR and CONTRAST, LINE CURVATURE, LINE ORIENTATION, LINE LENGTH, NUMBER OF LINES, and CLOSURE (i.e., whether a given circles was closed or contained a gap). Features, which seemed to be related to later visual processes (since no pop-out effect could be observed), include arrangement of lines (INTERSECTION, JUNCTURE, ANGLES) and topological properties such as CONNECTEDNESS, CONTAINMENT, and HEIGHT-TO-WIDTH RATIO. For most of the candidate features, Treisman and Gormican also found the expected asymmetry effects, pointing towards a specific "default" value of the feature (e.g., focal colors for the COLOR feature).

Relation to other studies

In addition to the orientation sensitivity as already found by Hubel and Wiesel [203] and as implicitly used in the first layer of the HMAX model [331], Treisman and Gormican thus provide a large variety of candidate features for earlier and later stages in the visual process. Especially the features relating to early vision can be linked to Bar's proposal [29] of using rough shape information in order to initiate top-down processes.

Other studies on basic shape features

Also other researchers have tried to establish basic shape features. For instance, Ons et al. [307] have explicitly asked participants to analyze given shapes according to their ASPECT RATIO and their MEDIAL AXIS CURVATURE. They found, that participants were able to attend to these two properties in a separable way (cf. Sections 1.2.1 and 2.1.1), which supports their status as basic shape features.

A data-driven approach

Also Huang [202] has attempted to find basic dimensions of pre-attentive shape features. However, instead of using a list of candidate features, he followed a more data-driven approach. He used 14 different filled shapes for his psychological experiment, which involved a texture segregation task: One of the shapes was selected as background and another one as foreground. These shapes were then placed in an overall grid, where the foreground shape made up a horizontally or vertically elongated rectangle and the background shape was used to fill the remainder of the grid. For both foreground and background shapes,

different rotations were used. Participants looked at the resulting image for 100 ms and were then asked to decide, whether the rectangle had been horizontally or vertically elongated. The underlying intuition behind this task is, that visually dissimilar shapes should be relatively easy to discriminate and should thus lead to higher accuracies.

Huang then used an optimization procedure similar to MDS in order to find an n -dimensional spatial representation, where distances between points correlated well with the observed average accuracies. He reports, that a three-dimensional space seemed to be sufficient to obtain high correlations. By manually rotating the resulting solution, he found three underlying shape features: The *segmentability* dimension was sensitive to the presence or absence of a clear intersection or joint, while the *COMPACTNESS* dimension seemed to detect holes or concavities. Finally, the *spikiness* dimension distinguished smooth contours from sharp spikes.

Analysis results

One of the most well-known psychological models of shape perception is the *recognition by components (RBC)* model as proposed by Biederman [64]. It uses a bottom-up process starting with an edge detection phase, which gives rise to a two-dimensional representation of the object similar to a line drawing. This edge-based representation is used to split the overall object into multiple components by searching for concave regions. For each of these components, different so-called *nonaccidental properties* are detected, which are used to make inferences about the three-dimensional real world based on the two-dimensional information:

Recognition by components

- *Collinearity*: Straight lines in a two-dimensional image correspond to straight lines in the three-dimensional world.
- *Curvilinearity*: Smoothly curved elements in 2D correspond to smoothly curved features in 3D.
- *Symmetry*: A symmetrical 2D image corresponds to a symmetrical 3D object.
- *Parallelism*: Parallel lines in 2D are also parallel in 3D.
- *Cotermination*: Lines, which meet in the 2D representation, also meet in the 3D world.

These properties are called nonaccidental, since it is highly unlikely to observe them "by accident". For instance, it is very unlikely that the curved line of a three-dimensional object is perceived as straight in a two-dimensional projection.

Nonaccidental properties

Using the nonaccidental properties, each component can now be represented by so-called *geons*. These geons represent different simple volumes such as boxes, cones, or horns. They can be parametrized by looking at their cross section: The *OUTER EDGES* of a geon can be either curved or straight. The overall *CROSS SECTION* can be either symmetric or asymmetric, and the *CROSS SECTION SIZE* may remain constant, it may

Geons and their properties

expand in one direction, or it may expand and contract again. Finally, the MAIN AXIS of the object (perpendicular to the cross section) may be straight or curved. Biederman now argues, that these four properties can be directly grounded in the nonaccidental properties introduced above: For instance, the distinction between curved and straight edges is based on collinearity and curvilinearity.

Examples

In order to illustrate the definition of geons, let us consider two elementary shapes. A box has straight edges, is symmetric, its cross section has a constant size, and the object's main axis is straight. A horn, on the other hand, has also a symmetric cross-section, but it has curved edges, an expanding size, and a curved main axis.

Objects as configurations of geons

Once the geons representing the individual parts of the object have been determined, this overall description can be matched to the respective conceptual description of several candidate objects. These descriptions do not only consider the type of geons used for the different parts, but also their configurations. This means, that the arrangement of the object's parts plays a crucial rule for object recognition.

Limitations of the model

The geons themselves only encode the rough shape, but not exact properties such as WIDTH OR ASPECT RATIO. Also spatial relations between these geons are only specified in coarse terms such as "above" or "left-of". Thus, the shapes of a cow and a horse have the same structural description based on geons, while belonging to different categories – here, finer-granular information (e.g., the length of the neck) is necessary to make a distinction between categories [146].

A language of thought for shapes

Sablé-Meyer et al. [342] have recently proposed a compositional language of thought for the SHAPE domain. They assume, that basic shapes can be represented as short programs, which contain the necessary instructions to draw the respective shape. In their account, shape recognition corresponds to searching for the shortest program, which correctly draws the perceived shape. Shape complexity can then be linked to the length of the minimal program for describing this shape (cf. the minimum description length (MDL) principle, Section 5.1.3). Their proposed language contains three instructions for pen movement, namely, turning (by a specified angle), moving (in a straight line for a specified distance), and tracing a curve (based on specified duration and curvature). Moreover, it includes three compositional structures, namely, concatenation, repetition, and call to a subprogram. The MDL of a given shape then corresponds to the number of nodes in the corresponding program's syntax tree.

Shape recognition

As Sablé-Meyer et al. note, searching for the shortest program is computationally quite demanding due to a combinatorial explosion. They propose to use an ANN-based program induction technique, where a neural networks predicts prior probabilities for the different language primitives based on the raw pixels of the image. The search is then biased towards the types of operation, which are more likely to be contained in the shortest program. For instance, if the input shape contains many straight lines, then the ANN may predict a higher probability for the movement operation than for the trace operation.

Sablé-Meyer et al. found, that the shortest programs in their proposed language tended to produce quite simple shapes, which are relatively widespread in different human cultures. In two psychological studies, they furthermore confirmed, that the MDL according to their model was a good predictor of shape complexity, according to both participants' response times in a recognition task and subjective complexity ratings. Since these observations also held for complex shapes (e.g., a rectangle made of small circles), Sablé-Meyer et al. take their results as supporting especially also their proposed compositional structures. This focus on compositionality can be linked to the RBC model [64].

Experimental results

A recent overview of different shape representation theories has been given by Erdogan et al. [146], who distinguish the following general approaches, based on the classifications made by Palmer [314]:

Classification of approaches

Feature-based accounts of shape perception assume, that an object's shape is represented by a list of feature values extracted from 2D images. These features are assumed to be the result of a hierarchy of feature extractors. The similarity of two shapes is then usually based on the Euclidean distance of their respective feature values. According to Erdogan et al. [146], this is the predominant view held in neuroscience. Typical models from this class of approaches include HMAX [331] and convolutional neural networks (cf. Section 6.2.2). Moreover, the studies by Treisman and Gormican [409], Ons et al. [307], and Huang [202] fall into this category, since they explicitly focuses on a feature-based description of stimuli. Also conceptual spaces with their small number of interpretable dimensions are a feature-based approach.

Feature-based approaches: shapes as lists of features

According to Erdogan et al. [146], the main shortcoming of feature-based approaches, is that one needs to specify the features. This can be done either explicitly (as in the HMAX model) or implicitly (as in convolutional neural networks). An explicit specification of all relevant features is unfortunately often difficult in practice. An implicit specification of the features, on the other hand, corresponds to learning them from data. Clear limitations of CNNs are, that they are hard to interpret, and that the individual features learned by the systems may not only depend on the chosen network architecture, but also on the training data being used, the training objective, and the training procedure.

Limitations of feature-based approaches

View-based approaches emphasize the viewpoint-dependency of shape perception. They represent the shape of an object by a collection of memorized views from different viewpoints, and are thus a special case of the exemplar approach to concepts (cf. Section 1.1.1). According to Erodgan et al. [146], view-based approaches are in principle agnostic about how exactly a view is represented, but often use a feature-list representation in practical applications. The recognition of a given stimulus shape is based on a comparison to all exemplar views. This comparison may involve finding an optimal alignment of the views or a transformation into a canonical view. From our discussion above, only the HMAX model [331] explicitly considers multiple views.

View-based approaches: shapes as sets of views

<i>Limitations of view-based approaches</i>	Erdogan et al. [146] argue, that the greatest weakness of view-based approach is the assumption, that the same set of features can be extracted from all views, which is necessary for making comparisons.
<i>Structural descriptions: shapes as hierarchical structures</i>	<i>Structural descriptions</i> assume, that shape perception is hierarchically structured. In these approaches, an object shape is represented by a list of shape primitives and their spatial relations. Typically, three-dimensional object-centered representations are being used. The most well-known model of this type is Biederman's RBC model [64], which has been introduced above. Also the approach proposed by Sablé-Meyer et al. falls into this category.
<i>Properties of structural descriptions</i>	Erdogan et al. argue, that the extraction of a structural representation from a given image may be complicated. On the other hand, structural descriptions are capable of representing part-whole structures, which gives them larger representational capabilities than the feature-based and view-based approaches.
<i>Vision as Bayesian inference</i>	Erdogan et al. [146] also propose a new model for shape perception, which aims at overcoming the shortcomings of the existing approaches. They assume, that the human visual system needs to infer three-dimensional shape information from the retina image. Based on the current viewpoint, this inference may come with different degrees of uncertainty about different parts of the object. This viewpoint-dependency is modeled by assuming a probabilistic object representation (with the underlying shape properties corresponding to random variables) and by treating vision as a Bayesian inference process. ⁷
<i>Experimental results</i>	In a behavioral experiment, Erdogan et al. collected human similarity judgments using the triad method (cf. Section 8.1.1) for a dataset of artificial block-based objects. In addition to their own model, they also employed different models from the above mentioned categories, including HMAX and CNNs (as feature-based models), a pixel-wise comparison of images and an alignment-based model (as view-based models), as well as a structural distance model using a tree-based object representation. Their results showed, that their Bayesian inference model agreed with the binary human responses in about 70% of the cases. The CNNs reached an agreement of around 60%, while almost all other approaches were only slightly better than chance.
<i>Limitations of their model</i>	Erdogan et al. acknowledge, that their model is limited in two respects: Firstly, it is intended only for unfamiliar objects. This controls for potential confounding factors such as object use. Erdogan et al. assume, that familiar objects such as faces may be processed by specialized systems. Secondly, their model is rather normative than descriptive – since Bayesian inference is unlikely to happen in the human brain, their approach describes an optimal way of solving the shape recognition problem, but not necessarily the actual psychological process.

⁷ Bayesian inference has already been introduced in Section 7.1.2 in the context of concept learning in conceptual spaces.

10.1.2 The Shape Domain in Conceptual Spaces

Since the *SHAPE* domain is quite critical for identifying and classifying objects, it has experienced prior treatment in the context of conceptual spaces. We will now review two concrete proposals from the literature, which can both be classified as structural approaches.

Structural descriptions of the SHAPE domain

In his book outlining the conceptual spaces framework, Gärdenfors also discusses the importance of the *SHAPE* domain [179, Section 3.10.2]. He mainly refers to the model proposed by Marr and Sishihara [278], which uses configurations of cylinders to describe shapes on varying levels of granularity. Marr and Sishihara have developed their model mainly based on computational considerations: By interpreting human vision as the computational problem of extracting relevant information from raw visual data, they argue, that any psychologically plausible shape representation should be object-centered (rather than viewer-centered), volumetric (rather than surface-based), and modular (rather than unorganized). Based on these three constraints, they claim, that a representation similar to stick figures is promising, where primitive components correspond to the major axes of the object. If one adds information about *LENGTH* and *DIAMETER*, each of these axes can be represented by a cylinder. The shape of an object is then represented as an arrangement of multiple cylinders, along with the respective lengths and diameters of the individual cylinders.

A cylinder-based model

This cylinder-based representation can be transformed into a coordinate system: Each cylinder can be represented with two parameters (its *LENGTH* and its *DIAMETER*). The overall configuration of cylinders can be described by defining for each cylinder its relative *LOCATION* (with respect to the three spatial dimensions) and *ROTATION* (using polar coordinates for the rotation angles). If the number of cylinders is fixed, one can thus derive a conceptual space for the *SHAPE* domain with a fixed number of dimensions. As Gärdenfors [179, Section 3.10.2] argues, it is relatively easy to define both a betweenness relation and a distance function in this space.

Deriving a similarity space

Marr and Sishihara propose to use this representation scheme at multiple levels in the hierarchy. In Figure 10.1, one can see, that each individual cylinder from a given level of granularity can again be modeled in more detail by using multiple cylinders. For example, the arm is modeled by a single cylinder in the overall model of the human. A more detailed model of the arm, however, makes a distinction into upper arm and fore-arm. It is important to note, that there is only a single level of granularity for the overall human, and that each finer-granular model only refers to a single part of its "parent model".

Part-whole relations

Figure 10.2 shows, that the proposal by Marr and Sishihara also accounts for conceptual hierarchies: Superordinate concepts (such as *BIPED* or *QUADRUPED*) determine the number of cylinders and their overall arrangement, but leave their relative proportions largely unspecified. Subordinate concepts (such as *COW* or *GIRAFFE*) then further constrain the parameters of the individual cylinders, e.g., by narrowing down their

Conceptual hierarchies

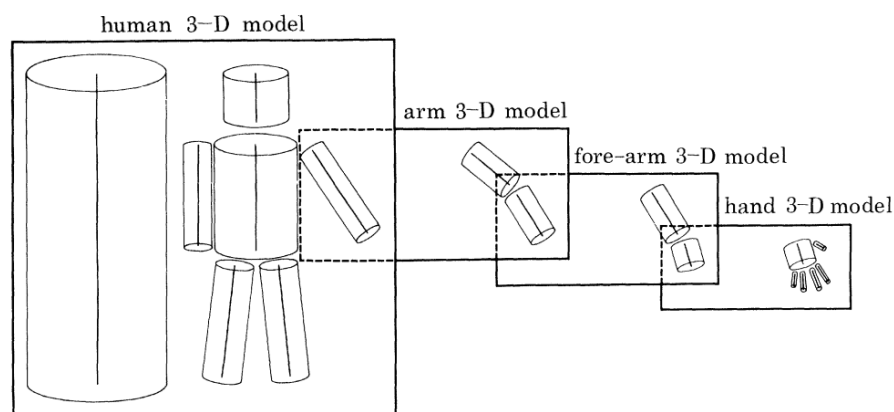


Figure 10.1: Part-whole relations in the proposal by Marr and Sishihara. Figure taken from [278], reprinted with permission.

respective LENGTH and DIAMETER. For instance, the cylinder representing the neck of a QUADRUPED exists in both COW and GIRAFFE at the same relative LOCATION. However, while it is relatively short for COW, it is quite long for GIRAFFE. These additional constraints on the individual parameters correspond to defining subintervals on the respective dimensions of the similarity space. Thus, subordinate concepts are represented as subregions of superordinate concepts.

*Psychological
plausibility*

While the model proposed by Marr and Sishihara follows a hand-crafting approach, there is some indirect psychological support for their proposal: Firstly, they derived their model based on constraints, which emerge from an interpretation of vision as a computational process. Secondly, they argue, that stick figures such as the ones in Figures 10.1 and 10.2 are easily recognized and classified by humans, even though they are a quite strong simplification of the actual shapes of the underlying objects. Therefore, the information contained in their model might be not too dissimilar from the information processed by humans when recognizing arbitrary shapes. Marr and Sishihara also note, that information about an object's main axes can be easily extracted from images based on simple computer vision algorithms.

*Relation to the RBC
model*

The proposal by Marr and Sishihara is limited to cylinders as basic components. It is thus incapable of faithfully representing round objects (such as APPLE or TENNIS BALL) or box-like objects (such as BOOK or WASHING MACHINE). It has therefore been criticized by Biederman [64], whose geons in the RBC model cover a wider range of elementary shapes. However, as already discussed in Section 10.1.1, these geons are only defined in a qualitative way, while the cylinders used by Marr and Sishihara can be quantitatively specified by denoting their LENGTH and DIAMETER.

*An approach based
on superquadrics*

Another proposal for representing the shape domain has been made by Chella et al. [97] (cf. Sections 1.2.4 and 2.4.3), who used conceptual spaces in the context of computer vision. In order to represent simple

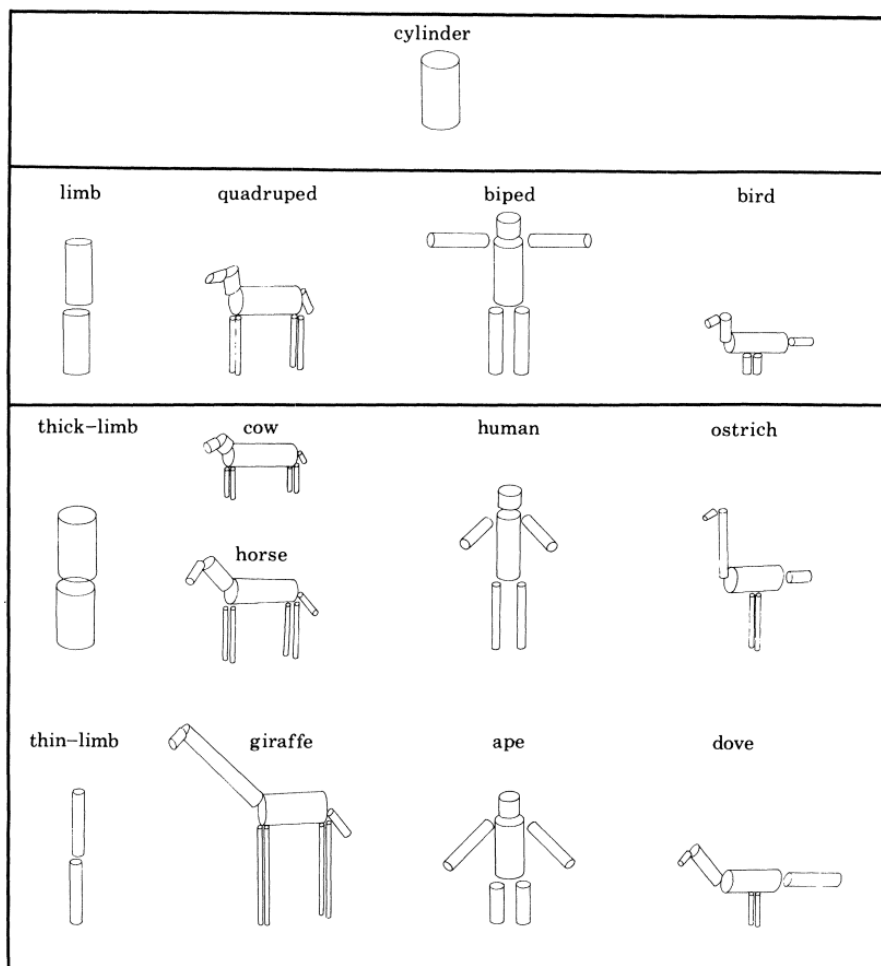


Figure 10.2: Conceptual hierarchies in the proposal by Marr and Sishihara.
Figure taken from [278], reprinted with permission.

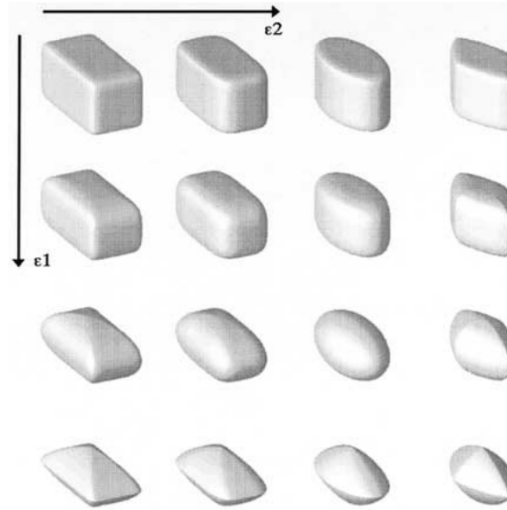


Figure 10.3: Different superquadrics based on varying form factors ϵ_1 (rows, ranging from 0.2 to 1.5) and ϵ_2 (columns, ranging from 0.2 to 1.5). Figure taken from [97], reprinted with permission.

objects, they used so-called *superquadrics*, which can be described by the following formula:

$$f(\eta, \omega) = \begin{bmatrix} a_x \cos^{\epsilon_1} \eta \cos^{\epsilon_2} \omega \\ a_y \cos^{\epsilon_1} \eta \sin^{\epsilon_2} \omega \\ a_z \sin^{\epsilon_1} \eta \end{bmatrix}$$

Superquadrics

The function f returns a vector containing the spatial coordinates of the points on the surface of a superquadric, dependent on the input angles $\eta \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ and $\omega \in [-\pi, \pi)$. The three parameters a_x, a_y, a_z represent the length of the superquadric with respect to the three axes of the coordinate system. The form factors ϵ_1, ϵ_2 influence the convexity/concavity of the resulting shape with respect to its longitude and latitude, respectively. Figure 10.3 illustrates different superquadrics for different values of ϵ_1 and ϵ_2 .

Representational power

Overall, any superquadric can be represented by a point in a five-dimensional similarity space using the aforementioned parameters $(a_x, a_y, a_z, \epsilon_1, \epsilon_2)$. Chella et al. argue, that superquadrics therefore provide a compact representation of basic shapes, while maintaining a sufficient degree of expressive power. More specifically, many simple geometric objects such as boxes, cylinders, and spheres form convex regions in this similarity space. Furthermore, Chella et al. claim, that the parameters of superquadrics are can be easily extracted from a given image with specialized computer vision techniques.

Part-whole relations

Chella et al. use multiple superquadrics to represent objects, which consist of multiple parts. For instance, a HAMMER is not represented holistically, but in the form of two separate parts, namely its HANDLE and its HEAD. Each of these parts is represented by a point in the overall SHAPE space, which is enriched by LOCATION and ROTATION information. An object consisting of multiple parts is therefore represented by a set

of points rather than a single point in the conceptual space (cf. Section 2.4.3, where we discussed their representation of part-whole structures in more detail). One should note, that the proposal by Chella et al. is not supported by any psychological evidence.

The proposal by Chella et al. is strongly related to the one by Marr and Sishihara, since both make use of a structural representation. Marr and Sishihara use cylinders as shape primitives, while Chella et al. use the more flexible class of superquadrics. This makes their representation more powerful, because both round and box-like objects can be easily represented. It differs from Biederman's geons [64] by specifying the parts not only qualitatively, but also quantitatively. In both the proposals by Chella et al. and Marr and Sishihara, complex objects are represented as a configuration of multiple simple objects. Marr and Sishihara impose a clear hierarchical structure, where each cylinder can be described as a configurations of cylinders on a lower level of granularity. In the approach by Chella et al., such an explicit hierarchical structure is, however, missing.

Comparison to Marr and Sishihara

The two approaches also differ with respect to the way, in which complex objects are represented: Chella et al. represent complex objects by a set of points in the same part-based similarity space. Objects with different numbers of parts are thus represented by a different number of points, respectively, using, however, the same underlying similarity space. This approach requires a more elaborate way of computing distances: Instead of defining similarity based on distance between points, one needs to consider distance between *sets* of points. Moreover, in the conceptual spaces framework, concepts are defined as regions in the similarity space, i.e., as sets of points. If complex objects are represented by sets of points, concepts need to be represented by sets of sets of points, adding another level of complexity.

Representing complex concepts in part space

Gärdenfors' usage of the model by Marr and Sishihara, on the other hand, uses a single overall similarity space for describing complex objects, where different subspaces are used to represent different parts. Objects or concepts with different numbers of parts are then represented by individual points in similarity spaces of different dimensionality. For instance, in Figure 10.2, cow is represented using eight cylinders, while HUMAN is represented using only six cylinders. This makes it quite complicated to compute the SHAPE-based similarity of two objects or concepts with a different number of parts, because they are represented in different similarity spaces. One could artificially fix the number of cylinders and thus the number of dimensions in the SHAPE space. However, one then needs to account for unused cylinders.

Representing complex objects in the product of all part spaces

As we have seen, neither of these approaches comes without problems. Moreover, both proposals (and most structural approaches in general) face an additional challenge: Even if two objects or concepts use the same number of parts, one still needs to align them correctly with each other to ensure that the computed distance is meaningful.

Drawbacks of structural approaches

Our approach for obtaining a SHAPE space (to be described in Chapter 11) therefore does not follow a structural approach. We rather focus on

Arguments for a feature-based approach

a feature-based representation using a holistic similarity space with a fixed number of dimensions. Moreover, unlike the proposals by both Marr and Sishihara [278] and Chella et al. [97], we base our *SHAPE* space directly on psychological data. Since the same space is used for all stimuli, we do not encounter the aforementioned complications when computing distances. We think, that part-whole structures should be added to the conceptual space as a whole (cf. the proposal by Fiorini [158] discussed in Section 2.4.3) instead of being enforced only in the *SHAPE* domain. According to this reasoning, the same *SHAPE* space can be applied to the whole object (representing holistic perception) and recursively to its parts, just like similarity spaces for *COLOR*, *SIZE*, and other domains. The overall configuration of individual parts would in this case be represented by the structure domain as proposed by Fiorini [158] (cf. Section 2.4.3).

*An angle-based
SHAPE space*

Our approach can thus also be related to the work by Li et al. [255], who constructed a stimulus space of 360 shapes. They noted, that the well-established CIE $L^*a^*b^*$ *COLOR* space can be projected onto a two-dimensional color wheel, if luminance is fixed. In this two-dimensional color wheel, similarity between different color stimuli can be measured by the angle they enclose. Li et al. aim to create such a circular space for shapes, where shape similarity can be measured through angles. Starting with a small set prototypes of simplistic shapes (simple outline drawings without any inner structure), they generated interpolations between these prototypes and then collected dissimilarity ratings to validate their respective psychological distances (using both the triad and the pairwise rating method, cf. Section 8.1.1). Li et al. then created similarity spaces from the resulting dissimilarity matrices using MDS. They then computed the circularity of the resulting configuration (i.e., whether stimuli lied approximately on a circle), and used this information to select a new subset of more equally-spaced prototypes for the next iteration of the aforementioned procedure. Overall, they obtained a similarity space with 360 shapes, which is approximately perceptually uniform.

*Limitations from the
conceptual spaces
perspective*

Although Li et al. were able to obtain a two-dimensional *SHAPE* space, one should, however, note, that the resulting similarity space is not a conceptual space: By considering the angle between stimuli in a circular arrangement, it implicitly targets the Cosine similarity rather than the Euclidean distance. Moreover, it is not based on interpretable dimensions and does not have any clear conceptual regions. Nevertheless, it may be useful for practical research, since it provides a continuous interpolation between different shape stimuli, which are normalized with respect to their pairwise similarity. It is, however, not adequate as a model for the *SHAPE* domain in the context of conceptual spaces. Our own approach, on the other hand, will explicitly focus on interpretable directions and meaningful conceptual regions.

10.1.3 Shape Features in Computer Vision

In classical computer vision, the overall process of classifying an image is split into two parts: In a first step, a relatively small set of manually defined features are extracted from the overall image. The image itself is then represented as a low-dimensional feature vector rather than by its raw pixel values. In a second step, standard machine learning algorithms (cf. Chapter 5) are applied to this feature-based representation in order to extract classification rules from large datasets of examples. With respect to shape perception, especially the first step of this process is of relevance, i.e., the extraction of shape-related features.

Two steps in classical computer vision

Feature detection is as a low-level preprocessing step in computer vision which takes the original pixel intensities as input, and which outputs basic information about the image's structure [256]. One can distinguish edge detection, corner detection and blob detection, which focus on detecting different important structures in the image: *Edge detection* aims to discover pixels, at which the image intensities change abruptly. By connecting neighboring edges, one can obtain the contour of an object. *Corner detection* can then be seen as the process of finding intersections of edges or points on a contour, where the direction of the contour changes abruptly. *Interest region detection* as a special case of *blob detection* can be related to image segmentation, and aims to identify closed sets of connected points. Li et al. [256] have given a recent overview of various algorithms for solving these feature detection problems. At this point, we should also highlight, that these processing steps can be easily related to visual processes in humans as discussed in Section 10.1.1, where both edges and corners are also viewed as elementary building blocks for higher-level representations.

Different types of feature detection

While feature detection allows us to identify the contours and regions in a given image, they are usually not sufficient for giving a concise description of the underlying object. Therefore, a wide variety of higher-level shape features have been developed in the computer vision community. We base our following summary on the overview articles by Zhang and Lu [456] and by Mingqiang et al. [293]. They consider the area of content-based image retrieval (where one aims to find similar images to a given query image), and classify different shape features according to different criteria:

Shape features on top of feature detection

- *Global approaches* represent the shape holistically, while *structural approaches* use a list- or tree-based representation of individual shape parts. This roughly reflects the distinction into feature-based and structural approaches from Section 10.1.1.
- *Contour-based methods* focus exclusively on the contour of the object, while *region-based methods* also take into account all pixels from the inside of the shape's region.
- Comparison between shapes can take place either in a *spatial domain* (i.e., based on pixels/points) or in the *transformed domain* (e.g., based on components of a Fourier transformation).

Level of analysis

Underlying structure

Domain of comparison

<i>Reconstruction capability</i>	<ul style="list-style-type: none"> • <i>Information preserving</i> features allow a faithful reconstruction of the original shape, while <i>non information preserving</i> features do not support such a reconstruction.
<i>Processing approach</i>	<ul style="list-style-type: none"> • Finally, one can distinguish various types of feature extraction processes, e.g., <i>spatial interrelation</i> (where a shape is described by the relations between its pixels), or <i>shape transformations</i> (e.g., a Fourier transformation, which transforms the pixel space into a frequency space).
<i>Organization</i>	The following overview will be organized around the distinction into global and structural approaches, commenting on the classification with respect to the other criteria where necessary and relevant.
<i>Simple shape descriptors: intuitive high-level features</i>	<p>The most straightforward way to describe a shape globally is given by <i>simple shape descriptors</i>, which compute simple geometric features of the given shape [293], and which can be applied both to the shape's contour and its region. Examples for such simple shape descriptors include ECCENTRICITY (which divides the length of the major axis by the length of the minor axis), CIRCULARITY RATIO (measured for example by dividing the shape area by the area of an circle with the same perimeter as the shape), and AVERAGE BENDING ENERGY (which looks at all arcs between points on the shape's contour and computes their average curvature).⁸ In general, simple shape descriptors have a clear intuitive meaning, but are only able to discriminate shapes with large differences [456]. They are thus only rarely used as standalone features, and are typically supplemented by other shape descriptors. Alternatively, they can be used as a first filtering step, which excludes clearly dissimilar shapes from further consideration. These simple shape descriptors are closely related in spirit to the features considered by Treisman and Gormican [409], which also target simple geometric properties, that are easy to interpret (cf. Section 10.1.1).</p>
<i>Correspondence-based shape matching: distance between sets of points</i>	<p>In <i>correspondence-based shape matching</i>, no explicit feature representation is extracted, but shapes are represented by the set of points defining their contours. A comparison of two shapes is then based on a modified version of the Hausdorff distance between the two sets of points [456].⁹ Correspondence-based shape matching is quite inefficient, because its underlying representation is not compact, and because the matching function is computationally expensive. Moreover, it is quite sensitive to noise. <i>Grid-based methods</i> are a variant of this approach, which create a binary feature vector by overlaying a grid on the shape and noting, whether a given grid cell is filled or not [456]. Comparisons between shapes can then be made by using the Hamming distance¹⁰, provided that the shapes have been normalized and rotated into a canonical orientation (e.g., by aligning the shape's major axis with the x-axis of</p>

⁸ See [293] for a more detailed overview of simple shape descriptors.

⁹ The Hausdorff distance $d_H(A, B)$ of two sets A and B is defined as $d_H(A, B) = \max(\sup_{a \in A} \inf_{b \in B} d(a, b), \sup_{b \in B} \inf_{a \in A} d(a, b))$, cf. Section 4.7.

¹⁰ The Hamming distance $d_{Ham}(\vec{x}, \vec{x}')$ of two vectors \vec{x} and \vec{x}' is defined as $d_{Ham}(\vec{x}, \vec{x}') = |\{1 \leq i \leq n \mid x_i \neq x'_i\}|$, cf. Section 5.2.5.

the coordinate system). However, this normalization and rotation preprocessing can result in very different representations for very similar shapes, if for example the difference between the major and the minor axis is quite small. Moreover, correspondence-based shape matching in general is limited to comparisons between shapes, but is incapable of giving a more concise, higher-level description of a stimulus.

Scale space approaches generate a sequence of increasingly simplified approximations of the shape boundary [293] and look for inflection points, which remain present during these simplifications, and which are then considered as important object characteristics [456]. It is, however, often difficult to give a clear semantic interpretation to the resulting features. Scale space approaches can be related to Bar's proposal [29] of using coarse-grained information to obtain a first guess about possible categories (cf. Section 10.1.1).

Scale space approaches: looking at simplified approximations

Another important global approach to describing shapes is given by *shape signatures*. A shape signature is a one-dimensional function, which is derived from the shape's boundary points. The most important example for a shape signature is the centroid distance function. It computes the centroid of the shape's contour, and then measures for each point on the boundary its distance to the centroid [456]. In general, shape signatures allow for a reconstruction of the original shape, but are usually quite sensitive to noise. Moreover, comparing two shapes is computationally costly due to the non-compact representation [456]. Also invariance with respect to translation, scale, and orientation often needs to be enforced manually [456]. Therefore, shape signatures are often not used directly as shape features, but as an intermediate processing step [293].

Shape signatures: representing the contour as a function

There are various ways of compressing the representation given by shape signatures. For instance, one can fit an *autoregressive model* (which predicts the next shape signature value based on a linear function of the k previous values), and use its coefficients as a feature vector [456]. The resulting feature vector is usually invariant with respect to translation, rotation, and scale, but the interpretation of the model coefficients is often difficult [456].

Autoregressive model: compressing a shape signature

Another way of transforming shape signatures is given by *boundary moments* [456]: Given the shape signature, one computes the r th moment (a generalization of the mean) and the r th central moment (a generalization of the variance) as higher-level features. While being easy to implement, the resulting features are often hard to interpret, especially for large values of r . Please note, that similar moment-based approaches can also be applied to the shape's region (which can be interpreted as a function of the coordinates in the image) [456]. They provide concise, robust, and computationally efficient representations. However, higher-order moments are again difficult to interpret.

Boundary moments: computing higher-order mean and variance

A third way of compressing shape signatures is given by *spectral transform* approaches, which apply a transformation from the function domain to the frequency domain, e.g., using a Fourier transformation [456]. Essentially, this corresponds to an approximation of the shape

Spectral transform: a sine-based approximation

signature function based on sine curves with different frequencies and amplitudes. Such a transformation is often simple to compute. Moreover, the resulting features can be associated with meaningful interpretations, and they tend to capture both global and local properties of the shape. Again, such transformations can also be directly applied to the two-dimensional shape region [293].

Chain code: contour as a sequence of unit-length line segments

Let us now turn to structural approaches, which treat the shape as a collection of parts. The *chain code representation* represents the contour of a shape as a sequence of unit-size line segments with a limited choice of orientations (e.g., steps of 45 degrees) [456]. This often results in a high-dimensional representation, which is quite sensitive to noise. Moreover, the starting point for describing this sequence needs to be chosen in a canonical way in order to make meaningful comparisons between shapes possible.

Polygonal representation: approximating the contour with a polygon

Instead of unit-size line segments with a limited choice of orientation, one can use line segments of arbitrary length and orientation to approximate the contour of a shape. The result is a *polygonal representation* of the shape, which ignores minor variations along the edges and thus eliminates noise [293]. The overall shape can then be represented either by the coordinates of the polygon's corners or by the LENGTH and ORIENTATION of the polygon lines. As Zhang and Lu [456] note, the polygon decomposition works well for man-made objects, but is not necessarily a good choice for natural objects, which cannot be easily approximated with polygons. Mingqiang et al. [293] point out, that a polygonal approximation of the shape's contour can also be used as a preprocessing step for filtering out noise before applying any other feature extraction approach. The polygonal approach can be further generalized by lifting the restriction to line segments and allowing for curved segments, adding CURVATURE as another parameter for describing each individual segment [456].

Convex hull: hierarchical violations of convexity

Not only the contour, but also the region of a shape can be decomposed into parts. For example, *convex hull based approaches* [456] compute the convex hull of the overall region, and detect concavities as the differences between the region and its convex hull. Each of these concavities is again approximated by its convex hull, and a second level of concavities can be detected. By recursively applying this procedure (until the remaining concavities are sufficiently small), one can thus generate a tree-like structure of concavities. Each of these concavities can be described by various parameters, including for example its AREA and the LENGTH of the line needed to "close" this concavity.

Medial axis: create a skeleton of the shape's region

Another structural approach for representing shape regions is based on its *medial axis* [456]. These medial axes can be constructed by representing the region with a set of disks, where each disk is as large as possible. One can then construct a skeleton of the region by considering the centers of these disks, which is called medial axis. Unfortunately, this computation of the medial axis is a computationally challenging problem, and the resulting representation tends to be sensitive to noise.

Please note, that the medial axis representation can be related to the proposals by Biederman [64] (Section 10.1.1), and by Marr and Sishihara [278] (Section 10.1.2), both of which consider the main axes of the respective object parts as important pieces of information.

The *adaptive grid resolution* representation of a shape's region is based on a recursive division of the image into four quadrants [293]. Each quadrant, which is partially filled by the shape, is further divided into sub-quadrants, while quadrants, which are completely filled or completely empty, are not divided further. This procedure results in a so-called *quad tree* decomposition of the shape. Two shapes can then be compared to each other by comparing their respective trees.

Finally, the *bounding box* representation starts by approximating the overall shape region with one bounding box before dividing this global bounding box into several vertical slices [293]. For each of these slices, one computes again the bounding box of the respective part of the shape region. This second-level bounding box is then split into horizontal slices, repeating the process until a sufficiently fine-granular division is reached. The overall shape region is then approximated by a hierarchically organized set of non-overlapping bounding boxes, which can be represented by their bottom left and top right corner point.

As one can see from our brief overview, a wide variety of shape features have been proposed in the computer vision community. The classification into global and structural approaches reflects the distinctions into feature-based and structural descriptions as introduced in Section 10.1.1. Many of these shape features are, however, not necessarily easy to interpret, since their main aim is to improve classification performance. Moreover, many of the shape features discussed in this section mainly aim at the direct comparison of two shapes, but do not consider a concise high-level summary of the respective input.

Recently, Morgenstern et al. [297] have proposed a similarity space for shapes based on a large set of computer vision features. Their underlying motivation was, that different shape descriptors have complementary strengths and weaknesses, and that they exhibit different sensitivities and invariances. In order to obtain a more stable sense of shape similarity, Morgenstern et al. thus proposed to combine a total number of 109 shape descriptors. They started with a dataset of 25,000 animal silhouettes, and computed the value of each shape descriptor for each image. They then normalized the individual descriptors and combined them into a single overall distance metric. Since they identified 22 clusters of related features, Morgenstern et al. then applied classical MDS (cf. Section 8.3) to the overall dissimilarity matrix for the animal silhouettes to obtain a 22-dimensional similarity space for shapes. This similarity space is called *SHAPECOMP*, and its dimensions are a weighted linear combination of the original shape descriptors.

In order to map a novel input into the *SHAPECOMP* space, Morgenstern et al. first obtained the respective values of all underlying shape descriptors. They then computed the dissimilarity of the resulting data point

Adaptive grid resolution: hierarchy of filled image quadrants

Bounding box: hierarchy of tighter and tighter boundaries

Limitations of computer vision features

A similarity space based on computer vision

Coordinates for novel stimuli

to 500 anchor stimuli, before applying classical MDS on the resulting dissimilarity matrix, obtaining again a 22-dimensional similarity space. This new similarity space was then rotated and translated in such a way that the position of the 500 anchor stimuli matched their position in SHAPCOMP as closely as possible. The coordinates of the novel stimulus were then obtained from this rotated and translated configuration. This indirect way of mapping novel stimuli into an MDS-based similarity space has already been mentioned in Chapter 8. It is applicable in this case, because the dissimilarity matrix can be computed automatically, i.e., without the need for psychological experiments (cf. Section 8.1.5).

*Psychological
plausibility*

Morgenstern et al. then used generative adversarial networks [173] (cf. Section 6.3.3) on their initial dataset of animal silhouettes to create novel stimuli. When comparing human dissimilarity ratings on these novel stimuli to the predictions obtained from the SHAPCOMP (using the above mentioned mapping procedure), they found a high correlation of $r = 0.91$, indicating, that SHAPCOMP is able to capture human shape similarity quite well.

*CNNs for a more
direct mapping*

In a final experimental step, Morgenstern et al. trained different convolutional neural networks (CNNs, cf. Section 6.2.2), which used either the two-dimensional coordinates of 384 points on the shape's contour, or a binary image with 40×40 pixels as input. These CNNs were then trained to predict the correct coordinates in the SHAPCOMP space. Despite being quite shallow with only up to three convolutional layers, and despite using a low-resolution input, the high-level activations of the trained CNNs yielded better predictions for human shape similarity than the representations of other off-the-shelf CNNs.

*Relation to our
hybrid proposal*

The work by Morgenstern et al. is certainly highly interesting in our context, since they provide a neural network, which is able to map novel stimuli into a shape similarity space, that has a good correlation with human dissimilarity judgments. However, their similarity space is rather based on an engineering approach with only limited psychological support. Moreover, they consider only animal silhouettes without any internal structure, while our own approach will deal with richer line drawings of everyday objects from various categories.

10.1.4 Shape Sensitivity of CNNs

*CNNs and human
vision*

Recent research in both cognitive psychology [318, 319] (see also Section 8.1.5) and neuroscience [106, 443] has found, that the internal representations of deep convolutional neural networks (CNNs, cf. Section 6.2.2) align surprisingly well with human representations for object classification. Some authors go as far as proposing to use such neural networks as a model of human visual processing [35, 151, 228] (cf. also the usage of CNNs in the study by Erdogan et al. [146], Section 10.1.1).

*Comparing CNNs
and humans on
silhouettes*

Kubilius et al. [235] speculated, that one reason for the good performance of CNNs is, that they implicitly learn to recognize important shape features. In a series of experiments, they investigated, whether

this hypothesis is true. In a first experiment, they used line drawings of everyday objects in three variants, namely colored, greyscale, and silhouettes. Clearly, the colored input is the richest, because the greyscale variant lacks color information, and the silhouettes also lack all texture information. On this dataset, Kubilius et al. compared the categorization accuracy of three pretrained deep convolutional networks to the one obtained by 30 human participants. Both humans and CNNs performed best on the colored images and worst on the silhouettes. While the networks were able to reach human level performance on the colored images, their performance dropped much stronger than the ones of humans when looking at greyscale or silhouette images. Nevertheless, their classification accuracy was still considerably above chance, indicating, that shape information alone is still sufficient for making reasonable classifications even though the networks had only been trained on natural images. Moreover, Kubilius observed a reasonably good correlation between human accuracy and CNN accuracy on individual items in the silhouette condition. Kubilius et al. interpreted these results as a first indication that convolutional neural networks implicitly learn shape features.

In their second experiment, Kubilius et al. looked at two different datasets. Their first dataset consisted of nine novel three-dimensional shapes, which varied with respect to their overall form (vertical, square, horizontal) and their surface structure (spiky, smoothie, cubie). The second dataset consisted of six letters from six novel font families, where the individual letters were not recognizable, but the different letters within a font family shared a common style. The dissimilarity of two shapes with respect to their physical form was in both cases computed as difference in pixels, whereas the dissimilarity with respect to their perceived shape was obtained through human ratings. For both datasets, Kubilius et al. found, that the early layers of deep neural networks correlated well with the physical form dissimilarities, while higher levels reflected perceived shape dissimilarity quite well. Several shallow models (such as pixel-wise comparisons) and the HMAX model [331] (cf. Section 10.1.1) were not able to reflect perceived shape similarity at all. According to Kubilius et al., these results show, that the shape representation used by CNNs does not only capture surface-level information, but also higher-level shape features, which are not readily available in raw pixels.

The third experiment of Kubilius et al. focused on nonaccidental properties, which we have already introduced in the context of Biederman's RBC theory [64] (cf. Section 10.1.1), and which are assumed to be invariant under possible natural variations. Typically, a change in a nonaccidental feature is more salient to humans than a change in an accidental feature, even though the pixel-based difference may be similar. For instance, the difference between an object with slightly curved edges and an object with straight edges can be expected to be quite salient (cf. also the study by Treisman and Gormican [409], Section 10.1.1). The difference between the same object with slightly curved edges and a third object with more pronounced curvature, on

*Surface level
information and
perceived shape*

*CNNs and
nonaccidental
properties*

the other hand, is expected to be less salient. On a dataset of 22 geons, Kibilius et al. found, that also for deep neural networks, modifications of nonaccidental properties cause larger differences in their internal representations than modifications of accidental properties. Again, this effect was mainly found in higher layers of the deep architectures. On the other hand, no such preference for nonaccidental properties was found for shallow models or HMAX. Kibilius et al. took these results as further support for their hypothesis, that deep neural networks learn human-like shape features.

Shape properties and semantic properties

As a motivation for their last experiment, Kibilius et al. note, that shape properties and semantic properties (such as category membership) are typically correlated, and that deep neural networks may learn shape features, because they are useful for the categorization task they are trained on. In order to investigate, whether the networks focus more on shape information or on category information, they used a dataset of photographs, where shape information and category information was as orthogonal as possible. This dataset consisted of 54 images from six semantic categories, where each category contained nine images, each of a fixed shape type (e.g., horizontally elongated). This means, that there were six semantic categories and nine shape types, and every combination of semantic category and shape type occurred exactly once in the dataset. Kibilius et al. collected human dissimilarity ratings on both shape similarity and category similarity, and used the internal representations of CNNs to predict these ratings. They found, that human shape judgments were better reflected by the CNNs than human category judgments. Nevertheless, the CNNs were able to capture at least some of the semantic structure, while shallow and HMAX models were only able to capture shape information. Kibilius et al. interpreted these results as an indication, that deep neural networks capture shape similarity rather than conceptual similarity in their representations.

Lessons learned

Overall, the investigations by Kibilius et al. provide evidence, that the internal representations of deep neural networks seem to be mainly shape-based. Even though these networks are typically trained on classifying images into different categories, they develop internal representations, which reflect the subjective shape judgments of humans. Since this was not observed for any of the simpler models under consideration, Kibilius et al. conclude, that the ability to represent shape information in a human-like way has been critical for the great success of deep convolutional networks. Moreover, they highlight, that the transformation of physical features in the early layers of the networks into higher-level perceptual features in the later layers corresponds to the different pieces of information processed in the early visual cortex and higher visual areas in the human brain.

Shape information and adversarial examples

Also Baker et al. [24] have used the human sensitivity to shape perception as a way to compare the internal operation of pretrained deep convolutional neural networks to the way, in which the human visual system works. They note, that the results by Kibilius et al. [235] indicate a certain shape sensitivity in CNNs, but that adversarial

examples [401] on the other hand drive the network to making incorrect classifications, while leaving the global shape information of an image intact. In their paper, Baker et al. conduct a series of five experiments on pretrained CNNs, investigating to which extent they make use of shape information for determining their classification decision.

In their first experiment, Baker et al. created forty example images by filling the silhouette of one object with the texture of another object. No background or other context was contained in the image. The purpose of this experiment was to evaluate, whether the CNN would predict the class describing the *SHAPE* of the input or the class describing its *TEXTURE*. The pretrained CNN used by Baker et al. (VGG-19 [373], which consists of 19 layers) predicted the correct *SHAPE* in eight cases and the correct *TEXTURE* in seven cases among its top five predictions, which corresponds to a relatively poor performance level. It seems, that neither *SHAPE* nor *TEXTURE* was a main driver for classification. Baker et al. furthermore observed, that object *SHAPE* was predicted more frequently for artifacts than for animals, while *TEXTURE* was predicted more often for animals than for artifacts.

Conflicting shapes and textures

In their next two experiments, Baker et al. used input images without texture information. Their second experiment used ten images of glass figurines, and their third experiment used forty outline drawings. In both experiments, Baker et al. assumed, that a correct classification can be made based on the shape of the object even in the absence of a texture. However, they found, that in a large majority of the cases, the two CNNs under consideration (AlexNet [232], an eight-layer architecture, and VGG-19) predicted the correct class with a probability below 0.1%. This corresponds to performance below the chance level, since the CNNs had 1,000 class output units. It thus seems, that the neural networks were not able to pick up on the cues provided by the shape of the object. However, both glass figures and outline drawings are rather an edge case in perception.

Glass figurines and outline drawings

Next, Baker et al. attempted to replicate the results reported by Kubilius et al. [235] for silhouettes. They used the images from their first experiment, but rather than filling the silhouettes with a non-matching texture, they filled them with a uniform black color. Baker et al. were able to replicate the findings by Kubilius et al., namely, that the CNNs were able to report the correct class in their top 5 predictions in 15 or 20 out of 40 cases (AlexNet and VGG-19, respectively). Even though no texture information was present, the neural networks were still able to make some meaningful predictions. However, Baker et al. noticed, that black silhouettes on white ground worked considerably better than white silhouettes on black ground or red silhouettes on white ground. Baker et al. explain the difference in performance between this experiment and their prior experiments on outlines and glass figures by the clearer separation between figure and ground and the complete absence of texture. However, since classification performance was still considerably worse on silhouettes than on photographs (in contrast to human performance, which remains at a very high level), Baker et al. hypothesize, that unlike humans, CNNs do not consider the *global*

Considering silhouettes

shape structure, but rather focus on *local* shape features when making their classification.

Global shape structure vs. local shape features

In order to test this hypothesis, they conducted a fifth and final experiment. In this experiment, they selected six silhouettes, which were correctly classified by VGG-19 (when considering its top 5 predictions), and they applied two kinds of modifications to these silhouettes: In the first condition, they scrambled the global shape, while leaving local shape features intact by re-arranging the parts of the objects. In the second condition, they kept the global shape structure intact, but added jagged edges to the silhouette's bounding contour, resulting in a saw-tooth effect and presumably scrambled local shape features. Baker et al. observed, that VGG-19 was able to correctly classify five of the six silhouettes with re-arranged parts, while it was unable to predict the correct class for any of the silhouettes with jagged edges. In a small study with human participants, they, however, found the reverse effect of essentially unchanged performance for scrambled local features, but a considerable performance drop for missing global shape structure.

CNNs only consider local shape information

Based on their experiments, Baker et al. conclude, that CNNs, which have been trained for object recognition on ImageNet, do not make explicit use of shape information in making their predictions. There seems, however, to be a limited implicit use of local shape features with respect to the contours of the object. Humans, on the other hand, seem to rely much heavier on shape information in their visual processing, focusing especially on the overall arrangement of parts. Baker et al. point out, that while their results indicate, that the pretrained networks were not sensitive to shape, this does not imply, that CNNs are in principle incapable of learning shape information. As we will see in Chapter 12, there are CNN architectures, which reach human-level performance in sketch recognition tasks. When classifying sketches, also no texture or context information is available – this task is thus quite similar to the outline drawings considered by Baker et al. in their third experiment.

Modified training data can weaken the texture bias

Also Geirhos et al. [163] came to similar conclusions: When using artificial images, where the texture and shape of two different objects were combined (e.g., the texture of an elephant's skin with the shape of a cat, similar to the first experiment by Baker et al. [24]), CNNs classified the resulting image mostly based on TEXTURE information, while humans showed a stronger bias towards SHAPE. Geirhos et al. then proposed to transfer the photographs from ImageNet to different randomly selected texture styles, resulting in a dataset called "Stylized-ImageNet". In this dataset, texture is no longer a relevant clue for classification. Therefore, CNNs trained on Stylized-ImageNet can be expected to rely heavier on shape information. Interestingly, Geirhos et al. found, that this also makes CNNs more robust to other types of image distortions. The work by Geirhos et al. again highlights, that regular photograph-based CNNs tend to base their classification decisions mainly on texture information. However, their work also shows, that this texture bias can be avoided by careful selection of the training data.

More recently, Singer et al. [374, 375] have investigated the ability of convolutional neural networks to generalize to higher levels of visual abstraction, namely, from photographs to line drawings and sketches. They used the VGG-16 architecture [373] and a small dataset of 42 objects, which were available as photographs (where objects were cropped from their background), line drawings by a professional artist (leaving contours intact, but strongly altering color and texture), and sketches (containing even less detail and stronger distortions of the object's contour, e.g., by exaggerating some features).

*Comparing
photographs, line
drawings, and
sketches*

Singer et al. used the classification accuracy of the trained network on their small dataset as means for evaluation. This classification accuracy was compared to human performance in a labeling task (i.e., giving a single-word label for a given image). Moreover, Singer et al. trained a linear support vector machine (cf. Section 5.2.3) on the CNN's internal representation at varying levels of depth to distinguish man-made from natural objects. Finally, they employed *representational similarity analysis* (RSA) [230] by computing the correlation between the dissimilarity matrices extracted from the CNN's internal representation and human dissimilarity ratings. This essentially corresponds to the ANN baseline considered in our analysis setup for the NOUN dataset in Chapter 9. Singer et al. used Pearson's r as a distance metric to convert activation patterns of the CNN into distances (yielding one separate dissimilarity matrix for each input type), while the corresponding human data was obtained using the triad method (cf. Section 8.1.1), where participants were asked to select the "odd one out" among a set of three stimuli. Human and CNN-based dissimilarities were then compared using Spearman's ρ .

*General experimental
setup*

In their first experiment, Singer et al. pretrained their CNN on ImageNet [120], before evaluating its performance with respect to their dataset. They reported a high accuracy on photographs, but poor performance on both line drawings and sketches, while humans performed almost equally well in all three conditions. The RSA showed a high correlation between photographs and drawings for the lower layers of the network, which, however, decreased in later layers. A similar pattern was observed for the correlation between photographs and sketches (albeit at a lower correlation level), while the representational similarity between line drawings and sketches also improved, but stayed then relatively constant in higher layers. This latter observation together with the fact, that the linear SVM was still able to separate man-made from natural objects for higher network layers in all conditions, indicates, that the observed effects are not based on a representational collapse (where all non-photographs would be represented by the same activation pattern). Overall, Singer et al. thus conclude, that the general object representation at lower and intermediate layers seems to be similar across all three conditions, but that this similarity starts to fade as the link to object categories becomes stronger in later layers, hinting at a bias towards natural images.

*Pretraining on
photographs
generalizes poorly*

In order to investigate, whether this drop in representational similarity is based on the texture bias, Singer et al. trained another VGG-16

*Removing the texture
bias with different
training data*

architecture on Stylized-ImageNet [163] (cf. our discussion from above). Since Stylized-ImageNet contains different types of textures for each target class, the trained network was expected to be less biased towards texture, and hence more reliant on shape. Singer et al. found similar performance on photographs and improved performance on line drawings, while the classification accuracy on sketches remained low. Overall, pretraining without a texture bias considerably improved the correlation between photographs and line drawings for later layers of the network. Also the correlation between photographs and sketches increased, while the drawing-sketch correlation remained on a similar level as in the first experiment. Overall, Singer et al. thus confirmed the results by Geirhos et al. [163]: reducing shape bias improved generalization from photographs to other types of images. Moreover, training on Stylized-ImageNet increased the correlation between human and CNN-based dissimilarities.

*Transfer learning via
fine-tuning*

In their third experiment, Singer et al. now considered the photograph-based CNN and fine-tuned the later layers of the network on the ImageNet-Sketch dataset [424], which contains about 50 line drawings for each of the 1,000 classes from ImageNet. Again, they found, that both the classification accuracy on the line drawings and the correlation of representations between the photograph and the line drawing inputs improved. Classification performance on sketches remained poor, but the representational correlation between sketches and photographs increased as well. Interestingly, the correlation between drawings and sketches was reduced through fine-tuning. Overall, the correlation between CNN-based and human dissimilarities was reduced for photographs, but improved for both line drawings and sketches.

Lessons learned

Taken together, the results by Singer et al. highlight again, that CNNs, which are trained on photographs, exhibit a bias towards texture, which makes generalization to line drawings and sketches difficult. The internal representation of shape similarity is typically limited to lower layers of the network, but both training on augmented data (such as Stylized-ImageNet) and fine-tuning on line drawings (such as ImageNet-Sketch) can help to reduce this texture bias, bringing CNNs closer to the shape-based perception postulated for humans.

*Comparing CNNs on
photographs and
sketches*

Also Fan et al. [151] have considered CNNs in the context of sketches, using the architecture provided by Yamins et al. [443] as a model of the ventral visual stream in the human brain. As a dataset for their study, they used a subset of 105 classes from the TU Berlin sketch corpus [143] (which will be introduced in more detail in Chapter 12), along with photographs of the corresponding classes from ImageNet [120]. In addition to relational similarity analysis (RSA, see above) [230] based on Spearman's ρ between the dissimilarity matrices for sketches and photographs, Fan et al. also trained a support vector machine (cf. Section 5.2.3) on the representation of each network layer in order to classify the input image into its correct class. The RSA showed larger correlations in higher network layers than in lower layers, and also the SVM was more successful in its classification task when using the

representation of later network layers. Based on a recognition study with human participants on sketches, Fan et al. also found a high correlation between human decisions and the network's predictions. It thus seems, that the perception of both photographs sketches uses the same underlying representation in both humans and CNNs.

Fan et al. then investigated the effect of sketching practice by training human participants to sketch objects of a given class, and by observing the effects on classification performance and sketch similarity. They found, that sketches improved with practice as indicated by a slightly improved classification accuracy of their machine learning system. This effect, however, only held for the specific classes, on which participants were trained, and generalized neither to related nor to unrelated classes. Moreover, Fan et al. observed, that later drawings of the same category were more similar to one another than earlier drawings (as judged by the CNN-based dissimilarity matrix), and that the distance of the trained classes to other classes in the CNN's representation increased. It is important to emphasize, that the CNN was not modified at all in this experimental setup, so the focus was rather on the inputs generated by the trained human participants. By comparing the results of this experiment to two control conditions (where participants did not draw the sketches, but either viewed finished drawings, or observed the stroke dynamics of a sketch), Fan et al. showed, that this training effect is based on active sketching practice rather than the mere exposure to additional sketches.

Fan et al. furthermore hypothesized, that the same internal representation for both photographs and sketches is not only used for recognition, but also for production. Since the results of the previous experiment could also be explained by improved motor control, they conducted a final experiment, where participants underwent the same procedure of repeated sketching as before, but were this time tested in a sketch recognition task both before and after training. This task used morphed versions of selected stimuli and involved a speeded forced choice between the two endpoints of the underlying morph. By modeling the decisions of the participants with a logistic regression function, Fan et al. were able to compare the slope of the fitted function before and after training. They reported, that the slope increased for objects from the trained categories, which they interpret as sketching practice improving recognition. No such effects were observed for control objects from other classes, or for observing stroke dynamics instead of drawing the sketch.

The study by Fan et al. confirms the observations from the other studies reported above by establishing a link between the representation of rich photographs and more abstract sketches, thus linking CNNs to shape perception. While other studies have focused on the circumstances, under which shape-based recognition can work with CNNs, Fan et al. have highlighted, that the same type of representation seems to be used both for recognition (i.e., classifying a photograph or a sketch) and production (i.e., creating a sketch).

The effects of sketching practice

Recognition and production

Lessons learned

*Human sensitivity to
geometric
regularities*

Most of the studies summarized above focus on the perception of shapes in the form of outlines, silhouettes, line drawings, or sketches. Recently, Sablé-Meyer et al. [343] have investigated, to which extent CNNs can account for human perception of geometric regularities in shapes. They first established, that human shape perception is sensitive to geometric regularities (namely, parallel lines, equal sides, equal angles, and right angles) by using an outlier detection task: They started with eleven reference shapes, which were all quadrilaterals, i.e., closed polygons with four vertices and four edges, and which varied in their regularity. In each trial, they generated five instances of one reference shape, varying in scale and orientation. They then added one deviant shape, where the bottom right vertex was displaced in such a way, that at least one of the geometric regularities did no longer hold. The participants of their study were then presented with the overall set of six stimuli and had to click on the outlier as fast and accurately as possible. This experiment was carried out with four different groups of participants, namely, French adults, French kindergardeners, Himba adults, and baboons (a nonhuman primate).

*Experimental results
for adults*

The performance of French adults could be well predicted by the number of geometrical regularities in the reference shape. A statistical analysis of the data revealed a main effect of shape (as specified by the four geometric regularities), while size, rotation, and position of the target stimulus had only minor or insignificant effects. Also subjective ratings of shape regularity correlated well with error rates. Sablé-Meyer et al. also noted, that there was no pre-attentive pop-out effect (cf. the study by Treisman and Gormican [409], Section 10.1.1), indicating, that geometric regularities require attentive processing.

*Comparing to other
groups*

In order to validate, whether the observed sensitivity to geometric regularities was based on mathematical education, Sablé-Meyer et al. also collected data for kindergardeners and Himba adults (a pastoral people of northern Namibia). For both control groups, the geometric regularity effect could be replicated (although to a somewhat lesser degree), indicating, that it seems to relate to an universal intuition in humans. Sablé-Meyer et al. then also trained baboons (a nonhuman primate with a visual system similar to humans) on the outlier task. The baboons were able to understand the overall task when prompted with other stimuli, but their performance collapsed, when the shape stimuli from the human study were used. Moreover, they did not show any regularity effect, leading Sablé-Meyer et al. to conclude, that detecting geometrical regularities seems to be a unique capability of humans.

*Modeling the
regularity effect with
CNNs*

Finally, Sablé-Meyer et al. attempted to find a model of human visual perception, which predicts the observed regularity effect. They probed different pretrained CNNs by obtaining their hidden representation for the six input stimuli at various layers in the network, and by then selecting the input, whose activation different most strongly from the overall mean. As an alternative to this CNN-based model, they assumed a symbolic description of quadrilaterals, where each input was represented by the presence or absence of the four geometric regularities (encoded as a vector of zeros and ones), and where the Manhattan

distance was used to detect an outlier. When comparing the two models on the psychological data, Sablé-Meyer et al. found, that the CNN-based approach predicted the choices of baboons well, but failed to do so for humans, while the symbolic model gave better predictions on the French adults, but could not account for the baboons' behavior. For both kindergardeners and Himba adults, a combination of both models yielded the best approximation of the experimental data. Sablé-Meyer et al. also report, that neither using different network architectures nor employing additional fine-tuning with a shape classification task improved the predictive power of the CNN-based model.

The study by Sablé-Meyer et al. differs from most other studies reported in this section by putting its focus on abstract geometric shapes and by conducting extensive psychological experiments. It can thus also be related to the studies by Treisman and Gormican [409] and Erdogan et al. [146], which have been discussed in Section 10.1.1. Its important contribution to the discussion of shape sensitivity in CNNs lies in the observation, that some empirical effects in human shape perception, such as the sensitivity to geometric regularity, are not captured by the representations learned by CNNs.

Lessons learned

10.2 OUR NEW DATASET

In order to extract conceptual similarity spaces for the SHAPE domain in a data-driven way, we have created a new dataset based on psychological experiments. We have conducted two series of behavioral experiments to investigate, how humans perceive two-dimensionally simplified, but compositionally complex shapes of everyday objects. Both series of experiments employed the same set of line drawings as stimuli. In Section 10.2.1, we describe the visual stimuli used in our study, and how they were selected.

A dataset based on line drawings

In a first series of experiments (Section 10.2.2), we collected the subjectively perceived degree of dissimilarity between two complex shapes. We investigated both general conceptual dissimilarity and shape-based visual dissimilarity.

Dissimilarity ratings

Our second series of experiments (Section 10.2.3) investigated, how primitive shape features of the complex shapes were perceived. We focused on three features, which are supposed to be critical for shape perception, namely the object's overall FORM (elongated vs. blob-like), its overall ORIENTATION (horizontal vs. diagonal vs. vertical), and the shape of its LINES (straight vs. curved).

Shape features

10.2.1 Stimuli

The set of complex shapes employed in the behavioral studies was taken from a prior study by Scheibel and Indefrey [349]. Unfortunately, copyright permissions of these images do not allow us to reprint them

The line drawings

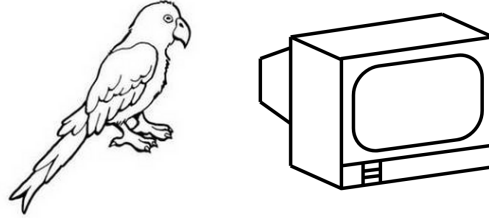


Figure 10.4: Example line drawings (PARROT and TELEVISION) similar to the ones used in our study.

Visual Type	Object Type	Category	Items
variable	artificial	DISHES	BOWL, GLASS, PLATE, TEA POT, SALT SHAKER
		BUILDINGS	CHURCH, IGLOO, LIGHTHOUSE, PYRAMID, WINDMILL
		VEHICLES	BICYCLE, BUS, CAR, SCOOTER, TRACTOR
	natural	INSECTS	ANT, BUTTERFLY, DRAGONFLY, FLY, LADYBUG
		FRUITS	APPLE, BANANA, CHERRY, LEMON, RASPBERRY
	VEGETABLES	ASPARAGUS, BELL PEPPER, BROCCOLI, CARROT, ONION	
coherent	artificial	TOOLS	AXE, HAMMER, PLIERS, SAW, SHOVEL
		APPLIANCES	DISHWASHER, MICROWAVE, TELEVISION, TOASTER, WASHING MACHINE
	natural	CLOTHES	BLOUSE, COAT, JACKET, SHIRT, UNIFORM
		PLANTS	DANDELION, PALM TREE, ROSE, SUNFLOWER, TULIP
		UNGULATES	COW, DONKEY, GOAT, HORSE, STAG
	BIRDS	EAGLE, OWL, PARROT, PIGEON, STORK	

Table 10.1: Overview of the 60 stimuli in our dataset, based on their categories.

in this book. They are, however, available upon request from the author. Two exemplary line drawings are shown in Figure 10.4. Our stimulus set contains 60 standardized images of line drawings, showing the shape of everyday objects in a simplified way. The original images were taken from different sources and adjusted, such that they matched in relative object size as well as object position and object orientation. Drawings containing too many details were additionally simplified. All drawings were fitted onto a 283 x 283 pixel white background and converted into grayscale images. The standardized images have a horizontal and vertical resolutions of 96 dpi, 32 bit depth. All of the reported image modifications were done by Scheibel and Indefrey.

The depicted objects belong to twelve different semantic categories. Half of these categories are labeled as visually coherent (VC; i.e., based on objects with similar shapes), and the other half as visually variable (VV; i.e., containing visually heterogeneous examples).¹¹ Half of the categories of each visual type were categories of natural objects and the other half categories of artificial, man-made objects. Table 10.1 gives an overview of the different categories, their labels, and their associated stimuli.

Category structure

Overall, this stimulus set fulfills several desiderata: The images show complex shapes, composed of different types of elementary components (e.g. straight and curved lines) and different types of shape components (e.g. elongated and round components).

Complexity

The depicted objects are familiar to adult humans of central Europe, to which the participants of the studies belonged (pre-tested by Scheibel and Indefrey [349]). We therefore assume, that evaluations on the shapes are based on stored shape knowledge rather than ad-hoc criteria.

Familiarity

Visual information available in the images is as strongly as possible restricted to SHAPE information. There is for example no COLOR or TEXTURE information, and no information about the real SIZE of the object. We therefore assume, that confounding influences from knowledge structures of other visual domains like COLOR, SIZE, or TEXTURE are sufficiently eliminated.

Focus on SHAPE

Moreover, the usage of line drawings makes it easy to relate our work to earlier studies on shape perception, which also typically focused on similar types of stimuli. This includes for example the RBC model [64] and the early vision study by Treisman and Gromican [409] (both Section 10.1.1), but also the machine learning experiments by Singer et al. [375] (cf. Section 10.1.4).

Relation to earlier work

10.2.2 Elicitation of Dissimilarity Ratings

In order to construct a similarity space of the SHAPE domain, we are interested in dissimilarity ratings with respect to visual similarity. However, we also need to ensure, that these ratings indeed refer to

Visual and conceptual similarity

¹¹ More details on the classification of the visual type of categories are given by Scheibel and Indefrey [349]

visual similarity and not to general conceptual similarity. Therefore, we collected not only visual dissimilarity ratings, but also conceptual dissimilarity ratings. In Section 10.3, we will compare these two sets of dissimilarity ratings and investigate, whether and to which extent they differ from each other.

*Ratings from
Scheibel and Indefrey*

Both rating studies were conducted by Scheibel and Indefrey [349] as pre-tests to control the material selection in their study. We reused their raw ratings (which they did not thoroughly analyze in their paper) as part of our dataset. In the following, we describe how these ratings were originally collected.

*Pairwise
comparisons*

The procedure of the two rating studies was identical. In both experiments, a direct rating method was used by considering pairwise comparisons with Likert scales (cf. Section 8.1.1). This method was selected, because it is applicable to both perceptual and conceptual similarity, and because it has a long tradition in psychology.

Image pairs

The 60 stimuli as described in Section 10.2.1 were combined to image pairs, such that all pairwise combinations of the images were obtained (in total 1770). In order to limit the length of the experimental sessions and to avoid fatigue effects, each participant only viewed a subset of the image pairs: The overall set of images was partitioned into four lists containing 442 or 443 image pairs each. Scheibel and Indefrey controlled, that all lists included a similar proportion of within- and between-category pairs, and that no category biases occurred. All lists were tested with an equal number of participants. The order of the image pairs was randomized for each participant individually. Image pairs were horizontally aligned, and the individual images were randomly assigned to the left or right position.

Technical platform

The rating study was conducted as web-based survey using SoSci Survey [250]. The link to the study was published on various online and offline platforms for students of the Heinrich Heine University (HHU) in Düsseldorf, Germany. The study was conducted in German.

*Experimental
instructions*

At the beginning of the survey, participants saw an introduction to the procedure and task, including two examples of picture pairs and possible ratings for these examples. For the elicitation of visual dissimilarities, it was stressed that only the visual similarity of the depicted shapes had to be judged (ignoring everything else they might know about the stimuli). For the experiment targeting conceptual dissimilarity ratings, the focus on semantic-conceptual similarity was emphasized. In both experiments, fast and spontaneous ratings were encouraged. Participants started the test phase individually by clicking on the "start" button.

User interface

In the test phase, the image pairs appeared automatically one after another at the center of the screen. A Likert scale with five points was displayed below the image pair on the bottom of the screen. For the elicitation of visual dissimilarities, it ranged from "totally dissimilar shapes" (numerically coded as 1) to "very similar shapes" (numerically coded as 5). In the experiment targeting conceptual dissimilarity, the endpoints were labeled as "conceptually similar / same category" (nu-

merically coded as 1) and "conceptually different / different categories" (numerically coded as 5).

Once a scale point was selected, the image pair disappeared from the screen and the next trial began. There was no possibility to change previous judgments. The study did not include a "back" button, and clicking the "back" button of the browser terminated the study. There was also no possibility to skip a trial. The study did not include a "next trial" button during the automatic part of the test phase – each image pair remained on the screen, until a scale point was selected. After a block of 21 to 23 trials, a break was offered. The complete survey lasted about 10 minutes on average.

Experimental workflow

62 students of HHU participated in the study on visual similarity. The data collection was fully anonymous, so no information about the age or sex of the participants were collected.

Visual similarity

48 students of HHU took part in the study on conceptual similarity. Two participants were non-native German speakers. Their data was discarded, because linguistic influences on the organization of semantic knowledge structures cannot be ruled out. The 46 native German speakers (32 female, 14 male) had a mean age of 25.26 years.

Conceptual similarity

10.2.3 Elicitation of Feature Ratings

As we have seen in Section 10.1, there exists a large variety of approaches for describing shape perception. For our study, we have selected three perceptual features as promising candidates for the underlying dimensions of a similarity space for the SHAPE domain.

Three perceptual features

The FORM feature refers to the global shape structure and distinguishes elongated from blob-like objects. This can be related to coarse-to-fine accounts such as Bar's model [29], to the HEIGHT-TO-WIDTH RATIO investigated by Treisman and Gormican [409] and Ons et al. [307], and to the SHAPE ENVELOPE considered by Op de Beeck et al. [308] (cf. Section 10.1.1). It is also implicitly coded in the LENGTH and DIAMETER of cylinders in Marr and Sishihara's proposal [278], and in the size parameters of the superquadrics considered by Chella et al. [97] (cf. Section 10.1.2). Also the ECCENTRICITY feature used in computer vision (cf. Section 10.1.3) is related to the FORM feature.

FORM: elongated vs. blob-like

The LINES feature describes, whether the lines of the object are mostly straight or mostly curved. This feature is presumed to be an important nonaccidental property in the RBC model [64] and has also been identified as an early vision feature by Treisman and Gormican [409], Op de Beeck et al. [308], and Ons et al. [307] (cf. Section 10.1.1). Moreover, it can be related to the form factors of the superquadrics considered by Chella et al. [97], which control the curvature of the object's surface (cf. Section 10.1.2). Finally, it is similar to the AVERAGE BENDING ENERGY considered in computer vision (cf. Section 10.1.3).

LINES: straight vs. curved

The ORIENTATION of the lines describes an object as having a horizontal, vertical, or diagonal orientation. This feature has been explicitly investi-

ORIENTATION: horizontal vs. diagonal vs. vertical

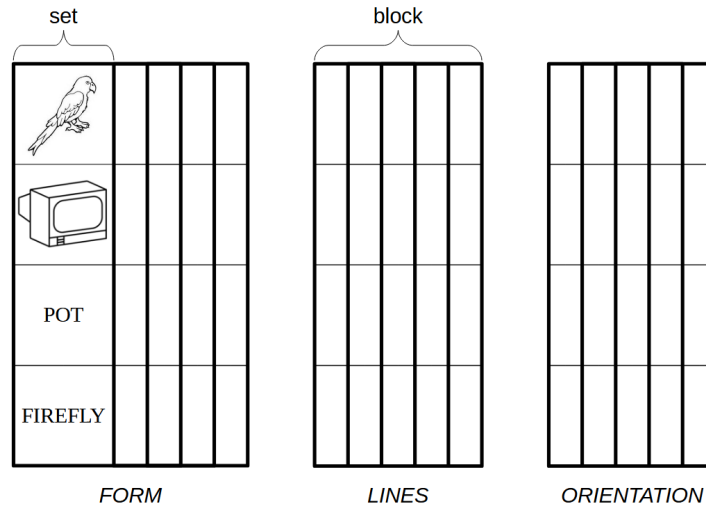


Figure 10.5: Trial structure for eliciting attentive feature ratings.

gated by Treisman and Gormican [409] (cf. Section 10.1.1). Moreover, the overall orientation of object parts (and thus their main lines) is explicitly coded in many structural approaches such as the models by Marr and Sishihara [278], or the system by Chella et al. [97] (cf. Section 10.1.2). Also neuroscientific findings (e.g., by Hubel and Wiesel [203]) and theories (such as HMAX [331]) emphasize, that individual neurons (especially in early visual processing) are sensitive to specific stimulus orientations (cf. Section 10.1.1).

*Attentive and
pre-attentive
perception*

According to the current understanding of visual perception, most of these primitive shape features are perceived very early, i.e. pre-attentively [409] (cf. Section 10.1.1). To test, whether the duration of perception changes the perceived main values of the primitive features, we investigated both attentive and pre-attentive perception by varying the viewing duration.

*Eliciting attentive
feature ratings*

Our first experiment investigated the *attentive* perception of the primitive shape features. Each primitive shape feature was considered separately. Overall, 27 students (15 female, 12 male, mean age = 24.07; all but one were native German speakers) of the Heinrich Heine University HHU in Düsseldorf, Germany, participated in the study. All had normal or corrected to normal vision. Attentive feature ratings were elicited for the 60 stimuli described in Section 10.2.1. Moreover, 24 additional filler items were used in warm-up trials and for demonstrations of the tasks.

Image sets

Figure 10.5 illustrates the trial structure of the experiment: The overall set of 60 images was partitioned into 15 sets of four images each. All image sets were categorically heterogeneous sets, i.e. each item belonged to a distinct semantic category, two of them were visually variable (VV) and two visually coherent (VC) categories. Half of the items of each set were artificial objects, the other half natural objects. Visual biases within image sets to a particular value of one of the features were avoided.

The image sets were then organized into three blocks with five image sets per block. Each block was assigned to one of the the three different features. Each participant therefore performed each rating task only on a small number of trials, which enhanced the sensitivity of the respective ratings and avoided image repetitions. The order of blocks, and thus also the order of tasks, was counterbalanced across participants by using a Latin square design¹². The order of image sets within a block was randomized for each participant individually. In order to avoid grouping effects, three different assignments of stimuli to sets were created, which were tested with an equal number of participants.

Organization into blocks

Subjects were tested in small groups of up to 15 people in a computer lab at HHU. The study was conducted with SoSci Survey software [250] and as web-based survey. The participants received individual passwords to the study to control the assignments of lists to participants. The study was conducted in German.

Technical platform

At the beginning, participants were introduced to the user interface of the software and the different rating tasks. The procedure of each rating task was demonstrated with an example image set, including an explicit introduction to the different rating scales. Participants were asked to ignore everything but the to-be-rated primitive shape feature of the images. Spontaneous ratings were encouraged. Participants started the test phase individually by clicking on a "start" button.

Experimental instructions

The test phase consisted of three parts, one for each primitive shape feature. Each part started with a reminder of the specific rating task and rating scale, including an example arrangement of example images. The first trial of each part was always a warm-up filler trial, then the five experimental trials were carried out. Between the experimental parts, a short break was offered.

Experimental structure

The rating scale and the four images of the current set were displayed on the screen. The images initially appeared at the bottom, below a continuous scale with pictorially and verbally labeled endpoints (FORM: "absolutely elongated" to "absolutely blob-like" ; LINES: "absolutely straight" to "absolutely curved"; ORIENTATION: "absolutely horizontal" through "diagonal in any angle" to "absolutely vertical").

User interface

The task consisted in arranging the images on the scale per mouse click, such that the final positions gave an accurate reflection of the feature values of each item. Positions could be corrected during a trial, but there was no possibility to change ratings from previous trials.¹³ There was also no possibility to skip a trial, because the "next trial" button only became active, when all images of a set were arranged on the scale. When participants clicked on the "next trial" button, the current arrangement of items was stored. The horizontal position of the items on the rating scale was mapped to a numerical value in the interval $[-1, 1]$. The scale ends corresponded to the borders of the

Task and workflow

¹² This means, that we ensured, that each block appeared in each position equally often when considering all participants.

¹³ The study did not included a "back" button, and clicking the "back" button of the browser terminated the study.

interval and a linear mapping was used for intermediate positions. The vertical position of the items was discarded. The complete session lasted about 10 minutes on average.

Post-processing

For subsequent analyses, the ratings (in total 9 ratings per item for each primitive shape feature) were aggregated for each item and each feature separately by using the mean. The 15 images with the highest average values were selected as examples for the positive endpoint of the respective scale, while the 15 images with the lowest average values were selected as examples for the negative endpoint of this scale.

*Eliciting
pre-attentive feature
ratings*

In a second line of experiments, we collected image-specific feature values which are based on *pre-attentive* image perception. We used an extremely short view duration of 50 milliseconds, after which participants had to select the feature value they believed to have seen. We conducted one separate experiment for each feature.

Participants

Each experiment was carried out with 18 subjects (no double participation).¹⁴ All participants were students at HHU, native German speakers, and had normal or corrected to normal vision. The FORM experiment had 16 female and 2 male participants with a mean age of 22.72. For the ORIENTATION experiment, we had 12 female and 6 male subjects with a mean age of 22.11. Finally, 12 female and 6 male subjects with a mean age of 22.28 participated in the LINES experiment.

Stimuli

In addition to the 60 images as described in Section 10.2.1, we used 6 filler items for a practice block. Each test shape was presented once to each participant. The order of images was pseudo-randomized for each participant individually.¹⁵

Technical setup

Subjects were tested in a sound-attenuated, dimly lit cabin. The distance to computer screen was approximately 60 cm. The experiment was controlled by Presentation Software (version 18.3). The stimuli were presented on a BenQ-monitor (XL2430) with a screen resolution of 1920 x 1080 pixel and a vertical refresh rate of 60 Hz. The study was conducted in German.

*Experimental
instructions*

At the beginning, participants received written instructions about the procedure and the task. It was stressed, that the task was not about shape identification, but only about a subjective perception of the respective primitive shape feature. A practice block of six trials had to be carried out by each participant to ensure, that the experimental procedure and task were understood and performed correctly.

Trial structure

Participants started each trial individually by pressing the "next trial" button. The structure of a trial is illustrated in Figure 10.6. All stimuli presented in a trial were presented at the center of the screen on a black screen background.

¹⁴ In two experiments, some subjects performed the tasks not correctly, e.g. pressed only one button, or too frequently the "no idea" button. The data of these subjects was discarded (ORIENTATION experiment: 4; LINES experiment: 1). The respective lists were tested twice in order to obtain an equal number of data for each item in all studies.

¹⁵ Maximally three items of the same category type (combination of visual type (VV vs. VC) and object type (artificial vs. natural)) appeared in successive trials. The same semantic category never appeared in successive trials.

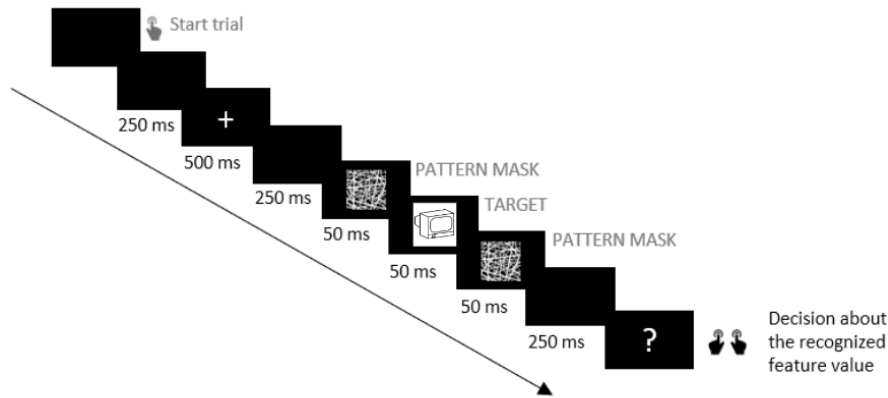


Figure 10.6: Trial structure in the masked recognition experiments for eliciting pre-attentive feature ratings.

The task was to decide as fast as possible by pressing a button, which value of the respective primitive shape features pertained to the target shape mostly. For the `FORM` feature, there were two response options, namely, "elongated" (numerically coded as -1) and "blob-like" (numerically coded as 1). Also the `LINES` experiment featured two response options, namely, "straight" (numerically coded -1) and "curved" (numerically coded as 1). The experiment targeting the `ORIENTATION` feature used four response options, namely "horizontal" (numerically coded as -1), "vertical" (numerically coded as 1), "diagonal from bottom-left to top-right", and "diagonal from top-left to bottom-right" (both numerically coded as 0).¹⁶ In all experiments an additional "no idea" response button (numerically coded as 0) was provided. The assignment of feature values to buttons were indicated on the respective keys of the button box and pictorially and verbally displayed on the screen whenever a button press was required. A testing session lasted about 4 minutes on average.

Task and workflow

For each primitive shape feature, the categorical responses (in total 18 per item, each numerically coded with -1, 0 or 1) were aggregated per item by using the arithmetic mean. Again, we took the 15 images with the highest values as examples for the positive scale end, while storing the 15 images with the lowest values as examples for the negative end of the respective scale.

Post-processing

Please note, that we have made a considerable simplification with respect to the perceptual feature of `ORIENTATION` by representing it as a scale with two endpoints in both types of feature rating experiments. However, orientation can be thought of as rotation around the object's center. It would thus be more natural to represent this feature as a circular dimension (e.g., by using the rotation angle). We nevertheless

A simplified notion of ORIENTATION

¹⁶ Although we distinguished between two senses of "diagonal", we have coded them identically. This was done, because the number of responses for the second variant was very low (only about 7%), and because all images in our stimulus set seemed to fall into the first category (cf. Table 10.1).

decided to use a linear scale, since this allows for easier analysis and increased comparability to the other two features under consideration. One can furthermore argue, that the two endpoints of our scale (horizontal and vertical) correspond to opposite positions on a circular dimension. By looking only at the horizontal-vertical contrast, we thus consider a subset of the information about an object's orientation.

10.3 ANALYSIS OF THE DATASET

Overview

In this section, we perform some analyses on the raw information from our dataset in order to ensure, that it is a good basis for a shape similarity space. First, in Section 10.3.1, we introduce the statistical tools used for making some of the subsequent analyses. In Section 10.3.2, we then compare the visual dissimilarity ratings to the conceptual dissimilarity ratings, taking into account also the distinction into visually coherent and visually variable categories. This is done to ensure, that the visual dissimilarity ratings elicited in our psychological experiments indeed refer to the SHAPE domain and not to general conceptual similarity (which may involve multiple domains). After this, we analyze the three psychological shape features in Section 10.3.3, where we compare the pre-attentive to the attentive ratings, and where we also investigate correlations between different features.

10.3.1 Statistical Tools

Our toolbox

In this section, we provide an overview of several statistical tools, which will be used in Sections 10.3.2 and 10.3.3 to analyze our dataset. This includes general statistical terminology, Student's t-test, cumulative link mixture models (CLMMs), and confidence intervals for correlation coefficients. Large parts of this section are based on the textbook by Field et al. [155].

General statistical terminology

When analyzing the results of a typical psychological experiment, researchers are usually interested in finding evidence for or against a specific prediction of a scientific theory. The prediction under investigation is usually called the *alternative hypothesis*. It states, that a given experimental intervention causes some observable effect [155, Section 1.7.5]. The complement to this alternative hypothesis is the so-called *null hypothesis*, which states, that no such effect exists [155, Section 1.7.5]. In any psychological experiment, we can only take a relatively small *sample* from a very large (or potentially infinite) *population*. For instance, in the experiments reported in Section 10.2, we used a sample of 60 images from a potentially infinite population of line drawings. Since we only have access to a small sample, we cannot empirically prove, that alternative hypothesis is true. Instead, one typically estimates the probability of observing the given data under the assumption, that the null hypothesis is true [268]. In other words, we estimate, how

likely it is to make these observations by chance (e.g., based on which samples were picked from the population), if there is no effect of the experimental intervention on the overall population.

The general approach for such investigations is called *inferential statistics*, and involves the construction of statistical models [155, Section 2.6]. In order to compute the aforementioned probability, so-called *test statistics* are used [155, Section 2.6]. They measure two types of variation in the data: *Systematic variation* can be explained by the model, while *unsystematic variation* reflects the remaining error (i.e., the difference between the model's predictions and the actual data). Most test statistics can be computed as the ratio of systematic variation and unsystematic variation. Since the underlying probability distribution for these test statistics is typically known, one can easily calculate the probability of obtaining the observed value of the test statistic, even if the null hypothesis was true. If this probability is sufficiently low (usually, a threshold of 5% is used), the null hypothesis can be rejected, and a statistically significant effect of the experimental intervention can be diagnosed [155, Section 1.7.5].

Statistical models

Since the inferential statistics approach is based on probabilities, there are two main types of errors, which can occur [155, Section 2.6]: A *type I error* describes a situation, where the null hypothesis is rejected, even if there is no true effect in the population. A *type II error* on the other hand occurs, if we fail to reject the null hypothesis, even though there is an actual effect in the population. The probabilities for both types of errors are influenced by the significance level α (usually set to 5%), but in opposite directions: For instance, decreasing α reduces the risk for type I errors, but increases the risk for type II errors.

Error types

One of the most popular statistical tests is called *Student's t-test*. It dates back to the work of William Sealy Gossett [396], who published his research under the pseudonym "Student" [268]. The t-test is based on the observation, that scientific hypotheses can often be formulated with respect to the value of the mean or the mean difference between two groups [396]. The null hypothesis in this case states, that there is no difference between the two groups. Therefore, large differences between the observed group means are quite unlikely under the null hypothesis [155, Chapter 9].

Student's t-test

Depending on the experimental design, one can distinguish two types of the t-test [155, Chapter 9]: A *dependent* t-test is used in the context of a so-called *repeated measures design*, where the same group of participants is exposed to different experimental manipulations at different points in time. Since each participant contributes to the data from both groups, the samples are dependent. If the samples of the two groups are on the other hand independent, an *independent* t-test can be used. This is for instance applicable to experiments with a so-called *between-group design*, where different people are exposed to different experimental manipulations.

Two types of t-test

Independent t-test

Let us first consider the *independent t-test*, where we compare the observed difference between the sample group means \bar{X}_1 and \bar{X}_2 to the expected difference between the population group means μ_1 and μ_2 under the null hypothesis. We can write this as $(\bar{X}_1 - \bar{X}_2) - (\mu_1 - \mu_2)$. Since the null hypothesis states that the groups are identical, the expected difference $\mu_1 - \mu_2$ between group means is zero [155, Chapter 9], so we only need to consider $\bar{X}_1 - \bar{X}_2$. This is the systematic variation, that can be explained by our model of the alternative hypothesis (namely, that the group means are different). In order to construct our test statistic, we also need to account for the unsystematic variation in the data. This is done by considering the *standard error of differences* between groups [155, Chapter 9].

Standard error of differences

This quantity can be interpreted as follows [155, Chapter 9]: For each group, one draws one sample from the underlying population, and then computes the difference between these two samples. The distribution across these differences is centered on the difference of their group means $\mu_1 - \mu_2$. The standard deviation of this distribution is a variant of the standard error. It describes, how far the observed difference between sample means is expected to deviate from the difference between the true population means [268]. The standard error of differences between groups can be expressed as $\sqrt{\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2}}$ with s_i and N_i representing the standard deviation and number of data points in group i , respectively.¹⁷ Overall, the t-statistic can be computed as follows:

$$t = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2}}}$$

Interpreting the t-statistic

The resulting value of t is proportional to the probability, that the observed difference between means is statistically significant [268]. In general, the numerical value of t increases for larger differences between means, smaller observed variances in the data, and a larger number of data points [268, 396]. The test for statistical significance is conducted by comparing the observed value of the t-statistic to the value of the probability density function of a t-distribution with the same degrees of freedom df [268]. These *degrees of freedom* correspond to the number of observations, which are free to vary [155, Section 2.4.2]. Since we compute a standard deviation s_i for the two groups, we make use of the respective group mean. This means that we hold one parameter (namely, the mean) constant: $N_i - 1$ out of N_i observations can vary, but the remaining one is determined by the value of the mean and the value of the $N_i - 1$ other observations [155, Section 2.4.2]. For the t-test, we compute a standard deviation for two groups, hence, the degrees of freedom is determined as $df = N_1 + N_2 - 2$ [155, Chapter 9].

Dependent t-test

Let us now turn to the *dependent t-test*, which is used, if the same participants contribute to both groups. Instead of first computing the group means and then considering their difference (as in the

¹⁷ See [155, Chapter 9] for a derivation of this formula.

independent t-test), we follow the opposite approach [155, Chapter 9]: For each participant, we first compute their difference between the two conditions, and then average these differences across all participants. This results in the average difference \bar{D} , which is an indicator of the systematic variation in the data. This systematic variation is now divided by the standard error of differences, which can be computed as $\sqrt{\frac{s_D^2}{N}}$ (see above). This division through the standard error contrasts the observed difference \bar{D} with the expected difference and furthermore normalizes the resulting value, such that it is independent from the original scale of measurement. Overall, the formula used for a dependent t-test looks as follows:

$$t = \frac{\bar{D}}{\sqrt{\frac{s_D^2}{N}}}$$

Again, one can compute the probability of obtaining the measured value of t based on a t-distribution with $N - 1$ degrees of freedom.

Using the t-statistic

The results of a t-test are usually reported by stating the mean and standard error for the respective groups, the degrees of freedom, the value of t , and the level of statistical significance (i.e., the probability of the data under the null hypothesis) [155, Chapter 9]. For instance, $t(11) = 2.47, p < .05$ means, that a t-test with 11 degrees of freedom was conducted, that the observed value for the t-statistic was 2.47, and that this corresponds to a probability of less than 5% for observing the given data, if the null hypothesis is true. Typically, instead of reporting the actual value of p , only the rough level of significance is reported, e.g., $> .5, > .1, > .05, < .05, < .01, < .001$, and $< .0001$.

Reporting t-test results

It is furthermore important to mention, that the t-test can only be applied, if several assumptions are met: The data must be measured at least at the interval level, such that computing a mean and a standard deviation is meaningful [155, Chapter 9] (cf. also Section 8.1.4). Moreover, one assumes, that the sampling distribution is approximately normal [268]. For an independent t-test, this means, that the individual values are normally distributed, while for a dependent t-test, the *differences* between the values need to follow a normal distribution [155, Chapter 9]. For the independent t-test, the samples of the groups need to be independent from each other, and the two groups are typically assumed to have an equal variance [155, Chapter 9]. Finally, outliers in the data and unequal sample sizes for the two groups can complicate an analysis with the t-test [268].

Assumptions and limitations

Some of our subsequent analyses make use of so-called *cumulative link mixed models* (CLMMs) and their implementation in the ordinal package of R [103]¹⁸. CLMMs can be used to analyze the dependence of *ordinal* response variables on a given set of predictors. They are especially useful, if the collected ratings cannot be assumed to be independent, since the data contains clusters [9]. Before introducing

Cumulative link mixture models

¹⁸ See <https://cran.r-project.org/web/packages/ordinal/index.html>.

CLMMs, we first need to describe their predecessors: cumulative link models (CLMs).

Cumulative link models

A *cumulative link model* (CLM) considers an ordinal response variable y with J different levels and a vector $\vec{x} = (x_1, \dots, x_n)$ of predictor variables. It tries to predict the probability of a given rating $y = j \in \{1, \dots, J\}$ by estimating the cumulative probabilities $\mathbb{P}(y \leq j)$ with a suitably transformed linear model $\alpha_j - \vec{x}^T \vec{\beta}$ [9, 104, 350]:

$$\mathbb{P}(y \leq j) = F(\alpha_j - \vec{x}^T \vec{\beta})$$

Components of the model

The monotonically increasing *transformation function* F is also called "link", which together with the focus on cumulative probabilities gives rise to the name "cumulative link model". The free parameters of the model include the *intercepts* α_j and the vector of *coefficients* $\vec{\beta} = (\beta_1, \dots, \beta_n)$. The intercepts α_j can be interpreted as thresholds for the different levels of the ordinal variable. The coefficients β_i model the influence of the respective predictor variable x_i on the response variable y : Positive values $\beta_i > 0$ cause the term $\alpha_j - \vec{x}^T \vec{\beta}$ to decrease for larger values of x_i . Since this corresponds to reducing the cumulative probability $\mathbb{P}(y \leq j)$, it implicitly increases the probability $\mathbb{P}(y > j)$ and thus leads to larger values of the response variable [350].

Interaction of variables

At this point, we should also note, that it is possible to model the interaction of predictor variables by including their product into the model [9]. The resulting model is then of course no longer linear, since it contains the nonlinear term $x_1 x_2$:

$$\mathbb{P}(y \leq j) = F(\alpha_j - \beta_1 x_1 - \beta_2 x_2 - \beta_3 x_1 x_2)$$

Transformation functions

In principle, different transformation functions F are possible. Using the logistic function $\sigma(z) = \frac{1}{1+e^{-z}}$ is referred to as a "logit link" and gives rise to a multi-category generalization of logistic regression [9] (cf. Section 5.2.2). Another popular choice is the cumulative distribution function $\Phi(z)$ of a standard normal distribution, which is then called a "probit link" [104].

From CLMs to CLMMs

CLMs implicitly make two assumptions [350]: Firstly, the coefficients β are independent of the response level j , which is often called the "proportional odds" assumption. Secondly, CLMs assume the independence of individual observations. The latter assumption is, however, often violated in practice, for instance, if there are clusters in the data (such as repeated measurements from the same individual subject) [9, 350]. Cumulative link *mixed* models (CLMMs) solve this problem by allowing random effects based on the presumed clusters. In general, CLMMs can be described as follows [9, 104, 350]:

$$\mathbb{P}(y \leq j) = F(\alpha_j - \vec{x}^T \vec{\beta} - \vec{z}_c^T \vec{u}_c)$$

Additional coefficients

Here, for each cluster c , \vec{z}_c is a vector of explanatory variables for the random effects, and \vec{u}_c contains additional coefficients for data

points in cluster c . The coefficients of \vec{u}_c are assumed to follow a normal distribution and allow us to model correlations between members of the same cluster [350]. On the one hand, they can be used for adding random intercepts, i.e., to represent that the cluster averages differ. For instance, one participant may consistently give higher ratings than another one. In the limiting case of $\vec{z}_c = \vec{1}$, \vec{u}_c corresponds to the deviation of the respective participant from the overall average rating. On the other hand, one can also add random slopes, for example, to model different participants as having a different sensitivity to a given predictor variable x_i . This can be modeled by also including x_i in \vec{z}_c , such that the respective entry of \vec{u}_c models the increased (or decreased) sensitivity of cluster c to the predictor x_i .

One often uses a shorthand notation for describing the structure of a CLMM. For instance, Christensen [102] considers a CLMM, which models the bitterness rating of various wines based on the predictor variables *temperature* and *contact* (i.e., whether juice and skin were separated while crushing the grapes). He assumes, that there may be systematic rating differences between judges, which he models through the variable *judge*:

An example

$$\mathbb{P}(y_{rating} \leq j) = \sigma(\alpha_k - \beta_{temperature} \cdot x_{temperature} - \beta_{contact} \cdot x_{contact} - u_{judge})$$

This overall relation can be expressed more compactly by using the following shorthand notation [102, 103]:

Shorthand notation

$$rating \sim temperature + contact + (1 \mid judge)$$

Here, $(1 \mid judge)$ denotes, that each judge has their own intercept. If we also include a group-level random slope for the *temperature* variable, the shorthand notation looks as follows [81]:

Adding random slopes

$$rating \sim temp + contact + (1 + temperature \mid judge)$$

Here, $(1 + temperature \mid judge)$ denotes, that both the intercept (i.e., a constant value of 1) and the coefficient with respect to the *temperature* depend on the grouping variable *judge*. Finally, the interaction of two predictor variables can be written as follows:

Interaction of variables

$$\begin{aligned} \mathbb{P}(y_{rating} \leq j) = \sigma(\alpha_k - \beta_{temperature} \cdot x_{temperature} - \beta_{contact} \cdot x_{contact} - \beta_{interaction} \cdot x_{temperature} \cdot x_{contact}) \\ rating \sim temperature \cdot contact \end{aligned}$$

CLMMs are used to analyze ordinal data by first specifying the model's structure and then estimating the parameters $\vec{\alpha}$, $\vec{\beta}$, and \vec{u}_c based on the available data. This is usually done by maximizing the log likelihood of the data (cf. Section 5.1.3) [9, 104, 350]. For evaluation purposes, one can optimize two separate models, which differ only with respect to one aspect (e.g., the inclusion or exclusion of a given predictor

Fitting CLMMs

variable). One can then measure, whether the difference between the two models causes statistically significant differences with respect to their ability to predict the ordinal response variable [104, 350]. This is usually done by resorting to the likelihood ratio of the two models.

Likelihood ratio

The *likelihood ratio* λ quantifies the relative likelihood of the observed data with respect to the two models under consideration [166]. It can be computed as follows, where p_k is the probability density function of the given model k , \vec{x} is the vector of observations, and $\hat{\theta}_k$ are the maximum-likelihood estimates for the parameters of model k [166]:

$$\lambda = \frac{p_2(\vec{x} | \hat{\theta}_2)}{p_1(\vec{x} | \hat{\theta}_1)}$$

Interpretation

For $\lambda > 1$, the second model is preferred over the first one, since it gives a better account of the observed data. Conversely, for $\lambda < 1$, the first model is preferred over the second one. In both cases, the order of magnitude of λ indicates the strength of the preference.

Statistical test for model differences

Wilks [435] was able to show, that the quantity $-2 \ln \lambda$ follows a χ^2 -distribution for sufficiently large sample sizes. The χ^2 -distribution is usually used in Pearson's χ^2 test, which analyzes the interaction of two categorical variables [155, Chapter 18]. Since the χ^2 -distribution is already well-known in the context of statistical tests, one can apply it also to the likelihood ratio in order to test for statistical significant differences between two models: One computes the term $-2 \ln \lambda$ based on the likelihood ratio λ , and then assesses the probability of obtaining this value based on a χ^2 distribution, whose degrees of freedom reflect the difference in free parameters between the two models. The results of the statistical test are reported by specifying the degrees of freedom, the obtained value for $-2 \ln \lambda$, and the significance level: For instance, $\chi^2(1) = 9.92, p < 0.05$ indicates, that the models differed with respect to a single parameter, that $-2 \ln \lambda = 9.92$, and that the probability of observing this value under the null hypothesis of no difference between the models is below 5%.

Statistical significance for Pearson's r

In our subsequent statistical analyses, we will also make use of Pearson's r . Remember from Section 9.1.2, that Pearson's r is a correlation coefficient, which measures the linear correlation between two quantities. In addition to computing the raw value of Pearson's r , we are also interested in conducting a test of statistical significance. The null hypothesis in this case is, that the true correlation in the underlying population equals zero [15]. We thus investigate, whether the correlation coefficient is significantly different from zero. This can for instance be done by resorting to a t-test, because the following expression follows a t-distribution with $N - 2$ degrees of freedom (where N is the number of observations) [155, Section 6.3]:

$$t = \frac{r \cdot \sqrt{N - 2}}{\sqrt{1 - r^2}}$$

In addition to a statistical test on the correlation coefficient, we will also provide a 95% *confidence interval* (CI). This confidence interval gives us a range of plausible values for the true correlation coefficient of the underlying population [365]. More specifically, we expect, that the true value of the correlation coefficient lies within this confidence interval for 95% of the samples, which can be drawn from the population [155, Section 2.5.2].

Confidence interval for Pearson's r

The distribution of r as measured from samples is unfortunately not necessarily normal, but often skewed, if the true correlation differs strongly from zero [365]. It is therefore difficult to directly assess the probability distribution of r in order to construct a confidence interval. Instead, a so-called z transformation is used, where we compute an auxiliary variable z as follows (see [365] and [155, Section 6.3]):

Using a z transformation

$$z = \frac{1}{2} \ln \left(\frac{1+r}{1-r} \right)$$

The resulting variable z is approximately normally distributed with a variance of $\sigma_z^2 = \frac{1}{N-3}$ (where N is the number of observations) [15, 365]. Notably, this variance is independent from the value of r . For the variable z , we can now easily define a 95% confidence interval $[z_{lower}, z_{upper}]$, since z follows a normal distribution [28, 365]:

Distribution of z

$$z_{lower} = z + v_{0.025} \cdot \sqrt{\frac{1}{N-3}} \quad z_{upper} = z + v_{0.975} \cdot \sqrt{\frac{1}{N-3}}$$

For both bounds, we start at our estimate of z and move away from this point estimate based on the standard deviation of z . The factors $v_{0.025}$ and $v_{0.975}$ reflect the thresholds for 2.5% and 97.5% of the probability mass of a standard normal distribution, respectively. They correspond to $v_{0.025} \approx -1.96$ and $v_{0.975} \approx 1.96$, respectively [155, Section 6.3]. This 95% confidence interval $[z_{lower}, z_{upper}]$ for z can now be transformed back into a 95% confidence interval $[r_{lower}, r_{upper}]$ for the correlation coefficient r by inverting the z transformation (cf. [28, 365] and [155, Section 6.3]):

Deriving the boundaries

$$r_{lower} = \tanh(z_{lower}) = \frac{e^{2 \cdot z_{lower}} - 1}{e^{2 \cdot z_{lower}} + 1}$$

$$r_{upper} = \tanh(z_{upper}) = \frac{e^{2 \cdot z_{upper}} - 1}{e^{2 \cdot z_{upper}} + 1}$$

10.3.2 Comparing Visual and Conceptual Similarity

Before extracting similarity spaces from the visual dissimilarity ratings, we need to ensure, that they indeed capture shape similarity and not general conceptual similarity. As argued above, information about the SHAPE of objects plays a critical role in conceptualizing them. Therefore, shape similarity can be seen as an important basis of conceptual similarity. However, conceptual similarity also has additional sources of information, such as SIZE, COLOR, and TEXTURE (which are also based

Visual and conceptual similarity

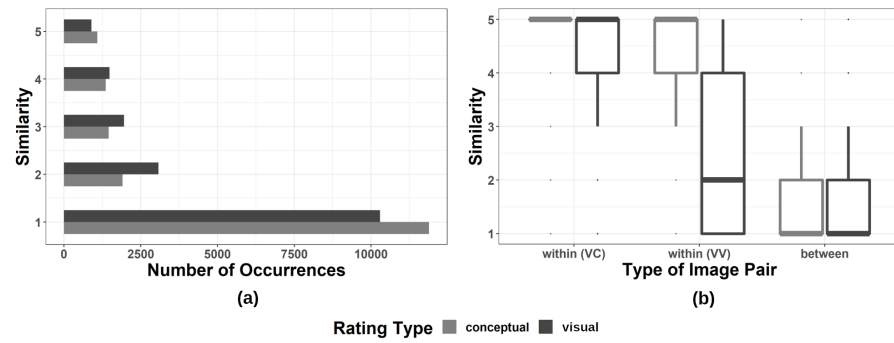


Figure 10.7: Comparison of the visual and conceptual similarity ratings. (a) Distribution of the ratings. (b) Boxplot of the ratings as a function of the type of image pair.

on visual input), or WEIGHT, SOUND, and USAGE (which are not based on visual input). Overall, we thus expect, that conceptual similarity and shape similarity are correlated to a certain extent, but that we are also able to identify clear differences between them.

CLMM setup

We randomly selected 10 of the 15 shape similarity ratings per image pair in order to have the same number of data points for each rating type. The numerical coding was standardized (1 for "very dissimilar" to 5 for "very similar") by inverting the scale of the conceptual dissimilarity ratings.¹⁹ For our analyses with CLMMs, we considered the variables *rating* (i.e., the similarity rating for a given pair of items), *ratingType* (conceptual similarity vs. visual similarity), *visualType* (visually coherent vs. visually variable categories), *pairType* (within-category vs. between-category) and *pairID* (i.e., a unique ID for each pair of items). Our basic CLMM used the following formula with a logit link:

$$rating \sim ratingType + (1 + ratingType | pairID)$$

Interpreting the CLMM

We assumed, that the value of the ordinal similarity rating is consistently affected by the type of similarity (visual vs. conceptual), but that there are also random effects with respect to both the intercept and the slope based on the individual item pairs. In other words, if we ignore the individual effects based on the respective item pairs, we expected to find a clear trend based on the rating type (visual vs. conceptual).

The overall distribution of ratings

Both types of ratings had a distribution across the full scale with a mode of 1 (see Figure 10.7a). The heavy bias towards non-similarity in both types of ratings is due to both the much higher number of

¹⁹ As Tversky [412] has pointed out, such a simple inverse relationship between similarity and dissimilarity, does, unfortunately, not always hold, when collecting psychological data. He argues, that judging the similarity of two stimuli incites participants to search for common features, while rating dissimilarity emphasizes distinctive features. Tversky also supports his claims with data from a small study. Since we analyzed a pre-existing set of ratings, which had been collected with two inverse scales, and since re-conducting the experiments was not feasible with our given resources, we nevertheless decided to make this simplifying assumption.

between-category pairs than within-category pairs, and the lack of shape similarities across categories for our stimulus set.²⁰ Although the rating distributions were roughly similar, the central tendency of the two rating types differed significantly ($\chi^2(1) = 178.33, p < .0001$), as a likelihood ratio test of the following two CLMMs revealed:²¹

$$\begin{aligned} rating &\sim ratingType + (1 + ratingType \mid pairID) \\ rating &\sim 1 + (1 + ratingType \mid pairID) \end{aligned}$$

Since the two models only differ with respect to a main effect based on the *ratingType* (i.e., visual vs. conceptual similarity), a statistically significant difference between the two CLMMs shows, that the *ratingType* has a significant impact on the overall *rating*, which cannot be explained by random effects based on the individual *pairIDs*. This indicates, that the perception of shape similarity was governed by different factors than the perception of conceptual similarity.²²

Interpretation

A separate analysis of the results of between- and within-category pairs further supports this conclusion. As the boxplot in Figure 10.7b shows, the level of conceptual similarity was clearly determined by category membership with small variances (inter-quartile ranges²³ between 0 and 1), whereas the level of shape similarity was poorly predicted by category membership (different medians in the three groups and partially overlapping inter-quartile ranges).

Within-category vs. between-category

We estimated with CLMMs, whether the two rating types also differed significantly, when analyzing the within-category pairs and between-category pairs separately. We first applied the following CLMMs only to the between-category pairs in our data:

Looking at between-category pairs

$$\begin{aligned} rating &\sim ratingType + (1 + ratingType \mid pairID) \\ rating &\sim 1 + (1 + ratingType \mid pairID) \end{aligned}$$

A likelihood ratio test revealed, that shape similarity tended to be significantly higher than conceptual similarity ($\chi^2(1) = 281.95, p < .0001$), verifying that semantic relationship is not required in order to perceive two shapes as visually similar.

The effect persists

The analysis of the within-category pairs was conducted with the following four CLMMs, where the first one was used as alternative

Considering within-category pairs

²⁰ As Verheyen et al. [418] noted, similarity ratings skewed towards dissimilarity are usually captured quite well by spatial models – a promising observation with respect our aim of extracting similarity spaces from these ratings.

²¹ Likelihood ratio tests were conducted using the `anova()` function.

²² One may argue, that this may to some extent be an artifact of our conversion of dissimilarities to similarities as discussed above. However, given the very low *p* value and the very similar elicitation method using Likert scales with labeled end points (cf. Section 10.2.2), it seems unlikely, that our results can be solely explained by the conversion of the ratings. This also holds for our subsequent analyses in the remainder of this section.

²³ The difference between the third and the first quartile of the data, illustrated by the height of the respective box.

hypothesis, while the other ones were used as null hypotheses to investigate main effects and interactions, respectively:

$$rating \sim ratingType * visualType + (1 + ratingType | pairID) \quad (10.1)$$

$$rating \sim ratingType + visualType + (1 + ratingType | pairID) \quad (10.2)$$

$$rating \sim visualType + (1 + ratingType | pairID) \quad (10.3)$$

$$rating \sim ratingType + (1 + ratingType | pairID) \quad (10.4)$$

Comparing VC with
VV categories

Here, we additionally included *visualType* as a predictor, since the type of category (visually coherent vs. visually variable) may play an important role for within-category ratings, especially with respect to visual similarity. Please recall from Section 10.3.1, that both shorthand notations $ratingType + visualType$ (as used for CLMM 10.2) and $ratingType \cdot visualType$ (CLMM 10.1) involve two individual linear terms $\beta_{ratingType} \cdot x_{ratingType}$ and $\beta_{visualType} \cdot x_{visualType}$, but that $ratingType \cdot visualType$ also adds a nonlinear interaction term $\beta_{interaction} \cdot x_{ratingType} \cdot x_{visualType}$ to the model.

Results

We found a significant difference between the shape similarity and the conceptual similarity ratings: Shape similarity tended to be lower than semantic similarity (i.e., main effect of *ratingType*: $\chi^2(2) = 152.09$, $p < .0001$, obtained by comparing the CLMMs 10.1 and 10.3). This effect was significantly stronger for visually variable (VV) categories than for visually coherent (VC) categories (significant interaction: $\chi^2(1) = 32.36$, $p < .0001$, comparison of CLMMs 10.1 and 10.2). This observation verifies, that semantic relationship is also not a sufficient condition for perceiving two shapes as visually similar. Finally, we also found a main effect of *visualType* ($\chi^2(2) = 116.19$, $p < .001$, comparing CLMMs 10.1 and 10.4), indicating, that visually coherent (VC) categories led to higher within-category ratings than visually variable (VV) categories.

Global similarity
matrices

We furthermore aggregated both types of similarity ratings into global similarity matrices using the median as an aggregator in order to meet the ordinal scale of measurement of the dependent variable, and the same ten ratings per item pair as used above.

Correlation of
aggregated ratings

Analyzing the Kendall rank correlation [219] (cf. Section 9.1.2) between the entries of the two matrices resulted in $\tau \approx 0.39$, indicating that there is some limited correlation between visual and conceptual similarity. Figure 10.8 shows a scatter plot for the entries of the two matrices. One cannot observe a clear linear or monotone trend, but there is a strong agreement on the lowest similarity value (visualized by the size of the circle). Based on the observation from Figure 10.7a, that both dissimilarity matrices contain a large amount of these values, this is, however, not a large surprise.

Looking at concrete
examples

Table 10.2 shows some item pairs with large and small differences between visual similarity (Sim_{Vis}) and conceptual similarity (Sim_{Con}) based on the entries of the respective dissimilarity matrix. As we can see, conceptual similarity seems to be much larger than visual similarity,

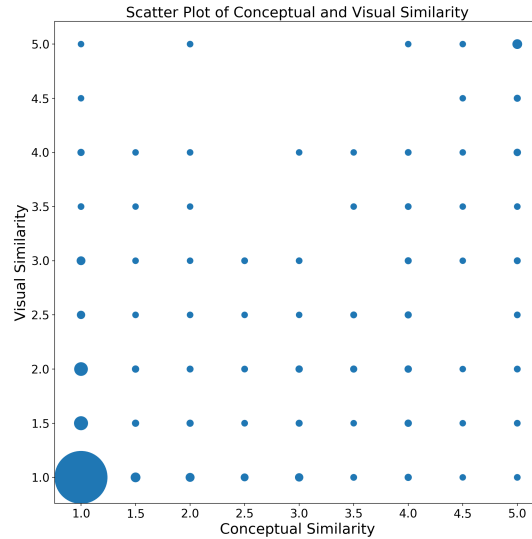


Figure 10.8: Scatter plot for the entries of the global similarity matrices based on conceptual and visual similarity ratings.

$Sim_{Vis} \ll Sim_{Con}$	$Sim_{Vis} \gg Sim_{Con}$	$Sim_{Vis} = Sim_{Con}$
CARROT – BELL PEPPER	TEA POT – APPLE	DISHWASHER – WASHING MACHINE
BANANA – CHERRY	IGLOO – LADYBUG	ANT – FLY
LADYBUG – HORSE	ASPARAGUS – AXE	LADYBUG – AXE
APPLE – ASPARAGUS	BANANA – SAW	CARROT – TEA POT

Table 10.2: Examples for item pairs with different relations of visual and conceptual similarity.

if the items belong to the same (super-)category (VEGETABLES, FRUIT, ANIMALS, and PLANTS, respectively), but differ in their visual appearance. On the other hand, visual similarity is much larger than conceptual similarity, if the items have a similar shape, but belong to different categories. Finally, visual and conceptual similarity are identical either for visually similar items from the same category, or for visually dissimilar items from different categories. The latter case illustrates again the large number of item pairs, which were rated as very dissimilar under both conceptual and visual similarity. While the overall distribution of the similarity ratings seems to be quite similar on a first glance, the examples from Table 10.2 indicate, that the statistically significant differences between visual and conceptual similarities as reported above can be easily interpreted and conform to our expectations.

As we have seen in our CLMM-based analysis above as well as in the box plot from Figure 10.7b, the distinction into visually coherent and visually variable categories seems to be reflected in the visual, but not in the conceptual similarity ratings. In the following, we investigate this observation in more detail. In addition to the item-based similarity matrices, we have also aggregated the raw similarity ratings based on

Item-based and category-based visual similarity

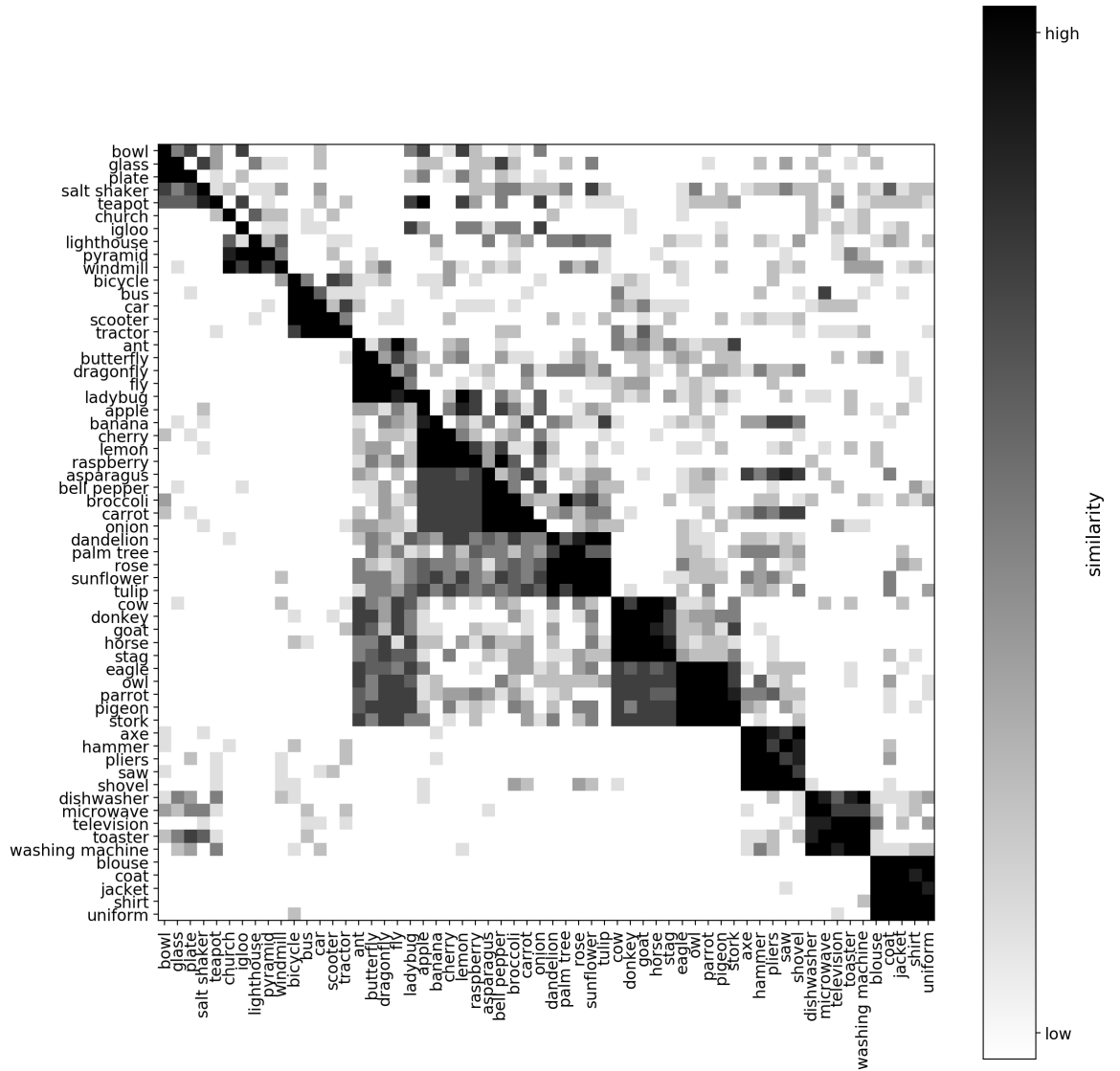


Figure 10.9: Item-based similarity matrices for visual (above the diagonal) and conceptual (below the diagonal) similarity ratings.

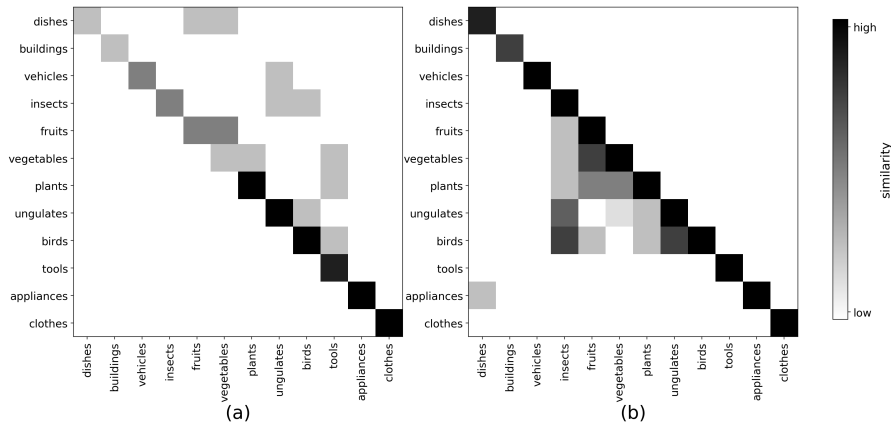


Figure 10.10: (a) Category-based similarity matrix for visual similarity. (b) Category-based similarity matrix for conceptual similarity.

the category structure, using the median to arrive at overall within-category and between-category dissimilarities. Figures 10.9 and 10.10 show the item-based and the category-based similarity matrices for both visual and conceptual similarity ratings.

The item-based shape similarity matrix is shown in Figure 10.9 as triangular area above the diagonal. One can see, that obviously similar shapes were consistently rated as very similar (black colored combinations, e.g. BLOUSE–JACKET OR TEAPOT–APPLE), and that obviously dissimilar shapes were consistently rated as very dissimilar (white colored combinations, e.g. ONION–CARROT OR PLATE–HAMMER), confirming a general sensitivity of the ratings to shape properties. For both high similarities and low similarities, we can find within-category and between-category pairs, indicating that the perceived similarity was not per se determined by semantic-taxonomic knowledge. However, high similarity combinations tend to be clustered within categories.

Looking at the average similarity of within-category pairs only (arranged at the diagonal of Figure 10.10a), one can see a clear distinction between visually variables (VV) categories (low to medium, see top left of the diagonal) and visually coherent (VC) categories (high for all categories, see lower right of the diagonal), supporting the visual type of the categories as classified by Scheibel and Indefrey [349].

Moreover, the average level of shape similarity for within-VV category pairs is not specific for these categories: Similar levels of shape similarity can also be observed for between-category pairs, which – crucially – can comprise semantically unrelated categories (such as UNGULATES and VEHICLES). This finding gives some evidence against large semantic confounds in the shape ratings.

The highest off-diagonal value in Figure 10.10a is given for the category combination FRUITS–VEGETABLES. While both categories have a low average intra-category visual similarity, we can observe high inter-category shape similarity for some specific combinations of stimuli. This is illustrated in Figure 10.11: High similarities occur mainly

Item-based shape similarity

Shape similarity of VV and VC categories

VV categories and between-category ratings

Visual similarity of FRUITS and VEGETABLES

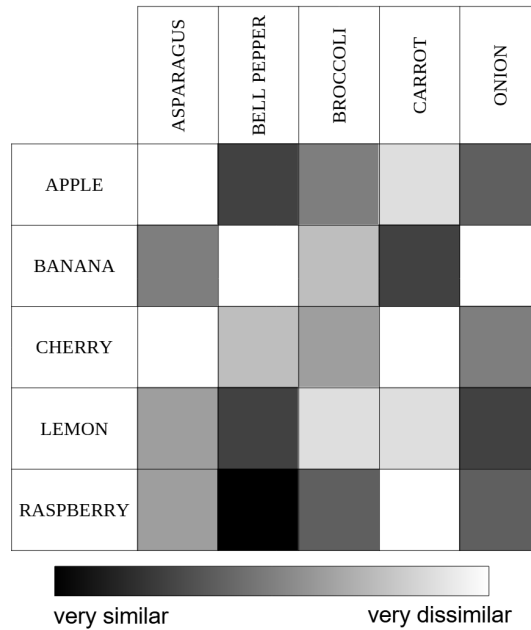


Figure 10.11: Aggregated visual similarity for item pairs from the categories VEGETABLES and FRUIT.

for combinations of the roundish objects {APPLE, RASPBERRY, LEMON} and {BELL PEPPER, ONION}, and for combinations of the elongated objects {BANANA} and {CARROT, ASPARAGUS}. This already indicates the relevance of the FORM feature for the perceived shape similarity. The relatively high similarity between RASPBERRY and CAULIFLOWER on the other hand seems to refer also to their internal structure and hence to the LINES feature. While both categories are visually heterogeneous, some subsets of items match well with each other resulting in a relatively high overall inter-category similarity.

Considering conceptual similarity

Let us now take a look at the conceptual similarity ratings. The category-based conceptual similarity matrix in Figure 10.10b shows, that the average similarity level was high within all categories (black colored, see the diagonal), and low for most of the between-category pairs (white colored), confirming a general sensitivity of the ratings to semantic-taxonomic knowledge. In Figure 10.10b, we can also find a prominent cluster at the center of the matrix, where high between-category similarities can be observed. This cluster is clearly delimited to pairs of taxonomically related categories, which have a common superordinate category (ANIMALS OR LIVING ORGANISMS), and corresponds to the six categories of natural objects. The only other pair of categories with a somewhat elevated conceptual similarity is APPLIANCES–DISHES. This cannot be explained by taxonomy, but may relate to shared contexts (e.g., a kitchen). As one can see in Figure 10.9 (entries below the diagonal), high conceptual similarity outside these clusters is limited to only a handful of exceptions, which might be negligible outliers.

The category-based similarity matrix in Figure 10.10b furthermore shows no difference in the intra-category level of conceptual similarity for visually coherent (VC) and visually variable (VV) categories. This indicates, that a strong semantic relationship between category members was given for all categories, such that common shape characteristics were secondary for making conceptual similarity judgments.

Category type and conceptual similarity

We can conclude our investigations by stating, that visual and conceptual similarity differ from each other in meaningful ways. The ratings obtained for visual similarity seem to refer to similarity of shapes rather than conceptual similarity. For the remainder of this dissertation, we will focus exclusively on the data concerning visual similarity.

Summary

10.3.3 Comparing Pre-Attentive and Attentive Feature Ratings

Let us now turn to the ratings with respect to the psychological shape features `FORM`, `LINES`, and `ORIENTATION`. In this section, we investigate, whether the three features are correlated with each other, and whether we can observe considerable differences between attentive and pre-attentive ratings. In order to compare attentive and pre-attentive ratings in a meaningful way, we used the aggregated ratings as described in Section 10.2.3 (i.e., a single average score per item across all participants). If we used the individual ratings instead (as in the analysis of dissimilarity ratings in Section 10.3.2), we would need to compare continuous attentive ratings to discrete pre-attentive ratings, which would have made a clean analysis much more cumbersome.

Overview

The distributions of the aggregated feature values are displayed in Figure 10.12, distinguishing between the three shape features and the type of rating experiment (attentive vs. pre-attentive perception). As the histograms illustrate, the ratings do not follow an obvious pattern such as being uniformly distributed, showing two distinct clusters at the ends of the scale, or following a normal distribution. In all cases, a variety of different feature values can be observed, indicating, that the features are able to differentiate between different stimuli.

Histograms

In a first step, we investigated, whether the different psychological features are correlated with each other. Since they are typically viewed as conveying orthogonal pieces of information, we expect them to be largely uncorrelated. Correlation analyses have been conducted using Pearson's r [315] (cf. Sections 9.1.2 and 10.3.1).²⁴

Are features uncorrelated?

Let us first focus on the attentive ratings. There was a small, but statistically insignificant positive relationship between `FORM` and `LINES` ($r = 0.23$, $CI = (-0.02, 0.46)$, $t(58) = 1.82$, $p > .05$). There was no correlation between `LINES` and `ORIENTATION` ($r = -0.08$, $CI = (-0.33, 0.18)$, $t(58) = -0.62$, $p > .5$). The features `FORM` and `ORIENTATION` showed a small negative, but not significant correlation ($r = -0.19$, $CI =$

Attentive ratings

²⁴ Pearson correlation coefficients as well as 95% confidence intervals (CI) and statistical significance values were estimated using the `cor.test()` function of R.

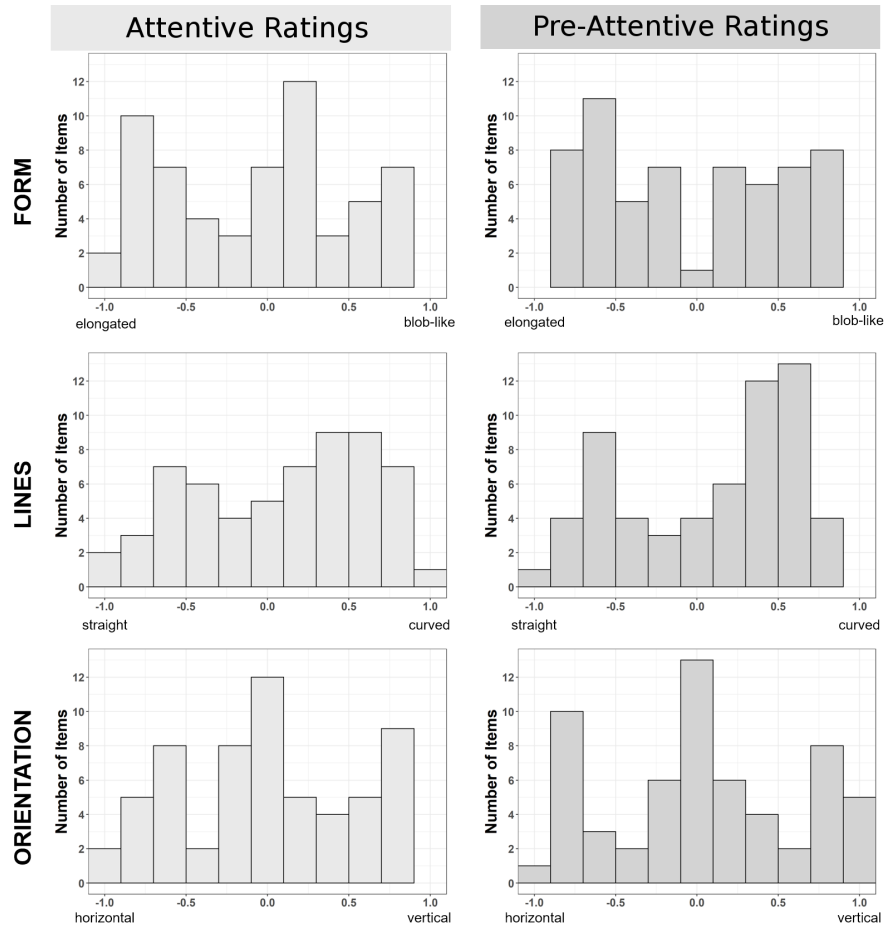


Figure 10.12: Histograms for both attentive and pre-attentive ratings with respect to FORM, LINES, and ORIENTATION.

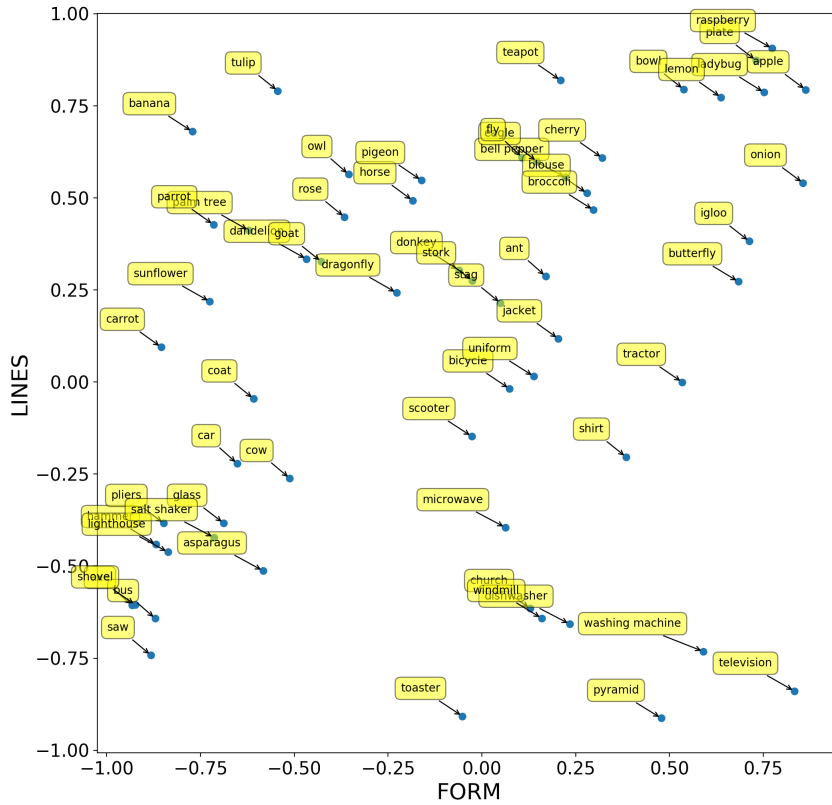


Figure 10.13: Scatter plot for the features FORM and LINES based on the attentive feature ratings.

$(-0.42, 0.07)$, $t(58) = -1.96$, $p > .1$), indicating that elongated shapes tended to have a vertical orientation and blob-like shapes a horizontal orientation. Since correlations between features were small and not significant, the inspected primitive shape features seem to refer to distinct shape qualities, which can be perceived selectively.

For the pre-attentive ratings, a similar picture emerged: There was a small, but insignificant positive relationship between FORM and LINES ($r = 0.22$, $CI = (-0.03, 0.45)$, $t(58) = 1.75$, $p > .05$), and we again found no relationship between LINES and ORIENTATION ($r = -0.01$, $CI = (-0.27, 0.24)$, $t(58) = -0.10$, $p > .5$). Also the features FORM and ORIENTATION showed no correlation in the pre-attentive data ($r = 0.06$, $CI = (-0.20, 0.31)$, $t(58) = 0.45$, $p > .5$). Overall, correlations between features were small and not significant, suggesting, that the inspected primitive shape features refer to distinct shape qualities.

Figure 10.13 shows a scatter plot for the feature pair FORM and LINES based on the attentive feature ratings. Even though we have found a small positive (but statistically not significant) correlation, we are not able to see a clear monotone trend in the scatter plot. Similar observations can be made on the scatter plots for the other feature pairs (both for attentive and pre-attentive ratings) which are reported in Appendix E.1.

Pre-attentive ratings

Visualizing FORM and LINES

Comparing attentive
and pre-attentive
ratings

Our second analysis of the feature ratings aimed at clarifying, whether the duration of perception systematically affected the feature values. In other words, we investigated, how similar attentive and pre-attentive ratings were for the three selected shape features. We considered each feature separately, using dependent t-tests to estimate the significance of the difference between attentive and pre-attentive feature values, and Pearson's r to estimate the strength of their correlation.

FORM: no difference,
high correlation

For the FORM feature, we observed no effect of viewing duration on the ratings ($t(59) = -0.32, p > .5$), indicating, that the global structure of a shape is rapidly, pre-consciously perceived, and that this perception is not changed by attentional mechanisms. Early FORM perceptions were strongly correlated with FORM perceptions after normal viewing times ($r = 0.77, CI = (0.64, 0.85), t(58) = 9.1, p < .0001$). The range of the confidence interval was small and did not cross zero, indicating, that the strong positive correlation can reliably be generalized.

ORIENTATION:
significant difference,
high correlation

When considering the ORIENTATION of the shapes, we found a statistically significant effect of viewing duration ($t(59) = 3.31, p < .01$), indicating, that the perception of shape orientation is changed by attentive processes. The attentive perception of the shapes' ORIENTATION (horizontal vs. vertical) was, however, highly correlated with the pre-attentive perception ($r = 0.81, CI = (0.70, 0.88), t(58) = 10.41, p < .0001$). The small confidence interval indicates, that this finding can reliably be generalized.

Interpreting the
difference

As argued in Section 10.1.1, the ORIENTATION of lines is usually associated with pre-attentive processes (cf. the studies by Treisman and Gormican [409] and Hubel and Wiesel [203], but also the HMAX model [331]). However, other studies (e.g., by Sagi and Julesz [344], by Palmer et al. [313], and by Töllner et al. [413]) have also found, that while pre-attentive processes are sufficient to decide about the presence or absence of a target stimulus, which differs in its ORIENTATION from the context objects, attentive processes are needed in order to determine the exact value of the ORIENTATION feature (e.g., whether the "odd one out" was horizontally or vertically oriented). The observed differences between pre-attentive and attentive ratings for the ORIENTATION feature are thus not entirely surprising, and match similar observations from the literature.

LINES: no difference,
high correlation

Finally, there was no significant effect of viewing duration on the perception of the main LINES of the items ($t(59) = 1.43, p > .1$), indicating, that the shape of the main lines is already pre-consciously perceived, and that the perception of the lines does not change under attentive viewing. The attentive perception of the lines was strongly correlated with the pre-attentive perception ($r = 0.74, CI = (0.60, 0.84), t(58) = 8.40, p < .0001$). Again, the small confidence interval indicates that this finding can reliably be generalized.

Differences in
ORIENTATION ratings

Figure 10.14 shows a scatter plot illustrating the values of the ORIENTATION feature for all items based on pre-attentive and attentive perception. We can see a monotone linear trend with some outliers, including CHERRY, STORK, STAG, and CHURCH (where pre-attentive ratings are higher than attentive ratings), as well as WINDMILL, WASHING MACHINE,

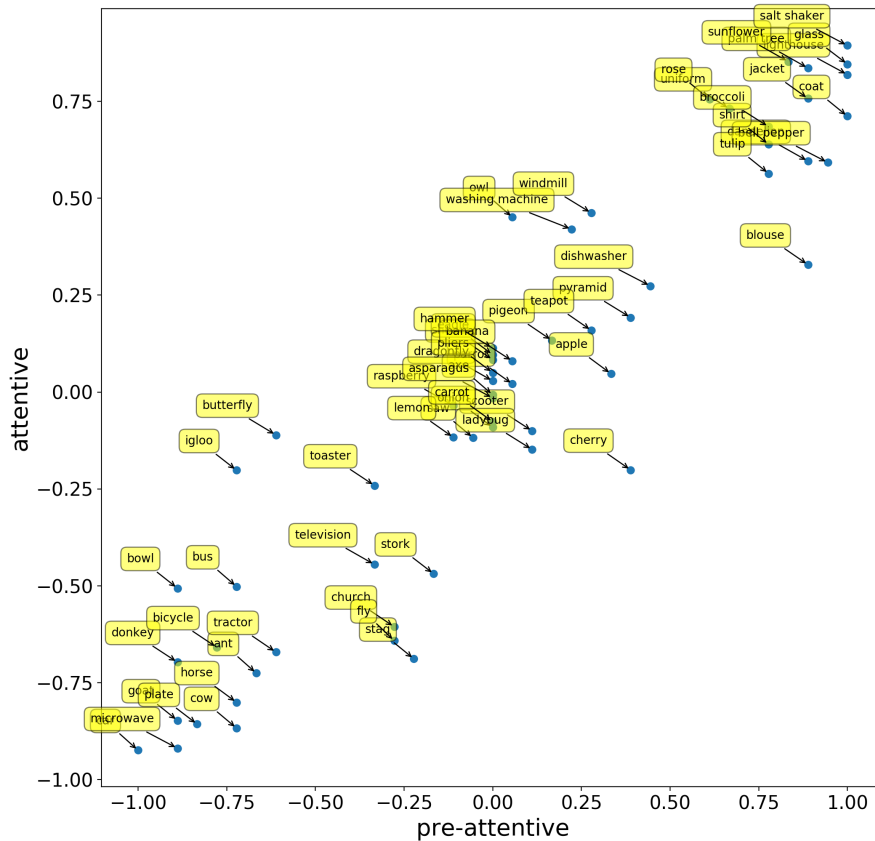


Figure 10.14: Scatter plot comparing pre-attentive and attentive ratings for the ORIENTATION feature.

and OWL (with higher attentive than pre-attentive ratings). If we take a closer look, these conflicts seem to make intuitive sense: For instance, for the CHERRY, pre-attentive judgments may have focused on the vertical stems, while attentive processing also took into account the overall convex hull of the object. For STORK, STAG, and CHURCH, the presence of both horizontal and vertical lines may have led to a pre-attentive judgment of "quite diagonal", while the overall horizontal extension of the objects may have played a more important role in attentive ratings. For WINDMILL, WASHING MACHINE, and OWL, we find many diagonal lines (on which pre-attentive processing may have focused), while semantic knowledge about the depicted objects may have led to ratings leaning towards vertical orientation. Analogous scatter plot for the features FORM and LINES are similar in nature and can be found in Appendix E.2.

The inspected primitive shape features FORM, ORIENTATION, and LINES seem to describe independent shape qualities, since we found no significant correlations between features. This general finding supports models on visual perception, which assume, that perceptual processing includes the perception of primitives or low-level qualities. Moreover, the perception of FORM and LINES seems to be rather unaffected by the duration of shape perception, since we found no difference between

Summary

attentive and pre-attentive results. This indicates, that the identification of reliable feature information about a shape is very early completed, which is in line with models assuming, that primitive features are identified at a pre-attentive level of visual processing [409]. The impact of viewing duration on the ratings for the `ORIENTATION` feature indicate, that this feature is less stable with respect to pre-attentive vs. attentive processes, a finding, which is in line with the literature.

*Expectations for
SHAPE space*

For our analysis of these features in the context of a similarity space for the `SHAPE` domain, we expect, that meaningful directions corresponding to these features can be found, independent from the underlying rating type (pre-attentive vs. attentive). Moreover, we expect, that the directions based on pre-attentive feature ratings are quite similar to the directions based on attentive feature ratings. Finally, the correlations between features should be reflected by the similarity of the extracted directions – for instance, the directions corresponding to `LINES` and `ORIENTATION` should be orthogonal in the similarity space, because they have been found to be uncorrelated in our analysis. We will compare these expectations to our results in Chapter 11.

10.4 SUMMARY

*Lessons learned: the
dataset...*

In this chapter, we have given an overview of relevant literature on shape perception from different subfields of cognitive science. Moreover, we have presented and analyzed a dataset containing line drawings of everyday objects. This dataset includes both conceptual and visual dissimilarity ratings for a set of 60 stimuli, as well as ratings with respect to the three shape features `FORM`, `LINES`, and `ORIENTATION`. Values for these features have been elicited both based on an attentive and a pre-attentive experimental setting. Moreover, the ratings contained in our dataset have been made publicly available, allowing other researchers to develop their own models of shape perception and to conduct further analyses. In addition to the conceptual spaces approach, which we will present in the following chapter, this may also include feature-based models as considered for example by Schubert and Endres [354].

...and its analysis

Our analysis of the dissimilarity ratings has confirmed, that visual dissimilarity ratings do refer to the `SHAPE` domain, and that they differ considerably from general semantic-conceptual dissimilarity ratings in meaningful ways. With respect to the three shape features, our analyses have confirmed their status as representing foundational aspects of shape perception. This can be seen by the high correlation of attentive and pre-attentive ratings, which indicate, that the features are already formed at a relatively early stage of visual processing, and that they remain largely unchanged later on. The `ORIENTATION` feature seems to be the only one influenced by attentive processes, but it still shows a high correlation between pre-attentive and attentive ratings.

*Open ends:
additional features*

In future work, one could investigate additional feature variants, e.g., by taking further inspiration from artificial systems. Some of

the computer vision features discussed in Section 10.1.3 are strongly related to the psychological features considered in our dataset (e.g., ECCENTRICITY seems to correspond to the FORM feature), but can be extracted in an automated way from input images. It may thus be interesting to analyze, to which extent psychological ratings can be replaced by mathematically defined features, relating also to the study by Morgenstern et al. [297] discussed in Section 10.1.3. Moreover, the computer vision community considers a variety of features, which seem to be different in nature from the psychological features considered in our work. One may thus investigate, to which extent approaches such as spectral transformations of the image are capable of explaining the dissimilarity ratings collected in our experiments.

At this point, we should again point out, that our study used a quite simplified version of the ORIENTATION feature, essentially restricting it to a horizontal-vertical contrast (cf. Section 10.2.3). Moreover, from the perspective of conceptual spaces, one may argue, that ORIENTATION is not part of the SHAPE domain, but rather associated with the overall CONFIGURATION of an object (which also includes its LOCATION, cf. Sections 2.4.3 and 10.1.2). Also the statistically significant impact of viewing duration on the feature ratings as reported in Section 10.3.3 highlight, that more research with respect to the ORIENTATION feature.

*The ORIENTATION
feature*

The dataset presented in this chapter will serve as a basis for our second exemplary application of the hybrid approach proposed in Section 8.7.1: In Chapter 11, we will apply multidimensional scaling (cf. Chapter 8) to the visual dissimilarity ratings in order to extract similarity spaces of different dimensionality for the SHAPE domain. The different candidate similarity spaces will then be analyzed based on both the category structure and the feature ratings from our dataset. In Chapter 12, we will then train convolutional neural networks (cf. Sections 6.2.2 and 10.1.4) on the task of mapping a given input image onto its coordinates in SHAPE space.

Outlook

11

A PSYCHOLOGICAL SIMILARITY SPACE FOR SHAPES

11.1	Obtaining Similarity Spaces with MDS	624
11.1.1	Comparing Mean Dissimilarities to Median Dissimilarities	624
11.1.2	Extracting the Similarity Spaces	626
11.2	Are Distances and Dissimilarities Correlated?	629
11.2.1	Methods	630
11.2.2	Baselines	631
11.2.3	Similarity Spaces	639
11.3	Are Conceptual Regions Well-Formed?	645
11.3.1	Overlap of Conceptual Regions	646
11.3.2	Size of Conceptual Regions	648
11.4	Are There Interpretable Directions?	650
11.4.1	Methods	651
11.4.2	General Observations	653
11.4.3	Individual Features	657
11.5	Summary	661

In this chapter, we use the dataset from Chapter 10 to obtain a psychological similarity space for the *SHAPE* domain. We follow a similar procedure as in Chapter 9 for the *NOUN* dataset, but include a more detailed analysis of the similarity spaces. This analysis is based on three criteria, namely, the correlation between distances and dissimilarities, the well-shapedness of conceptual regions, and the existence of interpretable directions. We thus make use of both Stress-based and interpretability-based evaluation methods as introduced in Section 8.6.

Analyzing SHAPE spaces

We first create similarity spaces of varying dimensionality in Section 11.1 using nonmetric multidimensional scaling (MDS). This also includes the distinction of two aggregator functions: Typical studies involving MDS compute the average dissimilarity ratings across all individual participants by using the arithmetic mean. However, since the original ratings are usually ordinally scaled, the median would be a more appropriate choice (cf. Section 8.1.4). We use both aggregators for constructing global dissimilarity matrices and similarity spaces in order to investigate possible effects.

Obtaining similarity spaces

Like in Chapter 9, we then analyze in Section 11.2 the correlation between the distances in the similarity spaces and the original dissimilarities. In addition to the pixel baseline and the ANN baseline already known from Chapter 9, we also make use of the feature ratings as a third baseline approach for predicting dissimilarities.

Correlations between distances and dissimilarities

Conceptual regions In Section 11.3, we then analyze the category structure of the MDS-based similarity spaces. Since the stimuli from our dataset are grouped into semantic categories, we investigate, whether these categories can be represented by small and convex, non-overlapping regions, as predicted by the conceptual spaces framework.

Interpretable directions Inspired by the research of Derrac and Schockaert [123] (cf. Section 8.6.2), we then use the ratings with respect to the three psychological features FORM, LINES, and ORIENTATION to search for interpretable directions in the similarity spaces in Section 11.4.

Summary Finally, Section 11.5 summarizes the main results from this chapter and gives an outlook on future work.

The research described in this chapter has been conducted in cooperation with Margit Scheibel.¹ Preliminary results have been published in [52, 53], and a publication of our final results is currently in preparation [54]. Both the dissimilarity and feature ratings from our dataset, and the source code necessary for reproducing our similarity spaces and their analysis are publicly available on GitHub [39].² Due to copyright issues, we are unfortunately not allowed to publicly share the line drawings used as stimuli in our dataset. They are, however, available upon request.

11.1 OBTAINING SIMILARITY SPACES WITH MDS

Overview In this section, we use the visual similarity ratings from Chapter 10 to extract shape similarity spaces of various dimensionality. In Section 11.1.1, we discuss the difference between using the mean and the median for aggregating ordinal dissimilarities. We also compare the resulting dissimilarity matrices. In Section 11.1.2, we then describe in more detail, how the similarity spaces were extracted using multidimensional scaling (MDS), and we take a look at the resulting Scree plots as a form of Stress-based evaluation.

11.1.1 Comparing Mean Dissimilarities to Median Dissimilarities

Mean and median aggregation Since the individual dissimilarity ratings from our dataset have been collected using a Likert scale (cf. Section 10.2.2), they are ordinally scaled. This means, that one can only talk about their order, but for example not about distances between them. Therefore, one is technically not allowed to aggregate them with the arithmetic mean, which uses distance computations, and which thus requires an interval scale. However, as already mentioned in Section 8.1.4, many studies use the

¹ As already noted in Chapter 10, Margit Scheibel has designed and conducted the psychological experiments discussed in Section 10.2. She has furthermore conducted several statistical analyses of the collected data (cf. Section 10.3). The extraction of similarity spaces and their subsequent analysis as reported in this chapter has been completely carried out by myself.

² See <https://github.com/lbechberger/LearningPsychologicalSpaces/>.

arithmetic mean for aggregating dissimilarity scores across participants [139, 248, 433]. Instead, one should use the median of the dissimilarity ratings, which assumes only an ordinal scale, and which furthermore tends to be more robust against outliers. However, as the median has a more limited co-domain than the mean (i.e., it produces a smaller amount of unique values), the resulting dissimilarity matrix comes at a lower "resolution".

In order to investigate the effect of the aggregator function (mean vs. median), we aggregated the visual dissimilarity ratings (taking 15 randomly chosen ratings per item pair and inverting the scale by defining $dissimilarity = 5 - similarity$) by using both the arithmetic mean and the median and compared the resulting dissimilarity matrices. We will refer to them as "mean matrix" and "median matrix", respectively.

Using both aggregators

The median and mean matrix differed mainly in two aspects: Firstly, the number of distinct values was much higher in the mean matrix than in the median matrix (59 vs. 5). This comes as no surprise, because the original ratings are based on a five-point Likert scale: The median over an odd number of ratings is thus also a natural number between zero and four. The arithmetic mean, on the other hand, may return any rational number in the interval $[0,4]$.

First difference: number of unique values

Secondly, the number of ties, i.e. the proportion of item pairs, which have exactly the same dissimilarity value, amounted to 1% in the mean matrix, but to 12% in the median matrix. This can be traced back to the aforementioned difference in the number of unique values: Since the median matrix contains only five unique values, ties are much more likely than in the median matrix with its 59 distinct values.

Second difference: number of ties

The number of ties is typically ignored when computing Stress in the context of multidimensional scaling (cf. Section 8.2.2). This also means, that the number of ties in a dissimilarity matrix of fixed size determines, how strongly a possible MDS solution is constrained: Each non-tied pair (x, y) of matrix entries enforces a constraint $x > y$ or $y < x$ on the similarity space. Similarity spaces based on the mean matrix are thus more constrained than similarity spaces based on the median matrix. These additional constraints could on the one hand provide valuable additional pieces of information, which could improve the quality of the similarity spaces. On the other hand, it is also possible, that these additional constraints mainly reflect random noise, which would presumably deteriorate rather than improve the MDS solutions.

Ties and constraints

Figure 11.1 compares the two matrices with a scatter plot of their entries. The correlation between the two matrices as measured with Spearman's ρ [384] (cf. Section 9.1.2) was statistically significant ($\rho = 0.8, p = 0.0$). This can be easily explained by the fact, that both matrices were created from the same raw data. Critically, despite the high correlation, the ordering of items pairs differed in a notable number of cases. There were item pairs with a larger median than mean (e.g., the item pair (TEA POT, STORK): median = 4, mean = 2.8), but also item pairs with a smaller median than mean (e.g. the pair (TEA POT, CAR):

Comparing the matrix entries

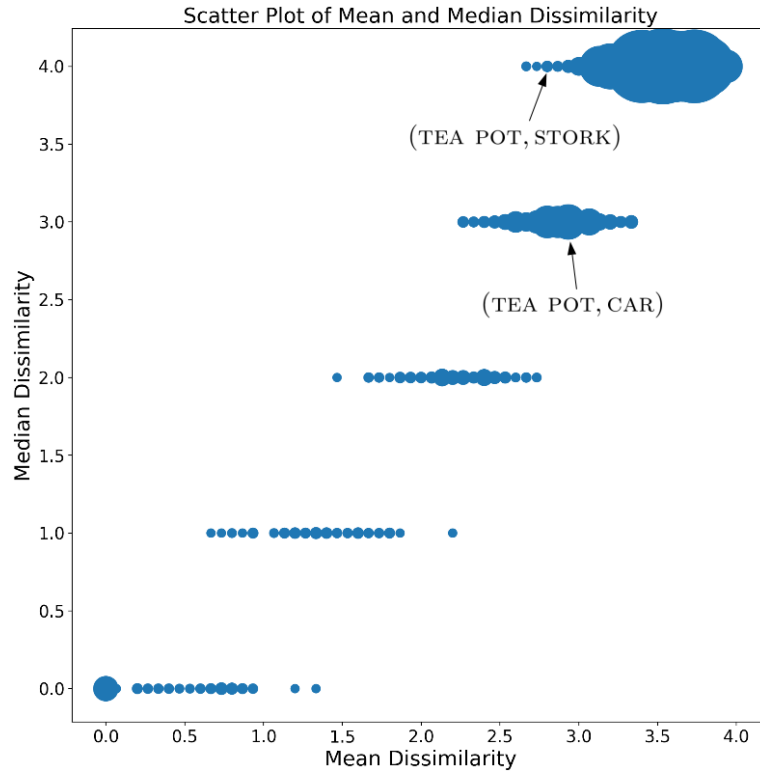


Figure 11.1: Scatter plot for the entries of the mean and median matrix.

median = 3, mean = 3.07). The ordering of these two example item pairs differs between the two matrices. Based on the mean matrix, *TEA POT* should be closer to *STORK* than to *CAR* (since $2.8 < 3.07$), while the median matrix enforces the opposite constraint (since $3 < 4$). This may lead to differences in the resulting similarity spaces.

11.1.2 Extracting the Similarity Spaces

*Using nonmetric
MDS*

As discussed above, the dissimilarity ratings from our dataset were collected with ordinal Likert scales using pairwise comparisons. Due to the ordinal scale, we only considered nonmetric MDS for our purposes. As we have seen in Chapter 9, even in cases, where one can argue, that the dissimilarity ratings are ratio scaled, using metric MDS does not yield any advantages. Therefore, the choice of using nonmetric MDS is unlikely to weaken our overall results.

SMACOF setup

We extracted similarity spaces from both the mean and the median matrix running nonmetric SMACOF [115] (cf. Section 8.5) with 256 random starts and a maximum number of iterations of 1,000.³ In our analyses, we considered similarity spaces with up to ten dimensions. Our setup was thus analogous to the one described in Section 9.1.2 for the NOUN dataset [199], including also the normalization of the spaces according to Kruskal's proposal [234] (i.e., centering the configuration

³ We used the original implementation in R's `smacof` library [116], see <https://cran.r-project.org/web/packages/smacof/smacof.pdf>

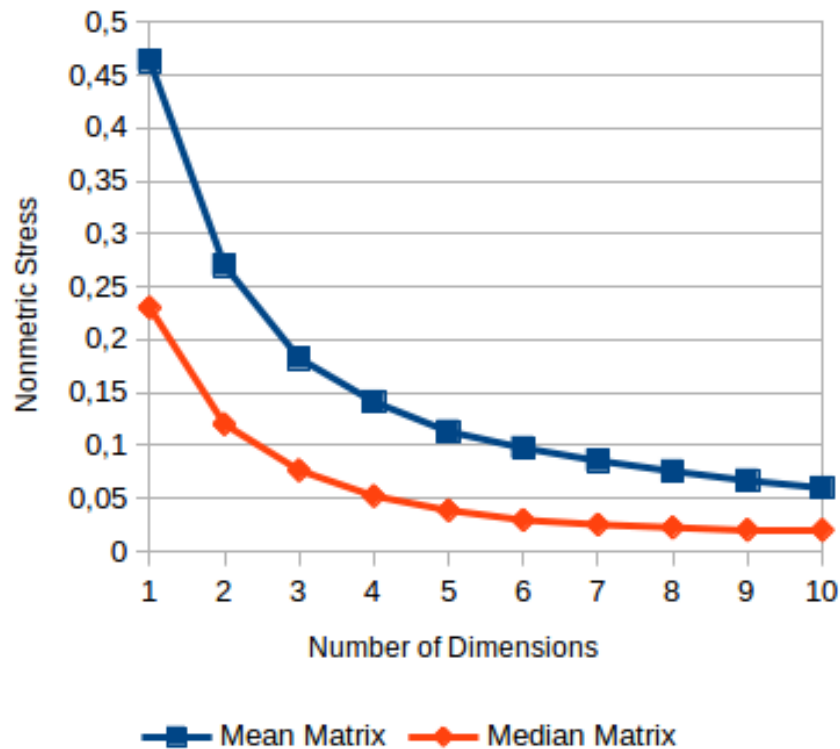


Figure 11.2: Scree plots showing the remaining nonmetric Stress as a function of the number of dimensions. (a) Comparison of mean and median matrices. (b) Comparison of shapes data and NOUN data.

around the origin, and ensuring, that the mean squared distance of the points from the origin equals one; cf. Section 9.2.1).

Figure 11.2a shows a joint Scree plot for both the mean and the median matrix, plotting remaining Stress against the number of dimensions. Possible candidates for elbows in the graph can be identified at two or three dimensions, while Stress seems to level off after five dimensions. This indicates, that spaces with two to five dimensions seem to be the most promising candidates. While the two curves have a similar overall shape, the absolute Stress values are about half as large for the median matrix as for the mean matrix. This does, however, not indicate, that the similarity spaces found for the median matrix are of a higher quality than the ones found for the mean matrix: Stress expresses the degree, to which the constraints from the dissimilarity matrix are violated in the similarity space (cf. Section 8.2.1). Please remember from Section 11.1.1, that the median matrix contains a much larger number of ties, i.e., a smaller number of constraints than the mean matrix. If the absolute number of constraints is lower, also the absolute number of *violated* constraints can be expected to be smaller, resulting in a smaller absolute value of Stress. A comparison of the absolute Stress values between the two matrices is therefore not informative.

Scree plot: mean vs. median

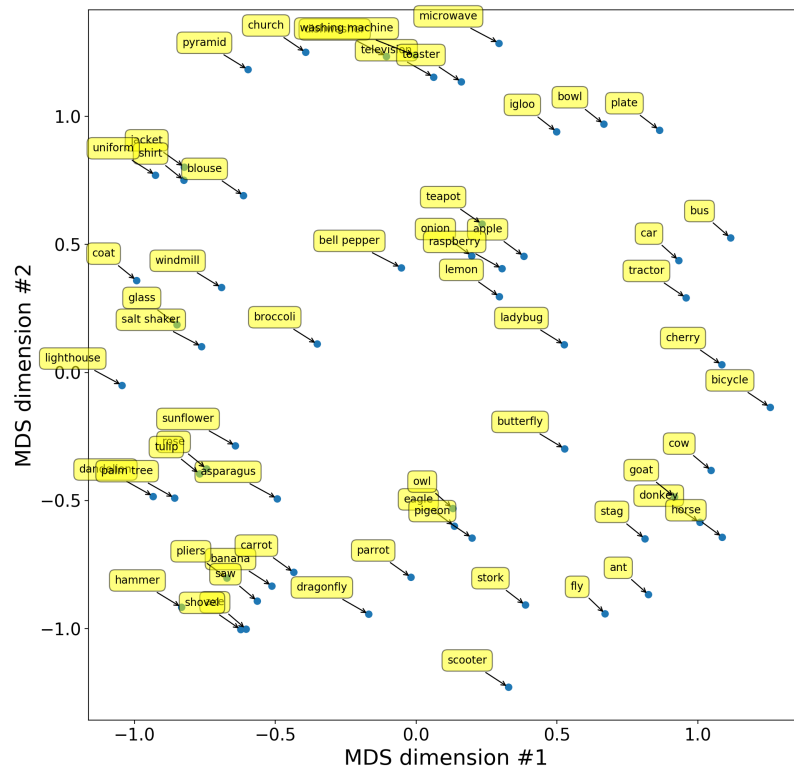


Figure 11.3: Two-dimensional similarity space based on the mean matrix.

*Scree plot: shapes vs.
NOUN*

We can, however, compare the Scree plot of the mean matrix to the analogous Scree plot for nonmetric SMACOF from our earlier study on the NOUN dataset [199] (cf. Section 9.1.3): In both cases, nonmetric Stress on a mean-based dissimilarity matrix was evaluated for the nonmetric variant of the SMACOF algorithm. Figure 11.2b illustrates both Stress curves in direct comparison. As we can see, the two curves are almost parallel and have a very similar general shape. Moreover, nonmetric Stress is slightly, but consistently lower for the shapes dataset than for the NOUN dataset. It thus seems, that the shape dissimilarity ratings are slightly easier to fit with a geometric model. One may speculate, that this is based on the fact, that we focus on a single domain, but other explanations (such as the usage of everyday versus novel objects, or the slightly lower number of stimuli) are also possible. This comparison should therefore be interpreted with caution. Possible future investigations of the conceptual similarity ratings for the line drawings could shed some further light on the observed differences.

Visual inspection

Finally, let us take a look at the extracted two-dimensional similarity spaces shown in Figures 11.3 and 11.4. Both spaces contain a meaningful grouping of items based on their visual appearance with logical transitions between groups. In fact, the rough category-based grouping in the two similarity spaces can be mapped onto each other through a reflection at the angle bisector of the first quadrant (i.e., the line with $y = x$). A first visual inspection of the similarity spaces therefore

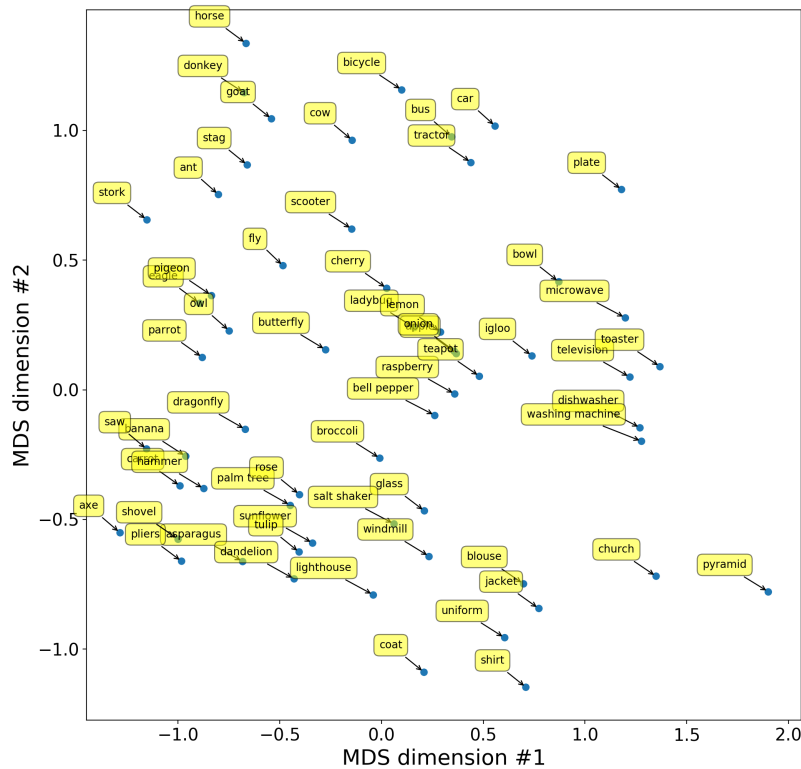


Figure 11.4: Two-dimensional similarity space based on the median matrix.

looks promising. In the following sections, we will analyze these shape similarity spaces in a more detailed, quantitative way.

11.2 ARE DISTANCES AND DISSIMILARITIES CORRELATED?

A central assumption of the conceptual spaces framework is, that large dissimilarities are represented by large distances in the similarity space. In order to assess, whether this expectation is fulfilled for our `SHAPE` spaces, we computed the correlation between distances and dissimilarities, and compared them to three baselines. In Section 11.2.1, we introduce the pixel baseline and the ANN baseline from Chapter 9, as well as a new baseline, which uses the three psychological features `FORM`, `LINES`, and `ORIENTATION` to predict the dissimilarities. Moreover, we describe our general evaluation setup. In Section 11.2.2, we then discuss the results obtained for these baselines, before focusing on the results obtained by our different `SHAPE` similarity spaces in Section 11.2.3. Overall, we expect, that the similarity spaces (especially high-dimensional ones) give a considerably better account of the dissimilarity ratings than our baselines.

Overview

11.2.1 Methods

Our baselines In order to analyze, how much information about the dissimilarities can be readily extracted from the images of the stimuli, we used the two baselines already introduced in Section 9.1.2.

Pixel baseline For our *pixel baseline*, we used the similarity of downscaled images: For each original image (with both a width and height of 283 pixels), we created lower-resolution variants by aggregating all the pixels in a $k \times k$ block into a single pixel (with $k \in [2, 283]$). We compared different aggregation functions, namely, minimum, mean, median, and maximum. The pixels of the resulting downscaled image were then interpreted as a point in a $\lceil \frac{283}{k} \rceil \times \lceil \frac{283}{k} \rceil$ dimensional space.

ANN baseline For our *ANN baseline*, we extracted the activation vectors from the second-to-last layer of the pre-trained Inception-v3 network [400] for each of the images from our dataset. Each stimulus was thus represented by its corresponding activation pattern. While the downscaled images represent surface level information, the activation patterns of the neural network can be seen as more abstract representation of the image.

Feature baseline In addition to this, we used the ratings with respect to the three psychological features FORM, LINES, and ORIENTATION as a third baseline, henceforth referred to as the *feature baseline*. More specifically, we used the scalar values obtained for the psychological features (both for pre-attentive and attentive ratings) as dimensions of a low-dimensional similarity space, and investigated, how well distances in this space reflect the dissimilarity matrix.

Distances For each of the four representation variants (downscaled images, ANN activations, feature values, and points in an MDS-based similarity space), we computed three types of distances between all pairs of stimuli, namely the Euclidean distance d_E , the Manhattan distance d_M , and the negated inner product d_{IP} (cf. Section 9.1.2):

$$d_E(\vec{x}^{(j_1)}, \vec{x}^{(j_2)}) = \sqrt{\sum_{i=1}^N w_i^2 \cdot (x_i^{(j_1)} - x_i^{(j_2)})^2}$$

$$d_M(\vec{x}^{(j_1)}, \vec{x}^{(j_2)}) = \sum_{i=1}^N w_i \cdot |x_i^{(j_1)} - x_i^{(j_2)}|$$

$$d_{IP}(\vec{x}^{(j_1)}, \vec{x}^{(j_2)}) = -1 \cdot \sum_{i=1}^N w_i^2 \cdot x_i^{(j_1)} \cdot x_i^{(j_2)}$$

Dimension weights

We only report results for the best choice of the distance function. For each distance function, we used two variants like in Chapter 9 for the NOUN dataset: One, where all dimensions are weighted equally, and another one, where optimal weights for the individual dimensions were estimated based on a non-negative least squares regression in a five-fold cross validation (cf. [319], who followed a similar procedure).

As an evaluation metric, we computed Kendall's τ [219] (cf. Section 9.1.2) between the distances predicted by the respective baseline and the target dissimilarity matrix:

$$\tau_{ab} = \frac{N_c - N_d}{N}$$

Please recall, that Kendall's τ considers all possible pairs of observations $(a_{l_1}, b_{l_1}), (a_{l_2}, b_{l_2})$. Here, N_c is the number of *concordant pairs* (i.e., pairs with $a_{l_1} > a_{l_2} \wedge b_{l_1} > b_{l_2}$ or $a_{l_1} < a_{l_2} \wedge b_{l_1} < b_{l_2}$), while N_d is the number of *discordant pairs* (i.e., pairs with $a_{l_1} > a_{l_2} \wedge b_{l_1} < b_{l_2}$ or $a_{l_1} < a_{l_2} \wedge b_{l_1} > b_{l_2}$), and N refers to the total number of pairs.

We chose Kendall's τ as a measure for monotone correlation over the more widely used Spearman's ρ , because its definition is based directly on the order of item pairs. This makes an easy interpretation especially with respect to the reversal of item pair orders as described in Section 11.1.1 much more straightforward. Moreover, by focusing only on concordant and discordant pairs, ties are automatically ignored. Therefore, Kendall's τ is especially recommended for small datasets with a large number of tied ranks [155, Section 6.5]. Finally, we have seen in Section 9.1.3, that the same qualitative observations can be made with both Kendall's τ and Spearman's ρ .

When computing Kendall's τ , we considered only matrix entries above the diagonal, because the matrices are symmetric, and because all entries on the diagonal are guaranteed to be zero. Our overall workflow for comparing dissimilarities and distances is illustrated in Figure 11.5 and was applied to both the median and the mean matrix.⁴

11.2.2 Baselines

Figure 11.6 illustrates the maximal correlation we were able to obtain for our various baselines. The expected correlation between randomly drawn configurations of points and the dissimilarity matrices is zero. Therefore, any value considerably above zero indicates a partial success in predicting the dissimilarity ratings. Let us first discuss the big picture from Figure 11.6, before elaborating on the individual baselines.

The feature baseline is able to obtain a correlation of $\tau \in [0.35, 0.38]$ and does not really profit from optimizing the dimension weights. On the other hand, both the ANN baseline and the pixel baseline benefit greatly from the weight optimization procedure. We think, that this difference can be explained based on the dimensionality of the underlying feature space: While the feature baseline is limited to a three-dimensional space, the activation vector of the ANN has 2048 entries. Since the number of dimensions directly corresponds to the number of weights to be estimated, the weight optimization procedure has considerably more degrees of freedom in the latter case.

⁴ Please note again, that this procedure of correlating the entries of dissimilarity matrices from different systems is also known as *relational similarity analysis* [230] in the literature (cf. Sections 9.1.2 and 10.1.4).

Correlation with Kendall's τ

Interpreting Kendall's τ

Motivation for Kendall's τ

Overall workflow

Overview

Effects of weight optimization

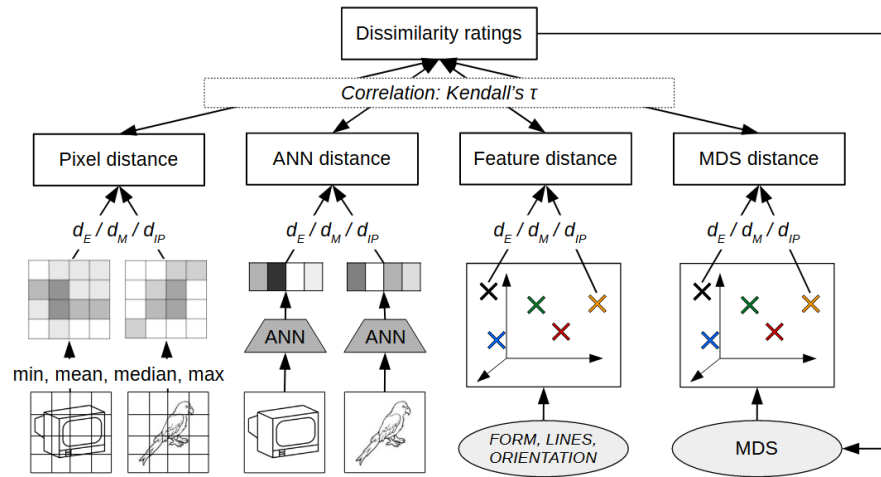


Figure 11.5: Illustration of our correlation analysis setup, which considers distances based on the pixels of downscaled images (far left), activations of an artificial neural network (middle left), values of psychological shape features (middle right), and similarity spaces obtained with MDS (far right).

ANN baseline

In the uniformly weighted case, the ANN baseline performs worse than the pixel baseline. However, when dimension weights are introduced, the ANN baseline becomes competitive with the feature baseline, while the pixel baseline shows a slightly inferior correlation. This observation is in line with the claims by Kubilius et al. [235], that ANNs are able to extract semantically meaningful shape information (cf. Section 10.1.4). However, one should keep in mind, that the ANN baseline still uses a feature space with 2048 dimensions in order to achieve comparable performance to a three-dimensional space based on psychological features, which seems to be very inefficient.

Mean and median matrix

When comparing the results between the mean matrix and the median matrix, the same qualitative effects can be observed. Also the values of Kendall's τ are in the same order of magnitude. However, it seems, that the correlations to the median matrix are always slightly higher except for the ANN baseline with optimized weights. This can, however, be easily explained by the larger amount of ties in the median matrix, which are not considered by Kendall's τ : There are simply less constraints, which need to be fulfilled, and thus fewer errors a predictor can make (cf. the lower Stress values reported in Section 11.1.1).

Considering the feature baseline

Let us now take a closer look at the individual baselines. We begin by considering the feature baseline. Table 11.1 gives a more detailed overview of the results for different variations of this approach. We can make the following observations:

Number of features

The correlation to the dissimilarity ratings always increases as more features are added. The improvement from using a single feature to using two features is considerably higher than the improvement when adding the third feature. Nevertheless, also the latter step leads to a

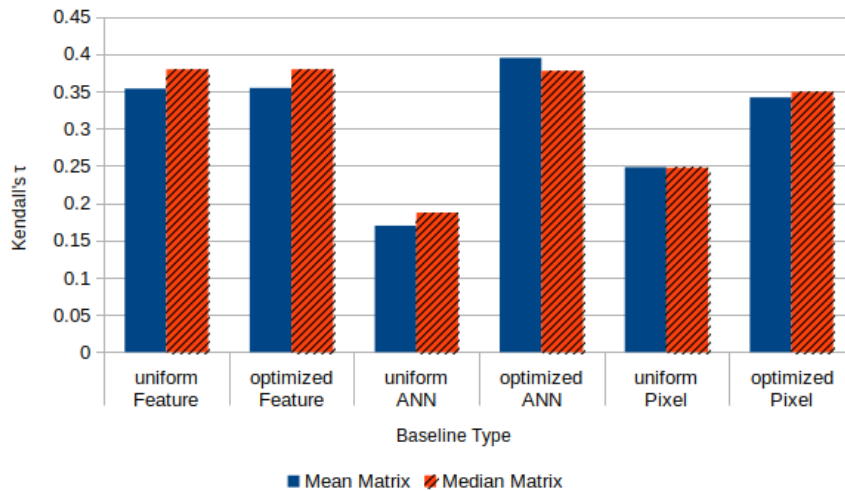


Figure 11.6: Optimal performance obtainable by the three baselines for both the mean matrix and the median matrix, using both uniform and optimized weights.

Matrix	Ratings	Weights	1D	2D	3D
Mean	Attentive	uniform	0.2278	0.3163	0.3535
		optimized	0.2294	0.3157	0.3544
	Pre-attentive	uniform	0.2223	0.3041	0.3507
		optimized	0.2221	0.3031	0.3466
Median	Attentive	uniform	0.2517	0.3357	0.3794
		optimized	0.2517	0.3340	0.3781
	Pre-attentive	uniform	0.2431	0.3207	0.3697
		optimized	0.2423	0.3214	0.3666

Table 11.1: Kendall's τ for different variations of the feature baseline.

notable improvement. In all cases, the best single feature for predicting the dissimilarities is FORM. ORIENTATION is always added as second feature, and LINES always comes last. One may speculate, that this order indicates the relative importance of the three features for the dissimilarity judgments on our stimulus set.⁵ In all cases, the negated inner product d_{IP} produced considerably better results than both the Euclidean and the Manhattan distance.

The inner product $a \cdot b = \|\vec{a}\| \cdot \|\vec{b}\| \cdot \cos \theta$ of two points a and b can be interpreted as a similarity measure, which depends both on the length of the corresponding vectors \vec{a} and \vec{b} and on the angle θ between them. Consider Figure 11.7, where we have two pairs of points (a, b) and (c, d) with the same Euclidean distance, but different distances from the origin. The vectors \vec{a} and \vec{b} are shorter than the vectors \vec{c} and \vec{d} , and

Interpreting the inner product

⁵ In Section 10.4, we remarked, that ORIENTATION may belong rather to the CONFIGURATION than to the SHAPE domain. As we have seen here, it is, however, relevant to predicting our visual dissimilarity ratings, and was hence rightfully included in our list of candidate features.

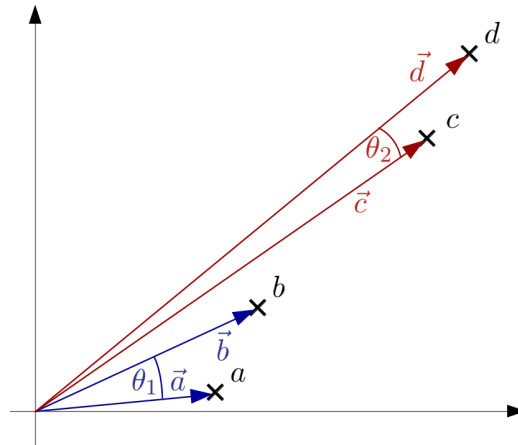


Figure 11.7: Example illustration for the inner product.

the angle θ_1 between \vec{a} and \vec{b} is larger than the angle θ_2 between \vec{c} and \vec{d} . This means that $a \cdot b < c \cdot d$, i.e., the pair (c, d) receives a higher similarity value than the pair (a, b) . Transferring this to our context, the inner product rates an agreement of feature values higher, if these feature values are located on the end of the scale rather than in the middle. In other words, stimuli are judged to be more similar, if they have extreme values on the psychological features, than if they have intermediate values, even if the differences are of equal size. The observation, that the negated inner product worked best in estimating dissimilarities based on the feature ratings, indicates, that the ends of the scales may be more important or reliable than the continuum between them.

Effects of weight optimization

We can furthermore observe in Table 11.1, that weight optimization is not very helpful for the feature baseline – we can only observe some modest improvements, and in many cases the optimized weights even cause a lower correlation to the dissimilarities than the uniform weights. This performance deterioration can be explained by the cross-validation scheme used in optimizing the weights: The overall dissimilarity matrix is repeatedly partitioned into a training set (on which the weights are optimized) and a test set (on which the correlation is computed). The performance deterioration we observed is thus simply the generalization error to unseen data. The observation, that weight optimization does not lead to a considerable increase in the observed correlation values, indicates, that the three features are of roughly equal importance on our dataset: If one feature was more crucial in determining the dissimilarity ratings, giving it a larger weight should considerably improve the predictions of the feature baseline.⁶

Attentive and pre-attentive ratings

When comparing attentive to pre-attentive ratings, one can observe, that using the feature ratings from the attentive perception study yields slightly higher correlation values in all cases. The observed difference is

⁶ Please note, that we consider only a monotone correlation in our analysis, but that the values of this monotone correlation may also be influenced by dimension weights, namely, if the order of pairs changes.

Matrix	Weights	Kendall's τ	Distance Function
Mean	uniform	0.1697	Manhattan
	optimized	0.3948	Euclidean
Median	uniform	0.1872	Manhattan
	optimized	0.3773	Euclidean

Table 11.2: Kendall's τ for different variations of the ANN baseline.

not very large, but seems to be quite consistent. One could potentially explain this by the fact, that the dissimilarities were also collected in an attentive fashion (cf. Section 10.2.2).

Finally, we can observe in Table 11.1, that Kendall's τ is slightly, but consistently higher for the median matrix than for the mean matrix. As already argued above, this can, however, be traced back to the larger amount of ties in the median matrix.

The observation, that the feature baseline performs far from perfect, indicates, that our set of features is not sufficient for accurately representing visual similarity. Introducing additional psychological features might further increase the correlations between the feature ratings and the overall dissimilarities, although we were able to observe a slight trend towards diminishing returns when adding further features in Table 11.1. With respect to the importance of the three given features, no clear picture emerges: While we observed a clear order, in which features were selected (namely, first FORM, then ORIENTATION, then LINES), using weight optimization was not able to increase the observed correlation values. This question about feature importance thus requires further research and additional analyses.

Table 11.2 shows the results obtained for the ANN baseline. Here, the negated inner product has always yielded the worst performance, the Manhattan distance was slightly preferable for uniform weights, and the Euclidean distance yielded the best results for optimized weights. This contradicts the assumptions of the study by Peterson et al. [319], where the inner product was used to obtain similarity judgments from the hidden activations of pre-trained ANNs (cf. Section 8.1.5). For both the mean and the median matrix, we can observe large performance increases through weight optimization. If we use uniform weights, the ANN baseline is the weakest among all baselines. We assume, that this is based on the style of the underlying images: The ANN had been trained on colored images of real objects, whereas the stimuli in our study are greyscale line drawings. However, by weighting the individual entries of the feature vector, we seem to be able to emphasize those parts of the activation vector, which are also useful on the line drawings (e.g., features, which focus on edges in the image), while ignoring irrelevant entries (e.g., features focusing on COLOR or TEXTURE). Finally, the ANN baseline obtains slightly higher correlation values on the median matrix for uniform weights, while reaching higher performance on the mean matrix when considering optimized weights.

Mean and median matrix

The need for more features

Considering the ANN baseline

Matrix	Weights	Kendall's τ	Distance Function	Aggregator	Block Size
Mean	uniform	0.2481	Euclidean	Minimum	24
	optimized	0.3418	Euclidean	–	1
Median	uniform	0.2475	Euclidean	Minimum	24
	optimized	0.3491	Euclidean	Minimum	24

Table 11.3: Kendall's τ for different variations of the pixel baseline.

*Comparison to the
NOUN dataset*

We can compare the results of the ANN baseline on the mean dissimilarity matrix to analogous results of this baseline reported in Section 9.1.3 for the NOUN dataset [199]. There, we observed values of $\tau \approx 0.1660$ for uniform weights (using the Manhattan distance) and $\tau \approx 0.3594$ for optimized weights (using the Euclidean distance). While the correlation observed for uniform weights barely differs between the two datasets, there is a considerable difference for the case of optimized weights. The underlying neural network is the same and had been trained on photographs. Even though the stimuli from the NOUN dataset are closer in nature to photographs than the line drawings we consider in the current study, the network makes better predictions of dissimilarities for our shape stimuli. One may speculate, that by weighting the individual dimensions of the network's representation, we can access the shape information inherent in the neural network (cf. Section 10.1.4). Since the dissimilarity ratings from the NOUN dataset are not confined to shape similarity, it may be more difficult to find an adequate subset of ANN-based features. Again, this hypothesis could be tested by running an additional study on the line drawings with the conceptual rather than the visual dissimilarity ratings – such a study could help to distinguish effects based on the input style (photographs vs. line drawings) from effects based on the type of similarity (shape similarity vs. overall similarity).

*Considering the pixel
baseline*

Table 11.3 summarizes the results obtained by the pixel baseline. In all cases, the Euclidean distance in combination with the minimum as aggregator function yielded the best results. Differences between the median matrix and the mean matrix are mostly negligible. For both matrices, we see considerable improvements when using weight optimization – putting more emphasis on certain parts of the raw image thus seems to be beneficial for explaining the dissimilarity ratings. The optimal configurations for uniform weights are identical for the mean matrix and the median matrix (block size of 24, resulting in downscaled images of size 12×12). When considering optimized weights, the optimal image size does not change for the median matrix, while the optimal results on the mean matrix were obtained when using the images in their full resolution.

*Comparison to the
NOUN dataset*

Again, we can compare the performance level obtained for the mean matrix to the results from Section 9.1.3 on the NOUN dataset [199]. There, we observed $\tau \approx 0.1675$ for uniform weights (using the

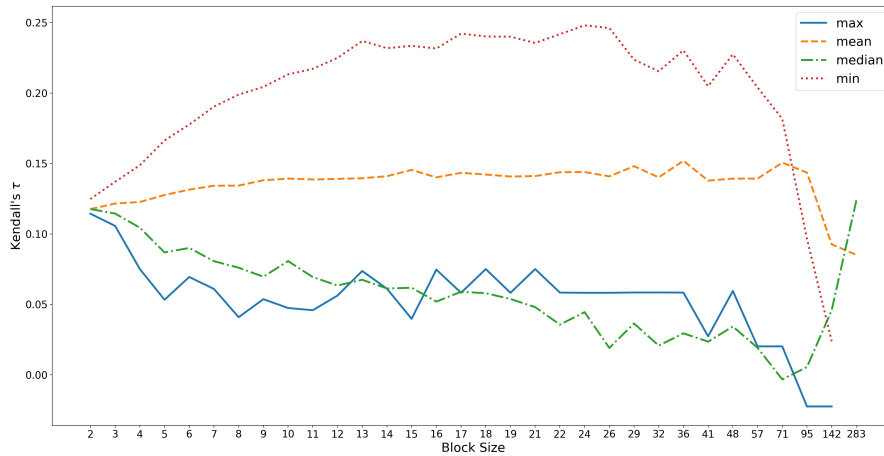


Figure 11.8: Kendall's τ between the pixel baseline and the mean matrix as a function of block size and aggregator function, using the Euclidean distance function with uniform weights.

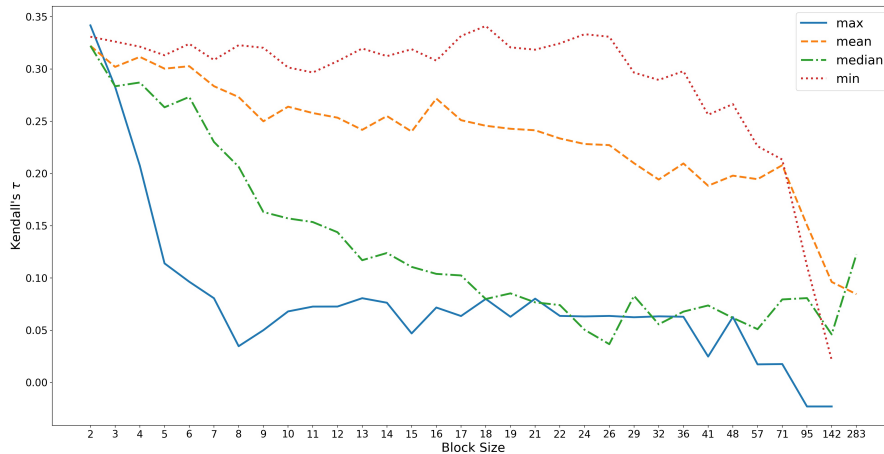


Figure 11.9: Kendall's τ between the pixel baseline and the mean matrix as a function of block size and aggregator function, using the Euclidean distance function with optimized weights.

Manhattan distance, the minimum aggregator, and a block size of 18), which increased to $\tau \approx 0.2608$ for optimized weights (Euclidean distance on the full image, i.e., a block size of 1). Both of these numbers are considerably lower than what we have seen in Table 11.3. One may speculate, that the pixel baseline is simply much more applicable to greyscale line drawings than to photographs, but also the focus on the *SHAPE* domain may have been helpful: The overall shape of an object can be approximated reasonably well by using a downscaled version of the image. Again, further investigations are needed in order to identify the source of the observed differences.

Figure 11.8 visualizes the correlation of the pixel baseline to the mean matrix as a function of the block size for the Euclidean distance function. As we can see, this performance is highly dependent on both the block size and the aggregator function being used. The performance of the

Block size and aggregator for uniform weights

mean aggregator is quite stable with respect to the block size. Both median and maximum perform relatively poorly, especially as the block size increases. The minimum function, however, clearly outperforms the other aggregators. It reaches a peak between block sizes 13 and 32, with both smaller and larger block sizes showing poorer performance. This is in line with our results from Table 11.3, where 24 is identified as optimal block size. It thus seems, that the pixel baseline can profit from smaller image sizes (presumably by removing irrelevant noise), but that it also needs a minimum image size to work properly (such that all relevant information is still contained in the image). Black pixels are encoded with a value of 0, while white pixels are encoded with a value of 255. Therefore, using the minimum when aggregating across pixels can be seen as a form of the logical OR operation: The resulting pixel in the downscaled image will be black, if any of the pixels inside the corresponding block of the original image have been black. The aggregation with the minimum thus results in an image, which represents the overall shape of the object by highlighting which regions are at least partially occupied by the object. This seems to work better than the alternatives of focusing on the average occupation (aggregation with mean and median) or on the regions, which are only completely occupied (aggregation with maximum).

Block size and aggregator for optimized weights

Figure 11.9 illustrates the performance of the pixel baseline for the case of optimized weights. Here, we can make some similar, but also some different observations than for the uniformly weighted case: On the one hand, the relative order of the different aggregators is preserved, i.e., the minimum outperforms the arithmetic mean, which in turn outperforms both the median and the maximum. On the other hand, we do no longer observe a performance improvement with increasing block size. While the minimum aggregator yields relatively constant performance (if the block size is not too large), the performance of all other aggregators clearly deteriorates with increasing block size. Optimal performance is in general observed for very small block sizes, i.e., very large feature vectors. We think, that this effect might be caused by the number of dimension weights, which can be optimized. We would finally like to highlight, that the performance of the minimum aggregator is relatively constant across different block sizes. This means, that also the correlation reached with a block size of 24 is close to optimal and might be preferable due to the smaller representational size.

Another comparison to the NOUN dataset

We can again compare the observed effects from Figures 11.8 and 11.9 to analogous visualizations from Section 9.1.3. There, we observed essentially the same order of aggregators (with the minimum performing best and the maximum performing worst) and also a performance deterioration for large block sizes. However, on the NOUN dataset, we were not able to observe a performance improvement for medium block sizes in the uniformly weighted case – there, the correlation was relatively constant across block sizes for the minimum aggregator. One may again explain this by the focus on general conceptual rather than purely visual similarity – a hypothesis, which should be further investigated with additional experiments and analyses. Additional

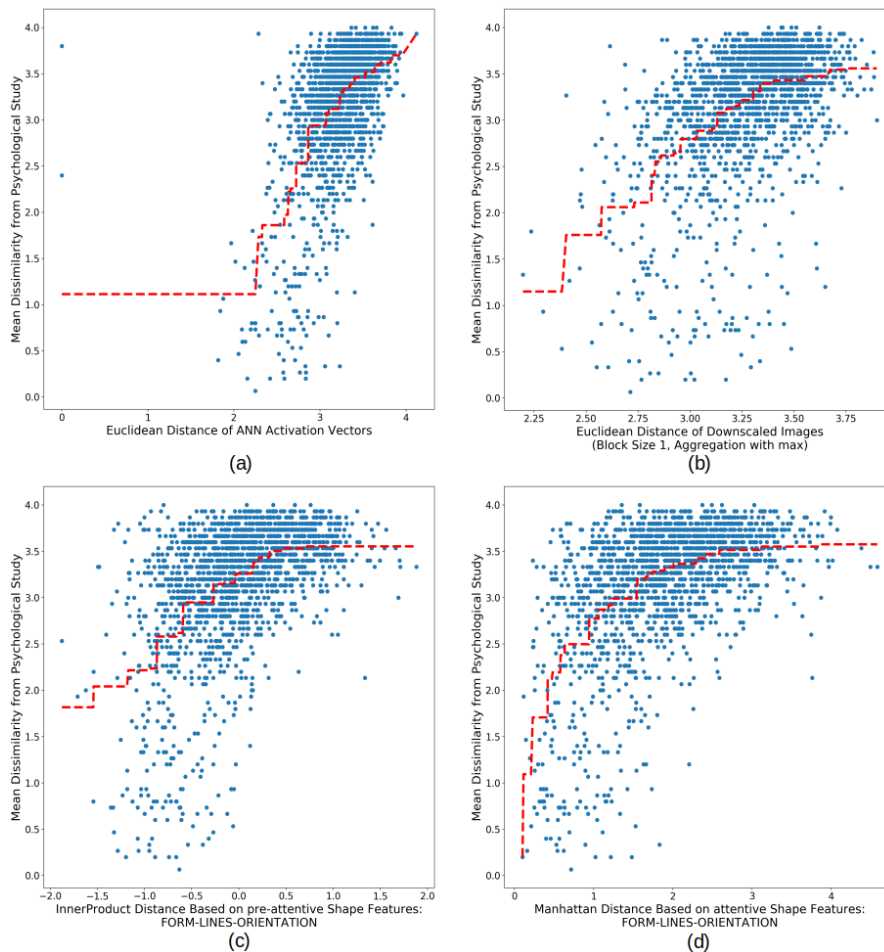


Figure 11.10: Shepard diagrams with respect to the mean dissimilarity matrix for the ANN baseline with optimized weights (a), the pixel baseline with optimized weights (b), and the feature baseline with uniform weights, based on pre-attentive (c) and attentive (d) feature ratings, respectively.

visualizations analogous to Figures 11.8 and 11.9, which consider also the median matrix and alternative distance functions, can be found in Appendices F.1 and F.2.

Finally, Figures 11.10 and 11.11 show Shepard diagrams, illustrating the match between distances as predicted by the baselines and the psychological dissimilarities from the mean and median matrices, respectively. In all cases, we can only observe a weak monotone trend in the data, highlighting, that all baselines are only partially successful in predicting dissimilarities.

11.2.3 Similarity Spaces

After having established the quality of our baselines, let us now turn to the similarity spaces extracted from the dissimilarity matrices. We

Shepard diagrams

Expectations

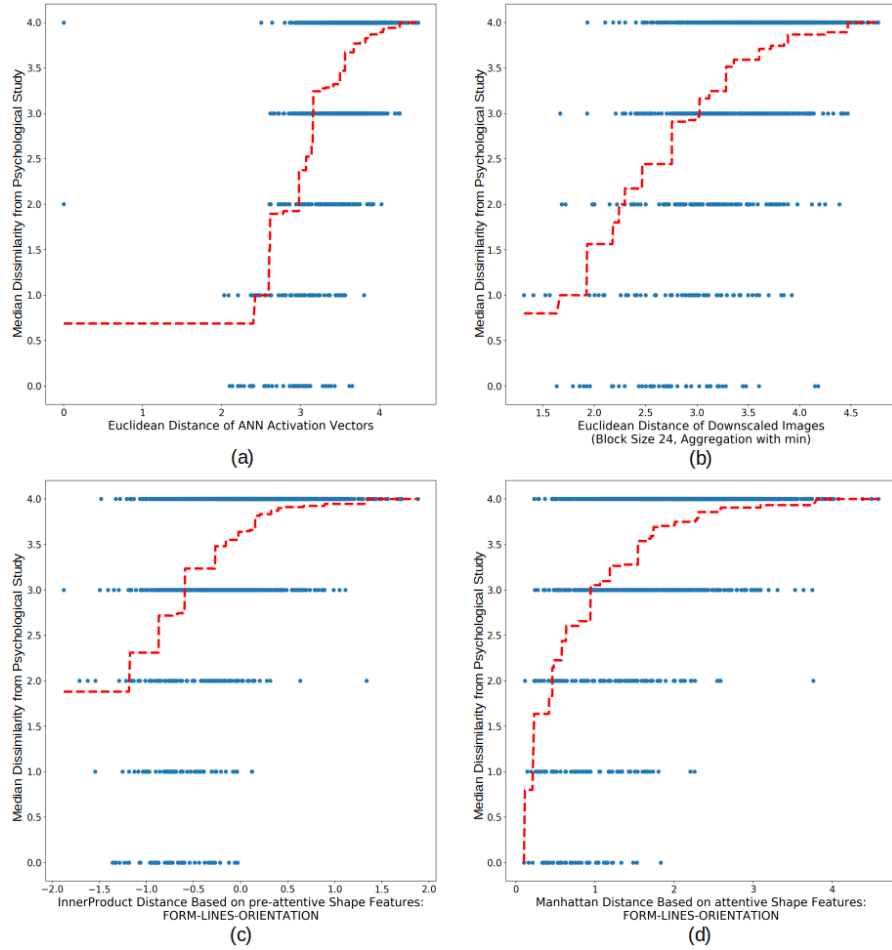


Figure 11.11: Shepard diagrams with respect to the median dissimilarity matrix for the ANN baseline with optimized weights (a), the pixel baseline with optimized weights (b), and the feature baseline with uniform weights, based on pre-attentive (c) and attentive (d) feature ratings, respectively.

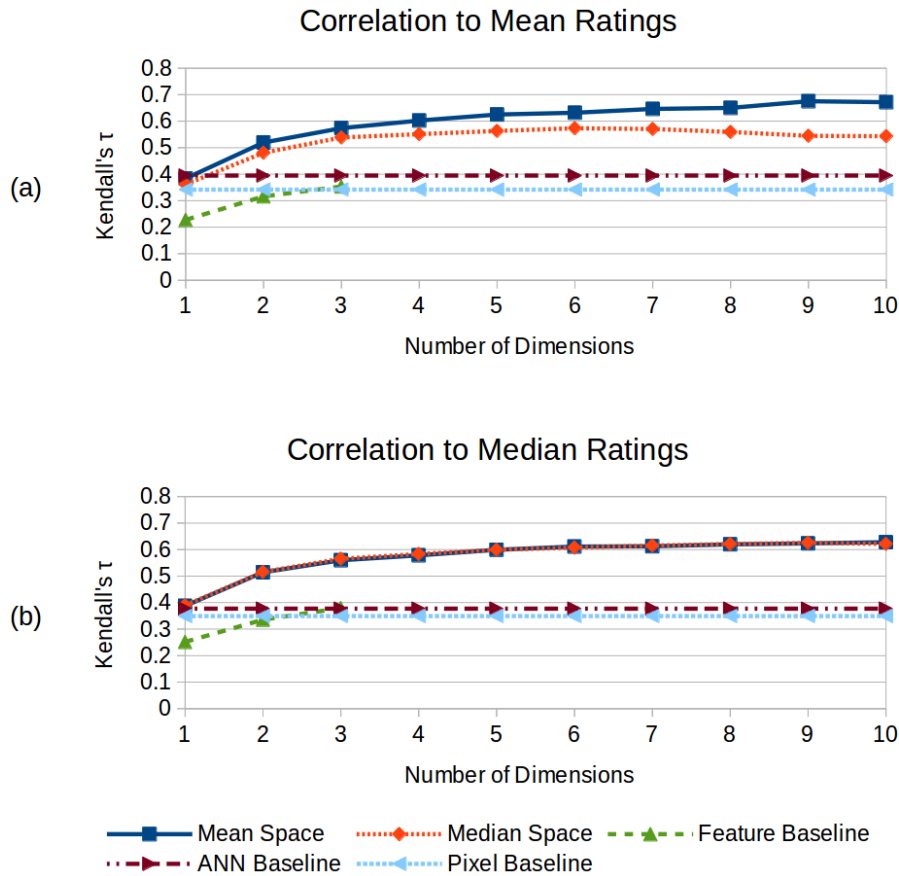


Figure 11.12: Correlation of the different similarity spaces and baselines to the mean matrix (a) and to the median matrix (b).

expect, that all MDS variants perform better than our baselines, and that they profit from a larger number of dimensions. Moreover, we expect overall higher correlation values for the median matrix than for the mean matrix, since the former contains more ties and hence a smaller number of discordant pairs. For each matrix of target dissimilarities, we compared the correlations of both the mean spaces and the median spaces in order to investigate, whether spaces constructed with one matrix correlate well with the other matrix. Kendall's rank correlation between the entries of the two dissimilarity matrices is $\tau \approx 0.70$, which gives an upper performance limit when generalizing between matrices.

For all similarity spaces, we found, that the Euclidean distance resulted in the highest correlations to the dissimilarities. Moreover, optimizing weights did not have any observable effect. Both observations are not surprising, given that the similarity spaces are constructed in such a way, that the unweighted Euclidean distance represents the dissimilarities as accurately as possible.

Figure 11.12 illustrates Kendall's τ as a function of the number of dimensions for both the mean matrix and the median matrix. The correlation curves for the similarity spaces show an elbow at two or

Distance metric and dimension weights

Dimensionality of the similarity space

three dimensions and seem to level off after five dimensions. This confirms our observations from the Scree plots (cf. Section 11.1.2).

*Comparison to the
baselines*

The similarity spaces created with MDS clearly outperform all baselines – already a one-dimensional similarity space is on par with the ANN baseline, which uses a 2048-dimensional feature space. The performance of the feature baseline grows roughly parallel to the performance of the MDS spaces. Its peak performance is, however, only comparable to that of a one-dimensional MDS space. This considerable offset indicates, that the dissimilarity matrices contain more information than represented by the three psychological features. It furthermore seems like MDS is capable of successfully compressing this information into very low-dimensional spaces. This, however, also implies, that the dimensions of the MDS-based similarity spaces combine information from multiple psychologically meaningful features. We speculate, that by adding more features to the feature baseline, we may be able to narrow the gap to the similarity spaces obtained through MDS.

*Comparison to the
NOUN dataset*

A comparison of Figure 11.12a to the results of the NOUN study (cf. Section 9.1.3) shows, that we are able to obtain similar correlation levels (around 0.6 for a five-dimensional space and a bit below 0.7 for a ten-dimensional space). There is, however, one major difference: The one-dimensional similarity space of the NOUN dataset had a very poor correlation of around 0.1 and performed thus worse than all baselines. This effect could not be observed in our shape similarity spaces, where a one-dimensional space performs comparably well as the best baseline.

*Comparison between
target matrices*

When comparing the absolute values achieved on the two different dissimilarity matrices, we observe, that the correlations to the mean dissimilarities are slightly but consistently higher than the correlations to the median dissimilarities. This is somewhat surprising and contradicts our expectations, since the larger number of ties in the median ratings should make them easier to fit, and is thus expected to increase the values for Kendall's τ .

*Predicting
dissimilarities across
aggregators*

Using the spaces based on the mean matrix to predict the dissimilarities from the median matrix works better than the other way around. We think, that this effect is again based on the much larger number of ties in the median matrix: Whenever two entries δ_A, δ_B from a given dissimilarity matrix Δ form a tie, we cannot define, how their corresponding distances d_A, d_B should be ordered. When extracting a similarity space from this dissimilarity matrix, we therefore do not enforce a constraint on this pair of distances (cf. Section 8.2.2). In the resulting space (yielding the distance matrix D), both orderings $d_A > d_B$ and $d_A < d_B$ are thus possible. Moreover, when computing the correlation of a given distance matrix D' to the dissimilarity matrix Δ , the relation between d'_A and d'_B does not influence the overall result because $\delta_A = \delta_B$. This is easy to see, when considering again the definition of Kendall's τ (cf. Sections 9.1.2 and 11.2.1):

$$\tau_{ab} = \frac{N_c - N_d}{N}$$

Here, N_c is the number of concordant pairs, N_d the number of discordant pairs, and N the overall number of pairs. Since $\delta_A = \delta_B$, the pair $(d'_A, \delta_A) - (d'_B, \delta_B)$ is neither concordant nor discordant, and thus does not influence the overall correlation coefficient.

Ties and Kendall's τ

There are many pairs of matrix entries, which are tied in the median matrix, but not in the mean matrix. In the median spaces, the corresponding pairs of distances are therefore not expected to respect any particular ordering. We can safely assume, that for at least some of them, the resulting order will be in conflict with the respective constraint from the mean matrix. When computing the Kendall correlation between the median spaces and the mean matrix, these pairs thus count as discordant and decrease the correlation coefficient. For the mean spaces, we expect, however, a larger number of distance pairs to match the requirements from the mean matrix.

Median spaces and mean matrix

On the other hand, when comparing the correlation of any given similarity space to the median matrix, all ties in the median matrix are disregarded. Even though the mean matrix enforces a certain ordering on these distance pairs for the mean spaces, this does not influence the overall correlation to the median matrix at all. This explains, why we observe no considerable difference between mean spaces and median spaces when comparing them to the median matrix.

Mean spaces and median matrix

The reversal of pairs observed in Section 11.1.1 is not capable of explaining these observations. Please recall, that on average, the mean ratings, which correspond to a given median rating of k , are smaller than the mean ratings, which correspond to a given median rating of $k + 1$. However, there are many examples violating this general trend. These individual conflicts with respect to the ordering constraints are expected to lead to different MDS solutions. Let us assume, that the poor correlation of the median spaces to the mean matrix was caused by the conflicting constraints. Since this conflict is symmetric, it should also impair the correlation of the mean spaces to the median matrix. We are, however, not able to observe this effect, thus the conflicts between the matrices cannot be the main cause of the observed difference. Overall, it thus seems, that the different number of ties has a greater influence on the resulting spaces than conflicting constraints.

Considering conflicting constraints

Finally, let us compare the one-dimensional to the five-dimensional similarity spaces to get a better intuition about how the number of dimensions influences the correlations. Figure 11.13 shows Shepard diagrams for the mean and the median matrix in comparison to the dissimilarities they were based on. In both cases, we see a clear improvement of the monotone trend when adding more dimensions. Clearly, the five-dimensional MDS spaces provide a better monotone trend than any of the baselines (cf. Figures 11.10 and 11.11). Moreover, it is interesting to note, that in the one-dimensional spaces many points are located in the top left corner of the Shepard plot. This means, that in a one-dimensional similarity space, there exist many pairs of points, which have a small distance in the similarity space, but whose corresponding

Shepard diagrams

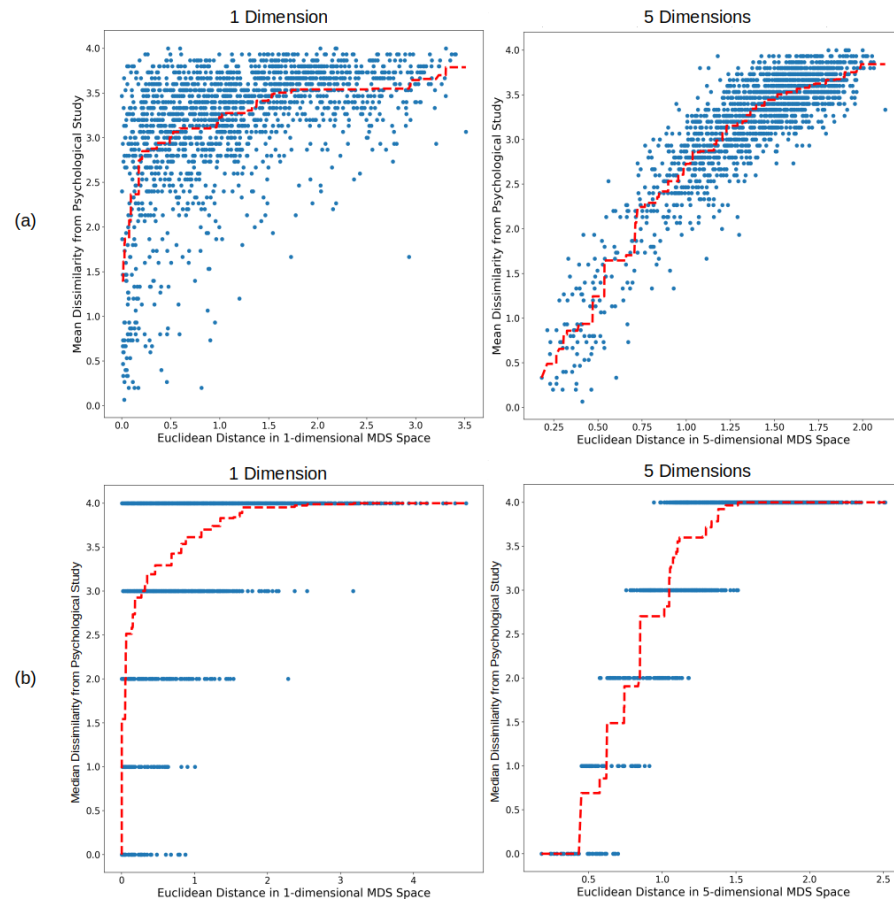


Figure 11.13: Shepard diagrams for the one-dimensional and the five-dimensional similarity spaces based on the mean matrix (a) and the median matrix (b).

items have been ranked as quite dissimilar. In the five-dimensional spaces, this effect has largely disappeared.

We can explain this observation as follows: The items in the one-dimensional space have to be represented by a single scalar value. When arranging the items on this single dimension, the largest distance is achieved between the two items at the ends of the scale. All other items need to be arranged between these two extreme values. Most pairs of items are expected to have at least one item, which is located somewhat close to the center of the scale. These pairs will therefore have a relatively small distance. Overall, we then have many small, but relatively few large distances in the space. As we have seen in Section 11.1.1 when analyzing the dissimilarity matrices, most item pairs are judged to be very dissimilar, and only few are viewed as similar. This inherent mismatch of the distribution of distances and dissimilarities leads to the effect described above. As we add more dimensions to the similarity space, the number of free parameter grows: Points can now differ in multiple dimensions, and the distribution of distances can now match the distribution of dissimilarities much better.

Dimensionality and distances

Overall, the results from our correlation analysis show, that both the mean and the median spaces clearly outperform the baselines. In both cases, spaces with two to five dimensions seem to be good candidates for further analysis. The first main difference, which we observed between mean spaces and median spaces, indicates, that the mean spaces are more successful in representing the median dissimilarities than vice versa. We assume, that this is based on the finer granularity level of the mean matrix, which causes a smaller amount of ties. The second main difference concerns the overall correlation level obtainable on the two matrices. Here, it seems, that higher correlations are possible on the mean matrix than on the median matrix, which is counter-intuitive, since the median matrix contains more ties and thus less constraints. Taken together, these two observations might be interpreted as an argument for preferring the mean spaces over the median spaces.

Summary

11.3 ARE CONCEPTUAL REGIONS WELL-FORMED?

One central idea of the conceptual spaces framework is to represent concepts as convex regions in similarity spaces. These regions typically do not overlap, unless one of them subsumes the other (such as GERMAN SHEPHERD being a specialization of the more general DOG concept). As discussed in Section 10.2.1, the stimuli in our dataset are grouped into twelve categories, half of which can be defined based on a common shape. We therefore expect, that the visually coherent categories are represented by non-overlapping convex regions in the extracted similarity spaces. Since the items from visually variable categories do not have a common shape prototype (cf. Section 10.3.2), we do not have any expectations for the regions representing these categories. In

Overlap of conceptual regions

Section 11.3.1, we will investigate, to which extent these expectations are fulfilled by our similarity spaces.

Size of conceptual regions

In addition to the overlap of the conceptual regions, we can also analyze their size. Again, we expect to find a difference between visually coherent and visually variable categories: Since visually coherent categories consist of objects with a very similar shape, the corresponding points in the similarity spaces should be close to each other, leading to a relatively small conceptual region. On the other hand, the points representing the visually variable categories can be expected to be quite scattered across `SHAPE` space, leading to larger distances and a relatively large conceptual region. We will analyze the size of conceptual regions in Section 11.3.2.

11.3.1 Overlap of Conceptual Regions

Analysis method

In order to measure, to which extent our expectation of non-overlapping convex regions is fulfilled, we have first created a convex hull for each category. We then counted for each pair of categories C_1, C_2 how often an item $i \in C_2$ lies inside the convex hull of category C_1 . By constructing the convex hull, we manually enforce convexity, and by checking for intruders, we test, to which extent the conceptual regions overlap. If a conceptual region is well-formed, then no items from another category should lie within that region. We look both at the overall number of violations of this well-formedness assumption for *all* categories, and at more fine-grained information concerning the distinction into visually coherent (VC) and visually variable (VV) categories.

Baselines

In order to have a meaningful baseline for comparisons, we also computed the expected number of violations for randomly drawn points. More specifically, we used three baseline variants, drawing points from a uniform distribution, a normal distribution, and the set of points in the respective MDS solution. In order to approximate the expected value for each of these baselines, we drew 100 random samples and averaged the number of violations across them.

Expectations

Overall, we expect, that all similarity spaces have a more meaningful structure and thus a lower number of violations than our three baselines. Moreover, we expect the number of violations to decrease with an increasing number of dimensions. Finally, visually coherent categories can be defined based on their shape. They should therefore form non-overlapping convex regions and hence cause less violations than visually variable categories.

Overlap and number of dimensions

Figure 11.14a shows the results from our analysis averaged across all categories. As one can see, even for our baselines, the expected number of violations becomes zero as soon as the space has five dimensions. This can be explained by the fact, that our categories consist of five examples each: It is very unlikely for a randomly chosen point in an n -dimensional space to lie inside the convex hull of n randomly chosen points. Consider for instance a two-dimensional space $[0, 1]^2$

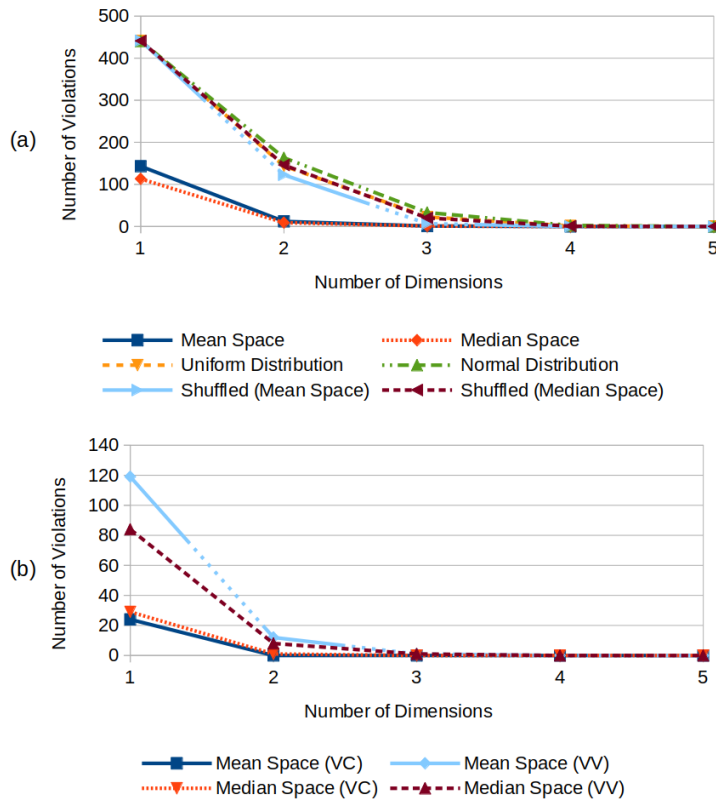


Figure 11.14: Overlap between conceptual regions. (a) Overall number of violations across all categories in comparison to three baselines. (b) Number of violations by category type.

and two points $x, y \in [0, 1]^2$. The convex hull of x and y corresponds to a line-segment (i.e., a one-dimensional structure). When drawing a third point $z \in [0, 1]^2$, it is highly unlikely, that z lies exactly on this line segment. Even if it lies only slightly above or below the line connecting x and y , it will not count as an intruder. The same reasoning applies also to higher-dimensional spaces such as convex hulls of three points in a three-dimensional space (which are in general triangles, i.e., two-dimensional structures), or categories with five examples in a five-dimensional space (as observed in Figure 11.14a).

Even though the number of violations is therefore not very informative for spaces with five or more dimensions, we are nevertheless able to make some observations for lower-dimensional spaces: The strongest difference between the baselines and our similarity spaces can be observed for one and two dimensions, where the similarity spaces produce considerably less overlap between conceptual regions than one would expect by chance. However, already for three-dimensional spaces, this difference vanishes. Moreover, it seems, that conceptual regions overlap slightly less in the one-dimensional median space than in the one-dimensional mean space.

Figure 11.14b distinguishes among the type of category being used for constructing the convex hull. There, we can clearly observe for the one-dimensional and also to some extent for the two-dimensional

Difference to baselines

Effect of category type

space, that visually coherent (VC) categories have a considerably smaller number of intruders inside their convex hull than visually variable (VV) categories. This fulfills our expectations, but the effect vanishes for higher-dimensional spaces: In three-dimensional spaces, only a single intruder is found (in both the mean and the median space for a VV category), and for higher-dimensional spaces, no violations are found for any category type. Moreover, the observed difference between visually coherent and visually variable categories seems to be slightly more pronounced in the mean space than in the median space.

Summary

Overall, we can conclude, that visually coherent categories form non-overlapping regions in our similarity spaces. This is, however, also expected to be the case for a random sample of points, if the space has enough dimensions. We are furthermore only able to observe minor differences between the mean spaces and the median spaces, when it comes to category overlap. In general, our analysis of conceptual overlap seems to be of limited informativeness. If the number of examples per category was considerably larger, one could investigate the robustness of the observed effects.

11.3.2 Size of Conceptual Regions

Analysis method

Let us now investigate, whether visually coherent categories are represented by smaller conceptual regions than visually variable categories. For each category, we have defined the category prototype as the centroid of the points belonging to this category. We have then computed the size of a conceptual region as the average distance of its category members to this prototype. Again, we have compared the numbers observed for the similarity spaces to points randomly sampled from a uniform distribution, a normal distribution, and the set of points in the similarity space. The expected values of these baselines have again been estimated by computing the average across 100 runs.

Category size and number of dimensions

Figure 11.15a plots the average category size as a function of the number of dimensions of the similarity space. First of all, we can observe, that the category size shrinks with an increasing number of dimensions for all of the baselines. This is caused by the normalization of the similarity spaces: All similarity spaces are centered around the origin and are re-scaled in such a way, that the root mean squared distance to the origin equals one [234] (cf. Sections 9.2.1 and 11.1.2):

$$\sqrt{\frac{1}{N} \cdot \sum_{j=1}^N \sum_{i=1}^n \left(x_i^{(j)}\right)^2} = 1$$

Mathematical explanation

This means, that on average, $\sum_i \left(x_i^{(j)}\right)^2 = 1$. If the number n of dimensions increases, then on average, the individual coordinates $x_i^{(j)}$ must become smaller, such that the sum can remain constant. Thus the

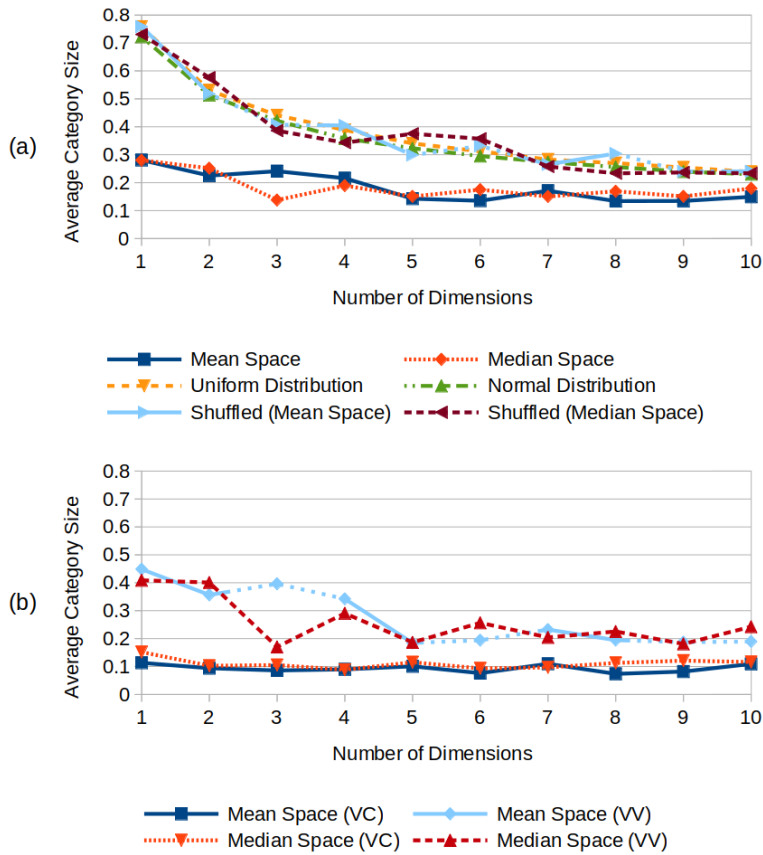


Figure 11.15: Size of conceptual categories, divided into average size of all categories (a), and average size by category type (b).

term $|x_i^{(j_1)} - x_i^{(j_2)}|$ used to compute the Euclidean distance between the points $x^{(j_1)}$ and $x^{(j_2)}$ is also bound to become smaller for each dimension i , resulting in an overall smaller distance between points. Hence, the observed effect of the number of dimensions on the expected category size is to be expected, but differs from the effect observed with respect to category overlap in Section 11.3.1.

In Figure 11.15a, we furthermore observe, that the average category size in the similarity spaces is considerably smaller than one would expect from the baselines. We are not able to observe any systematic differences between mean spaces and median spaces.

Figure 11.15b differentiates between visually coherent (VC) and visually variable (VV) categories. As expected, the average size of the visually coherent categories is considerably smaller than the average size of visually variable categories. Moreover, the size of the visually coherent categories stays relatively constant, and seems to be largely independent from the number of dimensions. The visually variable categories, on the other hand, show a similar pattern as observed for the baselines, which can be again mainly explained through the normalization of the similarity spaces.

Difference to baselines

Effect of category type

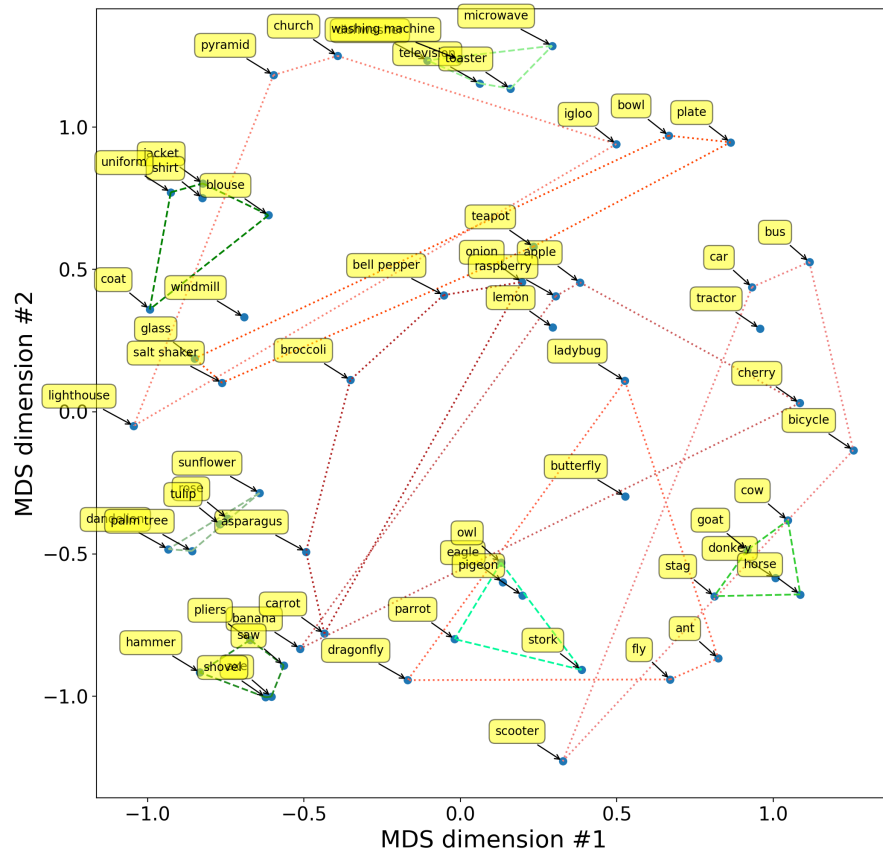


Figure 11.16: Convex hulls of categories in the two-dimensional similarity space based on the mean matrix.

Summary

Overall, we can conclude, that especially visually coherent categories form very small regions in the MDS-based similarity spaces. They are considerably smaller than visually variable categories, which are only slightly smaller than one would expect for randomly generated configurations. Figures 11.16 and 11.17 visualize the two-dimensional similarity spaces as obtained from the mean and the median matrix, respectively. In this visualization, the convex hulls of visually coherent categories are shown as green dashed lines, while the convex hulls of visually variable categories are illustrated with red dotted lines. Our observations from Figure 11.15b with respect to the size of conceptual regions, but also the findings from Section 11.3.1 about their respective overlap can both be verified in this two-dimensional visualization.

11.4 ARE THERE INTERPRETABLE DIRECTIONS?

Motivation

In general, the individual dimensions of a conceptual space should be interpretable. Since the solutions provided by multidimensional scaling are invariant under rotation, we can, however, not expect, that the coordinates of the similarity spaces correspond to meaningful features (cf. Section 8.6.2). Nevertheless, we should be able to find

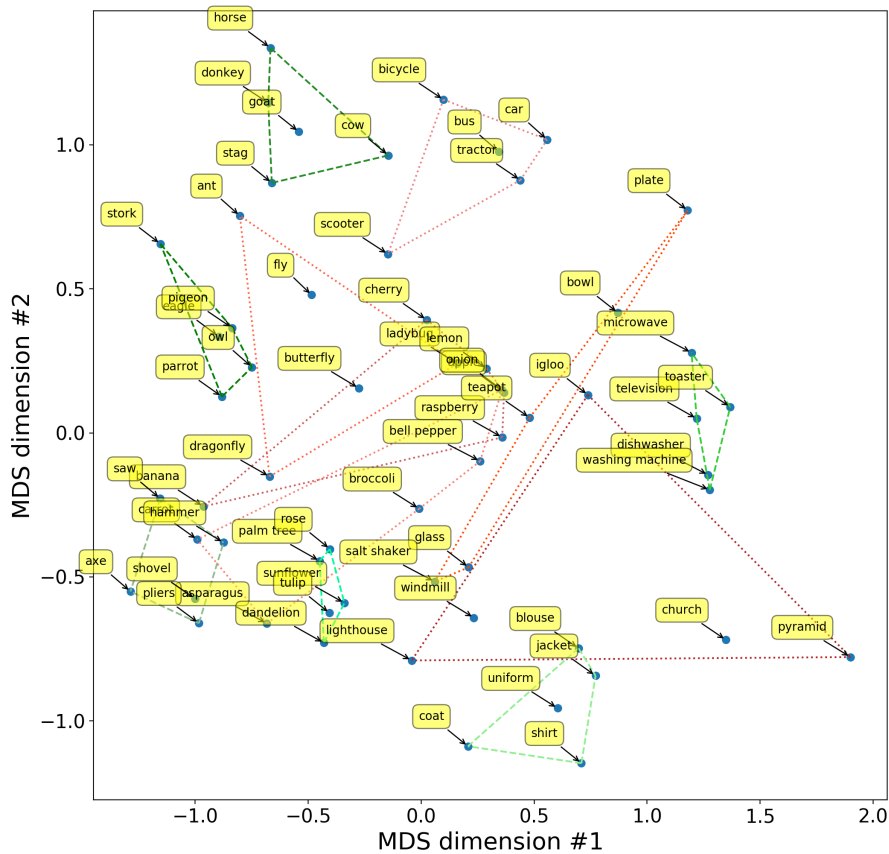


Figure 11.17: Convex hulls of categories in the two-dimensional similarity space based on the median matrix.

meaningful features as interpretable directions in the similarity space. This section is dedicated to searching for directions representing the three psychological features **FORM**, **LINES**, and **ORIENTATION** from our dataset in each of the similarity spaces.

In Section 11.4.1, we describe our overall analysis methods, taking inspiration from the work by Derrac and Schockaert [123]. We then present analyses of global effects averaged across all the psychological features in Section 11.4.2, before comparing the three psychological features to each other in Section 11.4.3.

11.4.1 Methods

In order to find a direction for each of the shape features from our dataset, we followed the procedure proposed by Derrac and Schockaert [123], which has already been introduced in Section 8.6.2. For each of the shape features and for each of the rating types (attentive and pre-attentive), we defined a binary classification problem by using the 15 line drawings with the highest feature values as positive examples, and the 15 line drawings with the lowest feature values as negative examples (cf. Section 10.2.3). Focusing on the scale ends and removing all stimuli with intermediate values can be expected to make the classification

Overview

*Finding directions
through classification*

problem easier. We then trained a linear support vector machine (cf. Section 5.2.3) on this classification problem, and used the normal vector of its hyperplane as a candidate direction for the respective feature.

Finding directions through regression

In addition to this classification-based approach, we also trained a linear regression to map from the points in the similarity space onto the scalar values associated with the feature. Here, we made use of the full set of stimuli. The weight vector found by the linear regression was then interpreted as candidate direction for the given feature (cf. Sections 5.2.1 and 8.6.2).

Baselines

In addition to the mean and median spaces of different dimensionality, we also considered random configurations of points as a baseline. Again, we used the average across 100 random configurations based on a uniform distribution, a normal distribution, and the configurations from the MDS solution.

Evaluation metrics

All candidate directions were then evaluated as follows: We projected the points from the similarity space onto the respective candidate direction, obtaining a single scalar value for each stimulus. On the one hand, we then computed Spearman's ρ [384] (cf. Section 9.1.2) as a measure of monotone correlation between the resulting ordering and the values of the original feature scale. On the other hand, we looked for an optimal threshold on these values, which separates the positive from the negative training examples. Here, we followed again Derrac and Schockaert [123] by using Cohen's κ [108] as evaluation metric. Please recall from Section 5.1.2, that Cohen's κ corrects the classification accuracy with the probability p_e of random agreement:

$$\kappa = \frac{Acc - p_e}{1 - p_e}$$

Evaluation scheme

Please note, that we use the same data points both for finding the directions, and for evaluating their quality, i.e., training set and test set were identical. It is well known, that this evaluation scheme tends to lead to an overestimation of the model's quality. The model should therefore be tested on an independent dataset (cf. Section 5.1.2), which would give a better impression of the model's generalization capability. In our case, a leave-one-out evaluation would be preferable due to the heavily limited number of data points. Like Derrac and Schockaert [123], we have, however, for our current analysis used all data points for both training and testing the models, since we are not interested in the generalization capability of the resulting predictor. The "test on train" methodology is already sufficient for answering the question, whether the feature values can in principle be mapped onto a direction in the similarity space. However, for future work, a more elaborate leave-one-out evaluation should seek to confirm our results.⁷

⁷ Thanks to Viviane Kakerbeck for this criticism.

In addition to the comparison between the extracted directions and the original feature ratings, we have also investigated the pairwise similarity of different candidate directions. In order to measure, whether the different candidate directions for a given feature agree with each other, we computed the average Cosine similarity between all pairs of candidate dimensions for this feature. In order to compare the directions of two different features, we considered all combinations of candidate directions, and averaged their individual Cosine similarities.

Similarity of directions

Please recall from Section 6.3.2, that the Cosine similarity of two vectors $\vec{x}^{(j_1)}$ and $\vec{x}^{(j_2)}$ is computed as their inner product $\vec{x}^{(j_1)} \cdot \vec{x}^{(j_2)}$ divided by the product of their lengths:

The Cosine similarity

$$\text{Sim}(\vec{x}^{(j_1)}, \vec{x}^{(j_2)}) = \frac{\vec{x}^{(j_1)} \cdot \vec{x}^{(j_2)}}{\|\vec{x}^{(j_1)}\| \cdot \|\vec{x}^{(j_2)}\|} = \cos \alpha$$

The Cosine similarity takes values in the interval $[-1, 1]$ and is equivalent to the Cosine of the angle α between the vectors $\vec{x}^{(j_1)}$ and $\vec{x}^{(j_2)}$: Values close to $+1$ indicate that both vectors point into the same direction, while values of -1 are obtained for opposite directions. A Cosine similarity of 0 indicates, that the two vectors are orthogonal to each other.

Interpreting the Cosine similarity

Overall, we expect to find more interpretable directions in higher-dimensional similarity spaces. Also the "confidence" in the extracted directions (measured by Cohen's κ and Spearman's ρ) is expected to increase with the number of dimensions, and to be considerably higher for the MDS-based similarity spaces than for the random configurations. Since we have not observed considerable differences between attentive and pre-attentive feature ratings in Section 10.3.3, we do not expect major differences with respect to the resulting directions. Finally, we expect, that the linear regression achieves a higher correlation (as measured with Spearman's ρ) to the feature values than the SVM, since the latter has only access to the end points of the scale. We do, however, not expect any differences with respect to classification performance (as measured with Cohen's κ).

Expectations for general effects

All candidate directions for the same feature are expected to point into roughly the same direction, which should be reflected by a large Cosine similarity. Since the three psychological features are considered to be relatively independent from each other, and since they have shown only very small correlation values in Section 10.3.3, we expect, that they are approximately perpendicular to each other, reflected by Cosine similarities of zero.

Expectations for individual feature directions

11.4.2 General Observations

Before considering the individual features, let us first investigate some general observations, which we expected to make across all features. More specifically, we will now analyze differences between MDS-based similarity spaces and random configurations, between attentive and pre-

Overview

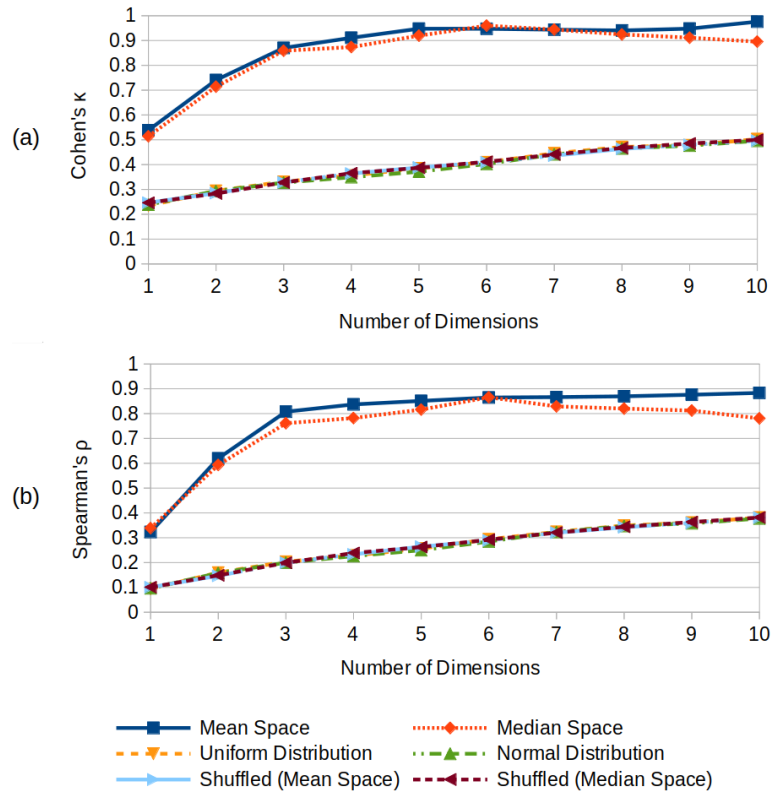


Figure 11.18: Average quality of extracted directions for random configurations and similarity spaces as measured by Cohen's κ (a) and Spearman's ρ (b).

attentive feature ratings, and between the two machine learning models.

*MDS spaces vs.
random
configurations*

Figure 11.18 illustrates the average quality of the extracted directions (aggregated across feature, rating type, and machine learning algorithm) for both the random configurations and the MDS-based similarity spaces. We can observe, that the performance of the different baselines is almost indistinguishable, and that it increases slightly with an increasing number of dimensions. This can be explained by the curse of dimensionality (cf. Section 5.1.4): A larger number of dimensions corresponds to a larger number of free parameters in the machine learning model. If the number of data points remains fixed, while the number of dimensions is increased, the machine learning problem becomes easier to solve, however, often at the cost of decreased generalization to previously unseen input. Since we do not consider generalization to unseen examples in our evaluation (cf. Section 11.4.1), the performance of the baselines increases as further dimensions are added, simply because the probability of randomly sampling a linearly separable configuration of points increases in higher-dimensional spaces. The observed effect is therefore not directly related to the dimensionality-based effects observed for the overlap and size of conceptual regions in Section 11.3.

Further observations

Both the mean spaces and the median spaces perform considerably better than the baseline spaces. The largest amount of improvement with

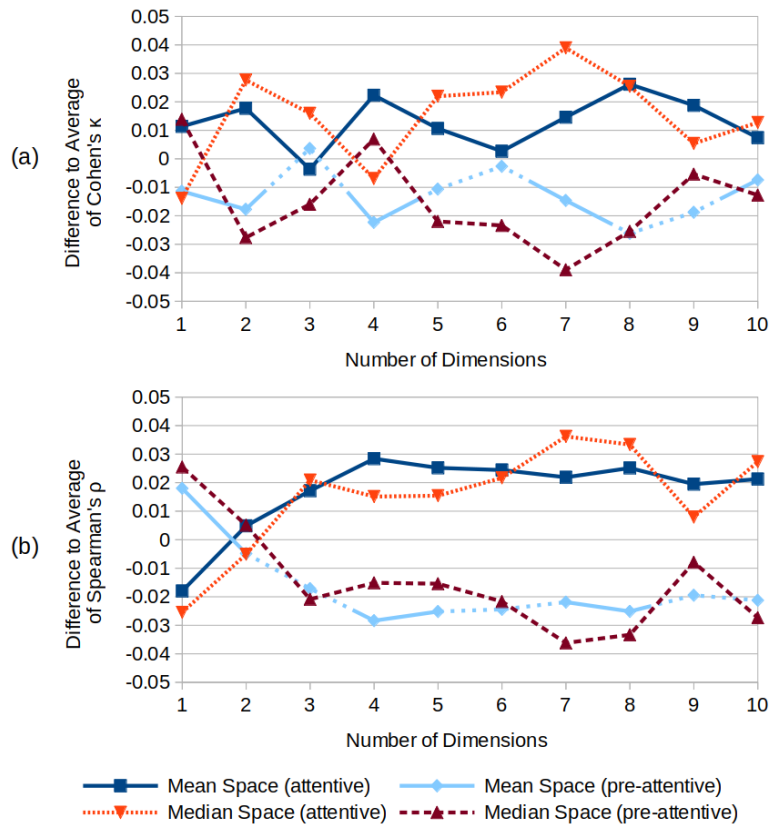


Figure 11.19: Relative average quality of extracted directions for pre-attentive and attentive feature ratings as measured by Cohen's κ (a) and Spearman's ρ (b).

respect to both evaluation metrics takes place up to three-dimensional spaces, and performance levels off at six dimensions. It furthermore seems that the directions can be identified slightly more successfully in the mean spaces than in the median spaces.

In Figure 11.19, we visualized differences between directions extracted based on attentive and pre-attentive ratings, respectively. Again, we averaged across all features and all machine learning algorithms, but this time, we visualized only the difference to the overall average obtained for the respective MDS space. This visualization of differences rather than absolute values has been chosen, because it allows for a more direct comparison, especially given, that all directions yielded quite similar evaluation results. For both Cohen's κ and Spearman's ρ , attentive ratings lead to slightly better results than pre-attentive ratings. This effect is much more consistent for Spearman's ρ than for Cohen's κ . Please recall from Section 11.2.2, that the attentive ratings were also more successful than the pre-attentive ratings in predicting the entries of the dissimilarity matrices. It thus seems, that the ranking obtained from attentive ratings is more closely reflected in the dissimilarity judgments. This may be based on the fact, that also the dissimilarity ratings were obtained in an attentive procedure. However, since the

Comparing attentive and pre-attentive ratings

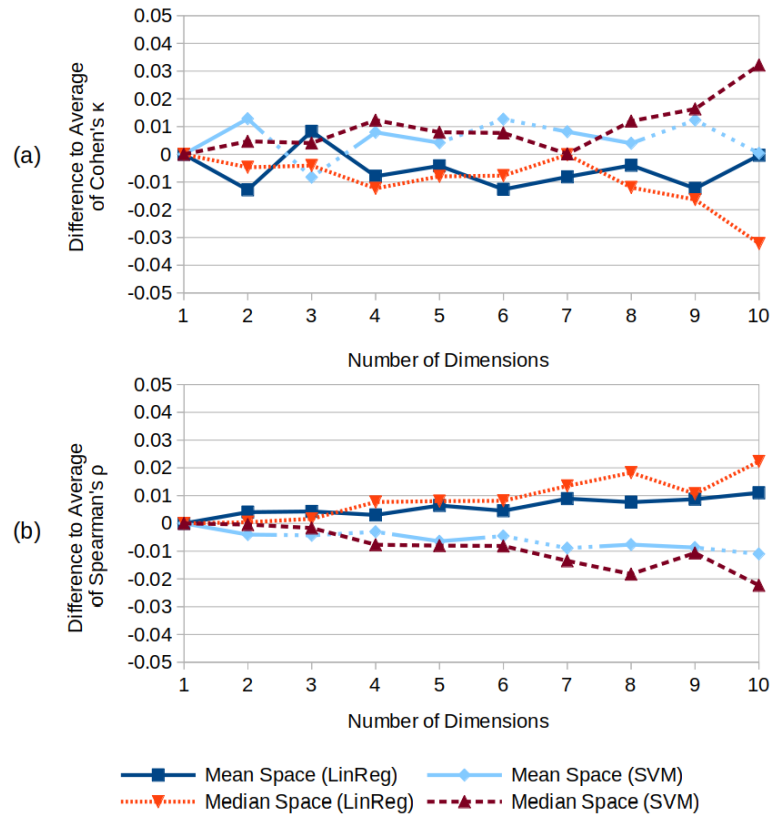


Figure 11.20: Relative average quality of extracted directions for a linear support vector machine (SVM) and a linear regression (LinReg) as measured by Cohen's κ (a) and Spearman's ρ (b).

observed difference is rather small, it may not be of practical relevance.

Comparing machine learning algorithms

In Figure 11.20, we finally compare the results obtained by the two machine learning algorithms to each other, averaging across all features and both rating types, and again showing the difference to the overall average. The results are almost identical, but we can see a very slight advantage for the support vector machine with respect to Cohen's κ , and a slight advantage for the linear regression with respect to Spearman's ρ . These small differences can be explained by the fact, that Cohen's κ evaluates classification performance (which is optimized by the support vector machine), while Spearman's ρ evaluates regression performance (which is optimized by the linear regression). Overall, the differences are, however, negligible.

Summary

In general, we can thus conclude, that most of our predictions have been fulfilled: We observed improved performance for higher-dimensional spaces. MDS-based similarity spaces made a discovery of interpretable directions considerably easier than randomly sampled points. Both with respect to the distinction into attentive and pre-attentive ratings, and with respect to the two machine learning models, we were only able to find negligible differences. Overall, spaces

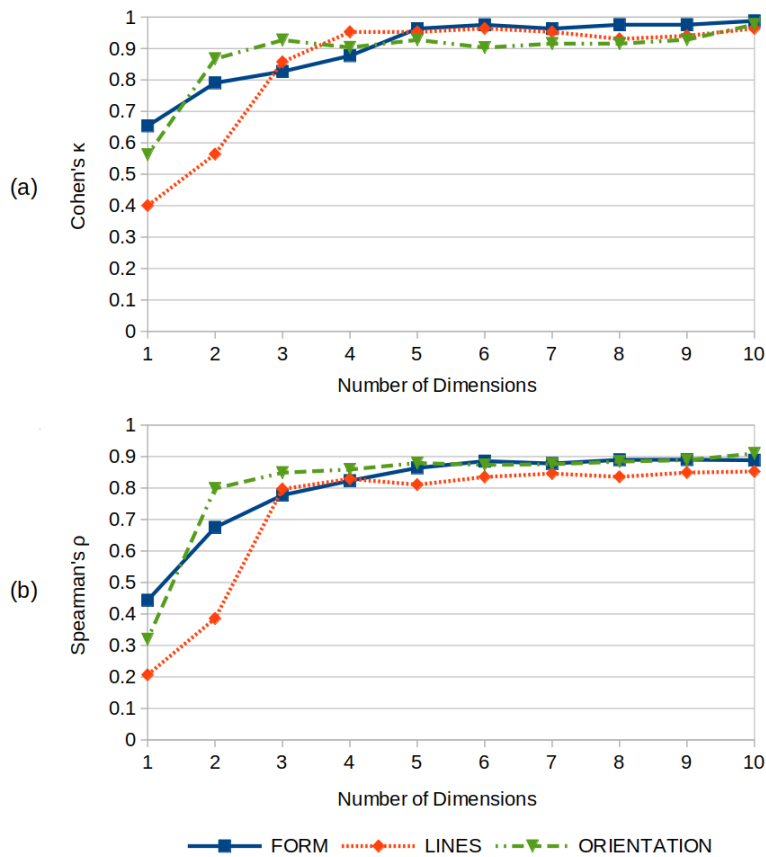


Figure 11.21: Average quality of the extracted directions for the individual features as measured by Cohen's κ (a) and Spearman's ρ (b) for the mean spaces.

with three to six dimensions seem to offer a good trade-off between compactness and expressiveness, which is in line with similar observations from Sections 11.1.2 and 11.2.3. Finally, the mean spaces seem to be slightly superior to the median spaces.

11.4.3 Individual Features

After having considered several global effects, let us now take a look at the individual features. Figures 11.21 and 11.22 show the evaluation results for the different features in the mean and median spaces, respectively, with respect to both Cohen's κ and Spearman's ρ . In all cases, we have averaged across both types of machine learning models (SVM and linear regression) and across both feature types (attentive and pre-attentive).

For the mean spaces (Figure 11.21), we can see, that the features FORM and ORIENTATION are already discovered in a two-dimensional space as indicated by the already relatively high values for Cohen's κ and Spearman's ρ . The LINES feature on the other hand can be reliably identified only from three dimensions onwards. Overall, the results are relatively stable as soon as we reach a five-dimensional space, where

Raw results

Mean spaces

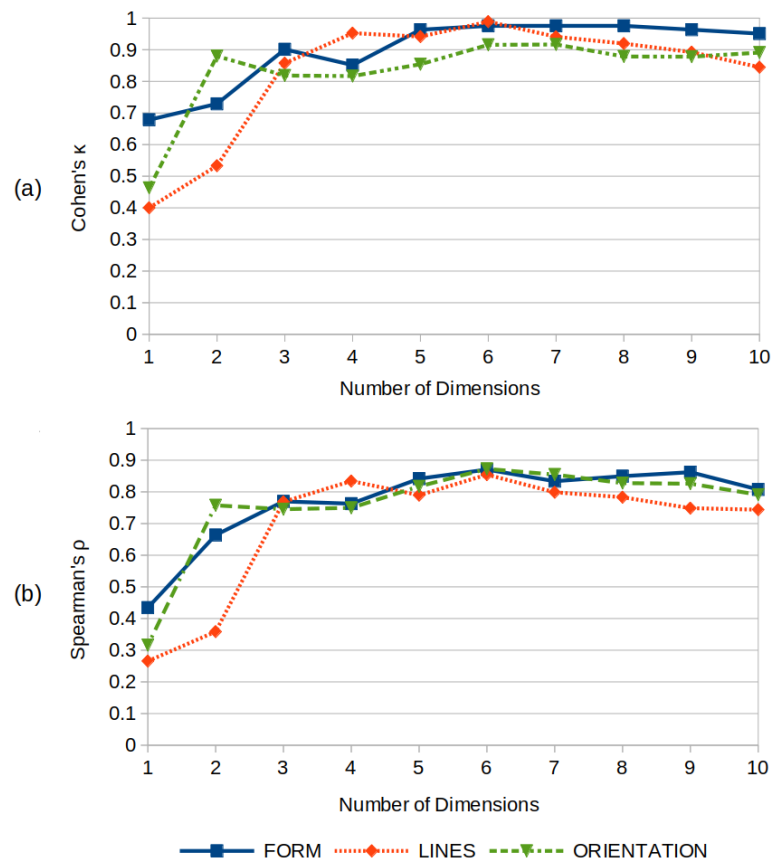


Figure 11.22: Average quality of the extracted directions for the individual features as measured by Cohen's κ (a) and Spearman's ρ (b) for the median spaces.

we observe near-optimal classification and regression performance for all three features.

Median spaces

In the median spaces (Figure 11.22), the features *ORIENTATION* and *FORM* are again successfully identified already in a two-dimensional space, followed by *LINES* in three dimensions. Performance again seems to saturate at five dimensions, reaching slightly lower levels than observed for the mean spaces. Moreover, the quality of the identified directions seems to fluctuate more strongly on the median spaces than on the mean spaces.

Lessons learned

Overall, we can thus reliably identify directions for all three psychological features already in a three-dimensional space, which highlights their status as fundamental aspects of shape perception. The fact, that a suitable direction for the *LINES* feature could only be identified in a three-dimensional space, indicates, that the two other features may be more relevant for our given dataset. This is in line with the observations made about the feature baseline in Section 11.2.2, where features were selected in the order *FORM*–*ORIENTATION*–*LINES*

Stability of the extracted directions

After having compared the quality of the extracted directions, let us now focus on their stability. Table 11.4 compares the different candidate

Space	Number of Dimensions	FORM	LINES	ORIENTATION
Mean Space	1	1.0000	1.0000	1.0000
	2	0.9732	0.9927	0.9938
	3	0.9837	0.9887	0.9891
	4	0.9808	0.9931	0.9900
	5	0.9767	0.9890	0.9801
	6	0.9808	0.9862	0.9741
	7	0.9753	0.9817	0.9657
	8	0.9778	0.9854	0.9717
	9	0.9699	0.9752	0.9627
	10	0.9602	0.9622	0.9713
Median Space	1	1.0000	1.0000	1.0000
	2	0.9647	0.9849	0.9960
	3	0.9844	0.9898	0.9951
	4	0.9826	0.9892	0.9821
	5	0.9754	0.9852	0.9812
	6	0.9706	0.9893	0.9853
	7	0.9760	0.9867	0.9712
	8	0.9668	0.9804	0.9747
	9	0.9630	0.9733	0.9703
	10	0.9394	0.9532	0.9444

Table 11.4: Average pairwise Cosine similarity of the different candidate directions for the three psychological features and similarity spaces of different dimensionality.

directions based on different rating types and ML models for each psychological feature, using the average pairwise Cosine similarity. As we can see, the candidate directions obtained by different machine learning models based on different rating types have a high average Cosine similarity for all features and all numbers of dimensions. This confirms again our observations from Section 11.4.2, where we found only negligible performance differences between attentive and pre-attentive ratings, and between the two machine learning models.

Figure 11.23 illustrates, to which extent the directions for different features point into similar directions, by considering the average Cosine similarity of the directions for all pairs of features.⁸ For both the mean spaces and the median spaces, we are able to make the same observations: From three dimensions onwards, the average cosine similarity is quite stable for all pairs of features, with FORM-LINES hovering around 0.5 (which indicates an angle of about 60 degrees), while the similarity of both FORM-ORIENTATION and LINES-ORIENTATION approaches zero.

Cosine similarity across features

⁸ Please note, that in a one-dimensional space, the Cosine similarity can only take values of plus one and minus one, since the directions can be either identical or opposite.

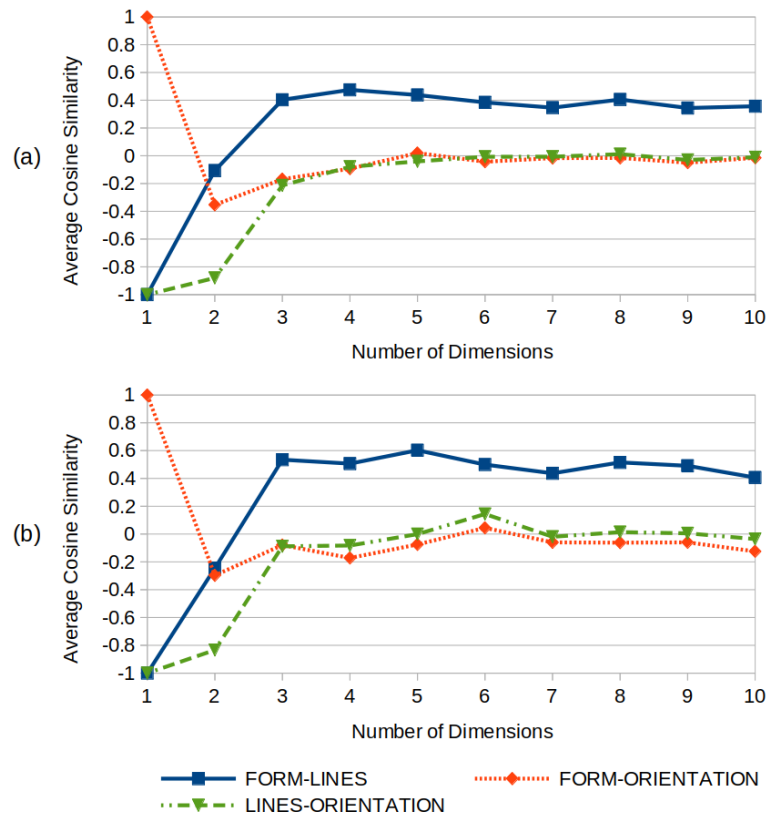


Figure 11.23: Average Cosine similarity of extracted directions for pairs of features based on the mean (a) and the median spaces (b).

Similarity and correlation

In Section 10.3.3, we observed, that the features `FORM` and `LINES` were moderately positively correlated (albeit this effect was not statistically significant), while the two other feature pairs were essentially uncorrelated. This positive correlation of `FORM` and `LINES` on our dataset is obviously reflected in the extracted directions – since stimuli with high values on the `FORM` scale are more likely to also have high values on the `LINES` scale, the two directions are not orthogonal to each other.

Summary

We can summarize our observations as follows: All psychological features can be found in all similarity spaces with at least three dimensions. Extraction performance tends to level off after five dimensions, and directions in the mean spaces tends to be more stable than on the median spaces. The Cosine similarity of the directions of the psychological features matches the correlations of their underlying scales reasonably well. Both the feature type and the model have only a negligible influence both on the resulting directions and their performance.

Visualization

Figures 11.24 and 11.25 illustrate the two-dimensional similarity spaces along with the interpretable directions for `ORIENTATION` and `LINES`. The directions shown in Figures 11.24 and 11.25 are averaged across rating type and machine learning model for all features, which reached a value of $\kappa \geq 0.8$. The directions representing `FORM` and `ORIENTATION` are found in both spaces and seem to be reasonable. The direction for `LINES`

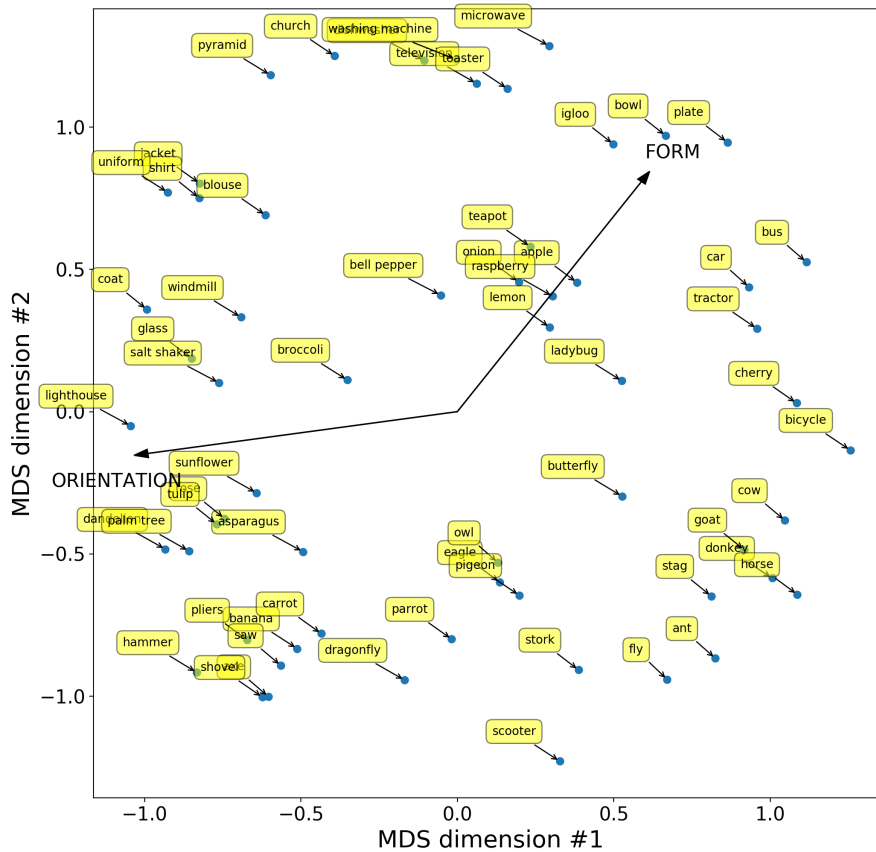


Figure 11.24: Two-dimensional similarity space based on the mean matrix, along with interpretable directions.

is not identified according to this criterion in any of the two-dimensional spaces.

11.5 SUMMARY

With the analysis presented in this chapter, we have shown that it is possible to extract a conceptual space for the *SHAPE* domain from dissimilarity ratings about the holistic visual similarity of complex shapes. This illustrates, that part-whole structures do not need to be explicitly incorporated into the definition of a *SHAPE* space (as done in prior accounts by Marr and Sishihara [278] and Chella et al. [97], cf. Section 10.1.2). We have furthermore seen, that the predictions of the conceptual spaces framework are largely fulfilled – dissimilarities can be represented by distances, shape-based categories are represented by small non-overlapping convex regions, and psychologically meaningful features can be identified as directions in the similarity spaces.

Since conceptual spaces are often obtained via MDS on ordinally scaled dissimilarity ratings, our comparison between the mean matrix and the median matrix may also be informative to other researchers: Even though an aggregation with the arithmetic mean is strictly speak-

A holistic SHAPE space

Mean vs. median aggregation

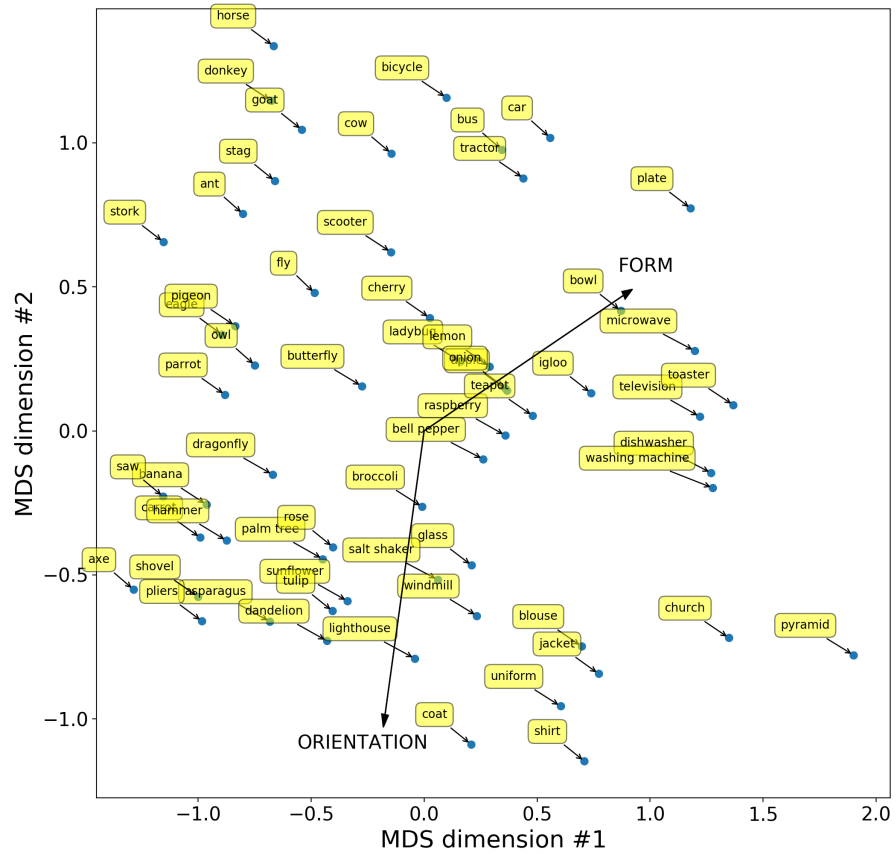


Figure 11.25: Two-dimensional similarity space based on the median matrix, along with interpretable directions.

ing not permitted on ordinal data, the resulting dissimilarity matrix does not have a lower quality than a dissimilarity matrix obtained by taking the median dissimilarities. We can even observe slight advantages for using the mean matrix in some cases, probably based on the fact, that it permits a larger range of possible values for the matrix entries, making the information more fine-grained and avoiding a large number of ties.

The optimal number of dimensions

Another overarching question in this chapter concerned the optimal number of dimensions for a *SHAPE* similarity space. Both the Stress values (cf. Section 11.1.1) and the correlation to the dissimilarities (cf. Section 11.2.3) indicate, that spaces with two to five dimensions are useful candidates for further analysis. Also when looking at the extraction of interpretable directions (cf. Sections 11.4.2 and 11.4.3), we observed the largest improvements up to a three-dimensional space and then a saturation of quality after six dimensions. Overall, it thus seems that three to five dimensions are the preferable choice for our dataset.

Relative importance of features

We have touched upon the relative importance of the three psychological features in multiple points of our analysis. When considering the feature baseline in our correlation analysis (cf. Section 11.2.2), we found, that the *FORM* feature by itself was most predictive, that the

ORIENTATION feature was the best addition, and that the LINES feature was chosen only if all three features were included. When searching for interpretable directions (cf. Section 11.4.3), we observed, that the features FORM and ORIENTATION tended to be identified first, followed by LINES in higher-dimensional spaces. It thus seems, that the LINES feature is less fundamental than the other two features. This is also indirectly supported by the observation, that the pixel baseline performed better for smaller image sizes if uniform weights were used (cf. Section 11.2.2). In such downscaled images, the overall FORM and ORIENTATION of an object can still be represented, but details (like the curvature of individual LINES) are lost.

Although we were able to identify the three shape features as directions in three-dimensional similarity spaces, the feature values themselves were only able to explain the dissimilarity ratings partially, as indicated by the relatively poor performance of the feature baseline in Section 11.2.2. In order to increase the explanatory power of the feature set, additional shape features from the literature should be considered in future analyses. Candidates for such features could include for example SYMMETRY OR LINE INTERSECTION, as proposed by both the RBC model [64] and the early vision study by Treisman and Gormican [409] (cf. Section 10.1.1).

The need for additional features

Furthermore, we have observed various effects of increasing dimensionality for random configurations of points: Both the overlap of conceptual regions (cf. 11.3.1) and their respective size (cf. Section 11.3.2) are expected to decrease with an increasing number of dimensions, while we observed, that the quality of extracted directions increased in higher-dimensional spaces (cf. Section 11.4.2). While these effects can all be explained in different ways, they highlight the dependence of results on the dimensionality of the underlying space. Related, but different effects observable for high-dimensional spaces include the "hubness" phenomenon [271] (i.e., the observation, that some "hub" data points tend to belong to the k nearest neighbors of many other data points) and the instability of proximity information [7] (i.e., the observation, that the ratio of the distances of the nearest and farthest neighbors to a given point approaches one).

Dimensionality-based effects

Just as for our study on the NOUN dataset [199] from Chapter 9, one may criticize, that we only applied multidimensional scaling with the Euclidean distance metric, but not with the Manhattan distance or an intermediate Minkowski metric.⁹ We thus implicitly assume, that the dimensions of the underlying similarity space are integral rather than separable (cf. Sections 1.2.1 and 2.1.1). Since our aim is to investigate the SHAPE domain, while ignoring information from all other domains, this assumption seems to be justified in the context of conceptual spaces. However, re-running our analyses based on Manhattan rather than

Distance metric

⁹ Again, thanks to Corina Strößner for this criticism.

Euclidean spaces would be a good way to double-check the validity of our assumption.

Involved domains

This may be especially useful, given, that the `ORIENTATION` feature may rather belong to the `CONFIGURATION` rather than to the `SHAPE` domain (cf. Section 10.4). The `ORIENTATION` feature was useful for predicting dissimilarities (cf. Section 11.2.2), and that it could be reliably identified as a direction in the similarity spaces (cf. Section 11.4.3). These observations may indicate, that the similarity spaces extracted in this chapter involve information from both the `SHAPE` domain and the `CONFIGURATION` domain, but in a less explicit way than structural approaches (cf. Section 10.1). Extracting spaces based on different distance metrics may shed further light on this issue, since it relaxes the single-domain assumption. For now, we can, however, treat our similarity spaces as a good approximation for the visual similarity of line drawings.

Considering conceptual dissimilarities

Moreover, it may be worthwhile to apply multidimensional scaling also to the set of conceptual rather than visual dissimilarity ratings. Differences with respect to the remaining Stress level, convexity of regions, and the existence of shape features as interpretable directions, but also with respect to the performance of our baseline models may illuminate the relation between shape similarity and general semantic similarity on our dataset. Since our dataset contains ratings for both cognitive and visual dissimilarity, one could furthermore attempt to extract a conceptual space, which explains the cognitive dissimilarity ratings, and which contains a dedicated subspace explaining the visual dissimilarities. This may involve some specialized MDS algorithm capable of incorporating subspace constraints, which relates for example to the work by Jameel et al. [208, 207] introduced in Section 6.3.2.

Outlook

Future work with respect to conceptual spaces for the `SHAPE` domain should seek to apply our procedure to other stimulus sets (e.g., three-dimensional shapes) in order to confirm the observed effects. Moreover, it would be interesting to investigate, to which extent other existing accounts of the `SHAPE` domain within conceptual spaces (e.g., the ones by Marr and Sishihara [278] and Chella et al. [97], cf. Section 10.1.2) are able to predict the shape dissimilarity ratings from our dataset. Finally, since the solutions provided by MDS are limited to the given set of stimuli, using machine learning techniques to automatically map arbitrary input images into the extracted similarity spaces (cf. Section 8.7.1) seems to be necessary for practical applications of the similarity spaces presented in this chapter. In the following chapter, we will investigate the capabilities of deep neural networks for learning such a mapping, re-using and considerably extending our experimental setup from Chapter 9.

12 | LEARNING A MAPPING INTO SHAPE SPACE

12.1	Sketch Recognition	667
12.1.1	Datasets	667
12.1.2	Approaches	670
12.2	Overall Approach	674
12.2.1	Data	675
12.2.2	Architecture	677
12.2.3	Training, Evaluation, and Hyperparameters	680
12.3	Obtaining Baseline Networks	684
12.3.1	Sketch Classification	684
12.3.2	Sketch Reconstruction	689
12.4	Transfer Learning	694
12.4.1	Photograph-Based Classification	695
12.4.2	Sketch Classification	699
12.4.3	Sketch Reconstruction	704
12.5	Multitask Learning	707
12.5.1	Sketch Classification	707
12.5.2	Sketch Reconstruction	709
12.6	Generalization to Other Target Spaces	712
12.6.1	Classification	712
12.6.2	Reconstruction	714
12.7	Summary	719

In the previous chapter, we have extracted similarity spaces of various dimensionality for the `SHAPE` domain, and we have showed, that they are in line with important properties of the conceptual spaces framework. This has essentially been the first step of the hybrid procedure we proposed in Section 8.7.1. In this chapter, we will now implement the second step of this hybrid procedure by training convolutional neural networks (CNNs, cf. Section 6.2.2) to map from raw input images to coordinates in the `SHAPE` similarity spaces.

Our hybrid approach

In Section 9.2, we used a photograph-based network as a starting point for our mapping experiments on the `NOUN` dataset [199]. Since the line drawings from our dataset differ in their characteristics considerably from photographs (e.g., by containing only black lines on white ground), we will also consider sketches as a source domain for our experiments. By comparing a photograph-based network to a sketch-based network, we will evaluate, whether the source domain has a considerable impact on mapping performance. In Sections 8.7.1 and 8.7.2, we argued, that learning the mapping task should be augmented by a secondary training objective in order to harness larger amounts of data and to reduce overfitting tendencies. In addition to the classification objective investigated in Section 9.2, we will also consider a reconstruction

Overall setup

objective in the form of a convolutional autoencoder (cf. Section 6.3.1) as secondary task. Moreover, we consider two different learning regimes (cf. Section 6.2.3): In *transfer learning*, we first train the CNN on its secondary task, and then train a (regularized) linear regression on top of the learned representation. This corresponds to two separate training phases for the two tasks. In *multitask learning*, on the other hand, performance on both the mapping task and the secondary task is being optimized jointly.

*Relation to Sanders
and Nosofsky*

The experiments presented in this chapter thus go beyond both the study by Sanders and Nosofsky [346, 347] and our experiments on the NOUN dataset [199] from Section 9.2: In addition to classification-based transfer learning, we also consider a reconstruction objective and a multitask learning setting. Moreover, our current investigations target a single cognitive domain, while both the work by Sanders and Nosofsky and our experiments on the NOUN dataset considered holistic similarity spaces without distinguishing individual cognitive domains.

Overview

The remainder of this chapter is structured as follows: In Section 12.1, we briefly summarize important datasets and modeling approaches from the area of sketch recognition. Section 12.2 then describes our general machine learning setup as used in our subsequent experiments. This includes a discussion of the data, the network architecture, and the general training and evaluation procedure. We start our practical experiments in Section 12.3 by training neural networks exclusively on the tasks of sketch classification and sketch reconstruction, respectively. For both types of networks, we optimize various hyperparameters in order to identify promising settings for our subsequent mapping experiments. In Section 12.4, we then apply the transfer learning approach already used in Chapter 9 to our pretrained sketch-based networks as well as to the pretrained photograph-based inception-v3 network [400]. Afterwards, we investigate in Section 12.5, whether multitask learning approaches (where multiple objectives are optimized jointly) are able to outperform our simple transfer learners. In Section 12.6, we then analyze the behavior of the different approaches on target spaces of varying dimensionality. Finally, Section 12.7 summarizes the main insights gained from our experiments and highlights some potential directions for future research.

The experiments described in this chapter have been published in [48]. The code necessary for reproducing our results is publicly available on GitHub [39].¹ The two sketch datasets used for training our networks have both been made publicly available by their respective authors², while the line drawings from our psychological study are unfortunately only available upon request due to copyright restrictions. All target similarity spaces are included in our aforementioned GitHub repository.

¹ See <https://github.com/lbechberger/LearningPsychologicalSpaces/>.

² See <http://cybertron.cg.tu-berlin.de/eitz/projects/classifysketch/> for the TU Berlin corpus [143] and <https://sketchy.eye.gatech.edu/> for the Sketchy dataset [348].

12.1 SKETCH RECOGNITION

Sketching can be regarded as a universal form of communication [151], which can potentially be used as a low-threshold visual input modality to computers [143, 442]. For instance, applications like sketch-based image retrieval may allow non-artist users to retrieve images with specific content based on a rough sketch instead of an elaborate natural language query [348].

Importance of sketching

Recognizing free-hand sketches poses, however, a difficult problem and differs from object recognition in photographs in various ways: When making a sketch, humans tend to use iconic, simplified and abstract representations focusing on salient object structures [143, 348, 446]. Moreover, individual object parts may be exaggerated, left out, or caricatured [143, 348]. Furthermore, sketches are inherently diverse, varying widely with respect to their level of detail and abstraction [151, 446]. Moreover, they typically consist of black lines on white ground, focusing only on a single object. This makes them much sparser than photographs, which contain also a wide variety of background and texture [442, 446]. Another crucial difference to photographs is the inherent sequential nature of sketches [151], which can be represented as an ordered list of strokes, while photographs are typically represented based on pixel intensities [442, 446]. Even though sketches are quite different from photographs, humans are still able to correctly recognize them [151, 348].

How sketches differ from photographs

Also the line drawings we considered in our psychological study from Chapter 10 have considerable differences to photographs, mostly with respect to the focus on a single object, the absence of background, and being made up of clearly visible individual lines. Generalizing from photographs to our line drawings may thus pose a more difficult problem than generalizing from sketches. In the following, we present three important datasets of sketches (Section 12.1.1) and give an overview of several neural approaches towards sketch classification (Section 12.1.2). This will then serve as a basis for our own experiments in Sections 12.3 to 12.6.

Overview

12.1.1 Datasets

Xu et al. [442] give a recent overview of neural approaches in the field of sketch recognition. Their overview also contains a short discussion of several datasets. Xu et al. classify them mainly based on their modality: *Single-modal* datasets contain only sketches, and are mainly used for training sketch recognition, retrieval, generation, or simplification tasks. *Multi-modal* datasets on the other hand pair the sketches with other sources of data such as photographs, 3D models of the depicted objects, textual descriptions, or videos. They can thus also be employed for tasks such as sketch-based retrieval of photographs or videos, or for sketch-based video generation. Xu et al. furthermore note, that standard web-crawling techniques, which are often used for gathering datasets of

Classification of datasets

photographs, are not well-suited for creating sketch datasets, because the sequence of strokes cannot be readily extracted from a pixel-based representation. In the following, we introduce the three largest and most popular datasets for sketches.

TU Berlin

The first large-scale dataset of sketches was collected by Eitz et al. [143], and is in the literature referred to as the *TU Berlin* dataset³. It is a single-modal dataset containing 20,000 unique sketches of everyday objects from 250 object categories. When defining the list of object categories, Eitz et al. considered three desiderata: Firstly, they wanted their overall set of categories to be exhaustive, i.e., to cover a wide variety of objects commonly encountered in everyday life. Secondly, the object categories needed to be recognizable based on a sketch. This implies, that rough shape information needs to be sufficient for determining category membership. Thirdly, Eitz et al. only included specific categories such as DOG or VIOLIN, but not superordinate concepts such as ANIMAL or MUSICAL INSTRUMENT in order to limit the visual variability within each object category.

Data collection

Sketches were collected from 1,350 participants through Amazon Mechanical Turk. Participants were instructed to only draw the outline of the object (i.e., no filled regions) without including any context around the object. Eitz et al. report a median drawing time of 86 seconds, and a median number of 13 strokes per sketch. For each of the 250 categories, Eitz et al. collected 90 sketches. After manually inspecting and cleaning the complete dataset (i.e., removing sketches, which were in the wrong category, or which did not follow the instructions), they truncated the dataset, leaving 80 sketches in each category. The sketches are available in a pixel-based format (png files with a resolution of 1111×1111 pixels) and in a stroke-based format (svg files, where each stroke is encoded as an individual spline).

Human classification performance

Moreover, Eitz et al. tested the ability of human participants to correctly classify the remaining sketches based on the 250 object categories. In order to make this task easier for the participants of their study, they arranged the categories into a hierarchy with 6 top-level super-categories and 27 mid-level categories such as ANIMAL or BUILDING. Eitz et al. found, that on average, humans reached a classification accuracy of 73.1%. They furthermore observed large differences between different categories, where confusions seemed to be most frequent between semantically similar categories such as PANDA and BEAR.

The Sketchy database

Sangkloy et al. [348] have published the *Sketchy* database⁴, which contains both sketches and photographs. In total, this multi-modal dataset contains 12,500 photographs in 125 categories and 75,471 sketches based on these photographs. Sangkloy et al. argue, that the TU Berlin dataset contains very iconic sketches, where the participants used poses and viewpoints, which are easy to draw or canonical. In order to increase

³ See <http://cybertron.cg.tu-berlin.de/eitz/projects/classifysketch/>.

⁴ See <https://sketchy.eye.gatech.edu/>.

the variety of poses and viewpoints, Sangkloy et al. primed their participants with a photograph of the object to be sketched, and encouraged them to adopt the pose and viewpoint from the photograph in their own sketch. When determining the set of object categories, Sangkloy et al. followed the same procedure as Eitz et al. [143] by considering an exhaustive set of recognizable and specific categories. Their set of categories is much smaller, but has a considerable overlap of 98 categories with the TU Berlin dataset.

Sangkloy et al. considered about 70,000 photographs from ImageNet [120] as initial candidates for their Sketchy dataset. These photographs were rated by a single volunteer according to a subjective five-point "sketchability" scale, which was intended to reflect the difficulty of sketching the object of interest based on this photograph. For each of their 125 categories, Sangkloy et al. kept 100 photographs (40 rated as "very easy", 30 rated as "easy", 20 rated as "average", 10 rated as "hard", and 0 rated as "very hard"). Like Eitz et al. [143], they used Amazon Mechanical Turk for collecting sketches. More specifically, they prompted participants with a particular photo for two seconds, which disappeared afterwards. Participants were then asked to sketch the object of interest with a pose similar to the one seen in the image, but without including any context or shading of regions. For each photograph, Sangkloy et al. collected five sketches from different participants.

In a manual post-processing step, Sangkloy et al. labeled each sketch AS CORRECT, AMBIGUOUS, INCORRECT POSE, ENVIRONMENTAL DETAILS, OR ERRONEOUS. Overall, 64,560 sketches were labeled as CORRECT. In contrast to the TU Berlin dataset, all other sketches were kept in the dataset as well. When analyzing the participants' sketching behavior, Sangkloy et al. observed a median sketching time of 85 seconds and a median number of 14 strokes per sketch, which is very similar to the numbers reported for the TU Berlin dataset. The sketches from the dataset are available both in a pixel-based representation (png files with a resolution of 256×256 pixels) and a stroke-based format (svg files).

The most recent dataset of free-hand sketches is called *QuickDraw*⁵ and was collected by Ha and Eck [186]. This single-modal dataset contains 50 million sketches for objects in 345 categories. These sketches were collected as part of a web-based game, where players were asked to draw objects belonging to a particular object class in less than 20 seconds. The players' goal was to draw the object in such a way, that a machine learning system was able to correctly classify it into the given category. For each of the categories, the dataset provides 70,000 training examples, 2,500 examples for validation and another 2,500 examples for testing. The sketches are available in a stroke-based format (ndjson files with metadata and individual strokes) as well as in a low-resolution pixel-based format (numpy bitmaps with a resolution of 28×28 pixels). The sketches from the QuickDraw dataset are very noisy, since players had only 20 seconds for making each sketch, and since

Data collection

Labeling

QuickDraw

⁵ See <https://github.com/googlecreativelab/quickdraw-dataset>.



Figure 12.1: Illustrations of a parrot: (a) Our dataset⁶. (b) TU Berlin [143]. (c) Sketchy [348]. (d) QuickDraw [186].

Criterion	TU Berlin [143]	Sketchy [348]	QuickDraw [186]
Number of Sketches	20,000	75,471	5,000,000
Number of Categories	250	125	345
Type of Sketch	Free	Based on Photograph	Free
Drawing Time	86 sec.	85 sec.	≤ 20 sec.
Strokes per Sketch	13	14	Unknown
Quality Control	Filtered	Labeled	None

Table 12.1: Comparison of the three sketch datasets.

the collected sketches have not been manually cleaned [216]. Moreover, players may not have finished a partial sketch, if it has already been classified correctly.

Comparison

Table 12.1 gives a systematic comparison of the three datasets, and Figure 12.1 compares an exemplary line drawing from our psychological study to images from the respective category of the three datasets. The QuickDraw dataset is considerably larger than the other two datasets, but due to the time pressure involved in its collection, it contains very large amounts of noise. It can thus be considered to be the poorest match to the images from our dataset, which tend to be much more detailed. Both the TU Berlin dataset and the Sketchy dataset were collected in similar ways and have similar properties with respect to drawing time and number of strokes. Moreover, they come with a manual quality control, which ensures that the sketches are associated with the correct category and follow the specifications. We therefore will use sketches from both TU Berlin and Sketchy for our experiments.

12.1.2 Approaches

Classical machine learning

Early approaches to sketch recognition were mostly based on hand-crafted computer vision features, which were originally designed for photographs [446] (cf. Section 10.1.3). For instance, Eitz et al. [143] used

⁶ Image license CC BY-NC 4.0, source: <http://clipartmag.com/cockatiel-drawing>.

features reflecting the line orientation within a small local neighborhood, and trained both a support vector machine (cf. Section 5.2.3) and a k nearest neighbors classifier (cf. Section 5.2.5) on the resulting feature space. More recently, Abdelfattah and Zakaria [1] used a stroke-based approach by first detecting visual primitives such as ellipses and polygons and then using their relations (e.g., with respect to size and position) as input to an SVM-based classifier. Since we aim to use sketch classification as a secondary task for a neural network, which also maps images onto points in a psychological similarity space, we will in the following focus on neural approaches to sketch classification.

First neural approaches to sketch classification simply re-used existing photograph-based architectures such as AlexNet [232], which was the first convolutional neural network (CNN, cf. Section 6.2.2) to beat the state of the art on ImageNet [120]. AlexNet (illustrated in Figure 12.2a) uses five convolutional layers, max-pooling with overlapping areas (pooling width of 3 and stride 2), and three fully connected layers. It has 60 million parameters in total and consists of 650,000 different neurons. While being considered a very large network at the time of its creation, it has been surpassed by other architectures such as GoogLeNet [399], which consists of 11 convolutional layers (most of which run convolutions of different sizes in parallel) and a single fully connected output layer.⁷

*AlexNet and
GoogLeNet*

Yu et al. [446] considered a three-fold cross validation on the TU Berlin dataset [143], where the best non-neural approaches were able to achieve 68.9% accuracy. A support vector machine trained on the features extracted from a pretrained AlexNet achieved an accuracy of 67.1%, while the same network structure trained exclusively on the sketch data achieved a slightly higher accuracy of 68.6%. This is still below the human performance level of 73.1% classification accuracy [143]. Sangkloy et al. [348] later used a pretrained version of AlexNet as a baseline for their own system, and further fine-tuned its weights on the TU Berlin dataset [143]. Using an 80-20 train-test split, they observed an accuracy of 77.29% of the fine-tuned AlexNet. Applying the same procedure to the deeper GoogLeNet resulted in an even higher accuracy of 80.85%. Both of these network architectures are thus in principle able to surpass the human performance level with respect to sketch classification.

*Applying
photograph-based
networks on sketches*

Sketch-a-Net [445, 446] was the first CNN specifically designed for the task of sketch recognition [442]. Like AlexNet, it consists of five convolutional layers, max-pooling with overlapping areas (pool width of 3 and stride 2), and three fully connected layers (see Figure 12.2b). However, its overall number of parameters is with 8.5 million considerably smaller than for the original AlexNet. This reduction in the number of parameters is obtained by using fewer kernels, smaller fully-

*Sketch-a-Net as a
variant of AlexNet*

⁷ Please note, that GoogLeNet is the predecessor of the inception-v3 network [400] (cf. Section 6.2.2) used in Chapters 9 and 11, and in some of our subsequent experiments in Sections 12.4.1 and 12.6.1.

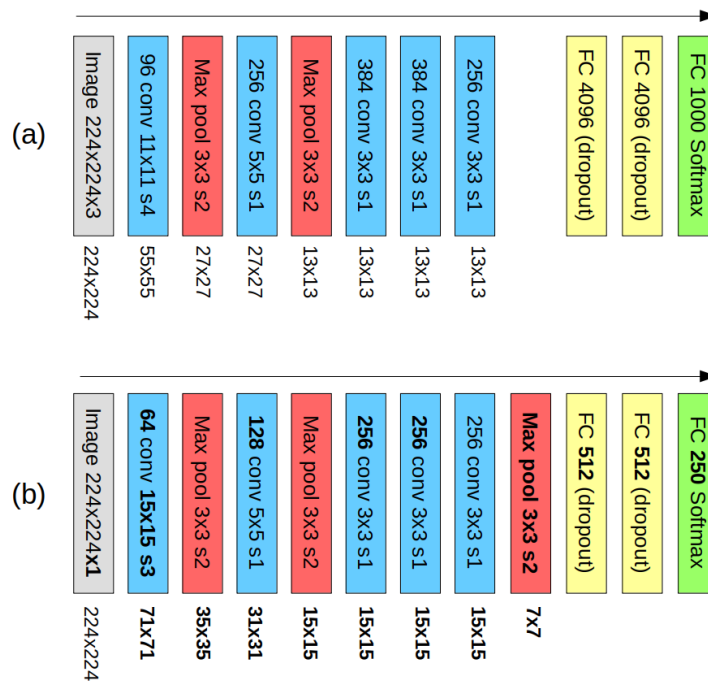


Figure 12.2: The overall structure of AlexNet [232] (a) and Sketch-a-Net [446] (b). Differences are highlighted in boldface.⁸

connected layers, and an additional max pooling layer to reduce the representation size. Yu et al. [446] emphasize, that the overall network structure is similar to standard photograph-based CNNs, since the number of filters increases with depth, the stride of convolutional layers is set to one for all but the first convolutional layer, and zero-padding is used for layers, where the image size is not supposed to change. The largest difference with respect to classical CNN architectures consists of the filter size used in the first convolutional layer: While AlexNet uses kernels of size 11×11 , and while other architectures such as GoogLeNet use even smaller kernel sizes, Yu et al. decided to use kernels of size 15×15 . They argue, that sketches lack texture information, and that one therefore needs to consider larger image patches in order to define low-level features.

In their first version of Sketch-a-Net [446], Yu et al. modified the basic network architecture in two ways in order to capture stroke order and multiple input scales. In order to model stroke order, they split the underlying strokes of each sketch into three sequential groups. For each group of strokes, they created an individual pixel representation, and used this as one channel of the network's input. Instead of using a single grey-scale input channel, they thus used six input channels (the three individual groups, the whole sketch, and combinations of two subsequent groups). This did not change the network's overall

First version: input channels and downsampling

⁸ Abbreviations: "96 conv 11×11 s4" = "convolutional layer with 96 kernels of size 11×11 and stride of 4", "Max pool 3×3 s2" = "max pooling layer with pool size 3×3 and stride of 2", "FC 4096 (dropout)" = "fully connected layer with 4,096 neurons, trained with dropout". Image sizes are shown next to the layers.

structure and added only a relatively small number of weights to the first convolutional layer. In order to incorporate multiple input scales, Yu et al. used downsampling and subsequent upsampling to create blurred versions of each input image. For each of their five downsampling sizes (256, 224, 192, 128, and 64 pixels), they trained an independent copy of the overall network. The resulting ensemble of networks used Bayesian fusion to obtain an overall prediction for the given input. In a three-fold cross-validation on the TU Berlin dataset [143], the resulting architecture obtained an accuracy of 74.9%, making it the first machine learning approach to beat humans with their reported 73.1% classification accuracy on this task.

Later, Yu et al. [445] have removed the multichannel, multi-scale aspects of the architecture. In the second version of Sketch-a-Net, they have dealt with stroke order and input scale through data augmentation steps (cf. Sections 6.2.3 and 7.2.1): In addition to translating, rotating, and reflecting the overall sketch, they progressively removed individual strokes from the overall sketch, where later and shorter strokes had a higher probability of being deleted. As an additional means of data augmentation, they further deformed the resulting sketch variants both on the local level (by slightly modifying individual strokes themselves) and on the global level (by deforming the convex hull of the overall sketch). The resulting updated version of Sketch-a-Net used the same network architecture, but needed less input channels and no ensemble of networks. This simplification of the model, however, comes at the cost of a more complex data preprocessing stage. After pretraining their model on edge maps extracted from ImageNet [120], Yu et al. reported an improved performance of 77.95% accuracy on TU Berlin [143].

Second version: data augmentation

Albeit being the first and most well-known example of sketch-based CNNs, the work of Yu et al. is not the only approach in this direction. For instance, Seddati et al. [357] have also trained a convolutional neural network on the TU Berlin dataset [143], reaching an accuracy of 75.4% in a three-fold cross validation. Their architecture consists of five convolutional layers and a single fully connected output layer, using a relatively small kernel size of 7×7 in their first convolutional layer. They observed, that larger kernels just before the final classification layer improved results, which they explain as sketch data requiring the network to take into account larger overall structures in the input. This is similar to the arguments by Yu et al., who, however, focused on the earlier network layers when increasing the size of the receptive field. Seddati et al. furthermore noted, that zero-padding of the original image was helpful in achieving better performance. They analyzed the representation learned by their network by using a bottleneck autoencoder (cf. Section 6.3.1) to reduce the dimensionality of intermediate layers to 1024 entries. When looking at the k nearest neighbors of a given sketch in these compressed feature spaces, Seddati et al. found, that lower layers seemed to focus on local features, while middle layers encoded global shape information, and higher layers represented

An alternative CNN architecture for sketches

semantic similarity. This is in line with similar findings discussed in Section 10.1.4 in the context of CNNs and shape perception.

*Supplementing
CNNs with a
shape-based network*

Zhang et al. [457] have noted, that shape descriptors are widely used in standard computer vision approaches (cf. Section 10.1.3), but that they are typically not learned by CNNs. They thus proposed to augment the high-level representation learned by a classical CNN with information extracted from a shape-focused neural network. This latter network did not take a pixel image as input, but a randomly selected set of points from the strokes making up the overall sketch. This set of points was then mapped into an aligned space using an input-dependent 2×2 affine transformation matrix in order to account for rotations and different scales, before being fed into a fully connected feedforward network. In the beginning, both networks were trained independently on a classification task. After training, Zhang et al. simply concatenated the features of the penultimate layers from both networks, and trained a support vector machine on top of this hybrid feature space. They reported an accuracy of 84.4% on the TU Berlin dataset [143] and an accuracy of 82.7% on the Sketchy database [348].

SketchRNN

Ha and Eck [186] were among the first to use recurrent neural networks for processing sketches. Their *SketchRNN* architecture corresponds to a sequence-to-sequence variant of the variational autoencoder framework introduced in Section 6.3.4. Instead of taking only a single fixed input, their recurrent neural network processes one stroke at each time step, keeping track of its internal hidden state over time. Overall, their model maps an input sketch to a single latent vector, which can then again be decoded into a sequence of strokes. In their experiments on their QuickDraw dataset, Ha and Eck found, that the learned latent code was able to create meaningful interpolations between sketches. Moreover, it also supported simple vector arithmetics as also reported for word embeddings (cf. Section 6.3.2).

*Recurrent neural
networks*

Starting with the publication of SketchRNN, most sketch recognition studies (e.g., [86]) have focused on recurrent neural networks, which interpret their input as a temporal sequence of individual strokes [442]. Since the line drawings used to create our SHAPE spaces are, however, only available in pixel-based format, we will use a CNN-based approach as a basis for our experiments.

12.2 OVERALL APPROACH

Overview

As stated before, we use sketches as additional data for training our neural networks, since the number of line drawings from our psychological study is considerably too low for applying machine learning techniques. In Section 12.2.1, we give a more detailed overview of our overall dataset, along with the data augmentation techniques being used. In Section 12.2.2, we then present the overall architecture we employ in our experiments, which aims to combine the three tasks of reconstruction, classification, and mapping. Finally, we discuss the train-

ing and evaluation procedure as well as important hyperparameters in Section 12.2.3.

12.2.1 Data

The dataset of line drawings from Chapter 10, which we used for the extraction of SHAPE similarity spaces in Chapter 11, is limited to 60 individual stimuli. These stimuli are all annotated with their respective coordinates in the target similarity space, and are thus our main source of information for learning the mapping task. However, the number of data points is clearly too low for training a neural network from scratch.

In addition to the 60 images from our dataset, we have 70 more line drawings in a similar style, which were used in preliminary investigations or as filler items, but not included into the main psychological studies for various reasons. Since their style is very similar to the stimuli from the study, we included them as additional data points in our training set. Please note, that these 70 additional line drawings, however, come without ground truth coordinates in the similarity space.

Since the number of data points is still considerably too low, we decided to use both the TU Berlin dataset [143] and the Sketchy dataset [348] as secondary data sources. From the TU Berlin corpus, we used all 20,000 sketches, while for the Sketchy corpus we selected a subset of 62,500 images by first keeping only the sketches, which had been labeled as CORRECT by the authors, and by then randomly selecting 500 sketches from each of the 125 categories to ensure a balanced distribution across classes. While the sketches from TU Berlin and Sketchy do not come with coordinates in a psychological similarity space, they are annotated with their respective class label, and thus lend themselves to being used in a classification task. TU Berlin contains 250 classes and Sketchy uses 125 classes, and both datasets overlap on a subset of 98 common classes. Overall, we thus consider a set of 277 distinct classes. The respective lists of classes can be found in Appendix G.1.

We used the following augmentation procedure to further increase the variety of inputs (cf. Section 6.2.3): For each original image, we first applied a horizontal flip with probability 0.5, and then rotated and sheared the image by an angle of up to 15 degrees, respectively. In the resulting distorted image, we identified the bounding box around the object, and cropped the overall image to the size of this bounding box. The resulting cropped image was then uniformly rescaled, such that its longer side had a randomly selected size between 168 and 224 pixels. Using a randomly chosen offset, the rescaled image was then put in a 224×224 image, where remaining pixels were filled with white. We used a uniform distribution over all possible resulting configurations for a given object, which makes smaller object sizes more likely, since they have more translation possibilities than larger object sizes. As a result, around 50% of all augmented images have objects with sizes between 168 and 182 pixels, while only about 10% of the generated

Our shape stimuli

Additional line drawings

Sketch data: TU Berlin and Sketchy

Dataset augmentation

Data Source	Number of Original Images	Factor	Number of Augmented Images
Shapes (Chapter 10)	60 (0.07%)	2,000	120,000 (16.00%)
Additional Line Drawings	70 (0.08%)	2,000	140,000 (18.67%)
TU Berlin [143]	20,000 (23.49%)	12	240,000 (32.00%)
Sketchy [348]	65,000 (76.35%)	4	250,000 (33.33%)
Overall	85,130	≈ 8.81	750,000

Table 12.2: Augmentation factors used for the individual data sources.

images contain objects of size 204 or larger. The overall average object size is 185 pixels, which corresponds to about 82% of the image size and is thus slightly larger than what has been reported for the raw TU Berlin and Sketchy images (where the largest dimension of the object covers around 78% of the overall canvas).

Differences between data sources

Please note, that we explicitly did not use the augmentation steps of horizontal flips, random shears, and random rotations on the line drawings from our shapes dataset, since the corresponding psychological similarity space contains an interpretable direction, which reflects the ORIENTATION of the object. If we used these transformation in the augmentation step, we would also have to update the target coordinates accordingly. For all other data sources (additional line drawings, TU Berlin, and Sketchy), we, however, did apply horizontal flips, shears and rotations in order to increase the input variety. We decided against the usage of random crops, because they might remove parts of the object. While this may be desirable for photographs (where the object of interest may not be fully visible in the image), we think, that it would remove important information from the already relatively sparse sketches and line drawings.

Overall augmentation setup

Instead of using all possible augmentation variants (i.e., all possible rescaling sizes and translations), we decided to use only a randomly selected subset in order to keep the size of the resulting overall data set at a manageable size. Data augmentation was done as an explicit pre-processing step rather than on the fly. While this approach comes with a higher storage requirement, it reduces the number of computations while training the system. Moreover, by doing the augmentation step once in the beginning, we can easily ensure, that the same randomly chosen augmentations are used in all experiments.

Augmentation factors

Table 12.2 lists the augmentation factors used for the different data sources. We chose these factors, such that each of the three sources (line drawings, TU Berlin, Sketchy) contributes a roughly equal amount of augmented data points to the resulting augmented dataset. This of course means, that the line drawings (for which only a very small number of examples is available) are more heavily augmented than the

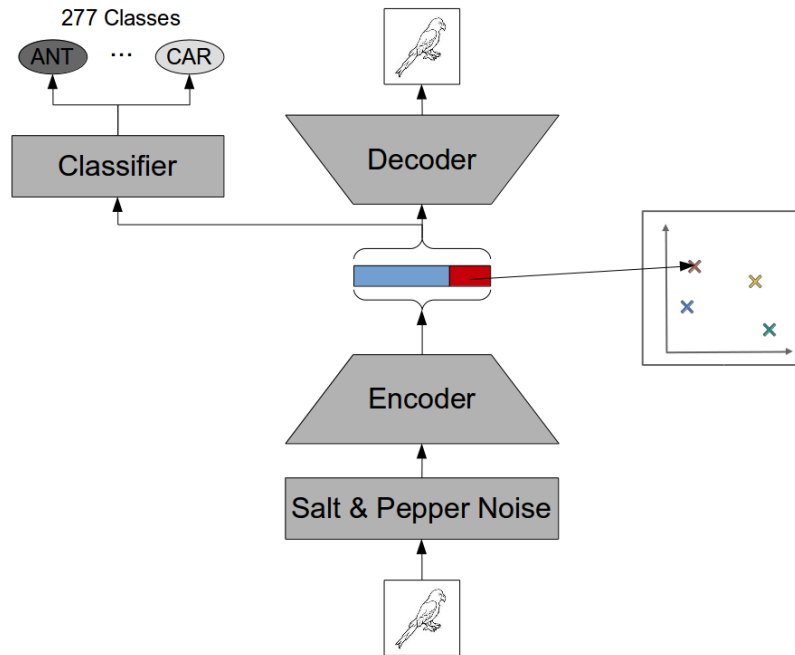


Figure 12.3: Illustration of our overall network architecture.⁹

sketch datasets (which already contain a larger number of examples). We nevertheless chose also a factor of four for the largest data source in order to ensure some variability with respect to object size and location.

From the overall augmented dataset, all 750,000 inputs can be used for training and evaluating a reconstruction objective (i.e., an autoencoder). Only the 120,000 inputs generated based on the stimuli from our shapes dataset can be used for learning and evaluating the mapping task, which corresponds to about one sixth of the overall augmented dataset. Finally, a classification of inputs into categories can be learned and evaluated based on both TU Berlin and Sketchy with their 490,000 data points, which make up about two thirds of the augmented dataset.

Data and tasks

12.2.2 Architecture

Figure 12.3 illustrates the overall architecture used in our experiments. The three tasks of classification, reconstruction, and mapping all share the bottom part of the architecture, namely the encoder network and the resulting hidden representation. The classification takes place on top of this shared hidden representation with a single fully connected softmax layer, while the reconstruction is obtained through a separate decoder network. In our experiments, we simply constrain the encoder to produce the coordinates in the psychological similarity space in a given subset of its outputs. We could also have included an additional linear layer, which translates from the common hidden representation to coordinates in the similarity space. This would, however, have

Overall architecture with shared encoder

⁹ Image license of parrot drawing CC BY-NC 4.0, source: <http://clipartmag.com/cockatiel-drawing>.

introduced additional parameters into the model without any obvious benefit. Moreover, since the mapping output is part of the common bottleneck layer, it also receives a gradient from the other two tasks, which provides an important factor of regularization. Our approach can be related to the work by Salakhutdinov and Hinton [345], who trained an autoencoder and constrained a part of its bottleneck layer with a classification task in order to enforce metric learning constraints (cf. Section 5.3.3).

*Joint classification
output*

Please note, that we use a single softmax layer with 277 outputs for the union of classes from TU Berlin and Sketchy. Of these 277 outputs, 152 are only relevant for the TU Berlin dataset, while 27 are only relevant for Sketchy. The remaining 98 classes are shared between the two datasets. A detailed list of all the classes can be found in Appendix G.1. During training, we optimize the prediction with respect to all 277 outputs, but we split the evaluation according to the two datasets.

Salt and pepper noise

In Figure 12.3, one can also see, that we apply salt and pepper noise (cf. Section 6.3.1) to the inputs, before feeding them through the encoder. This additional noise further increases the variety of the network's inputs, and can be seen as an additional form of data augmentation. We chose binary salt and pepper noise (which sets randomly selected pixels to their minimal or maximal value) rather than Gaussian noise, since the former seems to be more adequate for our near-binary inputs, where most of the pixels are either black or white. Please note, that while the encoder's input is corrupted with noise, the decoder has to reconstruct the original, uncorrupted image. We thus make use of a denoising autoencoder [419, 420] (cf. Section 6.3.1). This can be interpreted as a way of regularization: The function learned by the encoder needs to be able to remove irrelevant noise, and thus implicitly needs to learn about lines and strokes.

*Concrete network
structures*

The overall architecture sketched so far leaves the concrete implementation of both the encoder and the decoder network unspecified. There is a large number of possible network architectures one could employ for the encoder and the decoder. In order to drastically reduce the search space and to focus on our overall research question (namely, whether we are able to learn the mapping from images to points in the similarity space), we decided to base our own architecture on pre-existing models.

*Sketch-a-Net as an
encoder*

Figure 12.4a shows the structure of the Sketch-a-Net model [445, 446], which has been introduced in Section 12.1.2. It was the first CNN to achieve superhuman performance on sketch recognition, and thus seems to be a good starting point for our research. Although its performance has been surpassed in recent years by other approaches, the simplicity of its network structure makes it an ideal candidate for our encoder network.

*A decoder for
AlexNet*

Dosovitskiy and Brox [134] have used decoder networks consisting of multiple upconvolutional layers (which contain an unpooling step and a standard convolution, cf. Section 6.2.2) in order to reconstruct the original inputs from the high-level activations of pretrained convolutional neural networks. Figure 12.4b shows the network architecture they

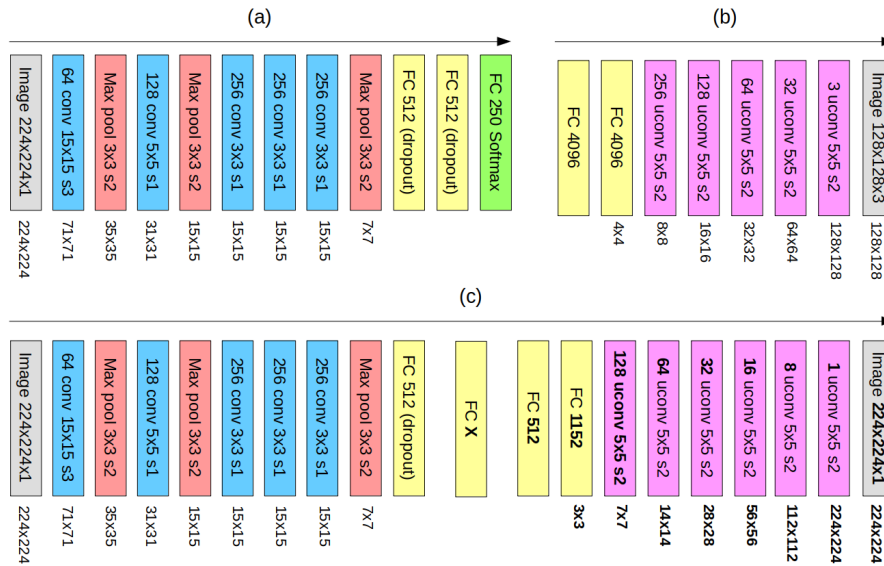


Figure 12.4: (a) Sketch-a-Net architecture [446]. (b) The decoder architecture used by Dosovitskiy and Brox [134] for AlexNet. (c) Our overall encoder-decoder structure (changes highlighted in boldface).¹⁰

used for the final layers of AlexNet [232]. Dosovitskiy and Brox have not only used this architecture to decode the features from a pretrained AlexNet architecture, but they have also trained the two networks jointly, obtaining a convolutional autoencoder. Since Sketch-a-Net can be considered a smaller variant of AlexNet, we assume, that a slightly modified version of the architecture used by Dosovitskiy and Brox is also applicable to Sketch-a-Net.

Figure 12.4c illustrates the configuration of our overall autoencoder (highlighting changes in boldface). The encoder part is almost completely identical to Sketch-a-Net. We used the last fully connected layer as bottleneck layer, whose size will be treated as a hyperparameter of the overall system. Moreover, we did not use dropout in the bottleneck layer and made use of linear units instead of ReLUs to allow the network to predict the MDS coordinates (which can also be negative) as part of its output. Overall, the encoder network has about 8.5 million weights.

Our decoder is based on the structure provided by Dosovitskiy and Brox with the following modifications: The first fully connected layer has 512 units in order to reflect the second-to-last layer of the encoder. In order to obtain an output size of 224×224 pixels, we furthermore introduced an additional upconvolutional layer and reduced the output size of the second fully connected layer. Since the images in our dataset contain only greyscale information, we need only a single color channel at the decoder's output. Since the desired output is of lower complexity, we furthermore decided to reduce the number of kernels by a factor of

Our encoder network

Our decoder network

¹⁰ Abbreviations: "96 (u)conv 11×11 s4" = "(up)convolutional layer with 96 kernels of size 11×11 and stride of 3", "Max pool 3×3 s2" = "max pooling layer with pool size 3×3 and stride of 2", "FC 4096 (dropout)" = "fully connected layer with 4,096 neurons, trained with dropout". Image sizes are shown next to the layers.

two, which further reduces the number of trainable parameters. Overall, our decoder network has about 1.5 million trainable parameters.

Our classifier network

The classifier (not shown in Figure 12.4) is implemented as a fully connected softmax layer with 277 outputs on top of the bottleneck layer. It comes with roughly 140,000 trainable weights. Taken together, the overall architecture (encoder, classifier, and decoder) has about 10 million parameters.

12.2.3 Training, Evaluation, and Hyperparameters

The overall loss function

In our experiments reported below, we will train the proposed overall architecture to minimize a linear combination of the classification, reconstruction, and mapping objective. In general, our loss function has the following form:

$$J = \lambda_1 \cdot J_{classification} + \lambda_2 \cdot J_{reconstruction} + \lambda_3 \cdot J_{mapping} \quad (12.1)$$

with $\lambda_1, \lambda_2, \lambda_3 \in \mathbb{R}_0^+$

Classification and mapping loss

The hyperparameters λ_1 , λ_2 , and λ_3 control the trade-off between the three different training objectives. Since the classification output is realized with a softmax layer, we use the softmax cross-entropy (cf. Section 6.2.1) for $J_{classification}$, considering all 277 classes. The mapping task is essentially a regression from image input to coordinates in the psychological similarity space. Therefore, $J_{mapping}$ is realized in the form of the mean squared error (MSE, cf. Section 5.1.2).

Reconstruction loss

For the reconstruction error $J_{reconstruction}$, we could also choose the mean squared error. However, even though our inputs and outputs are not binary, they are limited to the integer interval $[0, 255]$. Moreover, both the line drawings and the sketches mainly consist of white background and black lines, as documented in Table 12.3. There, we can see, that especially the sketches from TU Berlin and Sketchy are quite binary in nature, but that also the line drawings have most of their pixel intensities at the ends of the scale. We can thus approximate the distribution of pixel intensities reasonably well with a Bernoulli distribution. We normalize the pixel intensities to the interval $[0, 1]$ and use sigmoid units at the output layer of the decoder network, which allows us to use the sigmoid cross entropy loss (cf. Section 6.2.1) for $J_{reconstruction}$.

Data points and loss components

Each of the three loss components was only computed on the set of data points, to which it is relevant. While $J_{reconstruction}$ can be computed for all examples from the augmented dataset, $J_{classification}$ is only applicable to data points from TU Berlin and Sketchy, and $J_{mapping}$ is relevant only for the line drawings from our psychological study. Combining the three loss components on the overall dataset thus corresponds to a form of semi-supervised learning (cf. Section 5.1.2), because not all data points come with labels for all tasks.

Evaluation metrics

When evaluating the network's overall performance with respect to the three tasks, we make use of the following evaluation metrics: For the reconstruction task, we simply report $J_{reconstruction}$, i.e., the

Pixel Values	Shapes	Additional	TU Berlin	Sketchy	Overall
0 (total black)	0.28%	0.03%	0.00%	0.06%	0.07%
[0, 9] (black)	1.49%	0.82%	0.01%	0.63%	0.61%
[246, 255] (white)	86.97%	87.43%	93.14%	90.61%	90.25%
255 (total white)	69.15%	70.93%	91.08%	88.85%	83.07%

Table 12.3: Distribution of pixel intensities in the augmented dataset, split according to the data source.

binary cross entropy with respect to the original input and the denoised reconstruction. For the classification task, we report separate classification accuracies (cf. Section 5.1.2) for the TU Berlin and the Sketchy dataset, respectively, by using the respective examples from the test set. By evaluating performance on the two datasets individually, we can compare our obtained accuracies to the ones reported in the literature (cf. Section 12.1.2). Finally, for the mapping task, we report the same three metrics already used in Chapter 9, namely the mean squared error (MSE), the coefficient of determination R^2 , and the mean Euclidean distance (MED) between the predicted point and the ground truth:

$$\begin{aligned}
 MSE &= \sum_{i=1}^n \frac{1}{N} \sum_{j=1}^N \left(y_i^{(j)} - \hat{f}(\vec{x}^{(j)})_i \right)^2 \\
 MED &= \frac{1}{N} \cdot \sum_{j=1}^N \sqrt{\sum_{i=1}^n \left(y_i^{(j)} - \hat{f}(\vec{x}^{(j)})_i \right)^2} \\
 R^2 &= \frac{1}{n} \cdot \sum_{i=1}^n \left(1 - \frac{S_{residual}^{(i)}}{S_{total}^{(i)}} \right) \\
 &\quad \text{with } S_{residual}^{(i)} = \sum_{j=1}^N \left(y_i^{(j)} - \hat{f}(\vec{x}^{(j)})_i \right)^2 \\
 &\quad \text{and } S_{total}^{(i)} = \sum_{j=1}^N \left(y_i^{(j)} - \bar{y}_i \right)^2
 \end{aligned}$$

In the formulas from above, n refers to the number of dimensions in the target space and N to the number of data points, while $y_i^{(j)}$ and $\hat{f}(\vec{x}^{(j)})_i$ represent the target coordinate and the predicted coordinate, respectively. \bar{y}_i denotes the average coordinate value across all data points on the i th dimension. A more detailed explanation of the evaluation metrics and their interpretation is given in Section 9.2.1.

Notation

We only used salt and pepper noise during training, but not during evaluation in order to avoid random fluctuations on the validation and test set, which are only based on the different noise patterns applied to the input. This makes the validation and test set results more comparable across epochs and hyperparameter configurations.

Noise regime

Since the target coordinates used for learning and evaluating the mapping task are based only on 60 original stimuli, we decided to

Cross validation and fold structure

Setting	Shapes	Additional	TU Berlin	Sketchy
Classification (Section 12.3.1)	0	0	63	65
Classification + Mapping (Section 12.5.1)	25	0	51	52
Reconstruction (+ Mapping) (Sections 12.3.2 and 12.5.2)	21	24	41	42

Table 12.4: Creation of minibatches based on the four data sources for different training settings.

follow a five-fold cross validation scheme. We divided the original data points from each of the data sources into five folds of equal size, and then applied the augmentation steps from Section 12.2.1 to each fold individually. Therefore, all augmented images, which are based on the same original data point, are guaranteed to belong to the same fold, thus preventing potential information leaks between the different folds. We created folds in such a way, that they were stratified with respect to the given categories. With respect to the stimuli from our psychological study, each fold contained exactly one stimulus from each of the twelve categories. For the TU Berlin dataset, sixteen out of the 80 examples were selected for each category, and for Sketchy, we used for each fold 100 of the 500 example sketches per category.

*Cross validation
workflow*

In our overall evaluation process, we rotated through these folds in such a way, that we always used three folds for training, one fold for testing, and the remaining fold as a validation set to implement early stopping (cf. Section 6.2.5). For early stopping, we chose the epoch with the lowest loss on the validation set (classification and reconstruction loss for the initial hyperparameter optimizations in Section 12.3, and mapping loss for the multitask learning experiments in Section 12.5) across all of the 200 epochs. We ensured, that each fold was used once for testing, once as validation set, and three times as part of the training set. The reported numbers are always averaged across all folds. By using this five-fold cross-validation technique, we implicitly trained five neural networks with the same hyperparameter settings, but slightly different data. Our averaged results can therefore also be interpreted as an expected value of the neural network’s performance.

*Training algorithm
and minibatches*

During training, we used the Adam optimizer [223] (cf. Section 6.2.5) as a variant of stochastic gradient descent, with the initial learning rate set to 0.0001, the default parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, and a batch size of 128. We ensured, that each minibatch contained examples from all relevant data sources. Table 12.4 shows, how each minibatch was constructed for the different combinations of tasks, reflecting the relative size of each relevant data source as accurately as possible. During training, we only considered complete batches, which means, that some examples were discarded, since the dataset size was not integer divisible by the batch size. After each epoch, we shuffled the training examples to create a different assignment of examples to minibatches for the following epoch. For both the validation and the

test data, also incomplete batches were evaluated to ensure, that the complete set of examples was taken into account.

Since we need to estimate a large number of model parameters from a limited amount of data, we may need to introduce several regularization techniques (cf. Section 6.2.3). The form and strength of regularization was treated as a hyperparameter to the overall system, which needs to be manually tuned. More specifically, we considered the following regularization techniques:

Regularization

We added a *weight decay* penalty to the overall loss function, which incites the network to use small weights. A weight decay level commonly reported in the literature is 0.0005 [232, 302], which we will use as a starting point for our experiments. While some authors report, that applying weight decay only to the encoder part of an autoencoder network is preferable [300], others apply the weight decay also to the decoder [302]. We will explore both variants when considering the decoder network as part of our autoencoder experiments.

Weight decay

As already shown in Figure 12.4, *dropout* is commonly used in the fully connected layers of standard convolutional networks. We will investigate, whether using dropout in the penultimate layer of the encoder is necessary, and whether the fully connected layers of the decoder also profit from dropout regularization.

Dropout

Thirdly, one could use *batch normalization* as an additional regularization technique. Since, however, both original networks used as a starting point for our own architecture do not make use of batch normalization, we did not explore its effects in our studies.

Batch normalization

The additional *noise* injected into the inputs can be considered another form of regularization, although an indirect one. As stated above, we used salt and pepper noise, because our data is close to being binary. When introducing stacked denoising autoencoders, Vincent et al. [420] investigated salt and pepper noise added to 10%, 25%, and 55% of the pixels. We will use their proposed values as possible noise levels in our experiments. One should, however, note, that due to the very high number of white pixels in our images, salting (i.e., setting random pixels to white) will rarely have any observable effect.

Noise

Another way to regularize our architecture can be realized by varying the *size of the bottleneck layer*. The original Sketch-a-Net architecture [446] uses a fully connected layer with 512 neurons for making classifications on the TU Berlin dataset. In addition to this default value, we also investigated bottleneck layers with 256, 128, 64, 32, and 16 neurons.

Size of the bottleneck layer

Finally, the overall *multitask learning* setting used in our approach also acts as a regularizer: By considering multiple tasks at once, the encoder's weights are constrained to be useful for all of these tasks, making overfitting less likely. Whenever combining multiple objectives, we will consider multiple settings for the mapping coefficients λ_3 in order to investigate synergies and trade-offs between the tasks.

Multitask learning

12.3 OBTAINING BASELINE NETWORKS

Pretraining Sketch-a-Net Since no pretrained weights have been published for Sketch-a-Net [445, 446], we first need to train our sketch-based network from scratch in order to obtain a starting point for our classification-based experiments. We optimize our architecture for the classification task in Section 12.3.1, where we also investigate the influence of different hyperparameters on the network's performance.

Pretraining the autoencoder We are also not aware of any pretrained convolutional autoencoders for sketch data, hence we also need to train our proposed architecture from scratch on the reconstruction task. This is done in Section 12.3.2, where we also optimize the network's hyperparameters.

Correlation to dissimilarities For both tasks, we use the correlation between the learned hidden representation and the dissimilarities from our psychological study as an additional way of evaluating the expected usefulness of the given setup for the mapping task. This correlation is computed using Kendall's τ [219] as described in Section 11.2.1 for the different baselines.

12.3.1 Sketch Classification

Benchmark results on TU Berlin and Sketchy In our first experimental step, we used only the encoder network and the classification layer, and trained them exclusively on the classification task. We can compare the results obtained by our architecture to the best results obtained on the two sketch classification datasets so far. While human performance on the TU Berlin dataset seems to be around 73.1% accuracy [143], the Sketch-a-Net architecture was able to achieve up to 77.9% [445]. More recent approaches such as the one by Zhang et al. [457] (cf. Section 12.1.2) use specialized elements in addition to regular CNNs, and are able to achieve even higher accuracies of up to 84.4%. The Sketchy dataset is mostly used for evaluating sketch-based image retrieval rather than sketch classification, hence we have fewer reported classification accuracies. However, since Zhang et al. [457] report an accuracy of 82.7% for their hybrid network, we can assume, that also other approaches should obtain accuracies on Sketchy, which are in a similar order of magnitude as observed for the TU Berlin dataset.

Default hyperparameter setup We started our experiments by considering a default setup of the hyperparameters, which is based directly on Sketch-a-Net [445, 446] and AlexNet [232]: We used a weight decay of 0.0005, dropout in the first fully connected layer, and 512 neurons for the bottleneck layer. Moreover, we applied 10% of salt and pepper noise during training. As evaluation metrics, we considered the accuracies reached on TU Berlin and Sketchy, as well as the monotone correlation of distances in the feature space to the dissimilarity ratings from our shapes dataset (measured with Kendall's τ). The averaged results across all five folds are reported in Table 12.5 as default configuration.

Correlation to dissimilarities As we can see there, our default setup is able to achieve a correlation of $\tau \approx 0.27$ to the dissimilarity ratings from our psychological study.

This value was achieved using optimized weights and the Manhattan distance, but is considerably lower than the correlation of $\tau \approx 0.39$, which we reported for the inception-v3 network [400] in Section 11.2.2. It may thus seem, that the features extracted by our sketch-based network are less useful than the ones extracted by more complex photograph-based architectures. This difference may simply be based on the smaller dataset used in our study, on the considerably simpler network architecture, and on the smaller size of the bottleneck layer. It is interesting to observe, that for fixed dimension weights, the features from our network correlate better with the dissimilarity ratings ($\tau \approx 0.23$) than the ones from the inception-v3 network ($\tau \approx 0.17$). The optimization of dimension weights allows us to focus on some dimension of the feature vector, while largely ignoring others. This seems to be more useful for the photograph-based than the sketch-based network. Our observation may thus indicate, that while to photograph-based network contains more shape-related information than our sketch-based network, it also contains many irrelevant pieces of information (for instance with respect to domains such as TEXTURE OR COLOR).

When looking at classification performance, we observe, that the accuracy on the TU Berlin data is slightly above 63%, while we are able to reach 79% accuracy on the Sketchy dataset. These numbers are clearly lower than the ones reported above for the current state of the art with respect to CNNs. The observed difference can, however, be explained by our simpler network architecture and our less sophisticated augmentation procedure. Moreover, in comparison to Sketch-a-Net [445, 446], we have removed the nonlinearity in the second fully connected layer (cf. Section 12.2.2) in order to allow the network to represent also negative coordinates when used in the multitask setting. This lack of a nonlinearity in the second fully connected layer reduces the representational capacity of the network, since the two subsequent linear transformations before the softmax classification layer are mathematically equivalent to a single linear transformation. Nevertheless, our network is considerably better than chance on both datasets, where one would expect accuracies of 0.4% and 0.8% respectively, based on the 250 and 125 classes contained in the two datasets. This difference in the number of classes also explains, why the accuracy on Sketchy is higher than the one on TU Berlin. Although our network achieves only moderate performance levels, we think, that it is still a reasonable starting point for our further investigations.

Our choice of hyperparameters so far was motivated by earlier literature, but may nevertheless be somewhat arbitrary. Moreover, since our network architecture and dataset differ from other studies, we may need a different degree of regularization. Therefore, we investigated the influence of the individual regularization hyperparameters on the network's performance.

Conducting a full grid search with all of these potential hyperparameter settings would involve 180 different network configurations, each of which would need to be trained five times (once for each of the folds).

Classification performance

Hyperparameter optimization

A two-phase grid search

Hyper-parameter	Setting	Number of Epochs	Kendall's τ	Accuracy on TU Berlin	Accuracy on Sketchy
None	Default	188.6	0.2743	0.6320	0.7933
Weight Decay	0.0	12.2	0.2831	0.5514	0.7479
	0.0002	12.6	0.2931	0.5408	0.7364
	0.0005	188.6	0.2743	0.6320	0.7933
	0.001	163.8	0.2870	0.6110	0.7811
	0.002	168.0	0.2758	0.5761	0.7666
Dropout	<i>True</i>	188.6	0.2743	0.6320	0.7933
	False	4.8	0.3195	0.3963	0.6349
Noise Level	10%	188.6	0.2743	0.6320	0.7933
	25%	190.8	0.2824	0.6094	0.7828
	55%	138.6	0.2825	0.3873	0.7424
Bottleneck Size	2048	170	0.2994	0.6315	0.7885
	512	188.6	0.2743	0.6320	0.7933
	256	178.4	0.2777	0.6274	0.7908
	128	181.2	0.2621	0.6239	0.7875
	64	178.6	0.2340	0.6209	0.7813
	32	167.4	0.2332	0.5916	0.7597
	16	111.0	0.2196	0.4651	0.6780

Table 12.5: Results of our initial sketch classification experiments using the default setup and variations with respect to a single hyperparameter. For each hyperparameter, the default setup is shown in italics, and the best values for each of the three evaluation criteria are shown in boldface.

Since this would lead to an infeasible computational effort, we decided to first conduct individual investigations for the different hyperparameters. This reduces our initial search space to twelve different settings, and allows us to obtain some first insights into the regularization effects of the individual hyperparameters. Afterwards, we selected the default setting as well as the most promising other setting for each of the hyperparameters, leaving us with only 16 configurations for our grid search. This two-phase approach massively reduces the number of computations, while still yielding reasonable insights.

Effects of weight decay

Table 12.5 summarizes the results of our first explorations with respect to the individual hyperparameters. When considering the strength of the *weight decay*, we can observe, that our default setting is optimal for classification performance. Stronger regularization leads to a slight decrease in performance, while weaker regularization causes more pronounced performance drops. Optimal validation set performance is found after around twelve epochs for the low regularization settings, which indicates a stronger tendency towards overfitting. When considering the correlation to the dissimilarity ratings, however, all other weight decay settings are more promising than our default choice, the best results being yielded by a slightly smaller degree of regularization. Since a weight decay setting of 0.001 also leads to a notably improved

correlation without the heavy downside of reduced prediction accuracy, we will consider the two weight decay rates of 0.0005 (default) and 0.001 for our grid search.

When comparing our default setup to the configuration where *dropout* was turned off, we can make two somewhat contradictory observations: On the one hand, removing dropout from the first fully connected layer considerably increases the correlation to the dissimilarity ratings. On the other hand, we also get a considerable drop in classification performance. So although dropout clearly provides important regularization for the classification task, it causes the bottleneck representation to be less well correlated to human dissimilarity judgments. One may speculate, that this is caused by the fact, that during training, a random selection of inputs to the bottleneck layer are disabled, thus making its input less stable. Again, the observation, that optimal validation set performance is reached after only about five epochs, illustrates, that without dropout, overfitting is very likely to happen.

Effects of dropout

Let us now take a look at the different *noise levels*. As we can see in Table 12.5, a higher proportion of salt and pepper noise in the inputs can be at least partially beneficial: The correlation to human dissimilarity ratings increases considerably for both a noise level of 25% and 55%. On the other hand, classification accuracy suffers from stronger input noise. While this effect is relatively mild for 25% noise, it is quite pronounced for 55% noise. Since the correlation to the dissimilarity ratings is almost identical for the two higher noise levels, we will investigate 10% and 25% noise in our subsequent grid search.

Effects of noise level

As a final hyperparameter, we considered the *size of the bottleneck layer*.¹¹ Somewhat surprisingly, classification performance is relatively robust to a reduction of the number of units: Even for only 64 bottleneck units, classification accuracies are only slightly below the ones obtained in the default setting. Only if we reduce the size of the bottleneck layer to 32 or 16 units, we observe a relatively strong decrease in classification performance. The correlation to the dissimilarity ratings, however, seems to be more sensitive to the size of the bottleneck layer. For the grid search, we consider 512 units (as in the default setting, which has yielded the best classification performance) and 256 units, since this setup resulted in comparable performance to 512 units with respect to all three evaluation metrics. Moreover, a smaller representation size seems desirable for both transfer learning and multitask learning: When training a linear regression on top of the bottleneck layer in our transfer learning experiments, a reduction of the bottleneck size causes a reduction in the number of free parameters in the regression, thus reducing the risk of overfitting. Also when training the network on the classification task and the mapping task simultaneously in the multitask learning setting, a smaller overall size of the bottleneck layer implies, that a larger fraction of the bottleneck units is used to represent the

Effects of bottleneck size

¹¹ Due to considerably larger training times for larger number of bottleneck units, we only explored a single setting with more than 512 neurons. We decided to use 2048 rather than 1024 units, because this allows a better comparability to the inception-v3 network, which also uses an internal representation with 2048 neurons.

Bottleneck Size	Weight Decay	Dropout	Noise Level	Number of Epochs	Kendall's τ	Accuracy TU Berlin	Accuracy Sketchy
512	0.0005	True	10%	188.6	0.2743	0.6320	0.7933
512	0.0005	True	25%	190.8	0.2824	0.6094	0.7827
512	0.0005	False	10%	4.8	0.3195	0.3963	0.6349
512	0.0005	False	25%	150.8	0.2865	0.5067	0.6938
512	0.001	True	10%	163.8	0.2870	0.6110	0.7811
512	0.001	True	25%	172.6	0.2556	0.5420	0.7715
512	0.001	False	10%	5.6	0.3292	0.3636	0.6146
512	0.001	False	25%	186	0.2911	0.5344	0.7016
256	0.0005	True	10%	178.4	0.2777	0.6274	0.7908
256	0.0005	True	25%	181	0.2563	0.5816	0.7781
256	0.0005	False	10%	5.2	0.3066	0.3528	0.6172
256	0.0005	False	25%	191.4	0.2725	0.5478	0.7074
256	0.001	True	10%	170.6	0.2701	0.6138	0.7830
256	0.001	True	25%	187.8	0.2645	0.5420	0.7701
256	0.001	False	10%	5.6	0.3049	0.3405	0.6088
256	0.001	False	25%	177.8	0.2478	0.5296	0.6994

Table 12.6: Results of our grid search for the sketch classification network. The best values for each of the three evaluation criteria are highlighted in boldface.

psychological similarity space. Since they constitute a larger proportion of the overall bottleneck layer, the additional constraints coming from the classification layer may have a stronger regularization effect on the mapping task, potentially aiding generalization.

Investigating the interaction of hyperparameters

Overall, it seems that our default setup has already found a sweet spot with respect to classification performance. However, the correlation to the dissimilarity ratings can in principle benefit from different variations of its hyperparameters. So far, we have, however, only considered the different hyperparameters individually without taking into account potential interactions. For instance, it may be an interesting idea to deactivate dropout in order to increase the correlation to the dissimilarities, but to compensate for the loss in regularization by increasing the noise level. We therefore conducted a small grid search involving two possible settings for each hyperparameter. As motivated above, we considered weight decays of 0.0005 and 0.001, dropout being active and inactive, noise levels of 10% and 25%, and bottleneck layers with 512 and 256 units.

Optimal correlation

Table 12.6 shows the results of our grid search. As we can see, the highest correlation to the dissimilarity ratings can be achieved with a bottleneck size of 512 units, a weight decay of 0.001, enabled dropout, and a noise level of 10%: We improve from $\tau \approx 0.27$ for our default setup to $\tau \approx 0.33$. This is still somewhat lower than the correlation of $\tau \approx 0.39$ observed for the inception-v3 network, but nevertheless a considerable improvement. We will further investigate this configuration under the

name $C_{\text{CORRELATION}}$. When looking closer at Table 12.6, we can observe, that correlations of $\tau \geq 0.30$ are only achieved for disabled dropout and a very small number of epochs. We assume, that the small number of epochs is based on the missing regularization effect from dropout: The network quickly starts to overfit the training data, hence the lowest loss value on the validation set is observed quite early. The increased correlation reported for these configurations may thus be an artifact of simply terminating the training procedure much earlier.

When focusing on classification accuracy, our default configuration with a bottleneck size of 512, a weight decay of 0.0005, enabled dropout, and a noise level of 10% yields the best performance with accuracies of about 0.63 and 0.79 for TU Berlin and Sketchy, respectively. We will further investigate this configuration under the name C_{DEFAULT} . It seems, that as soon as dropout is disabled, the classification performance considerably drops. This effect seems to be independent from the number of epochs used for training. Also higher noise levels lead to a slight deterioration of classification performance.

We argued above, that a reduced bottleneck size of 256 instead of 512 units is attractive for our downstream mapping task. As our grid search revealed, halving the size of the bottleneck layer only slightly decreases performance. The configuration with 256 hidden units and otherwise default parameters will be investigated under the name C_{SMALL} . Moreover, for better comparability to the inception-v3 network with its 2048 hidden units, we will also consider a configuration named C_{LARGE} , which uses a bottleneck layer with 2048 units and otherwise default hyperparameter settings (see Table 12.5).

Overall, we have selected four configurations for our subsequent experiments: Our C_{DEFAULT} setup (which has yielded the highest classification accuracies), the configuration $C_{\text{CORRELATION}}$ with the highest correlation to the dissimilarity ratings, and two variations of our default setup, where the number of units in the bottleneck layer has been increased to 2048 (C_{LARGE}) or reduced to 256 (C_{SMALL}). By comparing the results obtained with these configurations, we can investigate which aspect is most important for a good performance in the mapping task: Correlation to the dissimilarities, classification accuracy, or compactness of representation.

12.3.2 Sketch Reconstruction

As an additional avenue of research, we explored the usage of an autoencoder instead of a classification network as a basis for our mapping task. Since autoencoders have to reconstruct the original pixels of the input image, they tend to focus on perceptual aspects of the input (cf. Sections 6.3.1 and 8.7.2) – a property, which may be useful in the context of shape similarity. Our default setup for the autoencoder makes use of the default encoder settings from Section 12.3.1 (weight decay of 0.0005, enabled dropout, 10% salt and pepper noise, and a bottleneck layer with 512 units) as well as an unregularized decoder (no weight

*Optimal
classification
performance*

*Different bottleneck
sizes*

*Selected
hyperparameter
settings*

Experimental setup

decay and no dropout). Similar to our sketch classification experiments, we again first conducted some initial experiments, where we varied individual hyperparameters in order to explore their influence on the overall results. For evaluation purposes, we used the reconstruction error on the test set as well as the correlation of distances in the learned representation to the original dissimilarity ratings (measured with Kendall's τ).

Performance of the default setup

Table 12.7 shows the results of this analysis. As we can see, our default setup results in a reconstruction error of approximately 0.13. This number is by itself difficult to interpret, but can nevertheless serve as a basis for relative comparisons between different hyperparameter settings. The correlation to the dissimilarity ratings of $\tau \approx 0.22$ is considerably lower than what we have observed for the default classification setting in Section 12.3.1, where we obtained $\tau \approx 0.27$.

Effects of encoder weight decay

When considering different *weight decay* settings for the *encoder* network, we observed two opposing effects: On the one hand, the reconstruction error improved considerably, if the weight decay penalty was decreased, and became larger for stronger regularization. On the other hand, the correlation to the dissimilarity ratings seems to behave in the opposite direction with stronger regularization leading to increased values of Kendall's τ . Since the gains with respect to reconstruction performance seem to be more pronounced than the improvements with respect to the correlations, we will consider the weight decay settings of 0.0 and 0.0005 for the encoder in our subsequent grid search.

Effects of decoder weight decay

In our default setup, we did not impose any *weight decay* on the *decoder* network. As Table 12.7 shows, introducing such a weight decay penalty on the weights of the decoder drastically reduced the correlation to the dissimilarities. It also considerably increased the reconstruction error.¹² We thus will not consider a weight decay for the decoder network in our subsequent experiments.

Effects of encoder dropout

In our classification experiments from Section 12.3.1, disabling *droupout in the encoder* network led to an improved correlation to the dissimilarities, but also resulted in considerably poorer classification performance. In our autoencoder setup, however, disabling dropout in the encoder yields clearly superior results with respect to both evaluation metrics. It therefore seems, that dropout is necessary for good classification performance, but makes reconstruction considerably more difficult. We therefore will not consider dropout in the encoder in our remaining investigations.

Effects of decoder dropout

Interestingly, enabling *dropout in the decoder* did not have such a pronounced effect: The results with respect to both the reconstruction error and the correlation became only slightly worse. Nevertheless, since dropout in the decoder did not seem to be beneficial, we excluded this way of regularization from our further considerations.

¹² When the decoder weight decay was set to 0.002, the correlation values for the second fold returned NaN (not a number). They have been excluded for computing the average value reported in Table 12.7. Since all correlation values for enabled decoder weight decay are very small, this does not considerably impact our overall results.

Hyperparameter	Setting	Number of Epochs	Kendall's τ	Reconstruction Error
None	Default	56.2	0.2228	0.1303
Weight Decay Encoder	0.0	178.8	0.1496	0.1098
	0.0002	49.0	0.2244	0.1220
	<i>0.0005</i>	56.2	0.2228	0.1303
	0.001	80.2	0.2318	0.1367
	0.002	168.0	0.2346	0.1454
Weight Decay Decoder	<i>0.0</i>	56.2	0.2228	0.1303
	0.0002	124.4	0.0641	0.1647
	0.0005	145.4	0.0651	0.1645
	0.001	170.2	0.0635	0.1651
	0.002	167.4	0.0610	0.1693
Dropout Encoder	<i>True</i>	56.2	0.2228	0.1303
	False	160.2	0.2745	0.1194
Dropout Decoder	True	75.6	0.2156	0.1315
	<i>False</i>	56.2	0.2228	0.1303
Noise Level	10%	56.2	0.2228	0.1303
	25%	24.0	0.2216	0.1396
	55%	156.0	0.1315	0.1527
Bottleneck Size	2048	101.2	0.2382	0.1319
	512	56.2	0.2228	0.1303
	256	72.8	0.2057	0.1290
	128	91.0	0.2057	0.1304
	64	87.8	0.1997	0.1297
	32	74.6	0.1847	0.1309
	16	73.2	0.1598	0.1328

Table 12.7: Results of our initial autoencoder experiments using the default setup and variations with respect to a single hyperparameter. For each hyperparameter, the default setup is shown in italics, and the best values for each of the three evaluation criteria are highlighted in boldface.

Effects of noise level

With respect to the *noise level*, our default setting of 10% salt and pepper noise seems to be optimal: Adding more noise made the denoising reconstruction task more difficult, and did not lead to improved generalization. While increasing the noise level to 25% only had a limited negative impact on performance, 55% noise heavily impaired both correlation and reconstruction error. Since 10% of salt and pepper noise have also turned out to be appropriate in our classification experiments (cf. Section 12.3.1), we also keep the noise level in our subsequent reconstruction experiments fixed to this value.

Effects of bottleneck size

The *size of the bottleneck layer* seems to have its main impact on the correlation values: More hidden units lead to an increased correlation to the dissimilarity ratings, while less hidden units cause the correlation to drop. With respect to the reconstruction error, the effects of a differently sized representation seem to be only minor: Even for only 16 bottleneck units, we still obtained a reconstruction error only slightly larger than observed in our default setup. It seems, that the sweet spot for the reconstruction error lies at 256 units, which is why we further investigate 512 and 256 units.

Relation to prior work

Our observations can be related to the results by Newson et al. [300], which were summarized in Section 6.4.2: Newson et al. investigated the performance of autoencoders on a dataset of disks, using only a single unit in the bottleneck layer. Their investigations showed, that the combination of a regularized encoder and an unregularized decoder led to the most stable results. Our dataset contains considerably more complex stimuli, and we employed a considerably larger bottleneck layer. Moreover, we used a denoising reconstruction objective, which can also be interpreted as a general form of regularization. While also our experiments showed, that an unregularized decoder is highly favorable, we also profit from an unregularized encoder.

Computational complexity of a grid search

In our classification experiments in Section 12.3.1, we have conducted a grid search using two promising candidate values for each of the four hyperparameters. Due to the usage of a second decoder network, we now have two additional hyperparameters to consider, which would result in $2^6 = 64$ instead of $2^4 = 16$ overall configurations to evaluate as part of the grid search. Due to the increased complexity of training an autoencoder (which has a more complex output and a larger number of weights to optimize), this was computationally not feasible with our given resources. Luckily, we observed clear tendencies for many of the individual hyperparameters in Table 12.7, and can thus simply fix them to their supposed optimal value.

Grid search setup

Overall, in our subsequent grid search, we used a fixed noise level of 10%, no dropout, and no weight decay for the weights of the decoder network. This leaves us with only two hyperparameters, namely, the size of the bottleneck layer (256 vs. 512 units) and the weight decay for the encoder network (0.0 vs. 0.0005).

Selected hyperparameter settings

As we can see in Table 12.8, the configuration with 512 bottleneck units and disabled weight decay scored best with respect to both Kendall's τ and the reconstruction error. It has therefore been selected

Bottleneck Size	Weight Decay Encoder	Number of Epochs	Kendall's τ	Reconstruction Error
512	0.0	192.4	0.3019	0.0828
512	0.0005	160.2	0.2745	0.1194
256	0.0	192.0	0.2764	0.0834
256	0.0005	156.2	0.2592	0.1170

Table 12.8: Results of our grid search for the autoencoder. The best values for each of the two evaluation criteria are highlighted in boldface.

for further investigations under the name R_{BEST} . Please note, that the highest correlation of $\tau \approx 0.30$, which we were able to achieve with the autoencoder, is notably lower than the highest values of $\tau \approx 0.33$ observed for the classifier in Section 12.3.1. However, this higher correlation was only obtained by sacrificing considerable amounts of performance on the classification task, while the selected configuration for the reconstruction task is optimal with respect to both the correlation and the reconstruction objective. This may indicate, that the reconstruction objective is more useful in our scenario, since improved reconstruction performance also leads to a more human-like representation. We will investigate in the following, whether this also results in improved mapping performance compared to a classification-based network. In order to see, whether the hyperparameter optimization conducted in this section has any impact on the mapping performance, we will compare the R_{BEST} configuration to our initial default setup, henceforth labeled as R_{DEFAULT} .

Figure 12.5 illustrates various reconstruction results for the PARROT stimulus from our original dataset. We only show reconstructions of network configurations, where the presented stimulus was part of the test set in order to focus on generalization performance. Figure 12.5a shows the original stimulus (without salt and pepper noise) as well as reconstruction results by the R_{DEFAULT} and R_{BEST} configurations, respectively. For both reconstructions, 10% salt and pepper noise had been applied to the raw input. Figure 12.5a nicely illustrates, how a reduction in the reconstruction error from 0.1303 to 0.0828 leads to a much cleaner reconstruction of the original image, where not only the overall shape of the object, but also some aspects of its inner structure (such as the wing) and more details of its contour (such as the beak and the feet) are captured.

Figure 12.5b shows reconstruction results with the R_{BEST} configuration for four different noise patterns based on 10% salt and pepper noise with different seeds. As we can see, the resulting reconstructions barely differ, which illustrates that the autoencoder succeeds in denoising its input. Finally, Figure 12.5c shows reconstruction results for various levels of noise. While the differences between 0% and 10% noise are negligible, we can observe, that the system breaks down for higher levels of noise: For 25% noise, the overall shape is roughly preserved,

Visualizing reconstruction performance

Robustness to noise

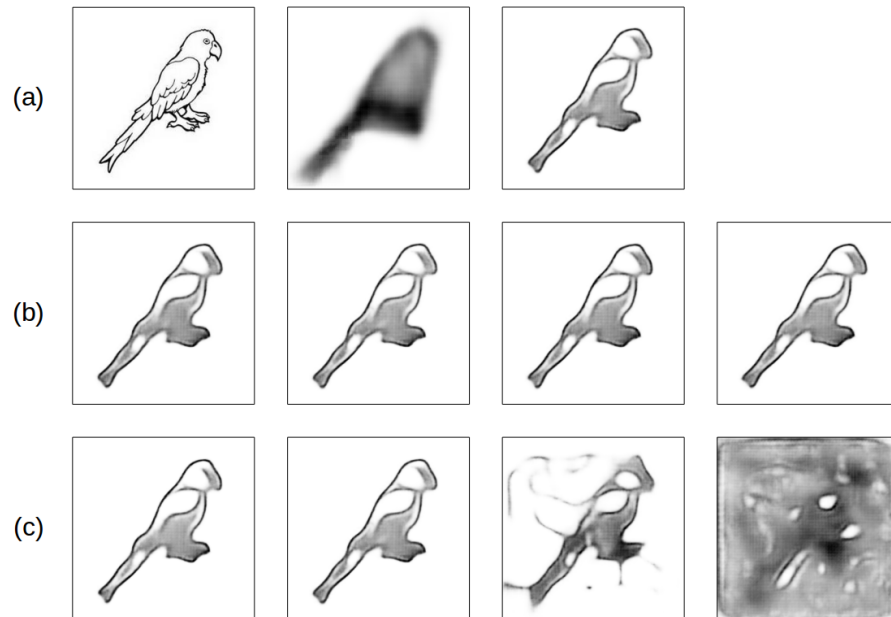


Figure 12.5: Reconstruction examples of the PARROT stimulus. (a) Original image¹² and reconstruction with the R_{DEFAULT} and R_{BEST} configurations. (b) Reconstruction with the R_{BEST} configuration for different noise patterns applied to the input. (c) Reconstruction with the R_{BEST} configuration for different noise levels (0%, 10%, 25%, 55%).

but some artifacts start to appear, while for 55% noise the shape of the original input is barely recognizable in the reconstruction.

12.4 TRANSFER LEARNING

Overview

In this section, we report on our first set of experiments for learning a mapping from images to our SHAPE similarity space. In our transfer learning setting (cf. Section 6.2.3), we will train a linear regression on top of the high-level activations of a pretrained neural network, resembling the setup of our machine learning experiments from Section 9.2. We will first investigate the performance of the photograph-based inception-v3 network [400] in Section 12.4.1, using a publicly available set of pretrained weights.¹³ In Sections 12.4.2 and 12.4.3, we will then apply the same transfer learning procedure to the classification-based and reconstruction-based networks from Sections 12.3.1 and 12.3.2, respectively. By comparing the results of these individual experiments, we hope to gain some insights with respect to the question, to which extent CNNs are capable of modeling human shape perception (cf. the discussions from Section 10.1.4).

The target space

In all of these experiments, we will employ the four-dimensional mean space from Chapter 11 as target space for the mapping task. The

¹² Image license CC BY-NC 4.0, source: <http://clipartmag.com/cockatiel-drawing>.

¹³ These weights can be found at <http://download.tensorflow.org/models/image/imagenet/inception-2015-12-05.tgz>.

choice of the target space has been motivated by the observation from Section 11.5, that similarity spaces with three to five dimensions give a good trade-off between compactness and expressiveness. Moreover, similarity spaces based on the mean dissimilarity matrix seemed to be slightly favorable over spaces based on the median matrix. Finally, using a four-dimensional target space allows us to make meaningful comparisons to our transfer learning experiments on the NOUN dataset [199] (cf. Section 9.2), which also used a four-dimensional target space.

In all experiments, we extracted the hidden representation of the respective network for each of the augmented line drawings. We then trained a linear regression (cf. Section 5.2.1) from these feature spaces to the four-dimensional target space. In addition to the linear regression, we also considered a lasso regression (which introduces an additional L1 penalty on the model's weights, cf. Section 5.2.1) with the following settings for the regularization strength β :

$$\beta \in \{0.001, 0.002, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10\}$$

We considered the mean squared error (MSE), the coefficient of determination R^2 , and the mean Euclidean distance (MED) as evaluation metrics in our five-fold cross-validation (cf. Sections 9.2.1 and 12.2.3).

12.4.1 Photograph-Based Classification

In our first experimental step, we replicate the setup from our NOUN experiments in Section 9.2 as closely as possible. We again used the 2048 features extracted by the photograph-based inception-v3 network [400] as a feature space (henceforth referred to as $C_{\text{INCEPTION}}$) for both a linear and a lasso regression. For the linear regression, we again considered both correct and shuffled target coordinates in order to check, whether the structure of the similarity space is necessary for successful generalization. In order to make the results more comparable to the ones obtained by our sketch-based networks, we furthermore considered three noise levels for salt and pepper noise during training, namely, 10%, 25%, and 55%. During testing, no salt and pepper noise was applied. Moreover, in this section, we consider four-dimensional spaces based both on the mean and the median matrix in order to validate, that our choice of the mean space has only minor impact on the results. We compared our results to the zero baseline, which always predicts the origin of the target space (cf. Section 9.2.1).

Tables 12.9 and 12.10 show the results obtained for a linear regression into the four-dimensional target spaces based on the mean and the median ratings, respectively. In both cases, we can observe the same general effects: The linear regression is clearly able to beat the zero baseline when trained on the correct targets with small to medium noise levels. However, if the assignment of input images to target coordinates is shuffled, then the linear regression fails to generalize to the test set, although its performance on the training set is only

Machine learning models

Evaluation

Overall setup

Results for a linear regression

Regressor	Noise Level	Targets	Training			Test		
			MSE	MED	R ²	MSE	MED	R ²
Baseline	All	All	1.0000	0.9940	0.0000	1.0000	0.9940	0.0000
Linear	0.1	Correct	0.0110	0.0961	0.9891	0.5506	0.7161	0.4513
		Shuffled	0.0207	0.1319	0.9792	1.2567	1.0997	-0.2592
	0.25	Correct	0.0196	0.1275	0.9805	0.6488	0.7835	0.3559
		Shuffled	0.0382	0.1781	0.9614	1.2614	1.0998	-0.2646
	0.55	Correct	0.1919	0.3932	0.8071	3.3461	1.7295	-2.2726
		Shuffled	0.2890	0.4892	0.7084	3.3354	1.7193	-2.3434

Table 12.9: Results for the zero baseline and a linear regression into the four-dimensional mean space, using different noise levels. Best results on the test set are highlighted in boldface.

Regressor	Noise Level	Targets	Training			Test		
			MSE	MED	R ²	MSE	MED	R ²
Baseline	All	All	1.0000	0.9781	0.0000	1.0000	0.9781	0.0000
Linear	0.1	Correct	0.0112	0.0969	0.9885	0.5410	0.6960	0.4366
		Shuffled	0.0199	0.1293	0.9797	1.2016	1.0655	-0.2206
	0.25	Correct	0.0198	0.1283	0.9793	0.6285	0.7596	0.3523
		Shuffled	0.0371	0.1749	0.9623	1.2260	1.0746	-0.2276
	0.55	Correct	0.1883	0.3889	0.8015	3.2534	1.7185	-2.4289
		Shuffled	0.2808	0.4801	0.7203	2.9972	1.6256	-2.0284

Table 12.10: Results for the zero baseline and a linear regression into the four-dimensional median space, using different noise levels. Best results on the test set are highlighted in boldface.

moderately impaired. This indicates, that the semantic structure of the target spaces is meaningful and required for generalization. With increased noise levels, both the training and the test set performance deteriorate. While the difference between 10% noise and 25% noise is relatively small, adding 55% noise to the inputs causes the regression to collapse completely: It is not able to generalize to the test set at all, being considerably worse than the baseline for both correct and shuffled target coordinates. Also in Section 12.3, a noise level of 10% was preferable for both sketch classification and sketch reconstruction networks. In the following, we therefore only consider the scenario with 10% salt and pepper noise.

Mean vs. median targets

A comparison between Tables 12.9 and 12.10 shows, that both target spaces result in similar performance levels. While MSE and MED are slightly better on the median space, the mean space reaches a slightly higher value for R^2 . The difference with respect to the MED of the zero baseline indicates, that the distribution of points in the mean space is closer to the surface of a unit hypersphere than the distribution of points in the median space (cf. Section 9.2.4).

The effect of target space dimensionality

So far, the difference between mean and median spaces seems to be negligible. In order to test, whether this observation generalizes

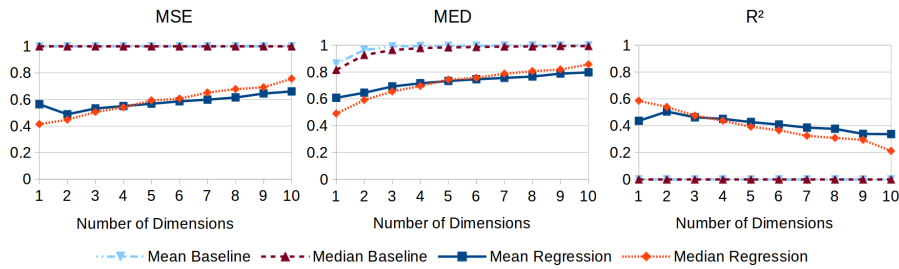


Figure 12.6: Test set results for the linear regression on $C_{\text{INCEPTION}}$ for both mean and median spaces of different dimensionality.

also to higher- and lower-dimensional target spaces, we obtained the performance of the zero baseline and the linear regression for all other possible target spaces (mean and median, one to ten dimensions), using a noise level of 10% during training and correct targets. Figure 12.6 visualizes the results of this experimental step. As we can see, performance with respect to all three metrics tends to get worse with an increasing number of dimensions of the target space. For the mean spaces, one or two dimensions seem to be preferable, while the median spaces reach their highest performance in a one-dimensional space. Overall, it seems, that the median spaces tend to perform slightly better for a smaller number of dimensions, while the mean spaces are slightly preferable for higher-dimensional spaces. This is somewhat surprising, since so such effect has been observed in our analysis of the similarity spaces in Chapter 11. Please note, that also for the zero baseline, the MED tends to increase, an effect, which has been observed before in Section 9.2.5, and which can be explained by the distribution of points becoming more uniform in higher-dimensional spaces.

We were furthermore able to observe, that an increase in the number of dimensions impacts test set performance much more than training set performance, which indicates, that overfitting plays a crucial role in our experiments. In fact, test set performance tended to be best when the degree of overfitting (computed as the fraction of test set error and training set error, cf. Section 9.2.4) was lowest. The preference for low-dimensional target spaces is in line with our observations from Section 9.2.5, and can again be explained by the reduced risk of overfitting. It is, however, somewhat surprising, that a one-dimensional similarity space is optimal for the median spaces, since one-dimensional spaces have shown a poor reflection of dissimilarity ratings in Section 11.2.3.

Overfitting tendencies

Table 12.11 shows the results of a lasso regressor on the four-dimensional target spaces, using 10% noise and varying values of the regularization strength β . A regularization strength of zero corresponds to the linear regression results reported in Tables 12.9 and 12.10. As we can see, a modest degree of regularization can help to improve the results slightly. The optimum is reached for $\beta \in \{0.005, 0.01\}$. If regularization becomes too strong, there are no improvements over the linear regression ($\beta \geq 0.5$ for the mean space, and $\beta \geq 0.1$ for the

Results for a lasso regressor

β	Mean Space			Median Space		
	MSE	MED	R^2	MSE	MED	R^2
0	0.5506	0.7161	0.4513	0.5410	0.6960	0.4366
0.001	0.5252	0.6972	0.4764	0.5234	0.6832	0.4560
0.002	0.5159	0.6898	0.4856	0.5137	0.6758	0.4676
0.005	0.5090	0.6828	0.4924	0.5035	0.6670	0.4801
0.01	0.5164	0.6861	0.4847	0.5008	0.6633	0.4834
0.02	0.5182	0.6859	0.4826	0.5025	0.6657	0.4800
0.05	0.5204	0.6872	0.4806	0.5211	0.6818	0.4613
0.1	0.5241	0.6934	0.4763	0.5541	0.7053	0.4306
0.2	0.5319	0.7005	0.4680	0.5942	0.7330	0.3898
0.5	0.6022	0.7506	0.3983	0.6579	0.7833	0.3192
1	0.7488	0.8525	0.2527	0.7871	0.8675	0.1872
2	0.9799	0.9839	0.0204	0.9633	0.9604	0.0246
5	1.0146	1.0010	-0.0147	1.0085	0.9821	-0.0084
10	1.0146	1.0010	-0.0147	1.0085	0.9821	-0.0084

Table 12.11: Test set results for the lasso regressor with respect to the four-dimensional target spaces, using 10% salt and pepper noise in during training and no noise during testing.

median space). For $\beta \geq 5$, the model collapses and performs slightly worse than the baseline.

Mean vs. median targets

Again, performance differences between the mean and the median spaces are relatively small, with the mean space performing better with respect to R^2 , and the median spaces yielding better values with respect to MSE and MED. Since we found only negligible differences between mean and median spaces for both regressors, we will only consider mean spaces for the remainder of this chapter.

Comparison to the NOUN experiments

Overall, we observe similar effects as for our study on the NOUN dataset [199] from Section 9.2: Like in Section 9.2.3, learning the correct mapping successfully generalized to the test set, while learning a shuffled mapping did not generalize well. We also again observed large overfitting effects, which can be somewhat mitigated through regularization. In Section 9.2.4, we compared similarity spaces obtained by different MDS algorithms, and found only negligible differences. In this section, we considered similarity spaces based on the mean and median dissimilarity ratings, respectively, and were again only able to find small performance differences. Again, low-dimensional target spaces tend to yield better results than high-dimensional target spaces, an effect already observed in Section 9.2.5 and attributed mainly to the increased risk of overfitting.

A more direct comparison

Table 12.12 compares the results from Chapter 9 with respect to the four-dimensional space obtained from nonmetric SMACOF to the

Dataset	Regressor	MSE	MED	R^2
NOUN	Linear	0.6076	0.7461	0.3706
	Lasso ($\beta \in \{0.005, 0.01\}$)	0.5937	0.7316	0.3853
Shapes	Linear	0.5506	0.7161	0.4513
	Lasso ($\beta \in \{0.005, 0.01\}$)	0.5090	0.6828	0.4924

Table 12.12: Comparison between the results from our NOUN study (Section 9.2.4) and our results on the shapes data.

results described in this section based on the four-dimensional mean space. Both target spaces have the same dimensionality, have been created with the same MDS algorithm, and are based on an aggregation of individual dissimilarities with the arithmetic mean. Moreover, the same machine learning approaches (linear and lasso regression on top of the inception-v3 network) have been applied. The only differences concern the underlying stimuli (photographs of novel objects vs. line drawings of everyday objects), the type of similarity judgments (overall similarity vs. visual similarity), and the elicitation method for the dissimilarity ratings (SpAM vs. pairwise ratings).

As we can see in Table 12.12, both the absolute performance level and the relative gains from regularization are higher for our SHAPE space than for the NOUN dataset. Since such effects are unlikely to be based on the elicitation method (cf. Section 8.1.2, where we discussed multiple studies that found a good match between the two elicitation methods), we can attribute them to the underlying stimulus set or the type of similarity. It thus seems, that our SHAPE spaces are somewhat easier to fit – either because they use simpler input images, or because they focus on a single cognitive domain. Additional investigations are required to disambiguate between these potential sources of performance differences.

Differences and potential reasons

12.4.2 Sketch Classification

We have argued in Section 12.1, that sketches may be a more useful source domain than photographs when it comes to processing line drawings. In this section, we test this hypothesis by applying the transfer learning approach from Section 12.4.1 to the sketch-based classification network from Section 12.3.1.

Motivation

We considered all four network variants, i.e., C_{DEFAULT} , C_{LARGE} , C_{SMALL} , and $C_{\text{CORRELATION}}$. For each of these four networks, we extracted the features of the bottleneck layer and trained both a linear and a lasso regression from the respective feature space into the four-dimensional mean space. We again applied 10% salt and pepper noise during training and no noise during testing. In Section 12.3.1, each network configuration has been trained on the sketch classification task in a five-fold cross-validation, resulting in five different sets of weights (one per fold). For each of these sets of weights, we trained and evaluated

Experimental setup

Feature Space	Regressor	Training			Test		
		MSE	MED	R^2	MSE	MED	R^2
Any	Zero Baseline	1.0000	0.9940	0.0000	1.0000	0.9940	0.0000
$C_{\text{INCEPTION}}$ (2048)	Linear	0.0110	0.0961	0.9891	0.5506	0.7161	0.4513
	Lasso ($\beta = 0.005$)	0.0173	0.1204	0.9828	0.5090	0.6828	0.4924
C_{LARGE} (2048)	Linear	0.0513	0.2073	0.9486	0.5540	0.6848	0.4451
	Lasso ($\beta = 0.2$)	0.1851	0.3983	0.8139	0.5065	0.6602	0.4929
C_{DEFAULT} (512)	Linear	0.0479	0.2004	0.9520	0.5567	0.6879	0.4409
	Lasso ($\beta = 0.05$)	0.1911	0.4062	0.8080	0.4775	0.6419	0.5216
C_{SMALL} (256)	Linear	0.0541	0.2129	0.9457	0.5373	0.6737	0.4575
	Lasso ($\beta = 0.02$)	0.1872	0.4009	0.8117	0.4737	0.6396	0.5246
$C_{\text{CORRELATION}}$ (512)	Linear	0.0563	0.2145	0.9433	0.7307	0.7825	0.2624
	Lasso ($\beta = 0.05$)	0.2184	0.4299	0.7801	0.5478	0.6815	0.4505

Table 12.13: Results of our transfer learning experiments based on the sketch classification network. The best values for each of the three evaluation criteria on the test set are highlighted in boldface.

the regressors in another five-fold cross validation analogously to the procedure from Section 12.4.1. We took the average regression performance across all of these five folds and across all five sets of network weights when reporting our results. We only considered correct targets, and report only the best performance of the lasso regressor along with the selected optimization strength. The full set of results for the lasso regressor can be found in Appendix G.2.

Experimental
Results

Table 12.13 shows the result of these transfer learning experiments and compares them to the results on $C_{\text{INCEPTION}}$ feature space from Section 12.4.1. As we can see, the linear regression performs considerably better than the zero baseline in all cases. Moreover, additional regularization through a lasso regression further helps to improve regression performance on all feature spaces.

Fitting the training
examples

Performance on the training set was comparable across all of our own network configurations and notably worse than training performance on $C_{\text{INCEPTION}}$. This indicates, that the feature space learned by our sketch classification network is somehow harder to fit than the feature space obtained from the inception-v3 network. Since training performance does not differ much between our different network setups, we can assume, that the size of the bottleneck layer (and thus the dimensionality of the feature space) does not play a major role for fitting the training set. We still observe a difference between $C_{\text{INCEPTION}}$ and C_{LARGE} , although both feature spaces have the same number of dimensions. One may thus conclude, that the ease of fitting the training set is more determined by the *type* of features rather than by their number.

Considering test set
performance

When comparing performance of the linear regression on the test set, we can again see barely any influence of the representation size on the results: The configurations C_{LARGE} , C_{DEFAULT} , and C_{SMALL} yield almost identical performance, which is quite similar to the performance

observed for the $C_{\text{INCEPTION}}$ feature space as reported in Section 12.4.1. However, we see slight advantages of our approach with respect to the mean Euclidean distance (MED). The $C_{\text{CORRELATION}}$ configuration (which sacrificed classification accuracy for a higher correlation to the psychological dissimilarity ratings) performs considerably worse than all other approaches with respect to all evaluation metrics. Even when using regularization in the form of a lasso regressor, it is barely able to reach the performance level obtained by a linear regression on the other feature spaces.

Regularization seems to be more helpful on our sketch-based feature spaces than on the photograph-based $C_{\text{INCEPTION}}$. It furthermore seems, that a smaller bottleneck size is beneficial in this case. Moreover, we can observe, that the regularization strength required for optimal performance of a lasso regressor is roughly proportional to the size of the bottleneck layer: Larger feature spaces require more regularization. This observation, however, only holds for our own feature spaces, which need considerably larger amounts of regularization than $C_{\text{INCEPTION}}$.

Overall, our feature spaces seem to be more robust towards overfitting than $C_{\text{INCEPTION}}$ as indicated by the smaller gap between training and test performance. At the same time, the regression benefits from larger degrees of regularization on our feature spaces, which shows, that a simple penalty on the size of the weights can effectively reduce overfitting tendencies even further. Summing up, transfer learning on sketch data seems to be more successful than transfer learning on photographs when considering a lasso regressor. Although the performance differences are certainly notable, the overall performance level remains quite low with $MSE \approx 0.47$, $MED \approx 0.64$, and $R^2 \approx 0.52$.

Unlike our sketch-based networks, the inception-v3 network [400] has not been explicitly trained with salt and pepper noise. One may thus speculate, that the observed performance difference is based on the robustness against noise. In order to investigate this hypothesis, we have compared the results of a linear regression for different noise regimes. Table 12.14 compares the behavior of a linear regression on the $C_{\text{INCEPTION}}$ and C_{DEFAULT} feature spaces, as well as on another variant of the sketch recognition network, which has been pretrained without any noise (labeled as $C_{\text{NO NOISE}}$). We compare the results of training the linear regression with 10% salt and pepper noise, while evaluating without any noise to two settings, in which the noise level on the training and the test set is identical (either 10% or 0%).

As we can see in Table 12.14, the linear regression on the $C_{\text{INCEPTION}}$ feature space is quite sensitive to different noise regimes: Using the same type of noise during training and testing considerably improves performance on the test set. This observation indicates, that a change in the noise level between training and evaluation is problematic for $C_{\text{INCEPTION}}$, presumably because the inception-v3 network is not able to successfully filter out the salt and pepper noise applied to its inputs. Somewhat surprisingly, training and evaluating on corrupted images yields even better results with respect to all three evaluation metrics

*Effects of
regularization*

*Comparison to
 $C_{\text{INCEPTION}}$*

*Robustness against
noise*

*$C_{\text{INCEPTION}}$ is
noise-sensitive*

Feature Space	Noise		Training			Test		
	Training	Test	MSE	MED	R ²	MSE	MED	R ²
$C_{\text{INCEPTION}}$	10%	0%	0.0110	0.0961	0.9891	0.5506	0.7161	0.4513
	0%	0%	0.0039	0.0573	0.9962	0.4413	0.6254	0.5604
	10%	10%	0.0110	0.0961	0.9891	0.3681	0.5672	0.6337
C_{DEFAULT}	10%	0%	0.0479	0.2004	0.9520	0.5567	0.6879	0.4409
	0%	0%	0.0450	0.1936	0.9549	0.5472	0.6841	0.4504
	10%	10%	0.0478	0.2004	0.9520	0.5249	0.6684	0.4727
$C_{\text{NO NOISE}}$	10%	0%	0.0919	0.2758	0.9077	65657.2780	96.1381	-66385.4946
	0%	0%	0.0389	0.1799	0.9609	0.5300	0.6734	0.4675
	10%	10%	0.0919	0.2758	0.9077	7.9197	0.8132	-7.2002

Table 12.14: Dependence of the linear regression results on the noise regime during training and testing. Best test set results for each feature space are highlighted in boldface.

Feature Space	0% Noise			10% Noise		
	d_E	d_M	d_C	d_E	d_M	d_C
$C_{\text{INCEPTION}}$ (2048)	0.5273	0.5307	0.7704	0.3794	0.3833	0.6103
C_{LARGE} (2048)	0.4115	0.3950	0.6411	0.4030	0.3866	0.6329
C_{DEFAULT} (512)	0.4147	0.4105	0.6448	0.4055	0.4014	0.6364
C_{SMALL} (256)	0.4045	0.4029	0.6347	0.3958	0.3943	0.6263
$C_{\text{CORRELATION}}$ (512)	0.3328	0.3328	0.5310	0.3225	0.3224	0.5180
$C_{\text{NO NOISE}}$ (512)	0.4059	0.4023	0.6341	0.2565	0.2556	0.4050

Table 12.15: Cluster analysis of the augmented images in the individual classification-based feature spaces using the Silhouette coefficient [339] (averaged across all folds, if applicable).

than always using uncorrupted images. One may speculate, that this effect is related to the texture-dependency reported for convolutional neural networks [24, 163, 375] (cf. Section 10.1.4): The added noise may appear to the network like an (irregular) texture, helping it to distinguish figure from ground.

C_{DEFAULT} is more robust

For our C_{DEFAULT} configuration, the performance differences between different noise regimes are much less pronounced. This indicates, that our sketch-based network is capable of filtering out salt and pepper noise, which is not surprising, since it has been explicitly trained in the presence of noise. Classification results of the $C_{\text{NO NOISE}}$ configuration were slightly better than in the C_{DEFAULT} case, but the network turned out to be extremely sensitive to noise: We only obtained reasonable regression results in the complete absence of noise, while an insertion of noise during training lead to a complete collapse of the regressor.

Outlook

Since we are interested in a mapping function that is robust in both the presence and absence of noise, the superior results of the $C_{\text{INCEPTION}}$ feature space for identical noise conditions are not of crucial importance for our overall investigations. However, they warrant further analysis in subsequent studies.

We have furthermore computed the Silhouette coefficient [339] (which measures, how well different classes are separated in a given feature space, cf. Section 9.2.2) for all our candidate feature spaces in order to investigate, how well the different augmented versions of our Shape stimuli from Chapter 10 are separated in the different feature spaces. The results of this analysis can be found in Table 12.15, where we report the Silhouette coefficient for three commonly used distance metrics (namely, the Euclidean distance d_E , the Manhattan distance d_M , and the Cosine distance d_C), and where we consider two noise settings (namely, 0% and 10% salt and pepper noise). Please recall from Section 9.2.2, that larger values of the Silhouette coefficient indicate a clearer separation of clusters. When looking at Table 12.15, we can see, that the structure of the feature space depends heavily on the noise level for both the $C_{\text{INCEPTION}}$ and the $C_{\text{NO NOISE}}$ feature space. For all other feature spaces, we observe only minor differences between the different noise settings, supporting our claim, that our network is able to successfully filter out noise. While the $C_{\text{INCEPTION}}$ feature space yields a better clustering for uncorrupted images, it is slightly inferior to our feature spaces in the presence of noise.

We can furthermore observe, that the size of the bottleneck layer (and hence the dimensionality of the feature space) does not seem to play a major role for separating the different clusters. Moreover, the $C_{\text{CORRELATION}}$ feature space is the least successful with regards to the Silhouette coefficient. Although the arrangement of the original stimuli in this feature space is better aligned with the original dissimilarity ratings (as indicated by the large value of Kendall's τ , cf. Section 12.3.1), the different augmented versions of the same original stimulus are not mapped onto similar feature vectors. This is even the case in the absence of noise, indicating, that the $C_{\text{CORRELATION}}$ setting is not capable of undoing the other augmentation steps of rescaling and translating the stimulus. We assume, that this is based on the fact, that the $C_{\text{CORRELATION}}$ setup has obtained its minimal validation set loss after only about five epochs (cf. Section 12.3.1): The network may simply not have had enough time to learn about these invariances.

Finally, we would like to point out, that the inception-v3 network [400] achieved state of the art classification results on ImageNet [120], while our own sketch classification network performs notably worse than the reported state of the art on sketch data (cf. Section 12.3.1). Moreover, the inception-v3 network has considerably more trainable parameters than our encoder network, and has been trained on a considerably larger set of target classes with a considerably larger number of training examples. If we further improved the performance of our own network on sketch classification, we may obtain better results also with respect to the mapping task, even in noise regimes, where the $C_{\text{INCEPTION}}$ feature space currently seems to have advantages.

Feature Space	Regressor	Training			Test		
		MSE	MED	R^2	MSE	MED	R^2
Any	Zero Baseline	1.0000	0.9940	0.0000	1.0000	0.9940	0.0000
$C_{\text{INCEPTION}}$ (2048)	Linear	0.0110	0.0961	0.9891	0.5506	0.7161	0.4513
	Lasso ($\beta = 0.005$)	0.0173	0.1204	0.9828	0.5090	0.6828	0.4924
C_{SMALL} (256)	Linear	0.0541	0.2129	0.9457	0.5373	0.6737	0.4575
	Lasso ($\beta = 0.02$)	0.1872	0.4009	0.8117	0.4737	0.6396	0.5246
R_{DEFAULT} (512)	Linear	0.2206	0.4288	0.7773	0.9709	0.9054	0.0168
	Lasso ($\beta \in \{0.02, 0.05\}$)	0.6378	0.7283	0.3591	0.8315	0.8739	0.1631
R_{BEST} (512)	Linear	0.1410	0.3383	0.8587	1.0791	0.9362	-0.0886
	Lasso ($\beta = 0.02$)	0.3610	0.5598	0.6383	0.7376	0.8102	0.2605

Table 12.16: Results of our transfer learning experiments with the autoencoder (R_{DEFAULT} and R_{BEST}) in comparison to the best results from Sections 12.4.1 ($C_{\text{INCEPTION}}$) and 12.4.2 (C_{SMALL}). The best values for each of the three evaluation criteria on the test set are highlighted in boldface for each type of pretraining objective.

12.4.3 Sketch Reconstruction

Experimental setup

Analogously to our transfer learning experiments on the classification networks, we again extracted the bottleneck layer activations of the pretrained autoencoders, and trained a linear regression and a lasso regression on top of this representation. We considered the two configurations R_{DEFAULT} and R_{BEST} , and trained the regression for each of the five network variants obtained from our cross-validation from Section 12.3.2. Again, a five-fold cross validation on the regression task was used, and averaged results across all folds are reported. We again used a noise level of 10% during training and no noise during testing.

Results

Table 12.16 shows the results of these reconstruction-based transfer learning experiments. As a reference, it also contains the best results obtained for transfer learning on the photograph-based and sketch-based classification networks. As we can see, performance on the reconstruction-based feature spaces is considerably worse than on the classification-based feature spaces: A linear regression is barely able to beat the zero baseline, and also the usage of moderate levels of regularization only leads to relatively poor performance. A lasso regression yields better results on the R_{BEST} configuration than on the R_{DEFAULT} setting, but the opposite can be observed for the linear regression: Here, the R_{BEST} configuration performs even worse than the baseline with respect to MSE and R^2 .

Cluster analysis

A contributing factor for this relatively poor performance can be seen in Table 12.17, where we analyze, how well the data points are clustered, using again the Silhouette coefficient [339] (cf. Sections 9.2.2 and 12.4.2): The different augmented versions of the same original line drawing do not form any notable clusters in the reconstruction-based feature spaces. The Silhouette coefficient is practically zero in most cases, which is a sharp contrast to our observations from Section 12.4.2.

Feature Space	0% Noise			10% Noise		
	d_E	d_M	d_C	d_E	d_M	d_C
$C_{\text{INCEPTION}}$ (2048)	0.5273	0.5307	0.7704	0.3794	0.3833	0.6103
C_{DEFAULT} (512)	0.4147	0.4105	0.6448	0.4055	0.4014	0.6364
C_{SMALL} (256)	0.4045	0.4029	0.6347	0.3958	0.3943	0.6263
R_{DEFAULT} (512)	0.0197	0.0223	-0.0359	0.0223	0.0238	-0.0300
R_{BEST} (512)	0.0351	0.0342	0.0818	0.0351	0.0344	0.0768

Table 12.17: Cluster analysis of the augmented images in both classification-based and reconstruction-based feature spaces (averaged across all folds, if applicable) using the Silhouette coefficient [339].

One may explain the observed difference in transfer learning performance through the distinction into global and local features, and the respective training objective from pretraining. For sketch classification, it is usually sufficient, if the network learns features relating to the global shape of the object, because small details are not expected to determine class membership. This was even an explicit criterion for the selection of classes in both TU Berlin [143] and Sketchy [348] (cf. Section 12.1.1). In the context of a reconstruction objective, however, also local shape features, which correspond to smaller details, need to be learned, because the reconstruction objective works in a pixel-based way. Thus, the autoencoder needs to learn both global and local shape features in its bottleneck layer. Thus, the type of representation learned by a classifier and by an autoencoder may be qualitatively different. Since our target similarity space tends to focus on global rather than local features (as indicated by interpretable directions for the three global features FORM, LINES, and ORIENTATION, cf. Section 11.4.3), classification-based pretraining may result in a more useful representation for our mapping task than reconstruction-based pretraining. Using a somewhat higher-level reconstruction objective, which abstracts away from individual pixel values (such as in the work by Larsen et al. [244], which was introduced in Section 6.3.4) could potentially alleviate this problem.

The explanation based on local features can, however, be criticized from different angles: Firstly, the study by Baker et al. [24] (cf. Section 10.1.4) has shown, that also classifier CNNs often focus on local rather than global shape features. One may argue, that their study considered only photograph-based networks and not sketch-based ones. However, we found good transfer learning performance also for the photograph-based $C_{\text{INCEPTION}}$, but not for R_{DEFAULT} and R_{BEST} . If the explanation about local vs. global shape features was correct, we would, however, not expect such a performance difference. Secondly, the need to encode both global and local shape features should imply the need for a larger bottleneck layer than for sketch-based classification, where only global features need to be encoded. However, when optimizing the hyperparameters of the autoencoder in Section 12.3.2, we could not find a strong dependence of reconstruction quality on the size of the bottleneck layer. In fact, the impact of bottleneck size seemed to be smaller than for the

*First explanation:
global vs. local
features*

*Limitations of this
explanation*

sketch classification network (cf. Section 12.3.1). Thirdly, the distinction into global and local shape features cannot explain the clustering effects from Table 12.17: In Table 12.17, we consider the similarity of the representations for different augmented versions of the same original line drawing. Please recall from Section 12.2.1, that our augmentation steps for the line drawings consist only in rescaling and translating the object – other modifications such as horizontal flips, rotations, and shears were not applied. Thus, our augmentation leaves both the global and the local shape structure intact – reconstruction-based representations should thus have a similar clustering quality as classification-based ones, which is, however, not observed in Table 12.17. Overall, this explanation for the inferiority of reconstruction-based pretraining thus seems to be at least questionable.

*Second explanation:
data augmentation*

We can also explain the observed difference between classification-based and reconstruction-based networks as follows: Data augmentation mainly consists in resizing and translating the object within the overall image. While the information about the exact size and location of the object is largely irrelevant in a classification context, it plays a crucial role for reconstruction. After all, the reconstruction error can only be minimized by reconstructing the object at the correct location in the correct size. Hence, variations of object size and location in the input need to be encoded in the bottleneck layer for a reconstruction task, but not for a classification task. Since different augmented versions of the same line drawing are thus represented in very different ways in the reconstruction-based feature space, the linear regression has difficulties in finding a straightforward mapping into the target similarity space. Using a reconstruction objective by itself therefore does not seem to be promising on our small dataset, where data augmentation is a necessary preprocessing step in order to obtain enough training examples. Using a more coarse-grained reconstruction objective may again be helpful to avoid this problem.

*Entangled
representation as root
cause?*

This second explanation points towards an underlying *entangled* representation (cf. Section 6.1.1): The autoencoder network is apparently able to represent the shape of the objects (as visualized by the reconstructions in Figure 12.5), but we are not able to directly access this information through a linear model, since it is tightly intertwined with information about the location and size of the object.

*Towards disentangled
representations*

Using approaches such as InfoGAN [101] (cf. Section 6.3.3) or β -VAE [196] (cf. Section 6.3.4), which promise to learn a *disentangled* representation of their inputs, may help to alleviate this problem: Supposedly, information about object shape and about its location and size would be encoded in different subspaces of the feature space induced by the bottleneck layer. In this case, a regression to points in our target similarity space should be relatively straightforward. Also multitask learning may solve this problem to some extent: The network can learn the coordinates of the target space as part of its bottleneck representation (which would then presumably encode the object's shape), while the remainder of the bottleneck layer can be used to

encode the location and size of the object. We will investigate this further in Section 12.5.2.

12.5 MULTITASK LEARNING

In our multitask learning experiments, we trained the different configurations of our sketch-based networks again from scratch, using, however, not only the classification or reconstruction loss as single training objective, but also directly incorporating the mapping loss. Instead of a two-phase process as used in the transfer learning setup, we therefore optimized both objectives at once. For these multitask learning experiments, we limited ourselves to the sketch-based networks, since the photograph-based network would have been too computationally costly to retrain given our resources. In Section 12.5.1, we will report on the multitask learning experiments for the classification network, before analyzing the behavior of our autoencoder in the multitask context in Section 12.5.2.

Again, we confined the evaluation of the mapping performance to the four-dimensional similarity space based on the mean dissimilarity ratings. We again used the mean squared error (MSE), the mean Euclidean distance (MED), and the coefficient of determination R^2 for evaluating performance (cf. Sections 9.2.1 and 12.2.3). As before, we applied a five-fold cross validation, and we will report the results as average across all five test folds. In all experiments, we use a noise level of 10% during training and no noise during testing.

12.5.1 Sketch Classification

For our classification-based multitask learning experiments, we considered three of the network configurations from Section 12.3.1. In addition to our C_{DEFAULT} setting, we also employed the C_{SMALL} configuration with its reduced bottleneck size, since it has shown promising results in the transfer learning experiments (cf. Section 12.4.2). Although the $C_{\text{CORRELATION}}$ configuration has not been very successful in the transfer learning approach, we gave it another chance in the multitask setting. We did not consider the C_{LARGE} configuration, since its performance barely differed from the one of the C_{DEFAULT} configuration, despite using a considerably larger hidden representation, which implies a higher demand for both computation time and memory.

When training the different network configurations, we set the weight λ_1 of the classification loss to one, and we varied the weight λ_3 of the mapping loss in order to explore different trade-offs between the two tasks (cf. Equation 12.1 from Section 12.2.3). We explored the following settings for λ_3 :

$$\lambda_3 \in \{0.0625, 0.125, 0.25, 0.5, 1, 2\}$$

Overview

General experimental setup

Hyperparameter configurations

Combining two loss components

Configuration	Task	λ_3/β	Corr.	Acc. TU Berlin	Acc. Sketchy	MSE	MED	R^2
Zero Baseline	Any	–	–	–	–	1.0000	0.9940	0.0000
$C_{\text{INCEPTION}}$	Transfer	0.005	0.3948	–	–	0.5090	0.6828	0.4924
C_{DEFAULT}	Transfer	0.05	0.2743	0.6320	0.7933	0.4775	0.6419	0.5216
	Multitask	0.0625	0.4141	0.5640	0.7510	0.4041	0.5920	0.5775
C_{SMALL}	Transfer	0.02	0.2777	0.6274	0.7908	0.4737	0.6396	0.5246
	Multitask	0.125	0.4118	0.6145	0.7818	0.4182	0.6020	0.5567
$C_{\text{CORRELATION}}$	Transfer	0.05	0.3292	0.3636	0.6146	0.5478	0.6815	0.4505
	Multitask	2	0.4534	0.5196	0.7018	0.4513	0.6115	0.5201

Table 12.18: Test set results of the multitask learning experiments for the sketch-based classification setup. Best values for each of the evaluation metrics are highlighted in boldface.

Motivation for selected values

A mapping weight of $\lambda_3 = 0.25$ in the context of a fixed classification weight of $\lambda_1 = 1$ approximately reflects the relative proportion of mapping examples in our augmented dataset, where we have 490,000 examples for the classification task and 120,000 for the mapping task (cf. Section 12.2.1). This value for λ_3 thus seemed to be a good starting point for our investigations. We also considered larger and smaller values of λ_3 based on the following intuitions: On the one hand, one may need a stronger emphasis on the mapping task, since the mapping loss (measured by the mean squared error) of a naive solution (namely, the zero baseline) equals 1, while the classification loss (measured by the categorical cross-entropy) of a naive solution (using the prior probabilities of the classes) is considerably larger than one. On the other hand, a smaller emphasis on the mapping task may be needed due to the limited number of target coordinates, which may create a strong risk for overfitting.

Experimental results

Table 12.18 summarizes the results of our multitask learning experiment and compares them to the transfer learning results from Sections 12.4.1 and 12.4.2. More detailed results can be found in Appendix G.4. As we can observe, mapping performance is considerably better in the multitask setting than in the transfer learning setting for all of the configurations under investigation. The best results are obtained for the C_{DEFAULT} configuration, which is followed closely by C_{SMALL} . The $C_{\text{CORRELATION}}$ configuration performs again considerably worse, although its best multitask results are still slightly superior to all transfer learning approaches. These observations indicate, that the multitask learning regime is more promising than the transfer learning approach. Moreover, classification accuracy seems to be a more helpful selection criterion for hyperparameter configurations than the correlation to the original dissimilarities, since C_{DEFAULT} and C_{SMALL} clearly outperform $C_{\text{CORRELATION}}$.

The optimal trade-off for good classifiers

When taking a closer look at the optimal values for λ_3 , we can note, that for both the C_{DEFAULT} and the C_{SMALL} setting, relatively small values

of $\lambda_3 \in \{0.0625, 0.125\}$ have been selected. It thus seems, that the risk of overfitting outweighs the difference in the absolute loss sizes. This is also reflected by the number of epochs, after which the early stopping procedure terminated training: It was reduced from 188.6 to 76.4 for the C_{DEFAULT} setting and from 178.4 to 123.2 for the C_{SMALL} configuration, when changing from singletask to multitask training. Since we always trained the network for 200 full epochs and then picked the epoch with the lowest mapping loss on the validation set, earlier termination indicates an increased tendency towards overfitting.

We can furthermore observe, that the increase in mapping performance is obtained by sacrificing a certain degree of classification performance for both C_{DEFAULT} and C_{SMALL} , with the effect being more pronounced for C_{DEFAULT} . This is somewhat surprising, since one would expect, that a larger bottleneck allows for more redundancy and thus more "slack" to cope with additional constraints. This observation thus urges for further investigations. In both cases, the introduction of the mapping loss leads to a considerable increase in the correlation τ to the dissimilarity ratings. This effect is, however, to be expected, since the mapping loss tries to align a part of the bottleneck representation with the coordinates of the similarity space, which is explicitly based on the psychological dissimilarity ratings.

*Effects on
classification
performance*

For the $C_{\text{CORRELATION}}$ configuration, we make somewhat different observations: Here, a relatively large mapping weight of $\lambda_3 = 2$ leads to the best mapping results. Interestingly, this rather strong emphasis on the mapping task also helps to improve performance on the classification task quite drastically. We also observed, that the best results on the validation set were observed after 96.6 epochs instead of 5.6 epochs as in the classification-only setup from Section 12.3.1. The mapping task thus seems to act as a kind of regularization term for the classification task, which counteracts overfitting to some degree. However, the best classification performance of the $C_{\text{CORRELATION}}$ configuration in the multitask learning setting is still notably worse than the results observed for the C_{DEFAULT} and the C_{SMALL} setting.

*Mapping task as
regularizer for
 $C_{\text{CORRELATION}}$*

The largest correlation to the dissimilarity ratings reported in Table 12.18 is obtained for $C_{\text{CORRELATION}}$. However, this is simply an artifact of the larger mapping weight of $\lambda_3 = 2$. Also for C_{DEFAULT} and C_{SMALL} , a strong emphasis on the mapping task (with $\lambda_3 \in \{1, 2\}$) led to comparable correlations of $\tau \approx 0.45$. The fact, that in these cases the best mapping performance was nevertheless obtained for smaller values of λ_3 , illustrates again, that a high correlation to the dissimilarity ratings is not necessarily a good predictor of success in the mapping task.

*Correlations to
dissimilarities*

12.5.2 Sketch Reconstruction

For the multitask learning setting based on our autoencoder, we considered the two configurations R_{DEFAULT} and R_{BEST} from Section 12.3.2. Again, we used the four-dimensional mean space as target for our

Experimental setup

Configuration	Task	λ_3/β	Corr.	Rec. Error	MSE	MED	R^2
Zero Baseline	Any	–	–	–	1.0000	0.9940	0.0000
C_{DEFAULT}	Transfer	0.05	0.2743	–	0.4775	0.6419	0.5216
	Multitask	0.0625	0.4141	–	0.4041	0.5920	0.5775
R_{DEFAULT}	Transfer	0.02, 0.05	0.2228	0.1303	0.8315	0.8739	0.1631
	Multitask	2	0.3533	0.1391	0.6211	0.7297	0.3369
R_{BEST}	Transfer	0.02	0.3019	0.0828	0.7376	0.8102	0.2605
	Multitask	0.25, 0.5, 2	0.4033	0.1114	0.5494	0.6846	0.4213
	Multitask	0.0625	0.3893	0.1023	0.5504	0.6851	0.4144

Table 12.19: Test set results of the multitask learning experiments for the sketch-based reconstruction setup. Best values of the reconstruction approach for each of the evaluation metrics are highlighted in boldface.

regression experiments, applying the same five-fold cross validation strategy as before. We again report MSE, MED, and R^2 as averaged across all test folds.

Combining loss components

In order to investigate the optimal trade-off between the reconstruction objective and the mapping objective, we fixed the reconstruction weight λ_2 in Equation 12.1 to a value of one, while varying the value of the mapping weight λ_3 in the same steps as used in Section 12.5.1:

$$\lambda_3 \in \{0.0625, 0.125, 0.25, 0.5, 1, 2\}$$

Experimental results

Table 12.19 summarizes the results of these experiments, comparing them to both our transfer learning experiments from Section 12.4.3 and to the best classification-based multitask learning results from Section 12.5.1. More detailed tables with the raw results can be found in Appendix G.5.

Multitask learning vs. transfer learning

As we can see, the multitask learning approach is able to beat both the zero baseline and the transfer learning approach with respect to all three metrics. This confirms our results from Section 12.5.1, where we observed a similar pattern also for the classification-based networks. Moreover, our speculations from Section 12.4.3 also seem to be somewhat fulfilled: Multitask learning is able to considerably outperform transfer learning in the reconstruction setting, which can potentially be explained by the pressure to disentangle location and size of the stimulus (which are only relevant to the reconstruction task) from its shape (which is relevant to both the reconstruction and the mapping objective). However, further investigations are needed to analyze, whether the observed performance difference can actually be traced back to such a (partial) disentanglement.

Considering R_{DEFAULT}

Let us now look at the two different hyperparameter configurations R_{DEFAULT} and R_{BEST} in more detail. For the R_{DEFAULT} configuration, a mapping weight of $\lambda_3 = 2$ was found to be optimal with respect to all three mapping evaluation metrics MSE, MED and R^2 . Since this is the largest

candidate value for λ_3 , which we investigated, it is certainly possible that even larger values (e.g., $\lambda_3 = 5$) would have further increased performance. This should be tested in potential follow-up studies. Overall, we see a considerable improvement with respect to the mapping performance, but also with respect to the correlation to the original dissimilarity ratings. This is achieved by sacrificing only a relatively small amount of reconstruction performance.

Just as in Section 12.4.3 for the transfer learning experiments, the R_{DEFAULT} configuration is considerably outperformed by the R_{BEST} configuration. There, however, no clear optimal setting for λ_3 could be determined: The optimal value for R^2 was achieved for $\lambda_3 = 0.25$, while the best value of the MSE was obtained with $\lambda_3 = 0.5$, and the lowest MSE was reported for $\lambda_3 = 2$. The setting of $\lambda_3 = 0.0625$ seems to be a reasonable compromise candidate, since it obtained the second best results for MSE and MED, and the third best result for R^2 . Interestingly, this compromise setting is considerably smaller than the three individually optimal settings, highlighting, that there is little systematicity with respect to the optimal mapping weight λ_3 . Also for R_{BEST} , we again observed a considerable improvement over the transfer learning setting. This improvement, however, comes at a notably higher cost with respect to the reconstruction error.

For both hyperparameter configurations, we furthermore observed a reduction in the number of epochs, for which the network was trained with early stopping: The R_{DEFAULT} configuration terminated after 42.2 epochs in the multitask setting (for the optimal setting of $\lambda_3 = 2$), while it used 56.2 epochs when trained on the reconstruction task exclusively. For the R_{BEST} setting, this effect was even more pronounced, reducing the optimal number of epochs from 192.4 to values between 4.6 and 9.8 (depending on the value of λ_3 under consideration). Since we always trained the network for 200 full epochs and then used the snapshot with the lowest mapping loss on the validation set, this may indicate, that there is a strong tendency towards overfitting during multitask learning.

If we compare the reconstruction-based multitask learning results from Table 12.19 to the results of C_{DEFAULT} , we can observe, that the reconstruction objective leads to systematically worse mapping results than the classification objective: Reconstruction-based multitask learning still performs slightly worse than classification-based transfer learning. It thus seems, that even though multitask learning considerably improves the results of an autoencoder network in comparison to the transfer learning case, using a classification objective is superior with respect to both performance and training time.

This observation may be somewhat surprising, given that the reconstruction objective is in theory closer to the perceptual level targeted by our shape similarity spaces.¹⁴ As already argued in Section 12.4.3, the poor performance of our reconstruction-based networks can potentially

Considering R_{BEST}

The number of epochs

Comparison to classification-based results

Interpretation of differences

¹⁴ Remember from Section 11.4, that the pre-attentive feature ratings could be identified as directions in our SHAPE spaces. This means, that the structure of the similarity space is at least partially available already in a pre-attentive stage of perceptual processing.

be explained by our augmentation procedure, where we resized and translated the object (cf. Section 12.2.1): While the location and size information is important for the reconstruction objective, it is irrelevant for the target coordinates in the similarity space. Apparently, although multitask learning helps to disentangle these pieces of information to some degree (as reflected by the improved performance in comparison to transfer learning), the resulting internal representation is still not as useful as a classification-based representation. It thus seems, that the increased overhead of training a full autoencoder (which contains also a decoder network and a more complex loss function, leading to considerably longer training times), is not worth the effort in comparison to a somewhat more straightforward classification objective.

12.6 GENERALIZATION TO OTHER TARGET SPACES

Overview

So far, we have only considered the four-dimensional shape space, which was based on the mean dissimilarity ratings. In this section, we aim to investigate, how well the different approaches generalize to target spaces of different dimensionality. We consider all mean spaces with one to ten dimensions as regression targets. Since the median spaces have shown similar performance in our preliminary investigations from Section 12.4.2, and since they seem to have a similar internal structure as the mean spaces according to our analyses from Chapter 11, we confine ourselves to the mean spaces. Again, we report the average MSE, MED, and R^2 of a five-fold cross-validation, using 10% noise during training and no noise during testing. We first consider classification-based approaches in Section 12.6.1, before discussing the reconstruction-based approaches in Section 12.6.2.

12.6.1 Classification

Experimental setup

In the context of our classification-based networks, we considered the following three setups: Firstly, we use the transfer learning approach on the photograph-based $C_{\text{INCEPTION}}$ feature space, where a lasso regression with $\beta = 0.005$ has given the best results for our four-dimensional target space (cf. Section 12.4.1). Secondly, we considered the best transfer learning configuration of our sketch-based classification network, which corresponds to a lasso regression with $\beta = 0.02$ on top of the C_{SMALL} configuration (cf. Section 12.4.2). Thirdly, we investigated the multitask learning approach with the C_{DEFAULT} configuration and a mapping weight of $\lambda_3 = 0.0625$ (cf. Section 12.5.1).

Testing generalization

These three setups were applied to all other target spaces with one to ten dimensions without any further modification or hyperparameter tuning. This means, that the respective predictor (lasso regressor or multitask network) was retrained from scratch on the new target spaces, using the hyperparameter settings for β and λ_3 as specified above. Tuning β and λ_3 on each target space individually may lead to better

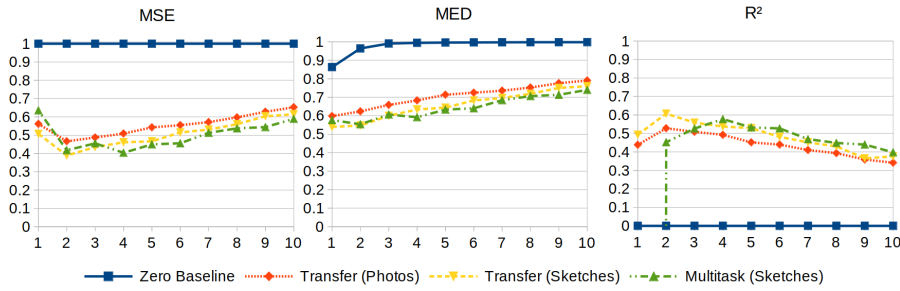


Figure 12.7: Results of our classification-based generalization experiment.

results, but keeping them fixed provides us with a "tougher" test of generalization performance. Moreover, retraining multiple versions of the multitask learner for each target space would have been computationally too costly.

Figure 12.7 visualizes the results of these generalization experiments for the three evaluation metrics with respect to the mapping task. Tables with the raw experimental results can be found in Appendix G.6. We can make the following general observations: Both transfer learning approaches reach their peak performance for a two-dimensional target space, even though they have been optimized on the four-dimensional similarity space. Only with respect to the MED, performance is best on the one-dimensional target space. However, also the MED of the zero baseline is smallest for a one-dimensional space. If we consider the relative MED (by dividing through the MED of the zero baseline), then the best performance is again obtained on a two-dimensional target space. In all cases, transfer learning on our C_{SMALL} configuration is slightly superior to transfer learning based on the $C_{\text{INCEPTION}}$ features.

For the multitask learning approach based on our C_{DEFAULT} configuration, a slightly different picture emerges: Here, peak performance is obtained for the four-dimensional target space with respect to both MSE and R^2 . With respect to the MED, the two-dimensional target space yields slightly better results than the four-dimensional space. Moreover, it seems, that the multitask learning approach is more sensitive to the dimensionality of the target space than the transfer learning approach: While our multitask learner consistently yields the best results for all spaces with four or more dimensions, it performs worse than the transfer learner for the two- and three-dimensional target spaces. It collapses completely for a one-dimensional target space, as illustrated by the negative value of R^2 .

Table 12.20 contains information about the training time and the classification performance of the multitask learner. When looking at the training time for the different target spaces, we can observe, that for both the one- and two-dimensional space, training was terminated after less than 25 epochs, while larger target spaces tended to lead to longer training. An early termination of the training procedure is an indicator of overfitting, since the validation set error reaches its minimum quite early in the training procedure. It is interesting to note,

Generalization pattern of transfer learning

Generalization pattern of multitask learning

The number of epochs

Number of Dimensions	Number of Epochs	Accuracy TU Berlin	Accuracy Sketchy
1	24.0	0.4421	0.6567
2	19.0	0.5090	0.7140
3	55.2	0.5253	0.7233
4	76.4	0.5640	0.751
5	75.4	0.5964	0.7725
6	25.6	0.5633	0.7534
7	95.2	0.5996	0.7760
8	104.8	0.6104	0.7813
9	135.6	0.6128	0.7852
10	145.6	0.6097	0.7843

Table 12.20: Training time and classification accuracy of the multitask learner for target spaces of different dimensionality.

that also classification accuracy seems to be monotonically related to the number of dimensions in the target space (and hence the number of epochs). By choosing different regularization regimes (e.g., stronger weight decay or a smaller weight for the mapping), one may thus be able to counteract these overfitting tendencies, and to obtain better results also for low-dimensional target spaces.

Summary

Overall, the results of this generalization experiment confirm the effects observed in Section 9.2.5 on the NOUN dataset [199], where we also found a performance sweet spot for a two-dimensional target space in a transfer learning setting. Again, we can argue, that this strikes a balance between a clear semantic structure in the target space and a small number of output variables to predict. While the transfer learning setup shows a very regular generalization pattern to target spaces of different dimensionality, the multitask learning approach seems to be somewhat more volatile. This indicates, that the target space should be carefully chosen before optimizing the multitask learner. Nevertheless, we are able to see, that the multitask learner is able to keep its performance advantage for a majority of the target spaces.

12.6.2 Reconstruction

Experimental setup

Let us now also take a look at the generalization capabilities of our reconstruction-based approaches. For the transfer learning setting, we considered a lasso regression on top of the pretrained R_{BEST} features with a fixed regularization strength of $\beta = 0.02$. For the multitask learning setting, we trained the R_{BEST} configuration from scratch using both the reconstruction objective ($\lambda_2 = 1$) and the mapping objective (with the compromise candidate of $\lambda_3 = 0.0625$ identified in Section 12.5.2). Just like in Section 12.6.1, both setups were applied to all

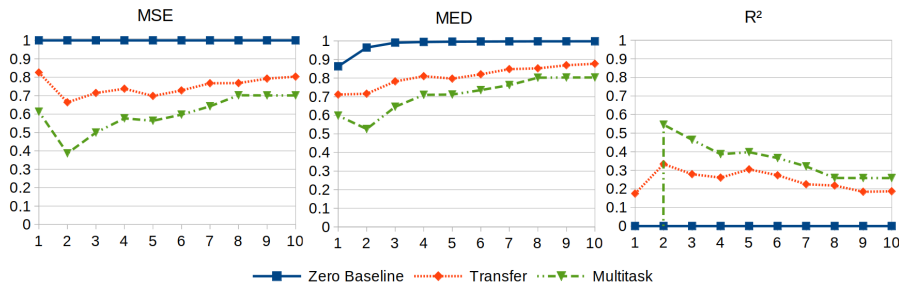


Figure 12.8: Results of our reconstruction-based generalization experiment.

Number of Dimensions	Number of Epochs	Reconstruction Error
1	11.2	0.1063
2	4	0.1133
3	6.6	0.1079
4	9.8	0.1023
5	8.2	0.1105
6	4	0.1124
7	5.6	0.1086
8	13	0.1121
9	10	0.1051
10	7.4	0.1071

Table 12.21: Training time and reconstruction error of the multitask learner for target spaces of different dimensionality.

target spaces from one to ten dimensions without further modifications.

Figure 12.8 illustrates the results of these experiments with respect to all three evaluation metrics targeting mapping performance. Tables with the raw results of these experiments can be found in Appendix G.7. As we can see, multitask learning is always more successful than transfer learning, except for R^2 on a one-dimensional target space, where the multitask learner again collapses completely. For both reconstruction-based approaches, we can observe a tendency for poorer performance on higher-dimensional spaces and optimal performance for a two-dimensional target space. Moreover, there seems to be a dent at five dimensions for most graphs, which in many cases yields better performance than both a four-dimensional and a six-dimensional target space. We assume, that this is, however, simply an artifact of the particular arrangement of stimuli in this five-dimensional space. As Table 12.21 shows, there is no clear pattern with respect to training time or reconstruction quality.

While similar generalization patterns have been observed for the classification-based transfer learning approaches in Section 12.6.1, the generalization pattern of the multitask learners seems to differ between the two settings: In Figure 12.8, the multitask learner follows essen-

General observations

Comparison to classification-based results

Configuration	Task	MSE	MED	R^2
C_{SMALL}	Transfer	0.3893	0.5472	0.6070
C_{DEFAULT}	Multitask	0.4178	0.5537	0.4525
	Multitask (4D)	0.4041	0.5920	0.5775
R_{BEST}	Transfer	0.6637	0.7159	0.3337
	Multitask	0.3868	0.5263	0.5454

Table 12.22: Comparison of best mapping performance in the generalization experiments. The best value for each evaluation metric is highlighted in boldface.

tially the same performance pattern as the transfer learner (with peak performance at two dimensions), while in Section 12.6.1, the multitask learner appeared to be less stable for lower dimensions and reached its best performance at four dimensions. Moreover, for the reconstruction-based network, we could not observe any clear effect of the target space dimensionality on training time or reconstruction performance.

Comparison of best performance

Table 12.22 summarizes generalization performance in the two-dimensional target space (which yielded the best performance for most approaches) for both classification-based and reconstruction-based learners. We additionally report the performance of the classification-based multitask learner on the four-dimensional target space, where it obtained its best performance.

Observations

As we can see in Table 12.22, the highest value for the coefficient of determination was obtained by the classification-based transfer learner with $R^2 \approx 0.61$, which is considerably larger than for all other approaches. Within the classification-based approaches, the transfer learning setup also achieved the best results with respect to MSE and MED. The reconstruction-based transfer learner is also in these generalization experiments clearly not competitive. However, the reconstruction-based multitask learner is capable of slightly outperforming the best classification-based approach with respect to both MSE and MED.

A surprising effect

This observation may be somewhat surprising, given the fact that reconstruction-based multitask learning was not competitive at all on the four-dimensional target space (cf. Section 12.5.2). Nevertheless, if we train this configuration on the two-dimensional target space (without further optimizing any hyperparameters such as the mapping weight λ_3), we are able to obtain very good results. How can this surprising effect be explained?

A possible explanation

Both multitask learners (based on the classification network and based on the autoencoder) used a mapping weight of $\lambda_3 = 0.0625$, i.e., the smallest value we investigated. While the influence of the mapping task can thus be considered to be approximately equal in both cases, the size of the classification and reconstruction loss has differed considerably: For the classification-based network, we observed a classification loss of around 1.3 to 1.6 on the test set, while for the autoencoder, the reconstruction loss had a range of 0.10 to 0.12 on

the test set. Since both losses are weighted with a weight of $\lambda_1 = 1$ and $\lambda_2 = 1$, respectively, the influence of the mapping objective on the overall optimization is greater in the classification-based multitask learner. This difference is not negligible, since it corresponds to a factor of approximately ten. One may therefore speculate, that even smaller values of λ_3 would have benefited the classification-based multitask learner for smaller target spaces.

Overall, one may thus need to conduct further experiments with a larger variety of candidate values for λ_3 , or, alternatively, with other fixed values for λ_1 and λ_2 , which normalize the respective objective onto the same order of magnitude. Our comparisons so far can thus not yet be deemed conclusive and urge for further research. Nevertheless, the observation, that reconstruction-based multitask learning gives competitive results in our generalization experiments can be seen as an argument for also further investigating autoencoders in the context of the mapping task.

Outlook

Figures 12.9 and 12.10 visualize the two-dimensional target space and the respective prediction performance of the individual approaches as indicated by their MED. In these visualizations, we assume, that the predicted point lies in the origin. The colored circles around the origin illustrate all points, which have a distance of the given regressor's MED to the origin, i.e., they show the average location of the ground truth, if the origin was predicted. The blue solid circle represents the results for the zero baseline, which simply reflects the overall distribution of the stimuli in the similarity space. As we can see, all other approaches (both classification-based and reconstruction-based, and both transfer learners and multitask learners) clearly outperform the baseline. In the classification setting (Figure 12.9), both sketch-based approaches yield comparable performance on this two-dimensional target space, and they clearly outperform the photograph-based transfer learner. Also the reconstruction-based approaches (Figure 12.10) show a similar level of performance, but with a clearer distinction between transfer learning (which is somewhat worse than photograph-based transfer learning) and multitask learning (which slightly outperforms the best classification-based approaches).

Visualizing the mapping performance

Overall, Figures 12.9 and 12.10 highlight, that the performance level achieved by our best regression is still far from satisfactory: While our regression might be capable of identifying the overall region of the similarity space correctly, the average distance between the prediction and the ground truth is still too large for any practical applications. This is a similar observation as made in Section 9.2.5 for the NOUN dataset [199], and it highlights, that further research and engineering effort is needed in order to improve the system's prediction quality.

The need for further research

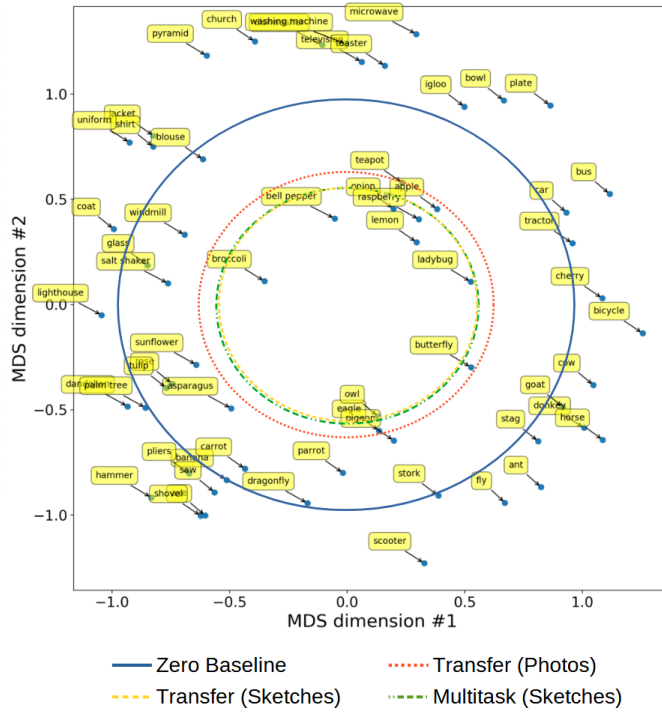


Figure 12.9: Visualization of the two-dimensional target similarity space and the performance of the individual classification-based regressors as measured with the MED.

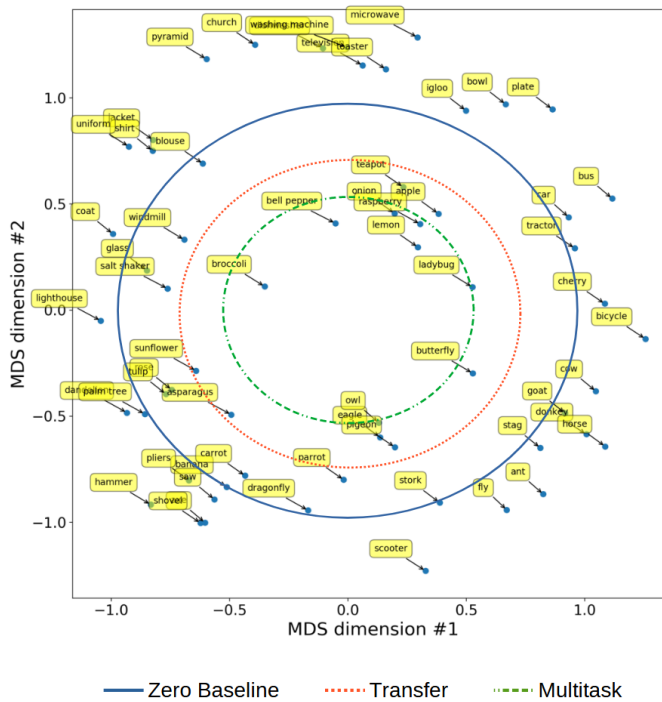


Figure 12.10: Visualization of the two-dimensional target similarity space and the performance of the individual reconstruction-based regressors as measured with the MED.

12.7 SUMMARY

In this chapter, we have aimed to learn a mapping from line drawings to their corresponding coordinates in the *SHAPE* spaces from Chapter 10. We have compared transfer learning to multitask learning, based on both classification and reconstruction as secondary training objectives.

Lessons learned

In Section 12.4, we replicated the transfer learning procedure from Section 9.2 by using a linear regression on top of the features extracted from a pretrained CNN. We were able to achieve better performance with our sketch-based classifier than with a photograph-based CNN, which seems to be mainly based on the increased robustness to varying noise regimes. Transfer learning on a reconstruction-based network was not competitive to the classification-based results, which we hypothesized to be the consequence of a highly entangled representation. Also Sanders and Nosofsky [346, 347] have used a similar transfer learning setup (cf. Section 8.7.3), but they also fine-tuned the weights of lower network layers. In contrast to that, our transfer learning approach only estimated the weights of the final regression, while leaving all other network parameters unchanged. We already observed strong overfitting tendencies in all of our transfer learning experiments. Therefore, fine-tuning the pretrained network was not investigated, because this can be expected to further worsen the overfitting tendencies.

Transfer learning

In Section 12.5, we then compared the results of the transfer learning approach to a multitask learning setting, where the target coordinates were predicted as part of the network's bottleneck layer. Our results showed, that a joint training regime in the form of multitask learning is more promising than a transfer learning approach. Again, the classification-based approach was more successful than the reconstruction-based approach.

Multitask learning

Finally, we tested the generalization to target spaces of different dimensionality in Section 12.6. Similar to what we had observed in Section 9.2.5 for the *NOUN* dataset [199], there seems to be a trade-off between a compact and an expressive representation in the target space: Lower-dimensional target spaces were preferable in most cases (presumably, because they pose an easier regression task), but a one-dimensional space seemed to be too unstructured for good generalization.

Generalization experiments

Our experiments in this chapter have shown, that a more complex setup can help us to improve performance over the more naive approach from Chapter 9. Nevertheless, the performance level achieved so far is still below the regression quality of $R^2 \approx 0.78$ reported by Sanders and Nosofsky [346, 347]. However, they used more complex network architectures and a considerably larger number of examples for the mapping task (360 instead of 60 stimuli). Also the results reported by Morgenstern et al. [297] (cf. Section 10.1.3) seem to surpass our own performance considerably. While they used considerably simpler CNN architectures, one may argue, that their stimuli were less complex (since they used silhouettes instead of line drawings), and that they had access to a much larger number of training examples (25,000 instead of 60).

Limitations of mapping performance

Finally, our best mapping results have been observed for photograph-based transfer learning, if 10% salt and pepper noise was used during both training and testing. These performance levels of $R^2 \approx 0.63$ have never been achieved in any of our sketch-based experiments. All of these observations urge for further experimentation with more complex architectures, larger datasets, different augmentation techniques, and additional regularization approaches.

Possibilities for improvement

There are several rather obvious ways, in which one could attempt to improve our system's performance. Firstly, using a larger stimulus set for obtaining the similarity spaces, and hence a larger number of training examples for the mapping task can be expected to improve performance. Secondly, more excessive hyperparameter tuning (e.g., by using a full grid search rather than our two-stage approach from Section 12.3, or optimizing hyperparameters for each target space individually) may lead to more promising configurations. Thirdly, as already remarked in Section 12.4.2, the decision to use only linear units in our bottleneck layer may have drastically reduced our network's capacity – using ReLUs for all bottleneck units, which do not predict coordinates, seems thus a straightforward next step. Fourthly, increasing the number of training examples for the classification and the reconstruction task by considering additional sources such as QuickDraw [186] (which has unfortunately a low drawing quality, cf. Section 12.2.1) or ImageNet-Sketch [424] (which was published after the majority of our experiments had been conducted, and whose drawings vary considerably in style) may be beneficial as well.

Conceptual vs. visual similarity

As already discussed in Chapter 11, it would be interesting to compare similarity spaces based on visual and conceptual dissimilarity ratings, respectively, which have been collected on the same set of stimuli. Intuitively, similarity spaces based only on visual information should be easier to learn by CNNs than similarity spaces which also take into account nonvisual, conceptual information (such as intended usage), since the latter cannot be directly extracted from images. Such comparative experiments could support or challenge this intuition and provide arguments for or against studying individual domains (as in this chapter) rather than holistic similarity spaces (as for instance done by Sanders and Nosofsky [346, 347]).

Evaluation with additional line drawings

In Section 12.2.1, we included 70 additional line drawings into our dataset. Since these line drawings are quite similar in style to the stimuli used to elicit dissimilarity ratings in Section 10.2, one could use them as additional means of evaluation. On the one hand, one could, for instance, predict the coordinates of these line drawings in a two-dimensional SHAPE space and visually inspect the resulting arrangement. On the other hand, since some of the additional line drawings belong to the same semantic categories as the original stimuli (e.g., BIRDS or CLOTHES, both of which are based on visual similarity), one could evaluate, whether their predicted coordinates lead to a correct classification base on simple prototype-based or convex-hull-based classification schemes. Such additional evaluation approaches would

further test the generalization capabilities of the different machine learning setups, which can yield informative insights.

With respect to our reconstruction-based experiments, we reasoned, that the poor mapping performance may be caused by an entangled representation, where information about LOCATION and SIZE on the one hand, and object SHAPE on the other hand are tightly intertwined (cf. Section 12.4.3). In Chapter 6, we have introduced approaches such as InfoGAN [101] (cf. Section 6.3.3) and β -VAE [196] (cf. Section 6.3.4), which claim to learn disentangled representations from visual input. Using such approaches instead of a regular autoencoder may be able to improve the reconstruction-based mapping performance. One could also investigate whether, such an approach would be able to discover the psychological shape features FORM, LINES, and ORIENTATION available in our dataset in an unsupervised way.

Towards disentangled representations

Our comparison of photograph-based and sketch-based approaches has been quite brief due to limited computational resources. More thorough investigations of photograph-based CNNs could include additional network architectures such as AlexNet [232], which is more comparable to our encoder, which was based on Sketch-a-Net [445, 446], than the more complex inception-v3 architecture [399], which we used as a representative of photograph-based CNNs in our experiments. Training the exact same architecture on different data would certainly be the cleanest way of comparing photograph-based networks to sketch-based CNNs. Additional experiments could then also include photograph-based reconstruction, multitask learning on photographs, and applying different noise regimes during training and testing. The results of these experiments could provide valuable additions to the debate on the shape sensitivity of CNNs (cf. Section 10.1.4).

Investigating photograph-based networks

The overall work presented in this chapter can be criticized for being only incremental with respect to prior work in this area (i.e., the study by Sanders and Nosofsky [346, 347], the system proposed by Morgenstern et al. [297], and our own experiments on the NOUN dataset [199] from Chapter 9). While we do indeed not propose a radically new approach, we provide valuable extensions of prior investigations by comparing transfer learning to multitask learning, and classification-based to reconstruction-based approaches. Our results thus provide further insights, which can serve as starting points for future investigations. Moreover, we put our focus on a single cognitive domain, where the target spaces have been thoroughly analyzed with respect to the conceptual spaces framework (cf. Chapter 11). Since a cognitively grounded and at the same time computationally accessible (in the sense of automatically obtaining coordinates for perceptual input) model of the SHAPE domain has so far not been proposed in this context, our work from the last three chapters fills an important gap of the conceptual spaces framework.

General criticism: incrementality ...

One may furthermore criticize, that our results are not surprising: One may already a priori expect, that multitask learning is superior to transfer learning, and that classifier-based approaches outperform

... and unsurprising results

reconstruction-based ones. While the former intuition is relatively clear, it is questionable, whether the additional effort of training a multitask network from scratch really pays off in comparison to simply reusing a pretrained network. Since our results showed considerable performance improvements for multitask learning, we provide evidence for the usefulness of such a computationally more demanding approach. When considering classification-based and autoencoder-based approaches, one may also argue, that autoencoders are closer to the perceptual level than classifiers, and may thus be more suitable for similarity spaces of the *SHAPE* domain. While our results in general favor classification over reconstruction, the generalization experiments from Section 12.6.2 justify further investigations of reconstruction-based approaches.

Outlook

Once the limitations of our experiments have been addressed, and once the quality of the learned mapping becomes reasonably good, the similarity spaces from Chapter 11 and be used in combination with the trained CNNs to equip artificial agents with a cognitively grounded and computationally accessible spatial representation of the *SHAPE* domain. Such an automated mapping from raw perceptual input to coordinates in *SHAPE* space can then serve as a basis for commonsense reasoning on top of this spatial representation, e.g., based on conceptual regions (cf. Section 11.3) or interpretable directions (cf. Section 11.4). In combination with other domains such as *COLOR* and *SIZE*, which can be more readily extracted from visual input, or non-visual domains like *WEIGHT* or *TASTE*, these *SHAPE* spaces can then be used together with our proposed formalization of conceptual regions from Part I of this dissertation (along with their supported reasoning processes) and possible learning mechanisms discussed in Chapter 7 to arrive at an overall system for an artificial agent based on conceptual spaces.

Part IV

SUMMARY

In the final part of this dissertation, we summarize the main research results obtained as part of our research. Moreover, we give a general outlook on potential future developments.

13 | CONCLUSIONS

13.1	Formalizing the Conceptual Layer	726
13.1.1	Lessons Learned	726
13.1.2	Open Ends	728
13.1.3	Outlook	730
13.2	Obtaining the Dimensions	734
13.2.1	Lessons Learned	734
13.2.2	Open Ends	737
13.2.3	Outlook	738
13.3	Learning Conceptual Regions	745
13.3.1	Lessons Learned	745
13.3.2	Outlook	746
13.4	The Big Picture	748

In Chapter 1, we have motivated the overall research presented in this dissertation by the need for cognitive AI [259, 274], i.e., artificial intelligence, which takes direct inspiration from human cognition in order to overcome the limitations of purely functional systems. We have moreover argued, that concepts as mental representations of real-world categories play an important role in human cognition, and that it is therefore important to equip artificial agents with a psychologically plausible representation of conceptual knowledge.

Our overall motivation

The cognitive framework of conceptual spaces [179, 181] proposes to represent concepts geometrically, which can be interpreted as an intermediate layer of representation between the traditional distinction into subsymbolic and symbolic approaches (cf. Section 1.2). Conceptual spaces can thus provide a bridge between these two types of systems, and therefore help to solve the symbol grounding problem [190].

Conceptual spaces

While the overall goal of building an artificial system, which bridges all three representational levels, is too ambitious to be reached within a single dissertation, we have identified three key problems in Section 1.3, which need to be solved as a prerequisite for progress with respect to the overall goal:

Three core problems

Firstly, practical applications of the conceptual spaces framework require an actual implementation in software. This implementation has to be based on a thorough mathematical formalization, which closes many of the open degrees of freedom inherent in the general proposal. We have provided such a formalization and implementation of the framework in Part I of this dissertation (Chapters 2, 3, and 4). This formalization also includes many operations, which can support different learning and reasoning processes (cf. Sections 3.6 and 4.6).

Formalizing the conceptual layer

Obtaining the dimensions

Secondly, one needs to define the structure of the conceptual space, especially its underlying dimensions. We have reviewed two approaches in detail: Representation learning with artificial neural networks (Chapter 6) and multidimensional scaling based on dissimilarity ratings from psychological experiments (Chapter 8). Since both approaches have their individual advantages and shortcomings, we have also proposed a hybrid approach in Section 8.7, which has then been tested in Part III of this dissertation (Chapters 9 to 12).

Learning conceptual regions

Thirdly, one needs to find a mechanism for identifying meaningful conceptual regions in a given conceptual space based on a set of examples. We have reviewed general machine learning algorithms in Chapter 5, and have then discussed more cognitively plausible approaches in Chapter 7, where we focused on the incorporation of prior knowledge, the ability to leverage unlabeled data, and the grounding of concepts in communication with other agents. We have proposed to consider logic tensor networks [21, 358] as one particularly interesting avenue for future research.

Structure

The structure of this conclusion chapter is organized around these three main issues: We first summarize our formalization of the conceptual layer in Section 13.1. Afterwards, we discuss in Section 13.2, what has been achieved with respect to learning the dimensions of a conceptual space, before summarizing our thoughts about cognitively plausible ways of identifying conceptual regions in Section 13.3. For all of these three topics, we provide a summary of our main results and contributions, as well as an outlook on possible future work in this area. Finally, Section 13.4 takes a step back and provides a general outlook with respect to the overall big picture.

13.1 FORMALIZING THE CONCEPTUAL LAYER

Overview

As a first step towards using conceptual spaces in artificial agents, we have developed a thorough mathematical formalization of the framework in Part I of this dissertation. In the following, we will first summarize the main contributions made in Chapters 2 to 4 in Section 13.1.1, before highlighting open ends and possibilities for further development in Section 13.1.2. We then give a more general outlook with respect to formalized conceptual spaces in Section 13.1.3.

13.1.1 Lessons Learned

Formalizing conceptual regions

In Chapter 2, we proposed a new formalization of the conceptual spaces framework. Our considerations started with the desire to represent correlations between domains in a geometric way, i.e., through the shape of the conceptual regions. Most other formalizations either do not consider correlations at all [3] or model them in a non-geometric way [329]. We observed in Section 2.2.1, that convexity under the Man-

hattan metric (which is used to combine multiple domains) leads to axis-aligned cuboids, which are not able to represent correlations. We therefore considered the more general notion of star-shapedness instead of Gärdenfors' favored constraint of convexity (cf. Section 2.2.2). Since a union of convex sets with a non-empty intersection is guaranteed to be star-shaped, we defined concepts based on multiple cuboids, whose intersection is interpreted as prototypical region (cf. Section 2.3.2). In order to allow for imprecise concept boundaries, we introduced a fuzzy membership function in Section 2.3.3, which assigns degrees of membership to all points in the conceptual space, and which is based on an exponential decay of the distance to the concept's crisp core.

In Chapter 3, we have then extended our formalization of the conceptual spaces framework with a comprehensive set of operations for creating new concepts based on existing ones: The *intersection* of concepts (cf. Section 3.1) can be used to define the concept GREEN BANANA based on the individual concepts GREEN and BANANA. Higher-level concepts such as CITRUS FRUIT can be obtained by taking the *union* of several sub-concepts such as LIME, ORANGE, and GRAPEFRUIT (cf. Section 3.2). As an intermediate step in reasoning, a *projection* of a concept onto a subset of domains (cf. Section 3.4) can be used to obtain properties such as SKIN-COLORED based on a full-fledged concept such as SKIN. Finally, we introduced the *axis-parallel cut* (cf. Section 3.5) as a way of splitting a concept into two sub-concepts based on a threshold on one of the dimensions. Moreover, we have sketched in Section 3.6, how this set of operations is useful for carrying out both learning tasks (e.g., concept formation) and reasoning tasks (such as concept combination).

*Operations for
combining concepts*

In Chapter 4, we have further extended our formalization of the conceptual spaces framework by providing mathematical ways for measuring relations between different concepts. The *size* of a concept can be used to describe its specificity (cf. Section 4.1): Smaller conceptual regions correspond to very specific concepts (such as GRANNY SMITH), while larger conceptual regions are typically used to describe more general concepts (such as APPLE or FRUIT). In order to extract a conceptual hierarchy (such as "GRANNY SMITH is a sub-concept of APPLE"), a fuzzy notion of *subsethood* has been introduced in Section 4.2. As we have argued in Section 4.3, this subsethood measure can also be used to quantify the degree of truth for a given *implication*: "APPLE \Rightarrow RED" is true to the degree, to which the projection of the APPLE concept onto the COLOR domain is a subset of the region representing RED. We also provided generalizations of the geometric notions of *betweenness* and *similarity* from points to fuzzy conceptual regions (cf. Sections 4.5 and 4.4, respectively). Again, we have sketched how the relations we defined can be used in the context of concept formation and commonsense reasoning (cf. Section 4.6).

*Measuring relations
between concepts*

The large set of operations, both for creating new concepts and for measuring relations between concepts, makes our formalization (to

*A comprehensive and
fully implemented
formalization*

the best of our knowledge) the most thorough and comprehensive formalization of conceptual spaces developed so far (cf. Sections 3.7 and 4.7). Also with respect to a list of representational desiderata (including for instance imprecise concept boundaries and a uniform treatment of properties and concepts), our formalization is superior to prior approaches (cf. Section 2.4.2 and 2.5). Moreover, we provided a comprehensive implementation of this formalization (cf. Section 2.3.4). Our implementation [51] and its source code are publicly available¹, and can be used by any scholar as a starting point for practical research on conceptual spaces.

13.1.2 Open Ends

Part-whole relations

Although we argued, that our formalization is superior to any of its predecessors, there are certain ways, in which the definition of conceptual regions from Chapter 2 can be further improved: We have sketched in Section 2.4.3, how part-whole structures could be included in our formalization, but we have not given a mathematical treatment. Future work could consist in formalizing Fiorini's proposal [158] in the context of our work.

Determining salience weights

Moreover, in our current account, the computation of the salience weights is under-determined. Future research could compare different ways of deriving those salience weights with respect to their respective advantages and shortcomings. For instance, the work by Sileno et al. [370] on contrast in conceptual spaces seems to be a promising avenue of research (cf. Section 2.3.3).

Convexity for properties

Also the question, whether the convexity constraint should be enforced for properties (by representing them with a single cuboid) has been left open for future studies (cf. Section 2.2.2).

Membership vs. typicality

Furthermore, our formalization currently does not differentiate between concept membership and typicality. However, as for instance Hampton [188] has argued, a given observation can be a full member of a concept without being very typical: A PENGUIN is definitely a BIRD, but not a very typical one. Another possible extension of our formalization would thus be the definition of a typicality function, which could for instance be based on the distance to the concept's central area P .

Repair mechanism

There are also several open ends with respect to the operations defined in Chapter 3: With respect to the intersection and union operations, we have argued to use a simple midpoint heuristic for restoring star-shapedness. The exact implications of using this heuristic instead of a numerical optimization for finding the optimal solution are unclear and should be investigated. Moreover, one can analyze, whether there are any benefits from returning multiple results for a single intersection instead of using a repair mechanism for ensuring star-shapedness.

Runtime of intersection

As we noted in our runtime analysis in Section 3.1.3, the intersection operation tends to become quite slow for both a large number of cuboids

¹ See <https://github.com/lbechberger/ConceptualSpaces/>.

per concept and a large number of dimensions in the similarity space. One should therefore investigate, whether our current implementation of the intersection can be further optimized, or whether fast and accurate approximations can be obtained.

Furthermore, we observed in Section 3.2.2, that the union of LEMON and ORANGE in our example fruit space yields unintuitive results. We explained this by the very simplified structure of our example space. This of course calls for more complex conceptual spaces to be used as further test cases for all the operations from our formalization.

*More complex
example spaces*

Moreover, our approach is so far lacking a meaningful negation operation (cf. Section 3.3). Given that this is an inherent weakness of the conceptual spaces framework itself, it might also be a worthwhile avenue of future research, which would not only benefit our formalization, but the conceptual spaces community as a whole.

Conceptual negation

Also our treatment of relations between concepts from Chapter 4 is still somewhat incomplete: For instance, the computation of a concept's size is quite complex (cf. Section 4.1.3). Even though computing the closed formula is significantly faster than numerically approximating the integral over the concept's membership function, we observed an exponential growth in runtime. For high-dimensional spaces, one may thus need to use a fast approximation of the concept size.

Runtime of size

Moreover, due to technical reasons laid out in Section 4.2.1, our proposed degree of subthood is not always confined to the interval $[0,1]$. Future research should investigate, whether there is a straightforward and computationally feasible way of overcoming this limitation. Moreover, we have argued in Section 4.3.1, that the implication relation between two concepts can be equated with their subthood relation. This argument could use some further empirical support by using the subthood relation as implication in a reasoning application.

*Subthood and
implication*

We have furthermore provided two different definitions for conceptual similarity in Section 4.4.2 – it remains an open question, which one of them is preferable in which contexts. Again, an empirical study in an application scenario can help to clarify this issue. Moreover, additional candidate definitions for conceptual similarity (e.g., based on the maximal Hausdorff distance over all α -cuts, inspired by [11]) could be investigated.

*Conceptual
similarity*

Finally, Derrac and Schockaert [123] have argued, that the degree of parallelism for pairs of difference vectors can be a useful measure for supporting analogy-based reasoning: If DOG relates to PUPPY like CAT relates to KITTEN, then the geometrical direction from the DOG concept to the PUPPY concept should be parallel to the corresponding direction from CAT to KITTEN. Obviously, a formalization of this intuition in our framework would be an interesting additional measure, which could support yet another approach to commonsense reasoning.

*Analogy-based
reasoning*

13.1.3 Outlook

Learning star-shaped concepts is difficult

We have argued in Part I of this dissertation, that our proposed formalization and its implementation are superior to many other proposals in the literature: We are able to geometrically represent correlations, and we offer a plethora of different operations. However, this increased expressive power comes at a cost, namely, at increased representational complexity. Other approaches only need to keep track of a single prototypical point for representing a given concept, whereas we need to store multiple cuboids, a set of domain weights, and a sensitivity parameter. This increases not only the computational and cognitive burden, but also makes learning more difficult:² It is much easier to learn a single prototype from a given set of examples than to find the optimal configuration for a star-shaped set, simply because the number of free parameters is lower. In machine learning terms, one could say, that our formalization has a higher capacity than a simple prototype-based representation (cf. Section 5.1.1). In order to make learning tractable, one may thus need to introduce appropriate regularization schemes (e.g., by penalizing the number of cuboids used for representing a concept, or by inciting more uniformly distributed salience weights).

The need for machine learning experiments

We therefore need practical machine learning experiments with our proposed formalization in order to understand, how problematic its increased capacity is in practice for learning conceptual regions. Additional experiments considering reasoning and concept combinations can inform us about the actual advantages gained from our more complex way of representing concepts. Our hope is, of course, that the additional predictions provided by our formalization outweigh the increased difficulty in learning concepts, but this can only be assessed based on practical experiments.

Natural multi-domain concepts

Strößner [395] has recently conducted a thorough philosophical analysis of multi-domain concepts, correlations, naturalness, and convexity in the context of conceptual spaces. She has noted, that one should not talk about "the" convexity criterion, because one can distinguish at least five different formulations from the literature, ranging from "convexity is required only for properties, but not for concepts" to "conceptual regions in the overall conceptual space must be convex". She furthermore points out, that convexity is currently only empirically supported within individual domains, but not for multi-domain concepts. These multi-domain concepts are, however, often easier to learn (cf. the complex first paradox [430] briefly discussed in Section 7.1.2) and more informative than single-domain concepts (i.e., properties). In order to specify the naturalness of multi-domain concepts, Strößner proposes the following altered version of criterion C:

Definition 13.1 (Natural Multi-Domain Concept)

A natural multi-domain concept is represented as a set of non-locational and characteristic regions in several independent conceptual spaces.

² Thanks to Leonard Frommelt for pointing me towards this issue.

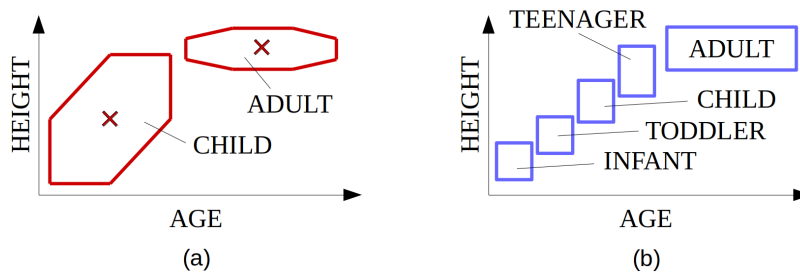


Figure 13.1: Reconsideration of our CHILD example from Section 2.2.1: (a) Our solution with a single star-shaped region. (b) Strößner's [395] solution with multiple convex sub-regions.

In Definition 13.1, non-locationality is used to exclude instable features (such as DAY OF BIRTH for the concept NEWBORN): Locational features describe the position of an entity in space-time, while non-locational features describe the entity itself and are thus less volatile. Characteristicness, on the other hand, is intended to capture, how well a given feature can predict concept membership – the given region should have a strong association to the given concept, but not to other concepts. For instance, RED in the COLOR domain is not very characteristic for the STRAWBERRY concept, because strawberries can also be green (when they are still unripe), and (more importantly) because also apples and tomatoes can be red. A much more characteristic feature for the STRAWBERRY concept is its region in the SHAPE domain.

Strößner furthermore claims, that multi-domain concepts capture correlations, i.e., probabilistic dependencies of individual features. This happens on two levels: From the outside, when determining concept membership ("dog-shaped animals tend to bark", cf. Section 1.1.2), but also from the inside of the concept, when discriminating among members (e.g., "red strawberries tend to be sweeter than green ones"). Exactly this latter type of correlation is explicitly targeted by the star-shaped regions of our proposed formalization.

Strößner argues, that the concept-internal correlations are used to prevent uninhabited regions in the similarity space (i.e., regions without any observations) from belonging to any concept. Referring to our example with the CHILD concept from Section 2.2.1, she notes, that our proposed solution (shown in Figure 13.1a) succeeds in excluding uninhabited regions (such as a newborn with a height of 1.50 meters) from the CHILD concept, but she also criticizes our approach in two ways: Firstly, she argues, that our proposed star-shaped region is a misrepresentation of the CHILD concept, since an extremely tall 13 year old is still a CHILD, albeit an atypical one. She also points out, that other important features for defining the CHILD concept (relating for example to cognitive development) are completely missing. As already noted in Section 2.2.1, our example is a quite simplified one. We did not attempt to provide a good definition of the CHILD concept, but merely an intuitively graspable example for illustrating the need to encode correlations in a geometric way.

*Non-locationality
and
characteristicness*

*Two types of
correlations*

*Criticism of our
approach: the CHILD
concept ...*

... and
star-shapedness

Her second and more important criticism targets our usage of star-shapedness instead of convexity for solving the representational problem. Strößner argues, that the CHILD concept can simply be split up into multiple sub-concepts such as INFANT, TODDLER, CHILD (in the narrow sense), and TEENAGER, all of which can be represented by a convex region. This proposal is visualized in Figure 13.1b. By representing the overall CHILD concept as a collection of multiple sub-concepts, one can thus exclude uninhabited regions, encode correlations, and keep the convexity requirement intact. In our point of view, there are, however, two issues with this proposal: Firstly, we now need to deal with two types of representations for concepts, namely, convex conceptual regions like TODDLER on the one hand, and sets of sub-concepts like CHILD on the other hand. This requires two separate learning and reasoning strategies for the different representation types, which is somewhat inelegant.³ Secondly, while such an approach with sub-concepts is easily applicable to CHILD, one may struggle to define similar sub-concepts for other concepts with internal correlations such as BANANA (with a correlation between COLOR and TASTE) or BIRD (with a correlation between SIZE and SINGING). We are thus convinced, that our star-shaped regions are nevertheless a valuable approach, which should be contrasted with other proposals based on empirical data.

Usage of our
formalization in the
smart home domain

A first practical usage of our formalization and implementation has been reported by Pol and Diaconescu [321, 322]. They considered automatic systems in the context of smart homes, and proposed a cognitive control system, which dynamically creates novel concepts at runtime and performs reasoning on them. The overall system consists of two parts: The *reactive subsystem* uses conceptual spaces for representing the current state of system variables (such as the current POWER CONSUMPTION, TEMPERATURE, OR LUMINOSITY). The *deliberative subsystem* on the other hand uses simulations, novelty detection, and argumentation procedures for generating new concepts and adaptation plans for fulfilling the system's goals. Pol and Diaconescu also provide a first simulation of their envisioned system based on a smart home dataset, and use our implementation of the conceptual spaces framework in order to implement their reactive subsystem. This illustrates, that our formalization can be used in practice.

Limitations

However, they do not report on a real-world application so far. Moreover, their work so far only uses a small part of our formalization, namely, the creation of concepts (which use a single point as core) and the computation of concept membership⁴. Pol and Diaconescu interpret basic concepts as being defined on a single dimension, and more abstract concepts as involving multiple dimensions of the same domain [322]. This differs from our own approach, where we can only distinguish properties (which are defined on a single, potentially multi-dimensional domain) from concepts (which involve multiple

³ This relates to our criticism of Rickard's formalization [329], where properties and concepts are represented in fundamentally different ways (cf. Sections 2.4.2 and 2.5.2).

⁴ Personal communication with Marius Pol, 11.09.2020.

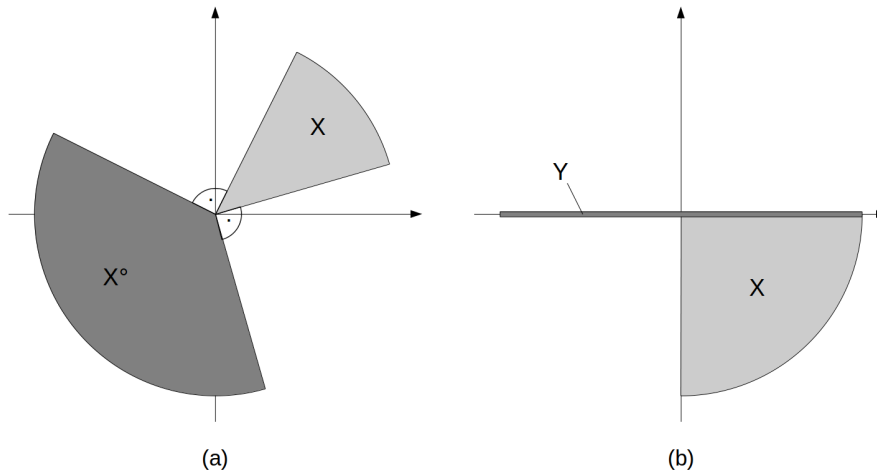


Figure 13.2: Cone-based representation as proposed by Özçep et al. [312]. (a) Convex cone X and its polar cone X° . (b) Axis-aligned cones $X = \mathbb{R}^+ \times \mathbb{R}^-$ and $Y = \mathbb{R} \times \{0\}$.

domains at once). Taken together, their work therefore provides only very limited empirical evidence for the usefulness of our formalization, and needs to be followed up by additional applications, which harness the expressive power of our formalization.

Finally, we would like to highlight, that our formalization of conceptual spaces should not only be compared to other formalizations of this framework, but to related ideas from other subfields. For instance, Özçep et al. [312] have recently proposed an embedding of ontologies into real-valued vector spaces based on axis-aligned convex cones. A region X is called a *convex cone*, if $\forall \lambda, \mu \geq 0 : v, w \in X \Rightarrow \lambda v + \mu w \in X$ (see Figure 13.2a). The *polar cone* X° of X can then be defined as $X^\circ = \{v \in \mathbb{R}^n \mid \forall w \in X : \langle v, w \rangle \leq 0\}$ (where $\langle \cdot, \cdot \rangle$ denotes the scalar product), and is a convex cone representing the negation of X (see Figure 13.2a). Özçep et al. now consider the special case of *axis-aligned cones*, which can be defined as $X = X_1 \times \dots \times X_n$, where $X_i \in \{\mathbb{R}, \mathbb{R}^+, \mathbb{R}^-, \{0\}\}$ (see Figure 13.2b). They note, that axis-aligned cones are convex, and that they are closed under intersection and negation. In order to represent a convex union of two axis-aligned cones X and Y , Özçep et al. make use of de Morgan's law⁵: $X \cup Y = (X^\circ \cap Y^\circ)^\circ$. Thus, they are able to represent conceptual intersection, union, and negation. Moreover, Özçep et al. derive a way of representing universal and existential quantifiers for encoding relations between concepts.

The proposal by Özçep et al. is simple and mathematically elegant. Its usage of convex cones can be related to Gärdenfors' convexity requirement for conceptual regions. Moreover, requiring the cones to be axis-aligned is similar to our own usage of axis-aligned cuboids. The main difference to the conceptual spaces framework concerns

Concepts as convex cones

Relation to conceptual spaces

⁵ De Morgan's law states, that the negation of a disjunction is equivalent to the conjunction of the negations, i.e., $\overline{A \cup B} = \overline{A} \cap \overline{B}$.

the underlying notion of similarity: While conceptual spaces consider the Euclidean and Manhattan distances as reflecting dissimilarity, the proposal by Özçep et al. makes use of the scalar product, and thus implicitly of the Cosine similarity. Hence, it can be related to word embeddings as introduced in Section 6.3.2. Moreover, Özçep et al. do not consider a domain structure as proposed by the conceptual spaces framework. Correlations between different dimensions can be implicitly encoded, but only to a certain degree: For instance, the axis-aligned cone X in Figure 13.2b can be interpreted as encoding a negative correlation between the two features corresponding to the x-axis and the y-axis, respectively. However, such a correlation could also be encoded through $X^\circ = \mathbb{R}^- \times \mathbb{R}^+$, i.e. the concept's negation. Overall, the proposal by Özçep et al. is certainly interesting and relevant, but differs from conceptual spaces in general and our own formalization in particular in some important aspects such as the underlying distance metric used in the representation space. Nevertheless, such comparisons may lead to further insights and are thus a worthwhile endeavor.

13.2 OBTAINING THE DIMENSIONS

Overview

As already discussed in Chapter 1, another main challenge for applying the conceptual spaces framework in practice is the correct identification of the dimensions, which span its domains. We have examined this problem from the two perspectives of machine learning (Chapters 5 and 6) and psychology (Chapter 8). Our main contribution consists of the proposal of a hybrid approach (cf. Section 8.7, which has been empirically tested in Part III of this dissertation. In the following, we first summarize the main insights from theory and practice in Section 13.2.1, before looking at open ends in Section 13.2.2, and general ongoing developments in this research area in Section 13.2.3.

13.2.1 Lessons Learned

Dimensionality reduction

In Section 5.3, we discussed different approaches for dimensionality reduction, which can be related to the translation of high-dimensional subsymbolic information to the lower-dimensional conceptual layer: In *feature selection* (cf. Section 5.3.1), one aims to select a subset of the original features, which is in some sense optimal for the downstream machine learning problem. A concrete example in the context of conceptual spaces is the approach taken by Banaee et al. [27], which also partitions the set of selected features into different conceptual domains. In *feature extraction*, on the other hand, one tries to create a new set of features by combining different original features. For feature extraction, we gave two examples: The popular principal components analysis (PCA, cf. Section 5.3.2) defines new features as linear combinations of the original features, such that variance in the data is captured as well as possible. Moreover, the area of metric learning (cf. Section 5.3.3) tries

to automatically obtain a distance metric between examples (which implicitly defines a similarity space), such that certain desiderata such as low intra-class distances are fulfilled.

In Chapter 6, we have then introduced artificial neural networks as a powerful machine learning algorithm, which is capable of learning abstract representations of its perceptual inputs: Each layer of the network computes a nonlinear transformation of its inputs, resulting in a hierarchy of more and more abstract features. We have covered different representation learning approaches such as word embeddings (cf. Section 6.3.2)⁶, InfoGAN [101] (cf. Section 6.3.3), and β -VAE [196] (cf. Section 6.3.4). Moreover, we introduced various evaluation metrics for quantifying the degree of disentanglement achieved in the resulting representations (cf. Section 6.1.2). An advantage of ANN-based representation learning is that a trained network is often able to successfully generalize to unseen inputs. However, training deep networks is a computationally very expensive process and often requires the complex tuning of several hyperparameters. Moreover, representations learned by neural networks can in general not claim any psychological validity.

*Representation
Learning*

In Chapter 8, we have then introduced the technique of multidimensional scaling (MDS), which can be used for constructing similarity spaces from dissimilarity ratings collected in psychological studies. We have covered different ways of collecting such dissimilarity ratings, and various algorithms for solving the MDS optimization problem. We have also discussed different evaluation approaches based on the match between distances and dissimilarities, and based on the interpretability of the resulting configurations. The main advantage of constructing conceptual spaces through MDS lies in its psychological grounding: Since the resulting similarity spaces are directly based on ratings elicited in psychological studies, they can be deemed to be cognitively plausible. However, MDS is not able to generalize to unseen examples – we can in general not predict the coordinates for novel stimuli, which were not part of the psychological study.

*Multidimensional
scaling*

In order to combine the strengths of both the ANN-based approach (namely, being able to generalize to unseen inputs) and the MDS-based approach (namely, being grounded in psychological data), we have thus made a hybrid proposal in Section 8.7: The similarity space is initialized by applying MDS to dissimilarity ratings from a psychological experiment, and ANNs are then subsequently used to learn a mapping from raw stimuli to their corresponding points in the similarity space. This hybrid approach can be considered our main theoretical contribution with respect to the grounding of dimensions, and has in a similar form been independently developed by Sanders and Nosofsky [347].

*Our proposed hybrid
approach*

We have tested our hybrid approach in Part III of this dissertation on two different datasets. In Chapter 9, we presented a first proof of concept

The NOUN study

⁶ In Section 6.3.2, as well as at various other points of this dissertation, we have discussed the work by Steven Schockaert. A recent article [75] gives a good overview of his research program, which aims to learn conceptual spaces from data.

using the NOUN database [199], which contains images and similarity ratings for novel objects. Here, we first compared the spaces obtained from different MDS algorithms, and found only minor differences. We then considered a transfer learning task, where we trained a linear regression on top of a pretrained neural network in order to map raw images into the obtained similarity spaces. Our results showed, that this approach seems to work in principle, although the overall performance level was still very low with strong overfitting tendencies. Moreover, we found, that the dimensionality of the target similarity space had a strong influence on regression performance, with a two-dimensional similarity space yielding the best overall results.

*Considering the
SHAPE domain*

In Chapter 10, we have then focused on the SHAPE domain by first summarizing related work from both psychology and computer vision, and by then analyzing a dataset of 60 line drawings belonging to 12 categories. In Chapter 11, an analysis of the similarity spaces obtained through MDS has shown, that our dataset can be represented with three to five dimensions. Moreover, we found, that the three psychological features FORM (elongated vs. blob-like), LINES (straight vs. curved), and ORIENTATION (horizontal vs. diagonal vs. vertical) can be successfully mapped onto linearly independent directions in these spaces. Furthermore, especially visually coherent categories formed small, non-overlapping convex regions. Since the original dissimilarity ratings are ordinally scaled, only the aggregation through the median is permitted, but many researchers nevertheless use the arithmetic mean. We have compared the impact of the aggregation function on the overall results, and were only able to observe relatively small differences, with the mean aggregator yielding slightly preferable results. Overall, the similarity spaces extracted from the visual dissimilarity ratings seem to be good candidates for representing the domain of shapes.

*Learning a mapping
into SHAPE space*

In Chapter 12, we have then attempted to learn a mapping from the original line drawings into the extracted SHAPE spaces, using convolutional neural networks. We compared photograph-based networks to sketch-based networks, classification networks to autoencoders, and transfer learning to multitask learning. Using a sketch-based classification network instead of a photograph-based classification network led to improved results in the transfer learning setting, which highlights the importance of the source domain: Since sketches are more similar to line drawings (they consist of a small amount of black strokes on white ground), the features extracted by a sketch classification network seem to be a better starting point for learning a successful mapping. Using a joint training phase for the sketch classification task and the mapping task further helped to improve performance, indicating that multitask learning can be more powerful than transfer learning. Also for the autoencoder, we found multitask learning to yield superior results than transfer learning. However, the feature spaces learned by an autoencoder seem to be less useful for the mapping task than classification-based features. We argued, that this may be caused by an entangled representation, which also needs to encode LOCATION and SIZE of the object in order to obtain a correct reconstruction. Overall,

our experiments in Chapter 12 showed, that additional effort (in the form of training a network from scratch in a multitask learning setting) is worthwhile and can considerably improve the results over the more naive approach taken in Chapter 9. Nevertheless, our results are still far from perfect and require further improvements.

13.2.2 Open Ends

In Section 6.4.1, we have argued, that representation learning may be a good fit for discovering the dimensions of a conceptual space in a bottom-up fashion. Especially approaches such as InfoGAN [101] (cf. Section 6.3.3) and β -VAE [196] (cf. Section 6.3.4) seem promising, since they claim to learn disentangled representations. In these disentangled representations, individual features correspond to interpretable factors of the data generating process. However, the success in disentangling these factors of variation has so far not been systematically evaluated due to a wide variety of available datasets and evaluation metrics. In Section 6.4.2, we have therefore proposed to use the domain of rectangles as a first test case for such an approach: Already in this very simple domain, stimuli can be described by two out of four candidate dimensions, namely, WIDTH, HEIGHT, AREA, and SHAPE (i.e., aspect ratio). One can thus use this setting to investigate, how stable the extraction of interpretable dimensions is in practice, and whether the results align with observations from psychological studies [71, Section 17.4].

Representation learning for the rectangle domain

In Chapter 9, we observed no considerable differences between metric and nonmetric MDS when applied to similarity ratings obtained through SpAM [169]. Since this is to our knowledge the first investigation in this direction, further analyses of other datasets should seek to confirm our results.

Metric vs. nonmetric MDS on SpAM ratings

With respect to the SHAPE spaces analyzed in Chapter 10, we have noted a good agreement with our theoretically derived predictions. However, there are still some open questions: For instance, we have hypothesized, that the differences between the mean and the median spaces are mainly caused by the different number of constraints from the respective matrices. A more fine-grained analysis should be conducted in order to evaluate this hypothesis. Moreover, we have seen, that the features under consideration (FORM, LINES, and ORIENTATION) were not sufficient for explaining the dissimilarity ratings. It may thus be worthwhile to elicit ratings with respect to additional shape features, and to re-run our analyses. Furthermore, one could attempt an analysis on the raw similarity and feature ratings with the aim to link the visually coherent categories to a description in terms of the shape features. Finally, an analysis of other shape-based dissimilarity ratings with our approach could be used to investigate the robustness of our results, while an application to a different cognitive domain can help to assess the generalization capability of our approach.

Analysis of SHAPE spaces

*Machine learning
experiments*

In Chapters 9 and 12, we have used deep convolutional neural networks for mapping images into psychological similarity spaces. In Chapter 9, we have given a first proof of concept, illustrating that transfer learning on top of a pretrained CNN is able to beat some simple baselines. In Chapter 12, we have extended this approach by also considering multitask learning. Although this resulted in notable improvements, the overall robustness and performance of our approach is still far from satisfactory. Larger datasets, more complex network architectures, and more sophisticated augmentation and regularization techniques need to be investigated in order to make the learned mapping function precise enough for practical applications. The work by Sanders and Nosofsky [346, 347] illustrates, that such improvements are in principle possible.

13.2.3 Outlook

Overview

Since the original inception of our hybrid approach for grounding conceptual spaces [40], many other researchers have made similar proposals. In the following, we try to give a short overview of this newly emerging research area, which aims to combine machine learning (mostly in the form of convolutional neural networks) with human data obtained from psychological experiments. We will first present four relevant datasets, before considering different modeling approaches and experiments.

*CIFAR-10H:
incorporating label
uncertainty*

Peterson et al. [320] have noted, that most computer vision datasets contain only a single ground truth label for each image in the collection. Since a single label is not able to reflect human uncertainty about the correct classification, they collected additional data in order to approximate a probability distributions over different labels. They considered the popular CIFAR-10 dataset [231], which contains 60,000 images in a resolution of 32×32 pixels, labeled with 10 distinct classes. Peterson et al. extended this dataset by collecting a total amount of 500,000 human categorizations for the 10,000 images in the CIFAR-10 test set. They refer to the resulting dataset as *CIFAR-10H*.

*Confusion
probabilities as
similarity measure*

Peterson et al. assumed, that human confusions between classes gives an indication of class similarity (cf. perceptual confusion tasks, Section 8.1.3), and can thus improve machine learning results. They therefore trained a convolutional neural network to minimize the cross-entropy of its output distribution to the human distribution of labels. Their results showed, that the trained network was more robust against adversarial attacks, where a small number of pixels is changed in order to provoke misclassifications. Moreover, it showed increased generalization capabilities. We would like to point out, that their approach can be interpreted as a variant of relational label smoothing [276] (cf. Section 7.4.3), which is based on psychological data rather than pre-defined class similarities.

In a related vein, Roads and Love [334] have collected human similarity judgments for the validation set of ImageNet [120]. They argue, that ImageNet is a very popular dataset for computer vision tasks, but that evaluation is often limited to task-specific metrics such as the classification accuracy. Roads and Love propose to use task-general metrics in the form of a correlation of the similarities of the system's internal representations to the human similarity judgments (cf. relational similarity analysis [230], Section 9.1.2). They collected their data through Amazon Mechanical Turk by asking participants to select the two most similar images to a given query image from a set of eight candidates. In order to present the most informative set of candidate images, Roads and Love used an active learning approach. In addition to the similarity ratings, their *ImageNet-HSJ* dataset also contains an embedding of these ratings into a similarity space.

ImageNet-HSJ: similarity ratings for ImageNet

Convolutional neural networks are the predominant approach used on ImageNet, therefore the extended dataset provided by Roads and Love would make a good test case for our hybrid proposal. However, since their similarity judgments refer to overall similarity, they are less useful from a conceptual spaces perspective, where we are mainly interested in individual domains.

Relation to conceptual spaces

Instead of augmenting an existing dataset from the computer vision community with human ratings, Mehrer et al. [289] have created a completely novel dataset, which they call *ecoset*. Their motivation stems from the observation, that deep convolutional neural networks are currently the best model of visual information processing in the primate brain. However, these models are typically trained on datasets such as ImageNet [120], which were created from an engineering perspective, but do not necessarily reflect human experience. For instance, ImageNet contains 120 different dog breeds, but no categories for humans. The human visual system, on the other hand, seems to be especially sensitive to human faces and body parts.

Ecoset: a dataset reflecting human experience

Ecoset consists of 1.5 million images from 565 basic-level categories, which were deemed more relevant to humans, as for instance indicated by their frequency in linguistic usage. Mehrer et al. compared convolutional neural networks trained on either ImageNet or *ecoset*, and observed, that training on *ecoset* led to a better alignment with both fMRI studies and human behavioral data.

Properties of ecoset

Hebart et al. [194] have recently created the *THINGS* dataset, which contains 1,854 images of objects from a wide variety of categories. They collected their data based on the triad method (where participants repeatedly had to select the "odd one out" among a set of three objects, cf. Section 8.1.1). They assumed, that the similarity between a pair of objects i and j can be approximated by the probability of choosing an arbitrary third object k as the "odd one out". Since a full similarity matrix would require an infeasible amount of one billion responses, Hebart et al. only sampled a random subset of unique trials, leading to an incomplete and relatively sparse similarity matrix.

THINGS: similarity ratings across many categories

*An embedding based
on human ratings*

Hebart et al. then used a shallow neural network for extracting a meaningful embedding for all objects. Their model predicted the human responses by comparing the dot product of the object embeddings. It was trained by gradient descent minimizing both cross entropy loss between its predictions and the observed behavior, as well as an L₁ regularization on the weights, inciting a sparse representation (cf. Section 5.2.1). The values on each dimension were restricted to be positive in order to improve their interpretability.

*Performance and
interpretability of the
embedding*

Hebart et al. were able to show, that the resulting model extracted 49 interpretable dimensions, which were able to predict similarity judgments on a held-out test set with an accuracy of 64.6%. Since the agreement among humans was estimated at 67.22%, this is an excellent result. Moreover, the similarity space spanned by these dimensions proved to be useful both for classification with a prototype-based classifier and typicality judgments. Hebart et al. also established, that the individual dimensions could be reliably named by participants, and that participants were also able to rate stimuli along these dimensions in large agreement with the model.

*Relation to
conceptual spaces*

Overall, the work by Hebart et al. combines an MDS-like approach (using continuous dimensions) with traditional feature-like approaches (which assume non-negative properties). Among the dimensions extracted from the data, the authors identified three groups, namely semantic membership in high-level categories (e.g., FOOD OR ANIMAL), conceptual properties (such as VALUABLE, DISGUSTING, OR HEAT-RELATED), and perceptual features (e.g., ROUNDNESS, ELONGATION, COLOR, OR SHININESS). Also in the conceptual spaces framework, both perceptual and conceptual dimensions exist, and they are furthermore grouped into domains based on their shared semantic properties. Semantic membership is, however, modeled through regions in these domains rather than as separate dimensions. While the embeddings obtained by Hebart et al. thus resemble the conceptual spaces approach to some extent, one can still find notable differences. It would be certainly interesting to analyze the THINGS dataset with our proposed hybrid approach, and to compare the results to the ones obtained by Hebart et al. with their embedding approach.

*CNNs and
higher-level
cognition*

Battleday et al. [35] have summarized recent research, which tries to use convolutional neural networks (CNNs) as a basis for modeling higher-level cognition such as similarity judgments and categorization. They argue, that CNNs are able to represent a large set of complex naturalistic stimuli, and that they can therefore be used as a substitute for the internal representations used by humans. Cognitive models on top of CNNs can therefore predict human behavior on naturalistic instead of artificial stimuli, which allows one to study human behavior in ecologically more valid settings. In the following, we will briefly summarize several relevant studies in this area.

*Predicting typicality
ratings with CNNs*

Lake et al. [240] were arguably the first researchers to combine CNNs with cognitive models. In their experiments, they used the activations of a pretrained CNN's output layer in order to predict category typicality

for a small set of photographs. When comparing the CNN's predictions to human ratings with Spearman's ρ , they observed a rank correlation of $\rho \approx 0.67$, indicating, that the network's classification probabilities can indeed reflect typicality to some non-trivial degree. By visually analyzing their set of stimuli (eight categories with 16 images each), they observed, that humans tended to rate typicality based on similarity to an idealized member, while the CNNs tended to judge typicality based on frequency. For instance, humans gave the highest typicality ratings for the BANANA class to an image of a yellow and spot-less banana, while the CNNs also gave high ratings for spotted bananas or a green plantain. Finally, Lake et al. observed, that later layers of the network gave better typicality predictions than earlier layers.

In Section 8.1.5, we have already discussed the studies by Peterson et al. [318, 319], who predicted human similarity ratings by using a weighted inner product of high-level CNN activations: The similarity s_{ij} between two stimuli i and j (with embeddings \vec{z}_i and \vec{z}_j , respectively) is computed as $s_{ij} = \vec{z}_i^T W \vec{z}_j$ with a diagonal weight matrix W . Attarian et al. [18] have generalized this approach by introducing a dimensionality reduction function f , yielding similarity predictions of the form $s_{ij} = f(\vec{z}_i)^T W f(\vec{z}_j)$. In their study, they implement this function f by applying a principal component analysis (PCA, cf. Section 5.3.2) and keeping the top k principal components. Moreover, they considered different constraints applied to the matrix W : In addition to diagonality, they also considered symmetry ($W = V^T V$, which implies, that $s_{ij} = (V f(\vec{z}_i))^T (V f(\vec{z}_j))$, and which thus relates their approach to metric learning [236], cf. Section 5.3.3), and a completely unconstrained matrix. Their results of estimating the entries of W through gradient descent showed, that removing constraints from the matrix improved performance, and that 512 principal components were sufficient for predicting similarity ratings, which corresponds to a considerably smaller space than the original 4096 units.

Instead of using a fixed dimensionality reduction function f and learning a weight matrix W , Jha et al. [209] have proposed to learn a linear function f for dimensionality reduction, and then to use the raw inner product in the resulting representation space: $s_{ij} = f_\theta(\vec{z}_i)^T f_\theta(\vec{z}_j)$. They argue, that a PCA as used by Attarian et al. [18] tries to preserve all information contained in the high-dimensional feature space, while a learned linear transformation is able to focus on the aspects relevant to human similarity judgments. Jha et al. found, that prediction performance as measured with R^2 was higher for a 64-dimensional compressed space than for the original CNN representation. Moreover, they observed a saturation in performance already after ten to twenty dimensions. In order to "orthogonalize" their low-dimensional representation (i.e., to rotate it onto meaningful directions), they applied a principal component analysis and found, that the resulting principal components corresponded to interpretable features.

Marjeh et al. [277] have recently argued, that CNN-based approaches are inherently limited to visual inputs. Their advantage with respect to

Using dimensionality reduction for predicting similarities

Learning the dimensionality reduction function

Predicting similarities from text

using dissimilarity ratings from humans lies mainly in their scalability, since there is usually one image per stimulus, while the number of dissimilarity ratings grows quadratically with the number of stimuli. Marjeh et al. proposed to use modern language models instead of CNNs in order to generalize outside the visual domain: Again, the number of textual descriptions grows linearly with the number of stimuli, but the descriptions are not limited to photographs, but can also be based on other domains such as sounds, or more complex entities or events. Marjeh et al. used the image dataset by Peterson et al. [318, 319] (cf. Section 8.1.5) as a first test bed, considering either the image labels or short descriptions from human subjects as textual representation of the stimuli. Using both word embeddings based on ConceptNet [267] (a knowledge base of concept descriptions) and the internal representation of BERT [125] (modern neural language model), they followed the same procedure as Peterson et al. by computing pairwise similarities as weighted dot product of the given feature vectors. Marjeh et al. report, that their text-based system outperformed the CNN-based approach originally proposed by Peterson et al. with a prediction performance of $R^2 = 0.69$ on the original similarity ratings. Their experiments thus show, that also textual representations can be used to similarities, relating their work to the approach by Derrac and Schockaert [123], who extracted conceptual spaces from textual data (cf. Section 8.1.5).

*A hybrid approach in
the context of music*

The studies summarized so far have mainly focused on predicting human similarity ratings. In contrast to this, and much in line with our own hybrid proposal and the work by Sanders and Nosofsky [346, 347], Esling et al. [147] have tried to learn a mapping from perceptual input into an MDS-based similarity space. However, instead of considering visual input, they focused on the domain of musical instruments and so-called *timbre spaces*, which try to capture the perceived differences in sound between pairs of instruments playing the same note at the same intensity. Esling et al. trained a β -VAE architecture [196] (cf. Section 6.3.4) on audio samples, and constrained the latent space of the network, such that the distances between encoded audio samples matched the distances of the respective instruments in a given timbre space, which had been obtained with multidimensional scaling from human dissimilarity ratings. They thus implicitly implemented our proposed hybrid procedure as described in Section 8.7, using, however, audio input instead of images. Esling et al. reported a good reconstruction quality of the resulting system, which also generalized to audio samples of previously unseen instruments. They moreover emphasize, that the decoder network can be used to synthesize sounds for any point in the timbre space, allowing for instance for meaningful interpolations between instruments. Their work adds further support to our proposed hybrid approach, highlighting, that the general procedure is not limited to the visual domain. Their evaluation focuses, however, mostly on the synthesis of sounds, and does not explicitly quantify the match between the learned representation and the original similarity space. It is thus

difficult to make a straightforward comparison to our own experiments or to the study by Sanders and Nosofsky [346, 347], where mapping performance was the main evaluation criterion.

Other researchers have put their focus on higher-level cognitive processes, which can operate on top of similarity spaces. For instance, Battleday et al. [34] have investigated the performance of several psychological categorization models when applied to the representations of pretrained CNNs. More specifically, they compared prototype models (where each category is represented by its centroid, cf. Section 1.1.1) to exemplar models (where each category is represented by the full set of examples, cf. Section 1.1.1), using the activations of a pretrained CNN as a feature space. Categorization is done based on similarity to prototypes and exemplars, respectively, where similarity is defined as an exponentially decaying function of a weighted Euclidean distance. Battleday et al. compared this to a baseline model, which used the class probabilities of a pretrained CNN as similarity measure. They found, that these cognitive classification models improved predictions for ambiguous images, and that prototype models were able to perform comparably to exemplar models, if dimension weights were used in the distance calculation.

*Prototypes,
exemplars, and
CNNs*

Also Sorscher et al. [383] have recently applied prototype learning on top of CNN representations, reaching very high accuracies of 92% for pretrained CNNs in a classification with 1,000 novel classes and only a single example per class. They argue, that each concept can be interpreted as a manifold in a high-dimensional feature space, and that classification performance for a prototype classifier is based on the signal-noise ratio between classes. This signal-noise ratio can be influenced through different factors, such as the pairwise distance between neighboring manifolds or their respective size.

*Prototypes and
signal-noise ratio*

Singh et al. [376] have also considered prototype and exemplar models in combination with CNNs. Instead of applying them on top of a pretrained network, they, however, propose to train an overall end-to-end system from scratch: In their work, the usual softmax output layer of a regular CNN was replaced by a probabilistic version of prototype or exemplar theory, which made use of Gaussian probability distributions: While prototype models used one Gaussian component per concept, exemplar models used one Gaussian component per example. Singh et al. furthermore considered an intermediate case, where the number of components was greater than one, but smaller than the number of examples. They trained several CNNs on CIFAR-10 [231], and evaluated the model's prediction on CIFAR-10H [320] by measuring the cross entropy between the model's output distribution and the human label distribution. They found, that their cognitive categorization approach performed better than regular CNNs, and that both prototype and exemplar models were outperformed by the intermediate case with a small number of receptive fields. Their system is somewhat reminiscent of ProtoNet [381], which had been introduced in Section 5.3.3 in the context of metric learning: Both systems use CNNs to learn a similarity

*Training an
end-to-end system*

space, where a simple prototype-based model is used for classification. Moreover, in both cases, the system is trained in an end-to-end manner. While ProtoNet focuses on classification performance in a classical machine learning setting, the approach by Singh et al. explicitly targets human classification behavior. Moreover, in addition to a prototype-based approach, it also considers exemplar models and intermediate cases with subconcepts.

A word of caution

All of the approaches discussed above assume either explicitly or implicitly, that neural networks, especially CNNs, are a good model of human (visual) perception. This view has been recently criticized by Bowers et al. [77]. They point out, that deep neural networks account for almost no results from psychological studies, which usually target a manipulation of independent experimental variables and their influence on the observed behavior. Bowers et al. furthermore list various shortcomings of CNNs with respect to cognitive plausibility, including their texture bias (cf. Section 10.1.4), which stands in contrast to the shape bias found in humans. They also criticize approaches based on relational similarity analysis (RSA) [230], where the correlation between similarity matrices of different systems is computed, by noting, that a high correlation does not imply a similar internal mechanism: A digital clock can predict the behavior of an analog clock with high accuracy despite using fundamentally different internal mechanisms. Bowers et al. conclude, that future research should focus on neural networks, which are able to predict key psychological phenomena such as the Gestalt principles of visual perception (i.e., visual input being organized based on proximity, similarity, continuity, connectedness, and closure). Also hybrid systems, which combine current CNNs with symbolic components are deemed a worthwhile area for future research. In the context of our current discussion, such more cognitively plausible networks may also be expected to provide better predictions of human data.

Outlook

Our summary of the literature in this emerging research area has been necessarily brief. It nevertheless shows, that research on the combination of deep neural networks with cognitive models is very active, since most studies have been published within the last two to three years. It furthermore illustrates, that there are different approaches with encouraging results. One can easily relate this strand of research to the conceptual spaces framework, since an intermediate geometric level of representation is used. This representation is learned by neural networks and supports the definition of similarity and categorization based on psychological models. A tighter integration with conceptual spaces (for instance through the incorporation of a domain structure) seems to be a promising direction for future endeavors, especially in the context of making these systems more cognitively plausible.

13.3 LEARNING CONCEPTUAL REGIONS

Once the dimensions of a conceptual space have been determined, one needs to identify conceptual regions within this space, and to connect them to appropriate symbols from the symbolic layer. While we were not able to conduct any practical experiments on this issue as part of this dissertation, we have discussed the concept learning problem on a theoretical basis in Part II, namely in Chapters 5 and (especially) 7. In the following, we briefly summarize the main points of this discussion (Section 13.3.1), before giving an outlook on further research directions (Section 13.3.2).

Overview

13.3.1 Lessons Learned

In Section 5.2, we have presented several standard machine learning algorithms without considering their cognitive plausibility. This included linear models (Sections 5.2.1, 5.2.2, and 5.2.3: linear regression, logistic regression, and support vector machines), rule-based approaches (such as decision trees, Section 5.2.4), and instance-based algorithms (such as k nearest neighbors, Section 5.2.5, which can be interpreted as an instantiation of the exemplar theory of concepts, cf. Section 1.1.1). In Chapter 7, we have then discussed on concept learning in the context of conceptual spaces. We have focused on three distinct aspects for making machine learning more cognitively plausible:

Classical machine learning

Firstly, we considered learning under knowledge-based constraints (cf. Section 7.2). In this context, we have introduced logic tensor networks (LTNs) [359] as a promising tool for exploring the incorporation of top-down information in the learning process. They can be applied to conceptual spaces by using a suitably defined membership function, for example the one proposed in Part I of this dissertation. Logic tensor networks can then be used to close the gap between the conceptual layer and the symbolic layer.

Knowledge-based constraints

Secondly, we discussed incremental concept formation algorithms (cf. Section 7.3), which process observations in an incremental fashion, and which do not need explicit access to ground truth labels. We introduced COBWEB [159], ART [175], and SUSTAIN [270] as three prime examples for this approach.

Concept formation

Thirdly, we discussed language games [389] (cf. Section 7.4) as a semi-supervised learning task, where multiple agents need to align their conceptualization of the world by developing a shared vocabulary. As we noted there, some researchers have already used conceptual spaces as a representation format in this setting [149, 421].

Language games

Although our discussion of concept learning in conceptual spaces has stayed on a theoretical level, and although it has consisted mostly of a literature review, the three strands of research discussed in Chapter 7 nevertheless highlight the integrative power of conceptual spaces. They thus provide useful starting points for experimental studies.

Summary

13.3.2 Outlook

*Other aspects of
concept learning*

In Chapter 7, we have only considered three selected aspects of concept learning. There are of course several other subfields, which are relevant, and which may spark interesting research programs. This includes active learning [361], learning with incomplete information and insufficient resources [425], and multi-modal embodied learning [301]. Also the grounding of abstract concepts (such as WISDOM OR BEAUTY) in more concrete concepts (such as BOOK OR SUNSET) [126] would be interesting in this context.

LTN experiments

We have proposed to use LTNs for learning conceptual regions under the influence of top-down constraints. Since this proposal is only of theoretical nature so far, practical studies are needed for empirically testing its potential. We hypothesize, that the additional top-down constraints are especially useful if only a (very) limited number of training examples is available, i.e., in a few shot learning scenario.

*Concept formation in
conceptual spaces*

Applying concept formation algorithms to conceptual spaces should be relatively straightforward, since all algorithms discussed in Section 7.3 assume, that instances are represented as points in feature spaces. Moreover, the presented algorithms make use of relatively straightforward representation schemes, which often relate to prototypes and thus to Gärdenfors' Voronoi representation. If one were, however, to use the formalization developed in Part I of this dissertation (e.g., because it is able to geometrically encode correlations), one may need to considerably alter the internal evaluation measures and update steps of these algorithms in order to reflect the higher representational complexity. Both theoretical and practical studies are needed in order to evaluate, whether such a modification is feasible and promising.

*Language games with
our formalization*

Conceptual spaces have already been successfully used in the context of language games [149, 421]. One could extend these studies by using our formalization of the framework, which would allow to also introduce operations on concepts: Agents could then for instance also pick a word that refers to the intersection operation from Section 3.1, and then use utterances such as "red intersect round" to describe an object, which is both red and round. Again, such an application of our formalization would, however, require to develop specific update procedures for the conceptual representations based on the results of the interactions.

*Combining multiple
aspects*

In the end, the final goal is of course to combine all three aspects discussed in Chapter 7 into a single algorithm. This single learning mechanism would then use an incremental procedure, which is able to work in an entirely unsupervised way, but which can also incorporate the indirect feedback from language games. Moreover, it would be able to extract structured knowledge (such as "all apples are round") from the geometric representation, and to use this background knowledge as constraints for subsequent learning tasks. One could, for instance, start such an endeavor by using logic tensor networks in an incremental fashion, or by applying them in the context of language games.

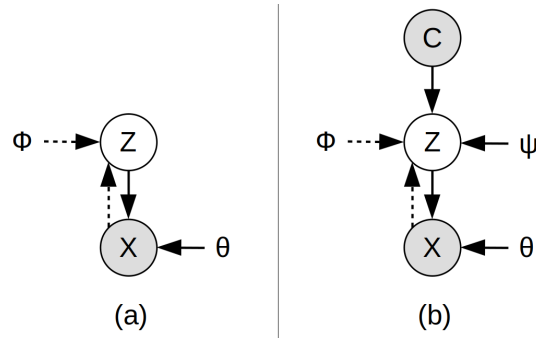


Figure 13.3: (a) Graphical model of a variational autoencoder [224]. (b) Graphical model of a conceptual VAE [363].

As a recent proposal for concept learning in conceptual spaces, we would now like to briefly introduce the work by Shaikh et al. [363], which is based on variational autoencoders [224] (VAEs). Please recall from Section 6.3.4, that the graphical model underlying the VAE approach assumes a latent code \vec{z} with a prior distribution $\mathbb{P}(\vec{z})$, which is then used to sample an observation \vec{x} based on the conditional distribution $\mathbb{P}_\theta(\vec{x}|\vec{z})$ (cf. Figure 13.3a). In VAEs, a recognition model $\mathbb{Q}_\phi(\vec{z}|\vec{x})$ is employed to make inferences about the generative factors based on an observation. VAEs implement both $\mathbb{P}_\theta(\vec{x}|\vec{z})$ and $\mathbb{Q}_\phi(\vec{z}|\vec{x})$ as neural networks, which predict the mean and variance of a Gaussian distribution, assuming that also $\mathbb{P}(\vec{z})$ is Gaussian. Shaikh et al. note, that the vanilla VAE model does not incorporate class labels – while it is possible to infer a latent code for a given observation, one cannot directly assign it to a category. Their *conceptual VAE* model adds an explicit representation of concepts by introducing a class label \vec{c} to the underlying graphical model. As one can see in Figure 13.3b, the starting point of the generative story now shifts from \vec{z} to \vec{c} : First, a class label \vec{c} is generated based on the prior distribution $\mathbb{P}(\vec{c})$, which is then used to sample a latent code \vec{z} based on $\mathbb{P}_\psi(\vec{z}|\vec{c})$. This distribution $\mathbb{P}_\psi(\vec{z}|\vec{c})$ is assumed to be a multivariate Gaussian and corresponds to the representation of \vec{c} in the latent space. The remainder of the model with $\mathbb{P}_\theta(\vec{x}|\vec{z})$ and $\mathbb{Q}_\phi(\vec{z}|\vec{x})$ remains unchanged.

Shaikh et al. argue, that the training procedure for a conceptual VAE is largely identical to the original VAE approach: Given an observation \vec{x} with a class label \vec{c} , one first uses the encoder $\mathbb{Q}_\phi(\vec{z}|\vec{x})$ to sample a latent code \vec{z}_s , whose decoder distribution $\mathbb{P}_\theta(\vec{x}|\vec{z}_s)$ gives rise to the reconstruction loss (cf. Section 6.3.4). Now instead of considering the KL divergence between $\mathbb{Q}_\phi(\vec{z}|\vec{x})$ and $\mathbb{P}(\vec{z})$ as in the original VAE, the conceptual VAE uses the KL divergence between $\mathbb{Q}_\phi(\vec{z}|\vec{x})$ and $\mathbb{P}_\psi(\vec{z}|\vec{c})$ as additional loss term, where now also the parameters ψ (i.e., the shape of the conceptual regions) must be optimized. Shaikh et al. show in their paper, that classifying a novel instance (which corresponds to finding $\arg \max_{\vec{c}} \mathbb{P}_{\theta, \psi}(\vec{c}|\vec{x})$) can be done by finding the class \vec{c} , whose representation $\mathbb{P}_\psi(\vec{z}|\vec{c})$ has the smallest KL divergence to the given observation's encoding $\mathbb{Q}_\phi(\vec{z}|\vec{x})$.

The conceptual VAE

Training and classification

Experimental results

In their experiments, Shaikh et al. considered concepts, which were based on the domains `SHAPE`, `COLOR`, `SIZE`, and `POSITION` – the classes \vec{c} consisted of a vector with one label per domain. Shaikh et al. found, that the conceptual VAE learned meaningful conceptual regions, which were defined on a single latent dimension representing the respective domain. Based on the labels, their system was able to disentangle the predefined domains. Interestingly, a meaningful ordering of labels within the domains was observed, for instance `MEDIUM` being located between `SMALL` and `LARGE` in the `SIZE` domain. Shaikh et al. also report very high accuracies for their proposed classification approach, as well as meaningful morphs between observations based on interpolations in latent space.

Relations to other work

Finally, Shaikh et al. establish a formal link of their model to the conceptual spaces framework by showing, that it fits nicely with work linking conceptual spaces to category theory [70, 411], if one assumes fuzzy concepts. The experiments by Shaikh et al. are somewhat limited by assuming only a single dimension per domain, and by using a predefined domain structure. Their work does not directly address any of the three topics considered in Chapter 7, but by allowing for partially filled class labels (representing, that only information with respect to a subset of domains is available), they implicitly allow for semi-supervised learning. While we have argued in Section 6.4, that VAEs may be a useful tool for discovering the dimensions of a conceptual space, Shaikh et al. rather apply them to concept learning. This difference in usage may, however, be a starting point for an integrated system: For instance, when considering our shape data from Chapter 10, one could use a conceptual VAE to learn a representation of multidimensional `SHAPE` space (potentially aided by psychological constraints, cf. Section 8.7.2) using $\mathbb{P}_\theta(\vec{x}|\vec{z})$ and $\mathbb{Q}_\phi(\vec{z}, \vec{x})$, while at the same time enforcing a structure with respect to the visually coherent categories or the interpretable features `FORM`, `LINES`, and `ORIENTATION` using $\mathbb{P}_\psi(\vec{z}|\vec{c})$. Finally, one can establish a link between the conceptual VAE and the SCAN network [197] as discussed in Section 6.4.1: Both approaches augment standard VAE approaches by additional structures for extracting domain-wise labels from the latent representation. While SCAN uses a second VAE, whose latent representation is tied to the latent space of the low-level VAE, the conceptual VAE achieves the integration of class labels by augmenting the underlying graphical model of the VAE with an additional variable.

13.4 THE BIG PICTURE

Conceptual spaces as "Lingua Franca"

Lieto et al. [260] have argued, that conceptual spaces can be used as a "Lingua Franca" for cognitive architectures, because they offer a way to integrate different representational approaches (cf. Section 1.2.4). This role of conceptual spaces as a powerful integration tool has also surfaced at many points during this dissertation.

Conceptual spaces provide a platform, which enables interdisciplinary exchanges in cognitive science (cf. Section 1.2): They are based on psychological similarity spaces and prototype theory, but also relate to the feature spaces commonly used in machine learning. Moreover, they can be linked to neuroscientific findings and provide a principled way of grounding different linguistic word types. From the perspective of AI, conceptual spaces can help to solve the symbol grounding problem [190], i.e., the problem of linking abstract symbols to the real world. This much-needed interdisciplinary perspective is crucial to progress in cognitive science.

An interdisciplinary perspective

Also when looking at the two learning processes in conceptual spaces [182], their integrative power becomes apparent: If we consider the grounding of the conceptual space in the subsymbolic layer, the framework allows us to link different strands of research, ranging from dimensionality reduction and metric learning (cf. Section 5.3) over word embeddings and representation learning (cf. Section 6.3) to multidimensional scaling (cf. Chapter 8). Also with respect to learning conceptual regions, we can use conceptual spaces to take a unified view on few shot learning, incremental processing, language games, and knowledge-based constraints (cf. Chapter 7).

Learning processes

In Section 1.3, we have identified three core issues for applying conceptual spaces in the context of artificial intelligence. As we have already summarized in Sections 13.1.1, 13.2.1, and 13.3.1, we were able to make progress on all three fronts:

Reconsidering the three core issues

With respect to the conceptual layer, we were able to provide a thorough formalization and implementation of the framework, which allows us to represent correlations in a geometric manner through the shape of the conceptual regions. This increased expressiveness allows for more powerful predictions, but comes at an increased complexity for learning concepts and reasoning on them. Our publicly available implementation can serve as a useful starting point for further practical research on conceptual spaces.

Formalizing the conceptual layer

We have also extensively discussed different ways of grounding the conceptual layer in subsymbolic computation. While a variety of approaches is possible in principle, we have focused our efforts on the integration of psychologically derived similarity spaces with deep neural networks. Learning a mapping from images to coordinates in the conceptual space proved to be a difficult task, limited mainly by the small amount of available stimulus-point mappings. As discussed in Section 13.2.3, the combination of neural networks and cognitive models is an emerging area of research with many recent contributions.

Perceptual grounding

This dissertation was not able to make any principled contributions to the problem of learning conceptual regions. Nevertheless, we have argued, that in theory, many off-the-shelf approaches can be adapted to the conceptual spaces framework. Here, the main modifications needed for such an adaption mostly concern the domain structure of the conceptual space and the constraint for conceptual regions to be convex or star-shaped. We have provided a first sketch, how logic

Learning conceptual regions

tensor networks [21] can be employed to learn conceptual regions under knowledge-based constraints.

*Towards an
integrated system*

Although we were able to make progress with respect to all three core issues, we are still relatively far away from the overall vision of an integrated system as sketched in Section 1.3.1.7. Before such an overall integrated system is feasible, further improvements on its individual components need to be made. While the formalization of the conceptual layer may already be in a good enough shape for first applications, especially the grounding of domains in neural networks needs some further work. Once the mapping learned by these networks is stable and precise enough, they can be used in a first prototype of an integrated system. Also for learning conceptual regions, some first feasibility studies are needed in order to identify promising approaches and configurations, before attempting an integration into an overall system.

Outlook

Nevertheless, we should not put off the integration of these three modules for too long: None of them will become optimal in the foreseeable future, but we may still obtain valuable insights from their combination. These insights about the interaction of multiple parts may be crucial in shaping the overall envisioned system. Overall, such an integrated prototype would be a valuable contribution to both aspects of cognitive AI [259]: It can serve as a partial explanation for the human mind with testable predictions, but it can also help us to build more intelligent artificial agents.

⁷ Also Galetić et al. [162] have recently proposed an explainable AI system in the aerospace manufacturing domain, which is based on conceptual spaces, and which employs our SHAPE spaces from Chapter 11. Since their paper provides only a high-level overview of the proposed system, but no implementational details or evaluation results, we assume, that their system is still in quite early stages of development.

BIBLIOGRAPHY

- [1] Ahmed M. H. Abdelfattah and Wael Zakaria. Employing a Restricted Set of Qualitative Relations in Recognizing Plain Sketches. In Gabriele Kern-Isberner, Johannes Fürnkranz, and Matthias Thimm, editors, *KI 2017: Advances in Artificial Intelligence*, volume 10505 of *Lecture Notes in Computer Science*, pages 3–14, Cham, 2017. Springer International Publishing.
- [2] Hervé Abdi and Lynne J. Williams. Principal Component Analysis. *WIREs Computational Statistics*, 2(4):433–459, 2010.
- [3] Benjamin Adams and Martin Raubal. A Metric Conceptual Space Algebra. In Kathleen Stewart Hornsby, Christophe Claramunt, Michel Denis, and Gérard Ligozat, editors, *Spatial Information Theory*, pages 51–68, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
- [4] Benjamin Adams and Martin Raubal. Conceptual Space Markup Language (CSML): Towards the Cognitive Semantic Web. In *3rd IEEE International Conference on Semantic Computing*, pages 253–260, 2009.
- [5] Thomas Ager, Ondřej Kuželka, and Steven Schockaert. Inducing Symbolic Rules from Entity Embeddings using Auto-Encoders. In *11th International Workshop on Neural-Symbolic Learning and Reasoning*, 2016.
- [6] Thomas Ager, Ondřej Kuželka, and Steven Schockaert. Modelling Salient Features as Directions in Fine-Tuned Semantic Spaces. In *22nd Conference on Computational Natural Language Learning*, pages 530–540, 2018.
- [7] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. On the Surprising Behavior of Distance Metrics in High Dimensional Space. In *8th International Conference on Database Theory*, pages 420–434. Springer Berlin Heidelberg, 2001.
- [8] Rakesh Agrawal and Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *20th International Conference on Very Large Data Bases*, pages 487–499, 1994.
- [9] Alan Agresti and Ranjini Natarajan. Modeling Clustered Ordered Categorical Data: A Survey. *International Statistical Review*, 69(3):345–371, 2001.
- [10] Ola Ahlqvist. A Parameterized Representation of Uncertain Conceptual Spaces. *Transactions in GIS*, 8(4):493–514, 2004.

- [11] Janet Aisbett and Greg Gibbon. A General Formulation of Conceptual Spaces as a Meso Level Representation. *Artificial Intelligence*, 133(1-2):189–232, 2001.
- [12] Muhannad Alomari, Paul Duckworth, David C. Hogg, and Anthony G. Cohn. Natural Language Acquisition and Grounding for Embodied Robotic Systems. In *31st AAAI Conference on Artificial Intelligence*, pages 4349–4356, 2017.
- [13] John R. Anderson. *How Can the Human Mind Occur in the Physical Universe?*, volume 3. Oxford University Press, 2009.
- [14] Martin Arjovsky and Léon Bottou. Towards Principled Methods for Training Generative Adversarial Networks. In *5th International Conference on Learning Representations*, 2017.
- [15] R. Artusi, P. Verderio, and E. Marubini. Bravais-Pearson and Spearman Correlation Coefficients: Meaning, Test of Hypothesis and Confidence Interval. *The International Journal of Biological Markers*, 17(2):148–151, 2002.
- [16] F. Gregory Ashby and Ralph E. Gott. Decision Rules in the Perception and Categorization of Multidimensional Stimuli. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 14(1):33–53, 1988.
- [17] Nicholas Asher. *Lexical Meaning in Context: A Web of Words*. Cambridge University Press, 2011.
- [18] Ioanna M. Attarian, Brett D. Roads, and Michael C. Mozer. Transforming Neural Network Visual Representations to Predict Human Judgments of Similarity. In *2nd Workshop on Shared Visual Representations in Human and Machine Intelligence*, 2020.
- [19] Fred Attneave. Dimensions of Similarity. *The American Journal of Psychology*, 63(4):516–556, 1950.
- [20] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. DBpedia: A Nucleus for a Web of Open Data. In Karl Aberer, Key-Sun Choi, Natasha Noy, Dean Allemang, Kyung-Il Lee, Lyndon Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, editors, *6th International Semantic Web Conference*, pages 722–735, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
- [21] Samy Badreddine, Artur d’Avila Garcez, Luciano Serafini, and Michael Spranger. Logic Tensor Networks. *Artificial Intelligence*, 303:103649, 2022.
- [22] Samy Badreddine and Michael Spranger. Injecting Prior Knowledge for Transfer Learning into Reinforcement Learning Algorithms using Logic Tensor Networks. *arXiv preprint, arXiv:1906.06576*, 2019.

- [23] Samy Badreddine and Michael Spranger. Extending Real Logic with Aggregate Functions. In *15th International Workshop on Neural-Symbolic Learning and Reasoning*, pages 115–125, 2021.
- [24] Nicholas Baker, Hongjing Lu, Gennady Erlikhman, and Philip J. Kellman. Deep Convolutional Networks Do Not Classify Based on Global Object Shape. *PLOS Computational Biology*, 14(12):1–43, 2018.
- [25] Christian Balkenius and Peter Gärdenfors. Spaces in the Brain: From Neurons to Meanings. *Frontiers in Psychology*, 7:1820, 2016.
- [26] Hadi Banaee, Erik Schaffernicht, and Amy Loutfi. An Elephant in the Dark: Creating Semantic Representations of Perceived Data with Conceptual Spaces. In *6th International Workshop on Artificial Intelligence and Cognition*, 2018.
- [27] Hadi Banaee, Erik Schaffernicht, and Amy Loutfi. Data-driven Conceptual Spaces: Creating Semantic Representations for Linguistic Descriptions of Numerical Data. *Journal of Artificial Intelligence Research*, 63:691–742, 2018.
- [28] Shipra Banik and B. M. Golam Kibria. Confidence Intervals for the Population Correlation Coefficient ρ . *International Journal of Statistics in Medical Research*, 5:99–111, 2016.
- [29] Moshe Bar. A Cortical Mechanism for Triggering Top-Down Facilitation in Visual Object Recognition. *Journal of Cognitive Neuroscience*, 15(4):600–609, 2003.
- [30] Andrea Baraldi and Palma Blonda. A Survey of Fuzzy Clustering Algorithms for Pattern Recognition. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 29(6):778–785, 1999.
- [31] Andrea Baraldi and Flavio Parmiggiani. Novel Neural Network Model Combining Radial Basis Function, Competitive Hebbian Learning Rule, and Fuzzy Simplified Adaptive Resonance Theory. *Applications of Soft Computing*, 3165:98–112, 1997.
- [32] Lawrence W. Barsalou. Frames, Concepts, and Conceptual Fields. In Adrienne Lehrer and Eva Feder Kittay, editors, *Frames, Fields, and Contrasts*, pages 21–74. Lawrence Erlbaum Associates, Inc, 1992.
- [33] Sugato Basu, Ian Davidson, and Kiri Wagstaff, editors. *Constrained Clustering: Advances in Algorithms, Theory, and Applications*. Taylor & Francis, 2008.
- [34] Ruairidh M. Battleday, Joshua C. Peterson, and Thomas L. Griffiths. Capturing Human Categorization of Natural Images by Combining Deep Networks and Cognitive Models. *Nature Communications*, 11(1):1–14, 2020.

- [35] Ruairidh M. Battleday, Joshua C. Peterson, and Thomas L. Griffiths. From Convolutional Neural Networks to Models of Higher-Level Cognition (and Back Again). *Annals of the New York Academy of Sciences*, 2021.
- [36] Lucas Bechberger. The Size of a Hyperball in a Conceptual Space. *arXiv preprint, arXiv:1708.05263*, 2017.
- [37] Lucas Bechberger. lbechberger/LearningPsychologicalSpaces v1.3: Study on Multidimensional Scaling and Neural Networks on the NOUN Dataset. <https://doi.org/10.5281/zenodo.4061287>, 2020.
- [38] Lucas Bechberger. Towards Conceptual Logic Tensor Networks. In *5th Workshop on Cognition And OntologieS*, 2021.
- [39] Lucas Bechberger. lbechberger/LearningPsychologicalSpaces v1.6: Learning a Mapping From Images Into MDS-Based Similarity Spaces. <https://doi.org/10.5281/zenodo.5524374>, 2022.
- [40] Lucas Bechberger and Elektra Kypridemou. Mapping Images to Psychological Similarity Spaces Using Neural Networks. In *6th International Workshop on Artificial Intelligence and Cognition*, pages 26–39, 2018.
- [41] Lucas Bechberger and Kai-Uwe Kühnberger. A Comprehensive Implementation of Conceptual Spaces. In *5th International Workshop on Artificial Intelligence and Cognition*, 2017.
- [42] Lucas Bechberger and Kai-Uwe Kühnberger. A Thorough Formalization of Conceptual Spaces. In Gabriele Kern-Isberner, Johannes Fürnkranz, and Matthias Thimm, editors, *KI 2017: Advances in Artificial Intelligence*, volume 10505 of *Lecture Notes in Computer Science*, pages 58–71, Cham, 2017. Springer International Publishing.
- [43] Lucas Bechberger and Kai-Uwe Kühnberger. Measuring Relations Between Concepts In Conceptual Spaces. In Max Bramer and Miltos Petridis, editors, *37th SGAI International Conference on Artificial Intelligence*, pages 87–100. Springer International Publishing, 2017.
- [44] Lucas Bechberger and Kai-Uwe Kühnberger. Towards Grounding Conceptual Spaces in Neural Representations. In *12th International Workshop on Neural-Symbolic Learning and Reasoning*, 2017.
- [45] Lucas Bechberger and Kai-Uwe Kühnberger. Formal Ways for Measuring Relations between Concepts in Conceptual Spaces. *Expert Systems*, 38(7):e12348, 2018.
- [46] Lucas Bechberger and Kai-Uwe Kühnberger. Formalized Conceptual Spaces with a Geometric Representation of Correlations. In Mauri Kaipainen, Frank Zenker, Antti Hautamäki, and Peter

- Gärdenfors, editors, *Conceptual Spaces: Elaborations and Applications*, volume 405 of *Synthese Library*, pages 29–58. Springer International Publishing, Cham, 2019.
- [47] Lucas Bechberger and Kai-Uwe Kühnberger. Generalizing Psychological Similarity Spaces to Unseen Stimuli – Combining Multidimensional Scaling with Artificial Neural Networks. In Lucas Bechberger, Kai-Uwe Kühnberger, and Mingya Liu, editors, *Concepts in Action: Representation, Learning, and Application*, pages 11–36. Springer International Publishing, Cham, 2021.
- [48] Lucas Bechberger and Kai-Uwe Kühnberger. Grounding Psychological Shape Space in Convolutional Neural Networks. In *3rd International Workshop on Cognition: Interdisciplinary Foundations, Models and Applications*, 2021.
- [49] Lucas Bechberger, Kai-Uwe Kühnberger, and Mingya Liu, editors. *Concepts in Action: Representation, Learning, and Application*, volume 9 of *Language, Cognition, and Mind*. Springer International Publishing, 2021.
- [50] Lucas Bechberger and Mingya Liu. Concepts in Action: Introduction. In Lucas Bechberger, Kai-Uwe Kühnberger, and Mingya Liu, editors, *Concepts in Action: Representation, Learning, and Application*, volume 9 of *Language, Cognition, and Mind*, pages 1–9. Springer International Publishing, Cham, 2021.
- [51] Lucas Bechberger and Marius Pol. Ibechberger/ConceptualSpaces: Version 1.3.2. <https://doi.org/10.5281/zenodo.5883567>, 2022.
- [52] Lucas Bechberger and Margit Scheibel. Analyzing Psychological Similarity Spaces for Shapes. In Mehwish Alam, Tanya Braun, and Bruno Yun, editors, *Ontologies and Concepts in Mind and Machine*, pages 204–207, Cham, 2020. Springer International Publishing.
- [53] Lucas Bechberger and Margit Scheibel. Representing Complex Shapes with Conceptual Spaces. In *2nd International Workshop "Concepts in Action: Representation, Learning, and Application"*, 2020.
- [54] Lucas Bechberger and Margit Scheibel. Modeling the Holistic Perception of Everyday Object Shapes with Conceptual Spaces. In preparation.
- [55] Radim Bělohávek and George J. Klir. *Concepts and Fuzzy Logic*. MIT Press, 2011.
- [56] Yoshua Bengio, Aaron Courville, and Psacal Vincent. Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

- [57] James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for Hyper-Parameter Optimization. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2546–2554. Curran Associates, Inc., 2011.
- [58] Pavel Berkhin. A Survey of Clustering Data Mining Techniques. In Jacob Kogan, Charles Nicholas, and Marc Teboulle, editors, *Grouping Multidimensional Data: Recent Advances in Clustering*, pages 25–71. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.
- [59] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web: A New Form of Web Content that is Meaningful to Computers will Unleash a Revolution of New Possibilities. *Scientific American*, 284(5):34–43, 2001.
- [60] Dimitri P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods*. Academic Press, 2014.
- [61] Tarek R. Besold, Artur d’Avila Garcez, Sebastian Bader, Howard Bowman, Pedro Domingos, Pascal Hitzler, Kai-Uwe Kühnberger, Luis C. Lamb, Priscila M. V. Lima, Leo de Penning, Gadi Pinkas, Hoifung Poon, and Gerson Zaverucha. Neural-Symbolic Learning and Reasoning: A Survey and Interpretation. In Pascal Hitzler and Md Kamruzzaman Sarker, editors, *Neuro-Symbolic Artificial Intelligence: The State of the Art*, volume 342 of *Frontiers in Artificial Intelligence and Applications*, pages 1–51. IOS Press, 2021.
- [62] Federico Bianchi and Pascal Hitzler. On the Capabilities of Logic Tensor Networks for Deductive Reasoning. In *AAAI Spring Symposium on Combining Machine Learning with Knowledge Engineering*, 2019.
- [63] Federico Bianchi, Matteo Palmonari, Pascal Hitzler, and Luciano Serafini. Complementing Logical Reasoning with Sub-Symbolic Commonsense. In Paul Fodor, Marco Montali, Diego Calvanese, and Dumitru Roman, editors, *Rules and Reasoning*, pages 161–170, Cham, 2019. Springer International Publishing.
- [64] Irving Biederman. Recognition-by-Components: A Theory of Human Image Understanding. *Psychological Review*, 94(2):115–147, 1987.
- [65] Dorrit Billman and Evan Heit. Observational Learning from Internal Feedback: A Simulation of an Adaptive Learning Method. *Cognitive Science*, 12(4):587–625, 1988.
- [66] Dorrit Billman and James Knutson. Unsupervised Concept Learning and Value Systematicity: A Complex Whole Aids Learning the Parts. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 22(2):458–475, 1996.

- [67] Joachim Bleys, Martin Loetzsch, Michael Spranger, and Luc Steels. The Grounded Color Naming Game. In *18th IEEE International Symposium on Robot and Human Interactive Communication*, 2009.
- [68] Kenneth P. Bogart. *Introductory Combinatorics*. Saunders College Publishing, Philadelphia, PA, USA, 2nd edition, 1989.
- [69] Josef Bolt, Bob Coecke, Fabrizio Genovese, Martha Lewis, Daniel Marsden, and Robin Piedeleu. Interacting Conceptual Spaces. In *1st Workshop on Semantic Spaces at the Intersection of NLP, Physics and Cognitive Science*, volume 221 of *Electronic Proceedings in Theoretical Computer Science*, pages 1–19, 2016.
- [70] Josef Bolt, Bob Coecke, Fabrizio Genovese, Martha Lewis, Daniel Marsden, and Robin Piedeleu. Interacting Conceptual Spaces I: Grammatical Composition of Concepts. In Mauri Kaipainen, Frank Zenker, Antti Hautamäki, and Peter Gärdenfors, editors, *Conceptual Spaces: Elaborations and Applications*, volume 405 of *Synthese Library*, pages 151–181. Springer International Publishing, Cham, 2019.
- [71] Ingwer Borg and Patrick J. F. Groenen. *Modern Multidimensional Scaling: Theory and Applications*. Springer Series in Statistics. Springer-Verlag New York, 2nd edition, 2005.
- [72] Ingwer Borg and Detlev Leutner. Dimensional Models for the Perception of Rectangles. *Perception & Psychophysics*, 34(3):257–267, 1983.
- [73] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A Training Algorithm for Optimal Margin Classifiers. In *5th Annual Workshop on Computational Learning Theory*, pages 144–152. Association for Computing Machinery, 1992.
- [74] Bernadette Bouchon-Meunier, Maria Rifqi, and Sylvie Bothorel. Towards General Measures of Comparison of Objects. *Fuzzy Sets and Systems*, 84(2):143–153, 1996.
- [75] Zied Bouraoui, Víctor Gutiérrez-Basulto, and Steven Schockaert. Integrating Ontologies and Vector Space Embeddings Using Conceptual Spaces. In Camille Bourgaux, Ana Ozaki, and Rafael Peñaloza, editors, *International Research School in Artificial Intelligence in Bergen*, volume 99 of *Open Access Series in Informatics (OASIs)*, pages 3:1–3:30, 2022.
- [76] Zied Bouraoui and Steven Schockaert. Learning Conceptual Space Representations of Interrelated Concepts. In *27th International Joint Conference on Artificial Intelligence*, pages 1760–1766, 2018.
- [77] Jeffrey S. Bowers, Gaurav Malhotra, Marin Dujmović, Milton L. Montero, Christian Tsvetkov, Valerio Biscione, Guillermo Puebla, Federico G. Adolphi, John Hummel, Rachel F. Heaton, Benjamin D.

- Evans, Jeffrey Mitchell, and Ryan Blything. Deep Problems with Neural Network Models of Human Vision. *PsyArXiv preprint, 5zf4s*, 2022.
- [78] Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001.
- [79] Rasmus Bro and Age K. Smilde. Principal Component Analysis. *Analytical Methods*, 6:2812–2831, 2014.
- [80] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large Scale GAN Training for High Fidelity Natural Image Synthesis. In *7th International Conference on Learning Representations*, 2019.
- [81] Fabian Bross. Using Mixed Effect Models to Analyze Acceptability Ratings. Technical report, <https://fabianbross.de>, 2019.
- [82] Luitzen E. J. Brouwer. Über eineindeutige, stetige Transformation von Flächen in sich. *Mathematische Annalen*, 69:176–180, 1910.
- [83] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems 33*, pages 1877–1901. Curran Associates, Inc., 2020.
- [84] Christopher P. Burgess, Irina Higgins, Arka Pal, Loic Matthey, Nick Watters, Guillaume Desjardins, and Alexander Lerchner. Understanding Disentangling in β -VAE. In *Workshop "Learning Disentangled Representations: from Perception to Control" at the 31st Conference on Neural Information Processing Systems*, 2017.
- [85] Angelo Cangelosi. Solutions and Open Challenges for the Symbol Grounding Problem. *International Journal of Signs and Semiotic Systems (IJSSS)*, 1(1):49–54, 2011.
- [86] Nan Cao, Xin Yan, Yang Shi, and Chaoran Chen. AI-Sketcher : A Deep Generative Model for Producing High-Quality Sketches. In *33rd AAAI Conference on Artificial Intelligence*, pages 2564–2571, 2019.
- [87] Susan Carey and Elsa Bartlett. Acquiring a Single New Word. *Papers and Reports on Child Language Development*, 15:17–29, 1978.
- [88] Rudolf Carnap. A Basic System of Inductive Logic, Part I. In Richard Jeffrey and Rudolf Carnap, editors, *Studies in Inductive*

- Logic and Probability*, pages 34–165. University of California Press: Los Angeles, 1971.
- [89] Rudolf Carnap. A Basic System of Inductive Logic, Part II. *Studies in Inductive Logic and Probability*, 2:7–155, 1980.
- [90] Gail A. Carpenter and Stephen Grossberg. A Massively Parallel Architecture for a Self-Organizing Neural Pattern Recognition Machine. *Computer Vision, Graphics, and Image Processing*, 37(1):54–115, 1987.
- [91] Gail A. Carpenter and Stephen Grossberg. ART 2: Self-Organization of Stable Category Recognition Codes for Analog Input Patterns. *Applied Optics*, 26(23):4919–4930, 1987.
- [92] Gail A. Carpenter and Stephen Grossberg. Adaptive Resonance Theory. In C. Sammut and G. I. Webb, editors, *Encyclopedia of Machine Learning and Data Mining*, pages 1–17. Springer Science + Business Media, 2016.
- [93] Gail A. Carpenter, Stephen Grossberg, and David B. Rosen. Fuzzy ART: Fast Stable Learning and Categorization of Analog Patterns by an Adaptive Resonance System. *Neural Networks*, 4(6):759–771, 1991.
- [94] Jon W. Carr, Kenny Smith, Jennifer Culbertson, and Simon Kirby. Simplicity and Informativeness in Semantic Category Systems. *Cognition*, 202:104289, 2020.
- [95] Antonio Chella, Silvia Coradeschi, Marcello Frixione, and Alessandro Saffiotti. Perceptual Anchoring via Conceptual Spaces. In *AAAI Workshop on Anchoring Symbols to Sensor Data*, 2004.
- [96] Antonio Chella, Haris Dindo, and Ignazio Infantino. Anchoring by Imitation Learning in Conceptual Spaces. In Stefania Bandini and Sara Manzoni, editors, *AI*IA 2005: Advances in Artificial Intelligence*, pages 495–506, Cham, 2005. Springer International Publishing.
- [97] Antonio Chella, Marcello Frixione, and Salvatore Gaglio. Conceptual Spaces for Computer Vision Representations. *Artificial Intelligence Review*, 16(2):137–152, 2001.
- [98] Antonio Chella, Marcello Frixione, and Salvatore Gaglio. Anchoring Symbols to Conceptual Spaces: The Case of Dynamic Scenarios. *Robotics and Autonomous Systems*, 43(2-3):175–188, 2003.
- [99] C. L. Philip Chen and Yuan Lu. FUZZ: A Fuzzy-Based Concept Formation System That Integrates Human Categorization and Numerical Clustering. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 27(1):79–94, 1997.

- [100] Tian Qi Chen, Xuechen Li, Roger B. Grosse, and David K. Duvenaud. Isolating Sources of Disentanglement in Variational Autoencoders. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems* 31, pages 2614–2624. Curran Associates, Inc., 2018.
- [101] Xi Chen, Yan Duan, Rein Houthoofd, John Schulman, Ilya Sutskever, and Pieter Abbeel. InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems* 29, pages 2172–2180. Curran Associates, Inc., 2016.
- [102] Rune H. B. Christensen. A Tutorial on fitting Cumulative Link Mixed Models with `clmm2` from the `ordinal` Package. Technical report, <https://cran.r-project.org>, 2019.
- [103] Rune H. B. Christensen. `ordinal`—Regression Models for Ordinal Data. <https://CRAN.R-project.org/package=ordinal>, 2019. R package version 2019.12-10.
- [104] Rune H. B. Christensen and Per B. Brockhoff. Analysis of Sensory Ratings Data with Cumulative Link Models. *Journal de la Société Française de Statistique*, 154(3):58–79, 2013.
- [105] Wang Chua-Chin and Don Hon-Son. A Modified Measure for Fuzzy Subsethood. *Information Sciences*, 79(3-4):223–232, 1994.
- [106] Radoslaw M. Cichy, Aditya Khosla, Dimitrios Pantazis, Antonio Torralba, and Aude Oliva. Comparison of Deep Neural Networks to Spatio-Temporal Cortical Dynamics of Human Visual Object Recognition Reveals Hierarchical Correspondence. *Scientific Reports*, 6:27755, 2016.
- [107] Viviane Clay, Peter König, Gordon Pipa, and Kai-Uwe Kühnberger. Fast Concept Mapping: The Emergence of Human Abilities in Artificial Neural Networks when Learning Embodied and Self-Supervised. *arXiv preprint, arXiv:2102.02153*, 2021.
- [108] Jacob Cohen. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement*, 20(1):37–46, 1960.
- [109] Silvia Coradeschi, Amy Loutfi, and Britta Wrede. A Short Review of Symbol Grounding in Robotic and Intelligent Systems. *KI – Künstliche Intelligenz*, 27(2):129–136, 2013.
- [110] Corinna Cortes and Vladimir Vapnik. Support-Vector Networks. *Machine Learning*, 20(3):273–297, 1995.
- [111] Artur d’Avila Garcez, Tarek R. Besold, Luc De Raedt, Peter Földiák, Pascal Hitzler, Thomas Icard, Kai-Uwe Kühnberger, Luis C. Lamb, Risto Miikkulainen, and Daniel L. Silver. Neural-Symbolic

- Learning and Reasoning: Contributions and Challenges. In *AAAI 2015 Spring Symposium on Knowledge Representation and Reasoning: Integrating Symbolic and Neural Approaches*, 2015.
- [112] Artur d'Ávila Garcez and Luis C. Lamb. Neurosymbolic AI: The 3rd Wave. *arXiv preprint, arXiv:2012.05876*, 2020.
- [113] Ernest Davis and Gary Marcus. Commonsense Reasoning and Commonsense Knowledge in Artificial Intelligence. *Communications of the ACM*, 58(9):92–103, 2015.
- [114] Roberto G. de Almeida and Lila R. Gleitman, editors. *On Concepts, Modules, and Language: Cognitive Science at its Core*. Oxford University Press, 2017.
- [115] Jan de Leeuw. Applications of Convex Analysis to Multidimensional Scaling. In J. R. Barra, F. Brodeau, G. Romier, and B. van Cutsem, editors, *Recent Developments in Statistics*, pages 133–146. North Holland Publishing, 1977.
- [116] Jan de Leeuw and Patrick Mair. Multidimensional Scaling Using Majorization: SMACOF in R. *Journal of Statistical Software, Articles*, 31(3):1–30, 2009.
- [117] Lieven Decock and Igor Douven. What Is Graded Membership? *Noûs*, 48(4):653–682, 2014.
- [118] Lieven Decock, Igor Douven, and Marta Sznajder. A Geometric Principle of Indifference. *Journal of Applied Logic*, 19:54–70, 2016. SI: Dynamics of Knowledge and Belief.
- [119] Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.
- [120] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *22nd IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.
- [121] George DeRise. Some n-Dimensional Geometry. *International Journal of Mathematical Education in Science and Technology*, 23(3):371–379, 1992.
- [122] Joaquín Derrac and Steven Schockaert. Enriching Taxonomies of Place Types Using Flickr. In Christoph Beierle and Carlo Meghini, editors, *8th International Symposium on Foundations of Information and Knowledge Systems*, pages 174–192, Cham, 2014. Springer International Publishing.
- [123] Joaquín Derrac and Steven Schockaert. Inducing Semantic Relations from Conceptual Spaces: A Data-Driven Approach to Plausible Reasoning. *Artificial Intelligence*, 228:66–94, 2015.

- [124] Jean-Louis Dessalles. From Conceptual Spaces to Predicates. In Frank Zenker and Peter Gärdenfors, editors, *Applications of Conceptual Spaces: The Case for Geometric Knowledge Representation*, pages 17–31. Springer International Publishing, Cham, 2015.
- [125] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding. In *2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186. Association for Computational Linguistics, 2019.
- [126] Alessandro Di Nuovo and Angelo Cangelosi. Abstract Concept Learning in Cognitive Robots. *Current Robotics Reports*, 2:1–8, 2021.
- [127] Gil Diesendruck and Paul Bloom. How Specific is the Shape Bias? *Child Development*, 74(1):168–178, 2003.
- [128] Richard Dietz. Comparative Concepts. *Synthese*, 190(1):139–170, 2012.
- [129] Stefan Dietze and John Domingue. Exploiting Conceptual Spaces for Ontology Integration. In *Data Integration Through Semantic Technology Workshop at 3rd Asian Semantic Web Conference*, 2008.
- [130] Carl Doersch. Tutorial on Variational Autoencoders. *arXiv preprint, arXiv:1606.05908*, 2016.
- [131] Ivan Donadello and Luciano Serafini. Compensating Supervision Incompleteness with Prior Knowledge in Semantic Image Interpretation. In *2019 International Joint Conference on Neural Networks*, pages 1–8, 2019.
- [132] Ivan Donadello, Luciano Serafini, and Artur d’Avila Garcez. Logic Tensor Networks for Semantic Image Interpretation. In *26th International Joint Conference on Artificial Intelligence*, 2017.
- [133] Derek Doran, Sarah Schulz, and Tarek R. Besold. What Does Explainable AI Really Mean? A New Conceptualization of Perspectives. In *1st International Workshop on Comprehensibility and Explanation in AI and ML*, 2017.
- [134] Alexey Dosovitskiy and Thomas Brox. Inverting Visual Representations With Convolutional Networks. In *29th IEEE Conference on Computer Vision and Pattern Recognition*, June 2016.
- [135] Alexey Dosovitskiy, Jost Tobias Springenberg, and Thomas Brox. Learning to Generate Chairs With Convolutional Neural Networks. In *28th IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

- [136] Igor Douven, Lieven Decock, Richard Dietz, and Paul Égré. Vagueness: A Conceptual Spaces Approach. *Journal of Philosophical Logic*, 42(1):137–160, 2011.
- [137] Igor Douven and Peter Gärdenfors. What are Natural Concepts? A Design Perspective. *Mind & Language*, 35(3):313–334, 2020.
- [138] Harris Drucker, Christopher J. C. Burges, Linda Kaufman, Alex J. Smola, and Vladimir Vapnik. Support Vector Regression Machines. In M. Jordan, M. Kearns, and S. Solla, editors, *Advances in Neural Information Processing Systems 10*, pages 155–161. MIT Press, 1997.
- [139] Matthew J. Dry and Gert Storms. Similar but not the Same: A Comparison of the Utility of Directly Rated and Feature-Based Similarity Measures for Generating Spatial Models of Conceptual Data. *Behavior Research Methods*, 41:889–900, 2009.
- [140] John Duchi, Elad Hazan, and Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011.
- [141] Ishan Durugkar, Ian Gemp, and Sridhar Mahadevan. Generative Multi-Adversarial Networks. In *5th International Conference on Learning Representations*, 2017.
- [142] Cian Eastwood and Christopher K. I. Williams. A Framework for the Quantitative Evaluation of Disentangled Representations. In *6th International Conference on Learning Representations*, 2018.
- [143] Mathias Eitz, James Hays, and Marc Alexa. How Do Humans Sketch Objects? *ACM Transactions on Graphics*, 31(4):1–10, 2012.
- [144] Chris Eliasmith. *How to Build a Brain: A Neural Architecture for Biological Cognition*. Oxford University Press, 2013.
- [145] Andreas K. Engel, Alexander Maye, Martin Kurthen, and Peter König. Where’s the Action? The Pragmatic Turn in Cognitive Science. *Trends in Cognitive Sciences*, 17(5):202–209, 2013.
- [146] Goker Erdogan and Robert A. Jacobs. Visual Shape Perception as Bayesian Inference of 3D Object-Centered Shape Representations. *Psychological Review*, 124(6):740–761, 2017.
- [147] Philippe Esling, Axel Chemla-Romeu-Santos, and Adrien Bitton. Generative Timbre Spaces: Regularizing Variational Auto-Encoders With Perceptual Metrics. In *21st International Conference on Digital Audio Effects*, pages 369–376, 2018.
- [148] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *2nd International Conference on Knowledge Discovery and Data Mining*, pages 226–231. AAAI Press, 1996.

- [149] Henrietta Eyre and Jonathan Lawry. Language Games with Vague Categories and Negations. *Adaptive Behavior*, 22(5):289–303, 2014.
- [150] Jiulun Fan, Weixin Xie, and Jihong Pei. Subsethood Measure: New Definitions. *Fuzzy Sets and Systems*, 106(2):201–209, 1999.
- [151] Judith E. Fan, Daniel L. K. Yamins, and Nicholas B. Turk-Browne. Common Object Representations for Visual Production and Recognition. *Cognitive Science*, 42(8):2670–2698, 2018.
- [152] Martha J. Farah. Is an Object an Object an Object? Cognitive and Neuropsychological Investigations of Domain Specificity in Visual Object Recognition. *Current Directions in Psychological Science*, 1(5):164–169, 1992.
- [153] Samuel J. Ferguson. A One-Sentence Line-of-Sight Proof of the Extreme Value Theorem. *The American Mathematical Monthly*, 121(4):331, 2014.
- [154] Matthias Feurer and Frank Hutter. Hyperparameter Optimization. In Frank Hutter, Lart Kotthoff, and Joaquin Vanschoren, editors, *Automated Machine Learning: Methods, Systems, Challenges*, The Springer Series on Challenges in Machine Learning, pages 3–33. Springer International Publishing, Cham, 2019.
- [155] Andy Field, Jeremy Miles, and Zoë Field. *Discovering Statistics Using R*. Sage Publications, 2012.
- [156] Michael Fink. Object Classification from a Single Example Utilizing Class Relevance Metrics. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 449–456. MIT Press, 2005.
- [157] Sandro R. Fiorini. *Similarity, Structure and Spaces: Representation of Part-Whole Relations in Conceptual Spaces*. PhD thesis, Universidade Federal do Rio Grande do Sul, 2014.
- [158] Sandro R. Fiorini, Peter Gärdenfors, and Mara Abel. Representing Part-Whole Relations in Conceptual Spaces. *Cognitive Processing*, 15(2):127–142, 2013.
- [159] Douglas H. Fisher. Knowledge Acquisition via Incremental Conceptual Clustering. *Machine Learning*, 2(2):139–172, 1987.
- [160] Stan Franklin, Tamas Madl, Sidney D’Mello, and Javier Snaider. LIDA: A Systems-level Architecture for Cognition, Emotion, and Learning. *IEEE Transactions on Autonomous Mental Development*, 6(1):19–41, 2014.
- [161] Leonard Frommelt. One-Shot Concept Learning in Conceptual Spaces. Bachelor thesis, Osnabrück University, 2021.

- [162] Vedran Galetić and Alistair Nottle. Inherently Interpretable Knowledge Representation for a Trustworthy Artificially Intelligent Agent Teaming with Humans in Industrial Environments. In *8th International Workshop on Artificial Intelligence and Cognition*, 2022.
- [163] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A. Wichmann, and Wieland Brendel. ImageNet-Trained CNNs are Biased Towards Texture; Increasing Shape Bias Improves Accuracy and Robustness. In *7th International Conference on Learning Representations*, 2019.
- [164] John H. Gennari, Pat Langley, and Doug Fisher. Models of Incremental Concept Formation. *Artificial Intelligence*, 40(1-3):11–61, 1989.
- [165] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Elsevier, 2004.
- [166] Scott Glover and Peter Dixon. Likelihood Ratios: A Simple and Flexible Statistic for Empirical Psychologists. *Psychonomic Bulletin & Review*, 11(5):791–806, 2004.
- [167] Jacob Goldberger, Geoffrey E. Hinton, Sam T. Roweis, and Russ R. Salakhutdinov. Neighbourhood Components Analysis. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 513–520. MIT Press, 2005.
- [168] Howard Goldowsky. (La)Place Cells for Robot Navigation. In *Computational Cognition Workshop*, 2021.
- [169] Robert Goldstone. An Efficient Method for Obtaining Similarity Data. *Behavior Research Methods, Instruments, & Computers*, 26(4):381–386, 1994.
- [170] Robert Goldstone. The Learning and Use of Concepts by Humans. <https://youtu.be/rUu7lIxjUnM?t=46>, https://youtu.be/Sa_GWjgFgl4?t=3317 and <https://youtu.be/-YzoLsNbNT8?t=105>, 2018. Lecture at CARLA Summer School 2018.
- [171] Asunción Gómez-Pérez, Mariano Fernández-López, and Oscar Corcho. *Ontological Engineering: With Examples From the Areas of Knowledge Management, e-Commerce and the Semantic Web*. Springer, 2004.
- [172] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [173] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In Z. Ghahramani,

- M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc., 2014.
- [174] John C. Gower. Some Distance Properties of Latent Root and Vector Methods Used in Multivariate Analysis. *Biometrika*, 53(3/4):325–338, 1966.
- [175] Stephen Grossberg. Competitive Learning: From Interactive Activation to Adaptive Resonance. *Cognitive Science*, 11(1):23–63, 1987.
- [176] Abhijeet Gupta, Gemma Boleda, Marco Baroni, and Sebastian Padó. Distributional Vectors Encode Referential Attributes. In *2015 Conference on Empirical Methods in Natural Language Processing*, pages 12–21. Association for Computational Linguistics, 2015.
- [177] J. Octavio Gutierrez-Garcia and Emmanuel López-Neri. Cognitive Computing: A Brief Survey and Open Research Challenges. In *3rd International Conference on Applied Computing and Information Technology / 2nd International Conference on Computational Science and Intelligence*, pages 328–333, 2015.
- [178] Isabelle Guyon and André Elisseeff. An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.
- [179] Peter Gärdenfors. *Conceptual Spaces: The Geometry of Thought*. MIT Press, 2000.
- [180] Peter Gärdenfors. How to Make the Semantic Web More Semantic. In Achille C. Varzi and Laure Vieu, editors, *3rd International Conference on Formal Ontology in Information Systems*, volume 114 of *Frontiers in Artificial Intelligence and Applications*, pages 17–34, 2004.
- [181] Peter Gärdenfors. *The Geometry of Meaning: Semantics Based on Conceptual Spaces*. MIT Press, 2014.
- [182] Peter Gärdenfors. From Sensations to Concepts: a Proposal for Two Learning Processes. *Review of Philosophy and Psychology*, 10(3):441–464, 2018.
- [183] Peter Gärdenfors. Primary Cognitive Categories Are Determined by Their Invariances. *Frontiers in Psychology*, 11:3429, 2020.
- [184] Peter Gärdenfors and Massimo Warglien. The Development of Semantic Space for Pointing and Verbal Communication. In Carita Paradis, Jean Hudson, and Ulf Magnusson, editors, *The Construal of Spatial Meaning: Windows into Conceptual Space*, pages 29–42. Cambridge University Press, 2013.

- [185] Fritz Günther, Luca Rinaldi, and Marco Marelli. Vector-Space Models of Semantic Representation From a Cognitive Perspective: A Discussion of Common Misconceptions. *Perspectives on Psychological Science*, 14(6):1006–1033, 2019.
- [186] David Ha and Douglas Eck. A Neural Representation of Sketch Drawings. In *6th International Conference on Learning Representations*, 2018.
- [187] James A. Hampton. Overextension of Conjunctive Concepts: Evidence for a Unitary Model of Concept Typicality and Class Inclusion. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 14(1):12–32, 1988.
- [188] James A. Hampton. Typicality, Graded Membership, and Vagueness. *Cognitive Science*, 31(3):355–384, 2007.
- [189] James A. Hampton and Yoad Winter, editors. *Compositionality and Concepts in Linguistics and Psychology*, volume 3 of *Language, Cognition, and Mind*. Springer International Publishing, 2017.
- [190] Stevan Harnad. The Symbol Grounding Problem. *Physica D: Nonlinear Phenomena*, 42(1-3):335–346, 1990.
- [191] Jeff Hawkins, Marcus Lewis, Mirko Klukas, Scott Purdy, and Subutai Ahmad. A Framework for Intelligence and Cortical Function Based on Grid Cells in the Neocortex. *Frontiers in Neural Circuits*, 12:121, 2019.
- [192] Haibo He and Edwardo A. Garcia. Learning from Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284, 2009.
- [193] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *29th IEEE Conference on Computer Vision and Pattern Recognition*, 2016.
- [194] Martin N. Hebart, Charles Y. Zheng, Francisco Pereira, and Chris I. Baker. Revealing the Multidimensional Mental Representations of Natural Objects Underlying Human Similarity Judgements. *Nature Human Behavior*, 4:1173–1185, 2020.
- [195] José V. Hernández-Conde. A Case Against Convexity in Conceptual Spaces. *Synthese*, 193:1–27, 2016.
- [196] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. β -VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. In *5th International Conference on Learning Representations*, 2017.

- [197] Irina Higgins, Nicolas Sonnerat, Loic Matthey, Arka Pal, Christopher P. Burgess, Matthew Botvinick, Demis Hassabis, and Alexander Lerchner. SCAN: Learning Abstract Hierarchical Compositional Visual Concepts. In *6th International Conference on Learning Representations*, 2018.
- [198] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural Networks for Machine Learning – Lecture 6: RMSProp. Lecture at University of Toronto, http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf, 2012.
- [199] Jessica S. Horst and Michael C. Hout. The Novel Object and Unusual Name (NOUN) Database: A Collection of Novel Images for Use in Experimental Research. *Behavior Research Methods*, 48(4):1393–1409, 2016.
- [200] Michael C. Hout, Stephen D. Goldinger, and Kyle J. Brady. MM-MDS: A Multidimensional Scaling Database with Similarity Ratings for 240 Object Categories from the Massive Memory Picture Database. *PLOS ONE*, 9(11):1–11, 2014.
- [201] Michael C. Hout, Stephen D. Goldinger, and Ryan W. Ferguson. The Versatility of SpAM: A Fast, Efficient, Spatial Method of Data Collection for Multidimensional Scaling. *Journal of Experimental Psychology: General*, 142(1):256–281, 2013.
- [202] Liqiang Huang. Space of Preattentive Shape Features. *Journal of Vision*, 20(4):1–20, 2020.
- [203] David H. Hubel and Torsten N. Wiesel. Receptive Fields of Single Neurones in the Cat’s Striate Cortex. *The Journal of Physiology*, 148(3):574–591, 1959.
- [204] Bärbel Inhelder and Jean Piaget. *The Early Growth of Logic in the Child: Classification and Seriation*. Routledge, 1964.
- [205] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In Francis Bach and David Blei, editors, *32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456. PMLR, 2015.
- [206] Anil K. Jain, M. Naramisha Murty, and Patrick J. Flynn. Data Clustering: A Review. *ACM Computing Surveys*, 31(3):264–323, 1999.
- [207] Shoaib Jameel, Zied Bouraoui, and Steven Schockaert. MEmBER: Max-Margin Based Embeddings for Entity Retrieval. In *40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 783–792, 2017.
- [208] Shoaib Jameel and Steven Schockaert. Entity Embeddings with Conceptual Subspaces as a Basis for Plausible Reasoning. In

- Gal A. Kaminka, Maria Fox, Paolo Bouquet, Eyke Hüllermeier, Virginia Dignum, Frank Dignum, and Frank van Harmelen, editors, *22nd European Conference on Artificial Intelligence*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 1353–1361. IOS Press, 2016.
- [209] Aditi Jha, Joshua Peterson, and Tom Griffiths. Extracting Low-Dimensional Psychological Representations from Convolutional Neural Networks. In *42nd Annual Meeting of the Cognitive Science Society*, pages 2180–2186, 2020.
- [210] Yichen Jiang, Asli Celikyilmaz, Paul Smolensky, Paul Soulos, Sudha Rao, Hamid Palangi, Roland Fernandez, Caitlin Smith, Mohit Bansal, and Jianfeng Gao. Enriching Transformers with Structured Tensor-Product Representations for Abstractive Summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4780–4793, Online, June 2021. Association for Computational Linguistics.
- [211] Mikael Johannesson. The Problem of Combining Integral and Separable Dimensions. Technical Report HS-IDA-TR-01-002, University of Skövde, School of Humanities and Informatics, 2001.
- [212] Susan S. Jones and Linda B. Smith. The Place of Perception in Children’s Concepts. *Cognitive Development*, 8(2):113–139, 1993.
- [213] Alan Jović, Karla Brkić, and Nikola Bogunović. A Review of Feature Selection Methods with Applications. In *38th International Convention on Information and Communication Technology, Electronics and Microelectronics*, pages 1200–1205, 2015.
- [214] Georg Jäger, Christian A. Mueller, Madhura Thosar, Sebastian Zug, and Andreas Birk. Towards Robot-Centric Conceptual Knowledge Acquisition. In *IROS 2018 Workshop "Robots that learn and reason"*, 2018.
- [215] Gerhard Jäger and Robert van Rooij. Language Structure: Psychological and Social Constraints. *Synthese*, 159(1):99–130, 2007.
- [216] Abdullah T. Kabakus. A Novel Sketch Recognition Model based on Convolutional Neural Networks. In *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications*, pages 1–6, 2020.
- [217] Daniel Kahneman. *Thinking, Fast and Slow*. Penguin Books, 2011.
- [218] Mauri Kaipainen, Frank Zenker, Antti Hautamäki, and Peter Gärdenfors, editors. *Conceptual Spaces: Elaborations and Applications*, volume 405 of *Synthese Library*. Springer, 2019.
- [219] Maurice G. Kendall. A New Measure of Rank Correlation. *Biometrika*, 30(1-2):81–93, 1938.

- [220] Alan W. Kersten and Dorrit Billman. Event Category Learning. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 23(3):638–358, 1997.
- [221] Hyunjik Kim and Andriy Mnih. Disentangling by Factorising. In Jennifer Dy and Andreas Krause, editors, *35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2649–2658. PMLR, 2018.
- [222] Joo-Kyung Kim and Marie-Catherine de Marneffe. Deriving Adjectival Scales from Continuous Space Word Representations. In *2013 Conference on Empirical Methods in Natural Language Processing*, pages 1625–1630. Association for Computational Linguistics, 2013.
- [223] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations*, 2015.
- [224] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations*, 2014.
- [225] Teuvo Kohonen. The Self-Organizing Map. *Neurocomputing*, 21(1):1–6, 1998.
- [226] Bart Kosko. *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence*. Prentice Hall, 1992.
- [227] David H. Krantz and Amos Tversky. Similarity of Rectangles: An Analysis of Subjective Dimensions. *Journal of Mathematical Psychology*, 12(1):4–34, 1975.
- [228] Nikolaus Kriegeskorte. Deep Neural Networks: A New Framework for Modeling Biological Vision and Brain Information Processing. *Annual Review of Vision Science*, 1(1):417–446, 2015.
- [229] Nikolaus Kriegeskorte and Marieke Mur. Inverse MDS: Inferring Dissimilarity Structure from Multiple Item Arrangements. *Frontiers in Psychology*, 3:245, 2012.
- [230] Nikolaus Kriegeskorte, Marieke Mur, and Peter Bandettini. Representational Similarity Analysis - Connecting the Branches of Systems Neuroscience. *Frontiers in Systems Neuroscience*, 2, 2008.
- [231] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Technical report, University of Toronto, 2009.
- [232] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

- [233] Joseph B. Kruskal. Multidimensional Scaling by Optimizing Goodness of Fit to a Nonmetric Hypothesis. *Psychometrika*, 29(1):1–27, 1964.
- [234] Joseph B. Kruskal. Nonmetric Multidimensional Scaling: A Numerical Method. *Psychometrika*, 29(2):115–129, 1964.
- [235] Jonas Kubilius, Stefania Bracci, and Hans P. Op de Beeck. Deep Neural Networks as a Computational Model for Human Shape Sensitivity. *PLOS Computational Biology*, 12(4):1–26, 2016.
- [236] Brian Kulis. Metric Learning: A Survey. *Foundations and Trends in Machine Learning*, 5(4):287–364, 2012.
- [237] Abhishek Kumar, Prasanna Sattigeri, and Avinash Balakrishnan. Variational Inference of Disentangled Latent Concepts from Unlabeled Observations. In *6th International Conference on Learning Representations*, 2018.
- [238] Elektra Kypridemou and Loizos Michael. Reaction Time as an Indicator of Instance Typicality in Conceptual Spaces. In *5th International Workshop on Artificial Intelligence and Cognition*, 2017.
- [239] John E. Laird. *The Soar Cognitive Architecture*. MIT Press, 2012.
- [240] Brenden Lake, Wojciech Zaremba, Rob Fergus, and Todd Gureckis. Deep Neural Networks Predict Category Typicality Ratings for Images. In D. C. Noelle, R. Dale, A. S. Warlaumont, J. Yoshimi, T. Matlock, C. D. Jennings, and P. P. Maglio, editors, *37th Annual Meeting of the Cognitive Science Society*, pages 1243–1248, 2015.
- [241] Christoph H. Lampert, Hannes Nickisch, and Stefan Harmeling. Attribute-Based Classification for Zero-Shot Visual Object Categorization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(3):453–465, 2014.
- [242] Barbara Landau, Linda Smith, and Susan Jones. Object Perception and Object Naming in Early Development. *Trends in Cognitive Sciences*, 2(1):19–24, 1998.
- [243] Thomas K. Landauer, Peter W. Foltz, and Darrell Laham. An Introduction to Latent Semantic Analysis. *Discourse Processes*, 25(2-3):259–284, 1998.
- [244] Anders B. L. Larsen, Søren K. Sønderby, Hugo Larochelle, and Ole Winther. Autoencoding Beyond Pixels Using a Learned Similarity Metric. In Maria F. Balcan and Kilian Q. Weinberger, editors, *33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1558–1566. PMLR, 2016.
- [245] Marek Lassak. Approximation of Convex Bodies by Rectangles. *Geometriae Dedicata*, 47(1):111–117, 1993.

- [246] Yann LeCun, Bernhard Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne Hubbard, and Lawrence D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551, 1989.
- [247] Ickjai Lee and Bayani Portier. An Empirical Study of Knowledge Representation and Learning within Conceptual Spaces for Intelligent Agents. *6th IEEE/ACIS International Conference on Computer and Information Science*, 2007.
- [248] Michael D. Lee and Kenneth J. Pope. Avoiding the Dangers of Averaging across Subjects when using Multidimensional Scaling. *Journal of Mathematical Psychology*, 47(1):32–46, 2003.
- [249] Tali Leibovich and Daniel Ansari. The Symbol-Grounding Problem in Numerical Cognition: A Review of Theory, Evidence, and Outstanding Questions. *Canadian Journal of Experimental Psychology*, 70(1):12–23, 2016.
- [250] Dominik Leiner. Sosci survey [computer software]. <https://www.sosciurvey.de>, 2016.
- [251] Alessandro Lenci. Distributional Models of Word Meaning. *Annual Review of Linguistics*, 4(1):151–171, 2018.
- [252] Martha Lewis. Towards Logical Negation for Compositional Distributional Semantics. *IfCoLoG Journal of Logics and their Applications*, 7(5):771–793, 2020.
- [253] Martha Lewis and Jonathan Lawry. Hierarchical Conceptual Spaces for Concept Combination. *Artificial Intelligence*, 237:204–227, 2016.
- [254] Pablo León-Villagrà, Isaac Ehrlich, Chris Lucas, and Daphna Buchsbaum. Recovering Human Category Structure Across Development Using Sparse Judgments. In *3rd International Workshop "Concepts in Action: Representation, Learning, and Application"*, 2021.
- [255] Aedan Y. Li, Jackson C. Liang, Andy C. H. Lee, and Morgan D. Barense. The Validated Circular Shape Space: Quantifying the Visual Similarity of Shape. *Journal of Experimental Psychology: General*, 149(5):949–966, 2019.
- [256] Yali Li, Shengjin Wang, Qi Tian, and Xiaoqing Ding. A Survey of Recent Advances in Visual Feature Detection. *Neurocomputing*, 149:736–751, 2015.
- [257] Zejian Li, Yongchuan Tang, and Yongxing He. Unsupervised Disentangled Representation Learning with Analogical Relations. In *27th International Joint Conference on Artificial Intelligence*, pages 2418–2424, 2018.

- [258] Antonio Lieto. Heterogeneous Proxytypes Extended: Integrating Theory-Like Representations and Mechanisms with Prototypes and Exemplars. In Alexei V. Samsonovich, editor, *8th Annual Meeting of the BICA Society*, pages 217–227, Cham, 2019. Springer International Publishing.
- [259] Antonio Lieto. *Cognitive Design for Artificial Minds*. Routledge, 2021.
- [260] Antonio Lieto, Antonio Chella, and Marcello Frixione. Conceptual Spaces for Cognitive Architectures: A Lingua Franca for Different Levels of Representation. *Biologically Inspired Cognitive Architectures*, 19:1–9, 2017.
- [261] Antonio Lieto, Enrico Mensa, and Daniele P. Radicioni. A Resource-Driven Approach for Anchoring Linguistic Resources to Conceptual Spaces. In Giovanni Adorni, Stefano Cagnoni, Marco Gori, and Marco Maratea, editors, *AI*IA 2016 Advances in Artificial Intelligence*, pages 435–449, Cham, 2016. Springer International Publishing.
- [262] Antonio Lieto, Andrea Minieri, Alberto Piana, and Daniele P. Radicioni. A Knowledge-Based System For Prototypical Reasoning. *Connection Science*, 27(2):137–152, 2015.
- [263] Antonio Lieto and Gianluca Pozzato. A Description Logic Framework for Commonsense Conceptual Combination Integrating Typicality, Probabilities and Cognitive Heuristics. *Journal of Experimental & Theoretical Artificial Intelligence*, 32(5):769–804, 2020.
- [264] Antonio Lieto, Daniele P. Radicioni, and Valentina Rho. A Common-Sense Conceptual Categorization System Integrating Heterogeneous Proxytypes and the Dual Process of Reasoning. In *24th International Joint Conference on Artificial Intelligence*, pages 875–881, 2015.
- [265] Antonio Lieto, Daniele P. Radicioni, and Valentina Rho. Dual PECCS: A Cognitive System for Conceptual Representation and Categorization. *Journal of Experimental & Theoretical Artificial Intelligence*, 29(2):433–452, 2017.
- [266] Antonio Lieto, Daniele P. Radicioni, Valentina Rho, and Enrico Mensa. Towards a Unifying Framework for Conceptual Representation and Reasoning in Cognitive Systems. *Intelligenza Artificiale*, 11(2):139–153, 2017.
- [267] Hugo Liu and Push Singh. ConceptNet – A Practical Commonsense Reasoning Tool-Kit. *BT Technology Journal*, 22(4):211–226, 2004.
- [268] Edward H. Livingston. Who was Student and why do we Care so much about his t-Test? *Journal of Surgical Research*, 118(1):58–65, 2004.

- [269] Bradley C. Love and Todd M. Gureckis. Models in Search of a Brain. *Cognitive, Affective, & Behavioral Neuroscience*, 7(2):90–108, 2007.
- [270] Bradley C. Love, Douglas L. Medin, and Todd M. Gureckis. SUSTAIN: A Network Model of Category Learning. *Psychological Review*, 111(2):309–332, 2004.
- [271] Thomas Low, Christian Borgelt, Sebastian Stober, and Andreas Nürnberger. The Hubness Phenomenon: Fact or Artifact? In Christian Borgelt, María À. Gil, Joao M. C. Sousa, and Michel Verleysen, editors, *Towards Advanced Data Analysis by Combining Soft Computing and Statistics*, volume 285 of *Studies in Fuzziness and Soft Computing*, pages 267–278. Springer Berlin Heidelberg, 2013.
- [272] Sebastian Löbner, Thomas Gamerschlag, Tobias Kalenscher, Markus Schrenk, and Henk Zeevat, editors. *Concepts, Frames and Cascades in Semantics, Cognition and Ontology*, volume 7 of *Language, Cognition, and Mind*. Springer International Publishing, 2021.
- [273] Gary Marcus. The Next Decade in AI: Four Steps Towards Robust Artificial Intelligence. *arXiv preprint, arXiv:2002.06177*, 2020.
- [274] Gary Marcus and Ernest Davis. *Rebooting AI: Building Artificial Intelligence We Can Trust*. Pantheon, 2019.
- [275] Eric Margolis and Stephen Laurence, editors. *The Conceptual Mind: New Directions in the Study of Concepts*. MIT Press, 2015.
- [276] Michael Marino, Pascal Nieters, Gunther Heidemann, and Joachim Hertzberg. Embedding Semantic Relationships in Hidden Representations via Label Smoothing. Withdrawn from the 9th International Conference on Learning Representations, 2021.
- [277] Raja Marjeh, Ilia Sucholutsky, Theodore R. Sumers, Nori Jacoby, and Thomas L. Griffiths. Predicting Human Similarity Judgments Using Large Language Models. In *1st Workshop on Learning with Natural Language Supervision*, 2022.
- [278] David Marr and Herbert K. Nishihara. Representation and Recognition of the Spatial Organization of Three-Dimensional Shapes. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 200(1140):269–294, 1978.
- [279] Yoshihiro Maruyama. Symbolic and Statistical Theories of Cognition: Towards Integrated Artificial Intelligence. In Loek Cleophas and Mieke Massink, editors, *Software Engineering and Formal Methods. SEFM 2020 Collocated Workshops*, pages 129–146, Cham, 2021. Springer International Publishing.

- [280] Margalida Mas, Marta Monserrat, Joan Torrens, and Enric Trillas. A Survey on Fuzzy Implication Functions. *IEEE Transactions on Fuzzy Systems*, 15(6):1107–1121, 2007.
- [281] Cynthia Matuszek, Michael Witbrock, John Cabral, and John DeOliveira. An Introduction to the Syntax and Content of Cyc. In *AAAI Spring Symposium on Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering*, 2006.
- [282] Julian J. McAuley and Jure Leskovec. From Amateurs to Connoisseurs: Modeling the Evolution of User Expertise through Online Reviews. In *22nd International Conference on World Wide Web*, pages 897–908, 2013.
- [283] John McCarthy. Situations, Actions, and Causal Laws. Technical report, Stanford University, 1963.
- [284] John McCarthy and Patrick J. Hayes. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In Bonnie L. Webber and Nils J. Nilsson, editors, *Readings in Artificial Intelligence*, pages 431–450. Morgan Kaufmann, 1981.
- [285] Michael McCloskey and Neal J. Cohen. Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. *Psychology of Learning and Motivation*, 24:109–165, 1989.
- [286] Stephen McGregor, Kat Agres, Matthew Purver, and Geraint A. Wiggins. From Distributional Semantics to Conceptual Spaces: A Novel Computational Method for Concept Creation. *Journal of Artificial General Intelligence*, 6(1):55–86, 2015.
- [287] Douglas L. Medin and Marguerite M. Schaffer. Context Theory of Classification Learning. *Psychological Review*, 85(3):207–238, 1978.
- [288] Douglas L. Medin and Edward J. Shoben. Context and Structure in Conceptual Combination. *Cognitive Psychology*, 20(2):158–190, 1988.
- [289] Johannes Mehrer, Courtney J. Spoerer, Emer C. Jones, Nikolaus Kriegeskorte, and Tim C. Kietzmann. An Ecologically Motivated Image Dataset for Deep Learning Yields Better Models of Human Vision. *Proceedings of the National Academy of Sciences*, 118(8):e2011417118, 2021.
- [290] Tomas Mikolov, Kai Chen, Greg S. Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. In *1st International Conference on Learning Representations*, 2013.
- [291] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. Distributed Representations of Words and Phrases and their Compositionality. In C. J. C. Burges, L. Bottou, M. Welling,

- Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.
- [292] George A. Miller. WordNet: A Lexical Database for English. *Communications of the ACM*, 38(11):39–41, 1995.
- [293] Yang Mingqiang, Kpalma Kidiyo, and Ronsin Joseph. A Survey of Shape Feature Extraction Techniques. *Pattern Recognition*, 15(7):43–90, 2008.
- [294] Marvin Minsky. A Framework for Representing Knowledge. In Dieter Metzger, editor, *Frame Conceptions and Text Understanding*, volume 5 of *Research in Text Theory*, pages 1–25. De Gruyter, 1979.
- [295] Tom M. Mitchell. *Machine Learning*. McGraw Hill, 1997.
- [296] Ananth Mohan, Zheng Chen, and Kilian Weinberger. Web-Search Ranking with Initialized Gradient Boosted Regression Trees. In Olivier Chapelle, Yi Chang, and Tie-Yan Liu, editors, *Proceedings of the Learning to Rank Challenge*, volume 14 of *Proceedings of Machine Learning Research*, pages 77–89. PMLR, 2011.
- [297] Yaniv Morgenstern, Frieder Hartmann, Filipp Schmidt, Henning Tiedemann, Eugen Prokott, Guido Maiello, and Roland W. Fleming. An Image-Computable Model of Human Visual Shape Similarity. *PLOS Computational Biology*, 17(6):1–34, 2021.
- [298] Gregory L. Murphy. *The Big Book of Concepts*. MIT Press, 2002.
- [299] Gregory L. Murphy and Douglas L. Medin. The Role of Theories in Conceptual Coherence. *Psychological Review*, 92(3):289–316, 1985.
- [300] Alasdair Newson, Andrés Almansa, Yann Gousseau, and Saïd Ladjal. Taking Apart Autoencoders: How do They Encode Geometric Shapes? Technical report, HAL archive, 2018.
- [301] Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y. Ng. Multimodal Deep Learning. In Lise Getoor and Tobias Scheffer, editors, *28th International Conference on Machine Learning*, pages 689–696, 2011.
- [302] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning Deconvolution Network for Semantic Segmentation. In *17th IEEE International Conference on Computer Vision*, pages 1520–1528, 2015.
- [303] Robert M. Nosofsky, Craig A. Sanders, Brian J. Meagher, and Bruce J. Douglas. Toward the Development of a Feature-Space Representation for a Complex Natural Category Domain. *Behavior Research Methods*, 50(2):530–556, 2018.

- [304] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional Image Synthesis with Auxiliary Classifier GANs. In Doina Precup and Yee W. Teh, editors, *34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2642–2651. PMLR, 2017.
- [305] Xenia Ohmer, Michael Marino, Michael Franke, and Peter König. Mutual Influence Between Language and Perception in Multi-Agent Communication Games. *arXiv preprint, arXiv:2112.14518*, 2022.
- [306] Xenia Ohmer, Michael Marino, Peter König, and Michael Franke. Why and How to Study the Impact of Perception on Language Emergence in Artificial Agents. In *43rd Annual Meeting of the Cognitive Science Society*, pages 1139–1145, 2021.
- [307] Bart Ons, Wouter De Baene, and Johan Wagemans. Subjectively Interpreted Shape Dimensions as Privileged and Orthogonal Axes in Mental Shape Space. *Journal of Experimental Psychology: Human Perception and Performance*, 37(2):422–441, 2011.
- [308] Hans P. Op de Beeck, Katrien Torfs, and Johan Wagemans. Perceived Shape Similarity among Unfamiliar Objects and the Organization of the Human Object Vision Pathway. *Journal of Neuroscience*, 28(40):10111–10123, 2008.
- [309] Daniel N. Osherson and Edward E. Smith. On the Adequacy of Prototype Theory as a Theory of Concepts. *Cognition*, 9(1):35–58, 1981.
- [310] Daniel N. Osherson and Edward E. Smith. Gradedness and Conceptual Combination. *Cognition*, 12(3):299–318, 1982.
- [311] Matías Osta-Vélez and Peter Gärdenfors. Category-Based Induction in Conceptual Spaces. *Journal of Mathematical Psychology*, 96:102357, 2020.
- [312] Özgür L. Özçep, Mena Leemhuis, and Diedrich Wolter. Cone Semantics for Logics with Negation. In *29th International Joint Conference on Artificial Intelligence*, pages 1820–1826, 2020.
- [313] John Palmer, Cynthia T. Ames, and Delwin T. Lindsey. Measuring the Effect of Attention on Simple Visual Search. *Journal of Experimental Psychology: Human Perception and Performance*, 19(1):108–130, 1993.
- [314] Stephen E. Palmer. *Vision Science: Photons to Phenomenology*. MIT Press, 1999.
- [315] Karl Pearson. Note on Regression and Inheritance in the Case of Two Parents. *Proceedings of the Royal Society of London*, 58(347-352):240–242, 1895.

- [316] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [317] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global Vectors for Word Representation. In *2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.
- [318] Joshua C. Peterson, Joshua T. Abbott, and Thomas L. Griffiths. Adapting Deep Network Features to Capture Psychological Representations: An Abridged Report. In *26th International Joint Conference on Artificial Intelligence*, pages 4934–4938, 2017.
- [319] Joshua C. Peterson, Joshua T. Abbott, and Thomas L. Griffiths. Evaluating (and Improving) the Correspondence Between Deep Neural Networks and Human Representations. *Cognitive Science*, 42(8):2648–2669, 2018.
- [320] Joshua C. Peterson, Ruairidh M. Battleday, Thomas L. Griffiths, and Olga Russakovsky. Human Uncertainty Makes Classification More Robust. In *22nd IEEE International Conference on Computer Vision*, pages 9617–9626, 2019.
- [321] Marius Pol and Ada Diaconescu. A Cognitive Control System for Managing Runtime Uncertainty in Self-Integrating Autonomic Systems. In *1st IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion*, pages 85–90, 2020.
- [322] Marius Pol and Ada Diaconescu. Multi-Level Online Learning and Reasoning for Self-Integrating Systems. In *2nd IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion*, pages 187–192, 2021.
- [323] Nina L. Poth. A Bayesian Approach Towards Concept Learning and an Answer to the Complex First Paradox. Master’s thesis, Ruhr University Bochum, 2016.
- [324] Nina L. Poth. Conceptual Spaces, Generalisation Probabilities and Perceptual Categorisation. In Mauri Kaipainen, Frank Zenker, Antti Hautamäki, and Peter Gärdenfors, editors, *Conceptual Spaces: Elaborations and Applications*, volume 405 of *Synthese Library*, chapter Poth2019, pages 7–28. Springer International Publishing, Cham, 2019.
- [325] Nina L. Poth and Peter Brössel. Learning Concepts: A Learning-Theoretic Solution to the Complex-First Paradox. *Philosophy of Science*, 87(1):135–151, 2020.

- [326] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. In *4th International Conference on Learning Representations*, 2016.
- [327] Martin Raubal. Formalizing Conceptual Spaces. In *3rd International Conference on Formal Ontology in Information Systems*, pages 153–164, 2004.
- [328] Russell Richie, Bryan White, Sudeep Bhatia, and Michael C. Hout. The Spatial Arrangement Method of Measuring Similarity Can Capture High-Dimensional Semantic Structures. *Behavior Research Methods*, 52:1906–1928, 2020.
- [329] John T. Rickard. A Concept Geometry for Conceptual Spaces. *Fuzzy Optimization and Decision Making*, 5(4):311–329, 2006.
- [330] John T. Rickard, Janet Aisbett, and Greg Gibbon. Knowledge Representation and Reasoning in Conceptual Spaces. *IEEE Symposium on Foundations of Computational Intelligence*, 2007.
- [331] Maximilian Riesenhuber and Tomaso Poggio. Hierarchical Models of Object Recognition in Cortex. *Nature Neuroscience*, 2(11):1019–1025, 1999.
- [332] Ryan Rifkin and Aldebaro Klautau. In Defense of One-vs-All Classification. *The Journal of Machine Learning Research*, 5:101–141, 2004.
- [333] Guendalina Righetti, Claudio Masolo, Nicolas Troquard, Oliver Kutz, and Daniele Porello. Concept Combination in Weighted Logic. In *5th Workshop on Cognition And Ontologies*, 2021.
- [334] Brett D. Roads and Bradley C. Love. Enriching ImageNet With Human Similarity Judgments and Psychological Embeddings. In *34th IEEE Conference on Computer Vision and Pattern Recognition*, pages 3547–3557, 2021.
- [335] Benjamin Rodatz, Razin A. Shaikh, and Lia Yeh. Conversational Negation using Worldly Context in Compositional Distributional Semantics. In *5th Workshop on Semantic Spaces at the Intersection of NLP, Physics, and Cognitive Science*, pages 53–65, 2021.
- [336] Eleanor Rosch, Carolyn B. Mervis, Wayne D. Gray, David M. Johnson, and Penny Boyes-Braem. Basic Objects in Natural Categories. *Cognitive Psychology*, 8(3):382–439, 1976.
- [337] Brian H. Ross, Susan J. Perkins, and Patricia L. Tenpenny. Reminding-Based Category Learning. *Cognitive Psychology*, 22(4):460–492, 1990.
- [338] Sascha Rothe and Hinrich Schütze. Word Embedding Calculus in Meaningful Ultradense Subspaces. In *54th Annual Meeting of the*

- Association for Computational Linguistics (Volume 2: Short Papers)*, pages 512–517, 2016.
- [339] Peter J. Rousseeuw. Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.
- [340] Enrique H. Ruspini. On the Semantics of Fuzzy Logic. *International Journal of Approximate Reasoning*, 5(1):45–88, 1991.
- [341] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 3rd edition, 2010.
- [342] Mathias Sablé-Meyer, Kevin Ellis, Joshua Tenenbaum, and Stanislas Dehaene. A Language of Thought for the Mental Representation of Geometric Shapes. *PsyArxiv preprint, 28mg4*, 2021.
- [343] Mathias Sablé-Meyer, Joël Fagot, Serge Caparos, Timo van Kerkoerle, Marie Amalric, and Stanislas Dehaene. Sensitivity to Geometric Shape Regularity in Humans and Baboons: A Putative Signature of Human Singularity. *Proceedings of the National Academy of Sciences*, 118(16):e2023123118, 2021.
- [344] Dov Sagi and Bela Julesz. Detection versus Discrimination of Visual Orientation. *Perception*, 13(5):619–628, 1984.
- [345] Ruslan Salakhutdinov and Geoffrey Hinton. Learning a Nonlinear Embedding by Preserving Class Neighbourhood Structure. In Marina Meila and Xiaotong Shen, editors, *11th International Conference on Artificial Intelligence and Statistics*, volume 2 of *Proceedings of Machine Learning Research*, pages 412–419. PMLR, 2007.
- [346] Craig A. Sanders and Robert M. Nosofsky. Using Deep-Learning Representations of Complex Natural Stimuli as Input to Psychological Models of Classification. In *40th Annual Meeting of the Cognitive Science Society*, 2018.
- [347] Craig A. Sanders and Robert M. Nosofsky. Training Deep Networks to Construct a Psychological Feature Space for a Natural-Object Category Domain. *Computational Brain & Behavior*, 3:229–251, 2020.
- [348] Patsorn Sangkloy, Nathan Burnell, Cusuh Ham, and James Hays. The Sketchy Database: Learning to Retrieve Badly Drawn Bunnies. *ACM Transactions on Graphics*, 35(4):1–12, 2016.
- [349] Margit Scheibel and Peter Indefrey. Top-Down Enhanced Object Recognition in Blocking and Priming Paradigms. *Journal of Experimental Psychology: Human Perception and Performance*, 3(49):327–354, 2023.
- [350] Jade Schmidt. Ordinal Response Mixed Models: A Case Study. Master's thesis, Montana State University, 2012.

- [351] Steven Schockaert and Shoaib Jameel. Plausible Reasoning Based on Qualitative Entity Embeddings. In *25th International Joint Conference on Artificial Intelligence*, pages 4078–4081, 2016.
- [352] Steven Schockaert and Henri Prade. Interpolation and Extrapolation in Conceptual Spaces: A Case Study in the Music Domain. In *5th International Conference on Web Reasoning and Rule Systems*, pages 217–231. Springer Nature, 2011.
- [353] Steven Schockaert and Henri Prade. Qualitative Reasoning about Incomplete Categorization Rules Based on Interpolation and Extrapolation in Conceptual Spaces. In Salem Benferhat and John Grant, editors, *5th International Conference on Scalable Uncertainty Management*, pages 303–316. Springer Berlin Heidelberg, 2011.
- [354] Moritz Schubert and Dominik Endres. Empirically Evaluating the Similarity Model of Geist, Lengnink and Wille. In Peter Chapman, Dominik Endres, and Nathalie Pernelle, editors, *Graph-Based Representation and Reasoning*, pages 88–95, Cham, 2018. Springer International Publishing.
- [355] Stephan Schulz, Simon Cruanes, and Petar Vukmirović. Faster, Higher, Stronger: E 2.3. In Pacal Fontaine, editor, *27th International Conference on Automated Deduction*, number 11716 in Lecture Notes in Artificial Intelligence, pages 495–507. Springer, 2019.
- [356] John R. Searle. Minds, Brains, and Programs. *Behavioral and Brain Sciences*, 3(3):417–424, 1980.
- [357] Omar Seddati, Stephane Dupont, and Saïd Mahmoudi. DeepSketch: Deep Convolutional Neural Networks for Sketch Recognition and Similarity Search. In *13th International Workshop on Content-Based Multimedia Indexing*, pages 1–6, 2015.
- [358] Luciano Serafini and Artur d’Avila Garcez. Learning and Reasoning with Logic Tensor Networks. In Giovanni Adorni, Stefano Cagnoni, Marco Gori, and Marco Maratea, editors, *AI*IA 2016 Advances in Artificial Intelligence*, pages 334–348, Cham, 2016. Springer International Publishing.
- [359] Luciano Serafini and Artur d’Avila Garcez. Logic Tensor Networks: Deep Learning and Logical Reasoning from Data and Knowledge. In *11th International Workshop on Neural-Symbolic Learning and Reasoning*, 2016.
- [360] Luciano Serafini, Ivan Donadello, and Artur d’Avila Garcez. Learning and Reasoning in Logic Tensor Networks: Theory and Application to Semantic Image Interpretation. In *Symposium on Applied Computing*, pages 125–130. Association for Computing Machinery, 2017.

- [361] Burr Settles. Active Learning Literature Survey. Technical report, University of Wisconsin-Madison, Department of Computer Sciences, 2009.
- [362] Razin A. Shaikh, Lia Yeh, Benjamin Rodatz, and Bob Coecke. Composing Conversational Negation. In *4th International Conference on Applied Category Theory*, 2021.
- [363] Razin A. Shaikh, Sara S. Zemljic, Sean Tull, and Stephen Clark. The Conceptual VAE. *arXiv preprint, arXiv:2203.11216*, 2022.
- [364] SharkD. File:HSL color solid dblcone.png. https://commons.wikimedia.org/wiki/File:HSL_color_solid_dblcone.png, 2015.
- [365] David Shen and Zaizai Lu. Computation of Correlation Coefficient and Its Confidence Interval in SAS. In *SAS Users Group International 31*, 2006.
- [366] Roger N. Shepard. The Analysis of Proximities: Multidimensional Scaling with an Unknown Distance Function. I. *Psychometrika*, 27(2):125–140, 1962.
- [367] Roger N. Shepard. Attention and the Metric Structure of the Stimulus Space. *Journal of Mathematical Psychology*, 1(1):54–87, 1964.
- [368] Roger N. Shepard. Toward a Universal Law of Generalization for Psychological Science. *Science*, 237(4820):1317–1323, 1987.
- [369] Giovanni Sileno, Isabelle Bloch, Jamal Atif, and Jean-Louis Dessalles. Similarity and Contrast on Conceptual Spaces for Pertinent Description Generation. In Gabriele Kern-Isberner, Johannes Fürnkranz, and Matthias Thimm, editors, *KI 2017: Advances in Artificial Intelligence*, volume 10505 of *Lecture Notes in Computer Science*, pages 262–275, Cham, 2017. Springer International Publishing.
- [370] Giovanni Sileno, Isabelle Bloch, Jamal Atif, and Jean-Louis Dessalles. Computing Contrast on Conceptual Spaces. In *6th International Workshop on Artificial Intelligence and Cognition*, pages 11–25, 2018.
- [371] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*, 529(7587):484–489, 2016.

- [372] John R. Sylvester. Determinants of Block Matrices. *The Mathematical Gazette*, 84(501):460–467, 2000.
- [373] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *3rd International Conference on Learning Representations*, 2015.
- [374] Johannes J. D. Singer, Martin N. Hebart, and Katja Seeliger. The Representation of Object Drawings and Sketches in Deep Convolutional Neural Networks. In *2nd Workshop on Shared Visual Representations in Human and Machine Intelligence*, 2020.
- [375] Johannes J. D. Singer, Katja Seeliger, Tim C. Kietzmann, and Martin N. Hebart. From Photos to Sketches – How Humans and Deep Neural Networks Process Objects across Different Levels of Visual Abstraction. *Journal of Vision*, 22(2):1–19, 2022.
- [376] Pulkit Singh, Joshua C. Peterson, Ruairidh M. Battleday, and Thomas L. Griffiths. End-to-end Deep Prototype and Exemplar Models for Predicting Human Behavior. In *42nd Annual Meeting of the Cognitive Science Society*, pages 634–640, 2020.
- [377] Burrhus F. Skinner. *Science and Human Behavior*. Macmillan, 1953.
- [378] C. R. Smith. A Characterization of Star-Shaped Sets. *The American Mathematical Monthly*, 75(4):386, 1968.
- [379] Edward E. Smith, Daniel N. Osherson, Lance J. Rips, and Margaret Keane. Combining Prototypes: A Selective Modification Model. *Cognitive Science*, 12(4):485–527, 1988.
- [380] Paul Smolensky. Tensor Product Variable Binding and the Representation of Symbolic Structures in Connectionist Systems. *Artificial Intelligence*, 46(1):159–216, 1990.
- [381] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical Networks for Few-Shot Learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4077–4087. Curran Associates, Inc., 2017.
- [382] Pia Sommerauer and Antske Fokkens. Firearms and Tigers are Dangerous, Kitchen Knives and Zebras are Not: Testing whether Word Embeddings Can Tell. In *2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 276–286. Association for Computational Linguistics, 2018.
- [383] Ben Sorscher, Surya Ganguli, and Haim Sompolinsky. The Geometry of Concept Learning. *bioRxiv preprint*, 10.1101/2021.03.21.436284, 2021.
- [384] Charles Spearman. The Proof and Measurement of Association between Two Things. *The American Journal of Psychology*, 15(1):72–101, 1904.

- [385] Michael Spranger and Katrien Beuls. Referential Uncertainty and Word Learning in High-Dimensional, Continuous Meaning Spaces. In *2016 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics*, pages 95–100, 2016.
- [386] Michael Spranger, Martin Loetzsch, and Luc Steels. A Perceptual System for Language Game Experiments. In Luc Steels and Manfred Hild, editors, *Language Grounding in Robots*, pages 89–110. Springer US, Boston, MA, 2012.
- [387] Michael Spranger, Jakob Suchan, and Mehul Bhatt. Robust Natural Language Processing – Combining Reasoning, Cognitive Semantics, and Construction Grammar for Spatial Language. In Subbarao Kambhampati, editor, *25th International Joint Conference on Artificial Intelligence*, 2016.
- [388] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.
- [389] Luc Steels. *The Talking Heads Experiment: Origins of Words and Meanings*. Language Science Press, 2015.
- [390] Luc Steels and Jean-Christophe Baillie. Shared Grounding of Event Descriptions by Autonomous Robots. *Robotics and Autonomous Systems*, 43(2):163–173, 2003.
- [391] Luc Steels and Tony Belpaeme. Coordinating Perceptually Grounded Categories Through Language: A Case Study for Colour. *Behavioral and Brain Sciences*, 28(04):469–529, 2005.
- [392] Stanley S. Stevens. On the Theory of Scales of Measurement. *Science*, 103(2684):677–680, 1946.
- [393] Corina Strößner. Integrating Conceptual Spaces in Frames. *Journal of Applied Logics*, 7(5):683–706, 2020.
- [394] Corina Strößner. Criteria for Naturalness in Conceptual Spaces. In *3rd International Workshop "Concepts in Action: Representation, Learning, and Application"*, 2021.
- [395] Corina Strößner. Criteria for Naturalness in Conceptual Spaces. *Synthese*, 200(2), 2022.
- [396] Student. The Probable Error of a Mean. *Biometrika*, 6(1):1–25, 1908.
- [397] Michael Subkoviak and Alan Roecks. A Closer Look at the Accuracy of Alternative Data Collection Methods for Multidimensional Scaling. *Journal of Educational Measurement*, 13(4):309–317, 1976.
- [398] Ron Sun. *Duality of the Mind: A Bottom-Up Approach Toward Cognition*. Psychology Press, 2001.

- [399] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper With Convolutions. In *28th IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.
- [400] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. In *29th IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.
- [401] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing Properties of Neural Networks. In *2nd International Conference on Learning Representations*, 2014.
- [402] Marta Sznajder. *Inductive Logic on Conceptual Spaces*. PhD thesis, University of Groningen, 2017.
- [403] Marta Sznajder. Inductive Reasoning with Multi-dimensional Concepts. *The British Journal for the Philosophy of Science*, 72(2):465–484, 2021.
- [404] Mariarosaria Taddeo and Luciano Floridi. Solving the Symbol Grounding Problem: A Critical Review of Fifteen Years of Research. *Journal of Experimental & Theoretical Artificial Intelligence*, 17(4):419–445, 2005.
- [405] Hirotaka Takeuchi and Ikujiro Nonaka. The New New Product Development Game. *Harvard Business Review*, 64(1):137–146, 1986.
- [406] Joshua B. Tenenbaum and Thomas L. Griffiths. Generalization, Similarity, and Bayesian Inference. *Behavioral and Brain Sciences*, 24:629–640, 2001.
- [407] Madhura Thosar, Sebastian Zug, Alpha M. Skaria, and Akshay Jain. A Review of Knowledge Bases for Service Robots in Household Environments. In *6th International Workshop on Artificial Intelligence and Cognition*, pages 98–110, 2018.
- [408] Warren S. Torgerson. Multidimensional Scaling: I. Theory and Method. *Psychometrika*, 17(4):401–419, 1952.
- [409] Anne Treisman and Stephen Gormican. Feature Analysis in Early Vision: Evidence from Search Asymmetries. *Psychological Review*, 95(1):15–48, 1988.
- [410] Eleni Triantafillou, Richard Zemel, and Raquel Urtasun. Few-Shot Learning Through an Information Retrieval Lens. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 2255–2265. Curran Associates, Inc., 2017.

- [411] Sean Tull. A Categorical Semantics of Fuzzy Concepts in Conceptual Spaces. In *4th International Conference on Applied Category Theory*, 2021.
- [412] Amos Tversky. Features of Similarity. *Psychological Review*, 84(4):327–352, 1977.
- [413] Thomas Töllner, Michael Zehetleitner, Klaus Gramann, and Hermann J. Müller. Stimulus Saliency Modulates Pre-Attentive Processing Speed in Human Visual Cortex. *PLOS ONE*, 6(1):1–8, 2011.
- [414] Arash Vahdat and Jan Kautz. NVAE: A Deep Hierarchical Variational Autoencoder. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems 33*, pages 19667–19679. Curran Associates, Inc., 2020.
- [415] Emile van Krieken, Erman Acar, and Frank van Harmelen. Analyzing Differentiable Fuzzy Logic Operators. *Artificial Intelligence*, 302:103602, 2022.
- [416] Sjoerd van Steenkiste, Francesco Locatello, Jürgen Schmidhuber, and Olivier Bachem. Are Disentangled Representations Helpful for Abstract Visual Reasoning? In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 14245–14258. Curran Associates, Inc., 2019.
- [417] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.
- [418] Steven Verheyen, Anne White, and Gert Storms. A Comparison of the Spatial Arrangement Method and the Total-Set Pairwise Rating Method for Obtaining Similarity Data in the Conceptual Domain. *Multivariate Behavioral Research*, 57(2-3):356–384, 2020.
- [419] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and Composing Robust Features With Denoising Autoencoders. In Andrew McCallum and Sam Roweis, editors, *25th International Conference on Machine Learning*, pages 1096–1103. Association for Computing Machinery (ACM), 2008.
- [420] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. *Journal of Machine Learning Research*, 11(110):3371–3408, 2010.

- [421] Paul Vogt. Self-Organisation of Conceptual Spaces from Quality Dimensions. In Frank Zenker and Peter Gärdenfors, editors, *Applications of Conceptual Spaces: The Case for Geometric Knowledge Representation*, volume 359 of *Synthese Library*, pages 141–163. Springer International Publishing, Cham, 2015.
- [422] Benedikt Wagner and Artur d’Avila Garcez. Neural-Symbolic Integration for Fairness in AI. In A. Martin, K. Hinkelmann, H.-G. Fill, A. Gerber, D. Lenat, R. Stolle, and F. van Harmelen, editors, *AAAI Spring Symposium on Combining Machine Learning and Knowledge Engineering*, 2021.
- [423] Alexander Waibel, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano, and Kevin J. Lang. Phoneme Recognition Using Time-Delay Neural Networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(3):328–339, 1989.
- [424] Haohan Wang, Songwei Ge, Zachary Lipton, and Eric P. Xing. Learning Robust Global Representations by Penalizing Local Predictive Power. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 10506–10518. Curran Associates, Inc., 2019.
- [425] Pei Wang. Insufficient Knowledge and Resources – A Biological Constraint and Its Functional Implications. In *AAAI Fall Symposium: Biologically Inspired Cognitive Architectures*, pages 188–193, 2009.
- [426] Xianfu Wang. Volumes of Generalized Unit Balls. *Mathematics Magazine*, 78(5):390–395, 2005.
- [427] Yaqing Wang, Quanming Yao, James T. Kwok, and Lionel M. Ni. Generalizing from a Few Examples: A Survey on Few-Shot Learning. *ACM Computing Surveys*, 53(3):1–34, 2021.
- [428] Massimo Warglien and Peter Gärdenfors. Semantics, Conceptual Spaces, and the Meeting of Minds. *Synthese*, 190(12):2165–2193, 2013.
- [429] Kilian Q. Weinberger, John Blitzer, and Lawrence K. Saul. Distance Metric Learning for Large Margin Nearest Neighbor Classification. In Y. Weiss, B. Schölkopf, and J. C. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 1473–1480. MIT Press, 2006.
- [430] Markus Werning. Complex First? On the Evolutionary and Developmental Priority of Semantically Thick Words. *Philosophy of Science*, 77(5):1096–1108, 2010.
- [431] Tom White. Sampling Generative Networks. *arXiv preprint, arXiv:1609.04468*, 2016.

- [432] Tom White and Ian Loh. Generating Animations by Sketching in Conceptual Space. In *8th International Conference on Computational Creativity*, pages 261–268, 2017.
- [433] Florian Wickelmaier. An Introduction to MDS. Technical report, Sound Quality Research Unit, Aalborg University, Denmark, 2003.
- [434] Dominic Widdows and Stanley Peters. Word Vectors and Quantum Logic: Experiments with Negation and Disjunction. In *8th Meeting on the Mathematics of Language*, pages 141–154, 2003.
- [435] Samuel S. Wilks. The Large-Sample Distribution of the Likelihood Ratio for Testing Composite Hypotheses. *The Annals of Mathematical Statistics*, 9(1):60–62, 1938.
- [436] Patrick Winston. 6.034 Artificial Intelligence – Lecture 16: Learning: Support Vector Machines. Massachusetts Institute of Technology. https://www.youtube.com/watch?v=_PwhiWxHK8o and <https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-034-artificial-intelligence-fall-2010>, 2010.
- [437] Ludwig Wittgenstein. *Philosophical Investigations*. Oxford: Blackwell, 1953.
- [438] Svante Wold, Kim Esbensen, and Paul Geladi. Principal Component Analysis. *Chemometrics and Intelligent Laboratory Systems*, 2(1–3):37–52, 1987.
- [439] Sewall Wright. Correlation and Causation. *Journal of Agricultural Research*, 20(7):557–585, 1921.
- [440] Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 82–90. Curran Associates, Inc., 2016.
- [441] Fei Xu and Joshua B. Tenenbaum. Word Learning as Bayesian Inference. *Psychological Review*, 114(2):245–272, 2007.
- [442] Peng Xu, Timothy M. Hospedales, Qiyue Yin, Yi-Zhe Song, Tao Xiang, and Liang Wang. Deep Learning for Free-Hand Sketch: A Survey and A Toolbox. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.
- [443] Daniel L. K. Yamins, Ha Hong, Charles F. Cadieu, Ethan A. Solomon, Darren Seibert, and James J. DiCarlo. Performance-Optimized Hierarchical Models Predict Neural Responses in Higher Visual Cortex. *Proceedings of the National Academy of Sciences*, 111(23):8619–8624, 2014.

- [444] Virginia R. Young. Fuzzy Subsethood. *Fuzzy Sets and Systems*, 77(3):371–384, 1996.
- [445] Qian Yu, Yongxin Yang, Feng Liu, Yi-Zhe Song, Tao Xiang, and Timothy M. Hospedales. Sketch-a-Net: A Deep Neural Network that Beats Humans. *International Journal of Computer Vision*, 122(3):411–425, 2017.
- [446] Qian Yu, Yongxin Yang, Yi-Zhe Song, Tao Xiang, and Timothy Hospedales. Sketch-a-Net that Beats Humans. In Xianghua Xie, Mark W. Jones, and Gary K. L. Tam, editors, *26th British Machine Vision Conference*, pages 7.1–7.12. BMVA Press, 2015.
- [447] Lorijn Zaadnoordijk, Tarek R. Besold, and Rhodri Cusack. Lessons from Infant Learning for Unsupervised Machine Learning. *Nature Machine Intelligence*, 4:510–520, 2022.
- [448] Lotfi A. Zadeh. Fuzzy Sets. *Information and Control*, 8(3):338–353, 1965.
- [449] Lotfi A. Zadeh. The Concept of a Linguistic Variable and its Application to Approximate Reasoning. *Information Sciences*, 8(3):199–249, 1975.
- [450] Lotfi A. Zadeh. A Note on Prototype Theory and Fuzzy Sets. *Cognition*, 12(3):291–297, 1982.
- [451] Matthew D. Zeiler and Rob Fergus. Visualizing and Understanding Convolutional Networks. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *13th European Conference on Computer Vision*, pages 818–833, Cham, 2014. Springer International Publishing.
- [452] Matthew D. Zeiler, Graham W. Taylor, and Rob Fergus. Adaptive Deconvolutional Networks for Mid and High Level Feature Learning. In *13th International Conference on Computer Vision*, pages 2018–2025, 2011.
- [453] Frank Zenker. From Features via Frames to Spaces: Modeling Scientific Conceptual Change Without Incommensurability or Aprioricity. In Thomas Gamerschlag, Doris Gerland, Rainer Osswald, and Wiebke Petersen, editors, *Frames and Concept Types*, volume 94 of *Studies in Linguistics and Philosophy*, pages 69–89. Springer International Publishing, 2013.
- [454] Frank Zenker and Peter Gärdenfors, editors. *Applications of Conceptual Spaces*. Springer Science + Business Media, 2015.
- [455] Alice Zhang. Reducing False Positives in the Construction of Adjective Scales. In *7th Global Wordnet Conference*, pages 372–377, 2014.

- [456] Dengsheng Zhang and Guojun Lu. Review of Shape Representation and Description Techniques. *Pattern Recognition*, 37(1):1–19, 2004.
- [457] Xingyuan Zhang, Yaping Huang, Qi Zou, Yanting Pei, Runsheng Zhang, and Song Wang. A Hybrid Convolutional Neural Network for Sketch Recognition. *Pattern Recognition Letters*, 130:73–82, 2020.
- [458] Junbo Zhao, Michael Mathieu, and Yann LeCun. Energy-based Generative Adversarial Network. In *5th International Conference on Learning Representations*, 2017.
- [459] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. In *19th IEEE International Conference on Computer Vision*, pages 2223–2232, 2017.

LIST OF FIGURES

Figure 1.1	A conceptual space for the COLOR domain (based on [364], image license CC BY-SA 3.0) with dimensions HUE, SATURATION, and LIGHTNESS, and with conceptual regions for BLUE and SKY BLUE.	15
Figure 1.2	Simple spatial representation of the pet fish example.	19
Figure 1.3	(a) Conceptual spaces as an intermediate layer for translating between symbolic and subsymbolic representations. (b) Mental image of three people sitting at a small round table and a modified TO THE RIGHT OF relation.	25
Figure 1.4	Different superquadrics based on varying form factors ϵ_1 (rows, ranging from 0.2 to 1.5) and ϵ_2 (columns, ranging from 0.2 to 1.5). Figure taken from [97], reprinted with permission.	28
Figure 1.5	Illustration of the envisioned AI system.	41
Figure 1.6	Structure of an autoencoder.	47
Figure 2.1	(a) Euclidean betweenness for stimuli on integral dimensions. (b) Manhattan betweenness for stimuli on separable dimensions.	56
Figure 2.2	Betweenness under the Minkowski metric for different values of r	57
Figure 2.3	Illustration of convexity and star-shapedness as based on the Euclidean vs. the Manhattan metric.	58
Figure 2.4	(a) Intuitive way to define regions for the concepts of ADULT and CHILD in a conceptual space spanned by the dimensions AGE and HEIGHT. (b) Representation of these concepts as convex regions. (c) Representation of these concepts as star-shaped regions.	61
Figure 2.5	Illustration of a core. (a) Three cuboids C_1, C_2, C_3 with nonempty intersection. (b) Core S based on C_1, C_2, C_3 with central region P	69
Figure 2.6	(a) Crisp conceptual boundaries lead to abrupt changes in membership. (b) Fuzzy conceptual boundaries allow for smooth changes in membership.	70
Figure 2.7	(a) Two concepts \tilde{S} and \tilde{S}' in a one-dimensional conceptual space. (b) A concept \tilde{S} in a two-dimensional conceptual space along with two of its α -cuts.	72
Figure 2.8	Class diagram of our implementation.	74

Figure 2.9	Screenshot from the ConceptInspector tool illustrating the fruit concepts in our example space. The concepts are labeled as follows: PEAR (1), ORANGE (2), LEMON (3), GRANNY SMITH (4), APPLE (5), BANANA (6).	76
Figure 2.10	Screenshot from the ConceptInspector tool illustrating the properties in our example space. The properties are labeled as follows: RED (1), GREEN (2), BLUE (3), NONSWEET (4).	78
Figure 2.11	Illustration of Fiorini’s proposal for representing part-whole relations in conceptual spaces.	87
Figure 2.12	Screenshot of the Concept Inspector tool for our proposed encoding of the APPLE concept. There are correlations within the TASTE domain as well as between the TASTE and the COLOR domain. The intersection of APPLE (1) with GREEN (2) has a large overlap with SOUR (3), indicating that green apples tend to be sour.	90
Figure 2.13	Illustration of the APPLE concept encoded under the formalization of Adams and Raubal [3]. There are correlations within the taste domain, but not between the taste and the color domain.	91
Figure 2.14	Visualization of the overall conceptual space as product space of the domains from Figure 2.13. The intersection of APPLE (1) with GREEN (2) has only a small overlap with SOUR (3), highlighting that this representation cannot encode cross-domain correlations.	92
Figure 2.15	Three-dimensional illustration of the encoding of the APPLE concept using a modified version of the formalization by Adams and Raubal [3]. The overall concept is defined as a polytope that is star-shaped with respect to a prototypical point under the combined distance.	92
Figure 2.16	A polytope defined by a number of inequalities that is not star-shaped under the Manhattan metric.	92
Figure 3.1	(a) The intersection of two cores in not necessarily star-shaped or even connected. (b) $P_1 \cap P_2 \neq \emptyset$ is not a necessary condition for a star-shaped intersection result.	103
Figure 3.2	Finding the minimal star-shaped hull of two cuboids C_1 and C_2 . (a) A point p_A outside of the bounding cuboid is dominated by another point p_B on the border of the bounding cuboid. (b) Two optimal solutions p_C and p_D which minimize the resulting volume. (c) Using p_E , the midpoint of the cuboids’ centers.	105

Figure 3.3	(a) Two cores S_1 and S_2 . (b) Their standard set intersection $S = S_1 \cap S_2$ and the midpoint p of this intersection's cuboids. (c) Extended cuboids C'_1 and C'_2 , resulting in S' 106
Figure 3.4	(a) The intersection $\mu_{\tilde{S}_1 \cap \tilde{S}_2}(x) = \min(\mu_{\tilde{S}_1}(x), \mu_{\tilde{S}_2}(x))$ of two concepts \tilde{S}_1 and \tilde{S}_2 . (b) Our modified intersection $I(\tilde{S}_1, \tilde{S}_2)$ 108
Figure 3.5	Possible results of intersecting two fuzzy cuboids. 109
Figure 3.6	Average runtime of the intersection operation for single-cuboid concepts as a function of the number of dimensions in the conceptual space. 113
Figure 3.7	Average runtime of the intersection operation as a function of the number of cuboids per concept in a four-dimensional conceptual space. 114
Figure 3.8	Screenshot of the ConceptInspector tool, illustrating the intersection of APPLE (1) and PEAR (2) in the three-dimensional fruit space, resulting in the APPLE-PEAR (3) concept. 115
Figure 3.9	(a) The union $\mu_{\tilde{S}_1 \cup \tilde{S}_2}(x) = \max(\mu_{\tilde{S}_1}(x), \mu_{\tilde{S}_2}(x))$ of two concepts \tilde{S}_1 and \tilde{S}_2 . (b) Our modified union $U(\tilde{S}_1, \tilde{S}_2)$ 117
Figure 3.10	Screenshot of the ConceptInspector tool, illustrating the concepts of APPLE (1) and PEAR (2) in the three-dimensional fruit space. 119
Figure 3.11	Screenshot of the ConceptInspector tool, illustrating the union of APPLE and PEAR in the three-dimensional fruit space. 119
Figure 3.12	Screenshot of the ConceptInspector tool, illustrating the concepts of ORANGE (1), LEMON (2), and APPLE (3) in the three-dimensional fruit space. 121
Figure 3.13	Screenshot of the ConceptInspector tool, illustrating the CITRUS FRUIT concept (1) (defined as the union of ORANGE and LEMON) and the APPLE concept (2). 121
Figure 3.14	(a) Set complement of a crisp conceptual region S in a two-dimensional space. (b) Set complement of a fuzzy conceptual region \tilde{S} in a one-dimensional space. 122
Figure 3.15	The set complement yields valid conceptual regions for half-spaces. (a) Convex half-space defined by a hyperplane. (b) Star-shaped half-space defined by half-open cuboids. 123
Figure 3.16	Screenshot of the ConceptInspector tool, illustrating the BANANA concept in the example fruit space. 128

Figure 3.17	Screenshot of the ConceptInspector tool, illustrating the projections of the BANANA concept onto the sets of domains $\Delta_1 = \{\delta_{\text{COLOR}}, \delta_{\text{TASTE}}\}$ (1) and $\Delta_2 = \delta_{\text{SHAPE}}$ (2).	129
Figure 3.18	Screenshot of the ConceptInspector tool, illustrating the intersection of the two projections of the BANANA concept.	130
Figure 3.19	Screenshot of the ConceptInspector tool, illustrating the APPLE concept in the example fruit space.	133
Figure 3.20	Screenshot of the ConceptInspector tool, illustrating the two resulting sub-concepts \tilde{S}^- (1) and \tilde{S}^+ (2) after applying the cut operation on the HUE dimension.	134
Figure 3.21	Screenshot of the ConceptInspector tool, illustrating the original concepts BANANA (1), GREEN (2), and BLUE (3) used for the concept combination example.	139
Figure 3.22	Screenshot of the ConceptInspector tool, illustrating the original BANANA concept (1) and the combined concepts GREEN BANANA (2) and BLUE BANANA (3).	140
Figure 4.1	α -cut of a fuzzified cuboid under the Euclidean distance (a) and the Manhattan distance (b), respectively.	150
Figure 4.2	(a) A two-dimensional hyperball (i.e., a disk) under the Euclidean metric (green, solid) and under the Manhattan metric (blue, dotted). (b) Stretching the disks along a single dimension results in ellipses.	152
Figure 4.3	(a) Illustration of a three-dimensional fuzzified cuboid's α -cut in a space consisting of two domains. (b) A hyperball in a three-dimensional space with two domains has the shape of a double cone.	154
Figure 4.4	Average runtime of the size operation for single-cuboid concepts as a function of the number of dimensions in the conceptual space.	158
Figure 4.5	Average runtime of the size operation as a function of the number of cuboids per concept in a four-dimensional conceptual space.	158
Figure 4.6	Illustrating the degree of subsethood with crisp sets in a two-dimensional space (a) and fuzzy concepts in a one-dimensional space (b).	163

Figure 4.7	Two problematic cases for Definition 4.6. (a) Perfect subsethood with different sensitivity parameters c results in $Sub(\tilde{S}_1, \tilde{S}_2) > 1$. (b) Overextension in the modified intersection causes $Sub(\tilde{S}_1, \tilde{S}_2) > 1$	164
Figure 4.8	Screenshot of the ConceptInspector tool, illustrating of the concepts and properties for the implication example: LEMON (1), APPLE (2), RED (3), GREEN (4), NONSWEET (5).	170
Figure 4.9	Two fuzzy sets \tilde{S} and \tilde{S}' with identical cores, weights, and sensitivity parameter, for which similarity functions based on the distance of cores violate the first criterion.	178
Figure 4.10	Screenshot of the ConceptInspector tool, showing all concepts used in our similarity computations. Concepts are labeled as follows: PEAR (1), ORANGE (2), LEMON (3), GRANNY SMITH (4), APPLE (5), and BANANA (6).	181
Figure 4.11	Scatter plot of similarity values obtained with Sim_S and Sim_J for 1,000 randomly generated pairs of concepts in a four-dimensional conceptual space.	183
Figure 4.12	(a) Illustration of Btw_1 as proposed by Derrac and Schockaert [122]. (b) Problematic case for Btw_1 as motivation for Btw_2	186
Figure 4.13	Two problematic cases for $B_{soft}^{inf}(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3)$ based on μ_0 (a), and c and W (b), respectively.	189
Figure 4.14	Average runtime of the betweenness operation for single-cuboid concepts as a function of the number of dimensions in the conceptual space.	191
Figure 4.15	Average runtime of the betweenness operation as a function of the number of cuboids used to represent a concept in a two-dimensional space.	192
Figure 4.16	Average runtime of the betweenness operation as a function of the number of α -cuts used in approximating the integral, using single-cuboid concepts in a two-dimensional space.	193
Figure 4.17	Scatter plot comparing the betweenness values obtained via B_{soft}^{inf} and B_{soft}^{int} (using 100 α -cuts) for 1,000 randomly generated triples of two-cuboid concepts in a four-dimensional conceptual space.	195
Figure 4.18	Scatter plot comparing the betweenness values obtained via B_{soft}^{int} for 1,000 randomly generated triples of two-cuboid concepts in a four-dimensional conceptual space, using 20 and 100 α -cuts, respectively.	196

Figure 4.19	Screenshot of the ConceptInspector tool illustrating all fruit concepts used in our betweenness example. Concepts are labeled as follows: PEAR (1), ORANGE (2), LEMON (3), GRANNY SMITH (4), APPLE (5), and BANANA (6).	197
Figure 5.1	Illustration of the curse of dimensionality for three values per feature and one (a), two (b), and three (c) dimensions in the feature space, respectively.	227
Figure 5.2	Comparison of lasso regression (a) and ridge regression (b) with MSE loss illustrated by solid red lines and regularization term illustrated by dashed blue lines.	231
Figure 5.3	Interpreting the weight vector of a linear model as direction in a one-dimensional feature space.	232
Figure 5.4	Interpreting the weight vector of a linear model as direction in a two-dimensional feature space.	233
Figure 5.5	(a) The sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$. (b) Illustration of gradient descent for a function \hat{f}	234
Figure 5.6	Illustration of the computation graph for a logistic regression and the partial derivatives involved in computing the gradient of the loss with respect to an individual weight.	236
Figure 5.7	(a) Non-optimal decision boundaries in a two-dimensional feature space. (b) Maximum margin separating hyperplane.	239
Figure 5.8	Computing the margin size s based on two support vectors \vec{x}^+ and \vec{x}^- and the weight vector \vec{w}	240
Figure 5.9	The XOR problem is not linearly separable.	244
Figure 5.10	(a) Example conceptual space, where PEAR is not linearly separable from the union of all other concepts. (b) Convex polytope of the PEAR concept based on all-vs-all SVMs. (c) Voronoi-like partitioning of the space is prohibited by unassigned regions.	249
Figure 5.11	Illustration of an example decision tree for deciding whether to take a walk.	250
Figure 5.12	Identifying domains as bicliques in the feature selection approach by Banaee et al. [26, 27].	263
Figure 5.13	Illustration for finding the vector \vec{w}^* by maximizing the variance of $X\vec{w}$ for a two-dimensional feature space.	267
Figure 6.1	Illustration of a multilayer feedforward network. (a) Individual hidden unit. (b) Network of units arranged in layers. (c) Layer-based visualization.	293

Figure 6.2	Different activation functions commonly used in artificial neural networks: linear (a), sigmoid (b), hyperbolic tangent (c), ReLU (d), and leaky ReLU (e).	296
Figure 6.3	Illustration of a one-dimensional discrete convolution for smoothing a noisy sensor at two different time indices t	299
Figure 6.4	Illustration of a two-dimensional convolution with a 3×3 kernel detecting horizontally oriented edges.	300
Figure 6.5	Illustrating the difference between a fully connected layer (a) and a convolutional layer (b).	301
Figure 6.6	Original inception module as described in [399].	303
Figure 6.7	(a) Convolution followed by max pooling. (b) Unpooling followed by convolution.	304
Figure 6.8	Illustration of backpropagation for a small example network.	310
Figure 6.9	Visualization of the general autoencoder architecture.	320
Figure 6.10	Illustration of the shallow network structures used by the CBOW model (a) and the Skip-gram model (b).	325
Figure 6.11	Structure of generative adversarial networks [173] with generator and discriminator.	332
Figure 6.12	Structure of InfoGAN [101], which adds a reconstruction objective for part of the noise vector to the general GAN framework.	336
Figure 6.13	Graphical model underlying variational autoencoders.	338
Figure 6.14	Implementation of variational autoencoders with neural networks for $\vec{x} \in \mathbb{R}^n$	340
Figure 6.15	Combination of two variational autoencoders and a denoising autoencoder as proposed by Higgins et al. [197].	351
Figure 7.1	(a) Learning prototypes in a conceptual space based on few examples. (b) Incremental update of one prototype and the resulting change in the Voronoi tessellation.	359
Figure 7.2	(a) Collated Voronoi diagram of a two-dimensional conceptual space. (b) Membership function based on a collated Voronoi diagram in a one-dimensional space.	363
Figure 7.3	Illustration of learning as a search through the hypothesis space. (a) Approximation and estimation error. (b) Improvement through data augmentation. (c) Improvement by reducing the hypothesis space. (d) Improvement through initialization.	372

Figure 7.4	Grounding of a formula containing a function symbol (<code>HOME_TOWN_OF</code>), a predicate symbol (<code>LIVES_IN</code>), and a constant symbol (<code>BOB</code>).	379
Figure 7.5	Different partitional membership functions in a one-dimensional conceptual space.	387
Figure 7.6	Unintended results for the soft Voronoi tessellation and different sensitivity parameters $c_1 \gg c_2$	388
Figure 7.7	Non-partitional membership functions in a one-dimensional conceptual space.	390
Figure 7.8	Workaround for zero gradient inside prototypical regions.	391
Figure 7.9	Difference between agglomerative hierarchical clustering using the single link method (a) and the complete link method (b).	395
Figure 7.10	Illustration of the k -means algorithm: Random initialization of cluster centroids (a), expectation step (b), and maximization step (c).	397
Figure 7.11	Illustration of the COBWEB operators. Splitting and merging can be used as approximately inverse operators (transition from (b) to (c) and backwards), while insertion increases the breadth of the tree.	403
Figure 7.12	Illustration of the overall ART system. Excitatory connections are marked with "+", inhibitory connections with "-".	407
Figure 7.13	Illustration of the overall SUSTAIN model with its three layers.	411
Figure 7.14	Illustration of the semiotic cycle.	418
Figure 8.1	The initial configuration X_0 for our fruit example.	458
Figure 8.2	(a) Initial configuration from Section 8.2.4. (b) Two-dimensional similarity space obtained by classical MDS for our fruit example.	468
Figure 8.3	Illustration of gradient descent for a function f	470
Figure 8.4	Illustration of initial configuration X_0 (a) and the update for Kruskal's algorithm (b).	479
Figure 8.5	Illustration of iterative majorization for a function f	481
Figure 8.6	Illustration of the update to the initial configuration X_0 for absolute (a) and ordinal (b) SMACOF.	489
Figure 8.7	Shepard diagram illustrating the relation between dissimilarities and distances (based on the study presented in Chapter 10).	492
Figure 8.8	Scree plot illustrating Stress as a function of n (based on the study presented in Chapter 10).	493
Figure 8.9	(a) Non-optimal decision boundaries in a two-dimensional feature space. (b) Maximum margin separating hyperplane.	495
Figure 8.10	Weight vector of a linear regression as direction in a two-dimensional feature space.	496

Figure 8.11	Illustration of the proposed hybrid procedure.	499
Figure 8.12	Potential network architectures for our proposed hybrid approach. Constraints are shown as dashed gray lines. (a) A standard feedforward network with two separate outputs for coordinates and class. (b) A standard feedforward network using the coordinates as intermediate step for predicting the class. (c) An autoencoder with a constrained bottleneck layer. (d) An autoencoder with an unconstrained bottleneck layer and a separate output for the coordinates. (e) An InfoGAN network [101] with an additional reconstruction constraint for labeled images.	502
Figure 9.1	Illustration of our analysis setup, which considers distances based on raw pixels (left), activations of an artificial neural network (middle), and similarity spaces obtained with MDS (right).	519
Figure 9.2	Scree plots showing the remaining metric (a) and nonmetric (b) Stress, respectively, as a function of the number of dimensions for the four MDS algorithms under consideration.	523
Figure 9.3	Pearson's r for pixel-based distances obtained through the Manhattan metric (using uniform weights) with respect to different aggregators and different block sizes.	525
Figure 9.4	Pearson's r for pixel-based distances obtained through the Euclidean metric (using optimized weights) with respect to different aggregators and different block sizes.	525
Figure 9.5	Correlation analysis for the different MDS solutions and the baselines, visualized as function of the number of dimensions.	527
Figure 9.6	Shepard diagram for the two-dimensional MDS space, based on nonmetric SMACOF and the Euclidean distance.	528
Figure 9.7	Shepard diagram for the ANN baseline, based on the Euclidean distance and optimized weights).	529
Figure 9.8	Two-dimensional similarity space based on classical MDS.	530
Figure 9.9	Two-dimensional similarity space based on Kruskal's algorithm.	531
Figure 9.10	Two-dimensional similarity space based on metric SMACOF.	531
Figure 9.11	Two-dimensional similarity space based on nonmetric SMACOF.	532
Figure 9.12	Illustration of the network structure used in our experiments.	535
Figure 9.13	Illustration of selected regression results from Table 9.5, considering only correct targets.	543

Figure 9.14	Illustration of the effect of correct vs. shuffled mappings on the ANN-based feature space from Table 9.5.	544
Figure 9.15	Visualization of the effects of regularization based on Table 9.6.	545
Figure 9.16	Visualization of the unregularized regression results for the different four-dimensional target spaces.	549
Figure 9.17	Visualization of the regularization effects on the ANN-based feature space for the different four-dimensional target spaces, based on Table 9.10.	552
Figure 9.18	Visualization of the regression results for MSE, MED, and R^2 as a function of the number of dimensions.	555
Figure 9.19	Visualization of the two-dimensional target similarity space based on nonmetric SMACOF, and the performance of the individual regressors as measured with the MED.	556
Figure 10.1	Part-whole relations in the proposal by Marr and Sishihara. Figure taken from [278], reprinted with permission.	572
Figure 10.2	Conceptual hierarchies in the proposal by Marr and Sishihara. Figure taken from [278], reprinted with permission.	573
Figure 10.3	Different superquadrics based on varying form factors ϵ_1 (rows, ranging from 0.2 to 1.5) and ϵ_2 (columns, ranging from 0.2 to 1.5). Figure taken from [97], reprinted with permission.	574
Figure 10.4	Example line drawings (PARROT and TELEVISION) similar to the ones used in our study.	592
Figure 10.5	Trial structure for eliciting attentive feature ratings.	596
Figure 10.6	Trial structure in the masked recognition experiments for eliciting pre-attentive feature ratings.	599
Figure 10.7	Comparison of the visual and conceptual similarity ratings. (a) Distribution of the ratings. (b) Boxplot of the ratings as a function of the type of image pair.	608
Figure 10.8	Scatter plot for the entries of the global similarity matrices based on conceptual and visual similarity ratings.	611
Figure 10.9	Item-based similarity matrices for visual (above the diagonal) and conceptual (below the diagonal) similarity ratings.	612
Figure 10.10	(a) Category-based similarity matrix for visual similarity. (b) Category-based similarity matrix for conceptual similarity.	613
Figure 10.11	Aggregated visual similarity for item pairs from the categories VEGETABLES and FRUIT.	614

Figure 10.12	Histograms for both attentive and pre-attentive ratings with respect to FORM, LINES, and ORIENTATION.	616
Figure 10.13	Scatter plot for the features FORM and LINES based on the attentive feature ratings.	617
Figure 10.14	Scatter plot comparing pre-attentive and attentive ratings for the ORIENTATION feature.	619
Figure 11.1	Scatter plot for the entries of the mean and median matrix.	626
Figure 11.2	Scree plots showing the remaining nonmetric Stress as a function of the number of dimensions. (a) Comparison of mean and median matrices. (b) Comparison of shapes data and NOUN data.	627
Figure 11.3	Two-dimensional similarity space based on the mean matrix.	628
Figure 11.4	Two-dimensional similarity space based on the median matrix.	629
Figure 11.5	Illustration of our correlation analysis setup, which considers distances based on the pixels of downscaled images (far left), activations of an artificial neural network (middle left), values of psychological shape features (middle right), and similarity spaces obtained with MDS (far right).	632
Figure 11.6	Optimal performance obtainable by the three baselines for both the mean matrix and the median matrix, using both uniform and optimized weights.	633
Figure 11.7	Example illustration for the inner product.	634
Figure 11.8	Kendall's τ between the pixel baseline and the mean matrix as a function of block size and aggregator function, using the Euclidean distance function with uniform weights.	637
Figure 11.9	Kendall's τ between the pixel baseline and the mean matrix as a function of block size and aggregator function, using the Euclidean distance function with optimized weights.	637
Figure 11.10	Shepard diagrams with respect to the mean dissimilarity matrix for the ANN baseline with optimized weights (a), the pixel baseline with optimized weights (b), and the feature baseline with uniform weights, based on pre-attentive (c) and attentive (d) feature ratings, respectively.	639
Figure 11.11	Shepard diagrams with respect to the median dissimilarity matrix for the ANN baseline with optimized weights (a), the pixel baseline with optimized weights (b), and the feature baseline with uniform weights, based on pre-attentive (c) and attentive (d) feature ratings, respectively.	640

Figure 11.12	Correlation of the different similarity spaces and baselines to the mean matrix (a) and to the median matrix (b).	641
Figure 11.13	Shepard diagrams for the one-dimensional and the five-dimensional similarity spaces based on the mean matrix (a) and the median matrix (b).	644
Figure 11.14	Overlap between conceptual regions. (a) Overall number of violations across all categories in comparison to three baselines. (b) Number of violations by category type.	647
Figure 11.15	Size of conceptual categories, divided into average size of all categories (a), and average size by category type (b).	649
Figure 11.16	Convex hulls of categories in the two-dimensional similarity space based on the mean matrix.	650
Figure 11.17	Convex hulls of categories in the two-dimensional similarity space based on the median matrix.	651
Figure 11.18	Average quality of extracted directions for random configurations and similarity spaces as measured by Cohen's κ (a) and Spearman's ρ (b).	654
Figure 11.19	Relative average quality of extracted directions for pre-attentive and attentive feature ratings as measured by Cohen's κ (a) and Spearman's ρ (b).	655
Figure 11.20	Relative average quality of extracted directions for a linear support vector machine (SVM) and a linear regression (LinReg) as measured by Cohen's κ (a) and Spearman's ρ (b).	656
Figure 11.21	Average quality of the extracted directions for the individual features as measured by Cohen's κ (a) and Spearman's ρ (b) for the mean spaces.	657
Figure 11.22	Average quality of the extracted directions for the individual features as measured by Cohen's κ (a) and Spearman's ρ (b) for the median spaces.	658
Figure 11.23	Average Cosine similarity of extracted directions for pairs of features based on the mean (a) and the median spaces (b).	660
Figure 11.24	Two-dimensional similarity space based on the mean matrix, along with interpretable directions.	661
Figure 11.25	Two-dimensional similarity space based on the median matrix, along with interpretable directions.	662
Figure 12.1	Illustrations of a parrot: (a) Our dataset ⁸ . (b) TU Berlin [143]. (c) Sketchy [348]. (d) QuickDraw [186].	670
Figure 12.2	The overall structure of AlexNet [232] (a) and Sketch-a-Net [446] (b). Differences are highlighted in boldface. ⁹	672
Figure 12.3	Illustration of our overall network architecture. ¹⁰	677

Figure 12.4	(a) Sketch-a-Net architecture [446]. (b) The decoder architecture used by Dosovitskiy and Brox [134] for AlexNet. (c) Our overall encoder-decoder structure (changes highlighted in bold-face). ¹¹	679
Figure 12.5	Reconstruction examples of the PARROT stimulus. (a) Original image ¹² and reconstruction with the R_{DEFAULT} and R_{BEST} configurations. (b) Reconstruction with the R_{BEST} configuration for different noise patterns applied to the input. (c) Reconstruction with the R_{BEST} configuration for different noise levels (0%, 10%, 25%, 55%).	694
Figure 12.6	Test set results for the linear regression on $C_{\text{INCEPTION}}$ for both mean and median spaces of different dimensionality.	697
Figure 12.7	Results of our classification-based generalization experiment.	713
Figure 12.8	Results of our reconstruction-based generalization experiment.	715
Figure 12.9	Visualization of the two-dimensional target similarity space and the performance of the individual classification-based regressors as measured with the MED.	718
Figure 12.10	Visualization of the two-dimensional target similarity space and the performance of the individual reconstruction-based regressors as measured with the MED.	718
Figure 13.1	Reconsideration of our CHILD example from Section 2.2.1: (a) Our solution with a single star-shaped region. (b) Strößner’s [395] solution with multiple convex sub-regions.	731
Figure 13.2	Cone-based representation as proposed by Özçep et al. [312]. (a) Convex cone X and its polar cone X° . (b) Axis-aligned cones $X = \mathbb{R}^+ \times \mathbb{R}^-$ and $Y = \mathbb{R} \times \{0\}$	733
Figure 13.3	(a) Graphical model of a variational autoencoder [224]. (b) Graphical model of a conceptual VAE [363].	747
Figure B.1	Cut above the central region. (a) Original core $S = \langle \Delta_S, \{C_1, C_2, C_3\} \rangle$. (b) Resulting cores $S^- = \langle \Delta_S, \{C_1^{(-)}, C_2^{(-)}, C_3^{(-)}\} \rangle$ and $S^+ = \langle \Delta_S, \{C_1^{(+)}, C_2^{(+)}\} \rangle$	842
Figure C.1	Illustration of distance computations with respect to d^*	862
Figure D.1	Pearson’s r for pixel-based distances obtained through the Euclidean metric (using uniform weights) with respect to different aggregators and different block sizes.	867

Figure D.2	Pearson's r for pixel-based distances obtained through the negated inner product (using uniform weights) with respect to different aggregators and different block sizes.	868
Figure D.3	Spearman's ρ for pixel-based distances obtained through the Euclidean metric (using uniform weights) with respect to different aggregators and different block sizes.	868
Figure D.4	Spearman's ρ for pixel-based distances obtained through the Manhattan metric (using uniform weights) with respect to different aggregators and different block sizes.	869
Figure D.5	Spearman's ρ for pixel-based distances obtained through the negated inner product (using uniform weights) with respect to different aggregators and different block sizes.	869
Figure D.6	Kendall's τ for pixel-based distances obtained through the Euclidean metric (using uniform weights) with respect to different aggregators and different block sizes.	870
Figure D.7	Kendall's τ for pixel-based distances obtained through the Manhattan metric (using uniform weights) with respect to different aggregators and different block sizes.	870
Figure D.8	Kendall's τ for pixel-based distances obtained through the negated inner product (using uniform weights) with respect to different aggregators and different block sizes.	871
Figure D.9	Coefficient of determination R^2 of a monotone regression for pixel-based distances obtained through the Euclidean metric (using uniform weights) with respect to different aggregators and different block sizes.	871
Figure D.10	Coefficient of determination R^2 of a monotone regression for pixel-based distances obtained through the Manhattan metric (using uniform weights) with respect to different aggregators and different block sizes.	872
Figure D.11	Coefficient of determination R^2 of a monotone regression for pixel-based distances obtained through the negated inner product (using uniform weights) with respect to different aggregators and different block sizes.	872
Figure D.12	Pearson's r for pixel-based distances obtained through the Manhattan metric (using optimized weights) with respect to different aggregators and different block sizes.	873

Figure D.13	Pearson's r for pixel-based distances obtained through the negated inner product (using optimized weights) with respect to different aggregators and different block sizes.	873
Figure D.14	Spearman's ρ for pixel-based distances obtained through the Euclidean metric (using optimized weights) with respect to different aggregators and different block sizes.	874
Figure D.15	Spearman's ρ for pixel-based distances obtained through the Manhattan metric (using optimized weights) with respect to different aggregators and different block sizes.	874
Figure D.16	Spearman's ρ for pixel-based distances obtained through the negated inner product (using optimized weights) with respect to different aggregators and different block sizes.	875
Figure D.17	Kendall's τ for pixel-based distances obtained through the Euclidean metric (using optimized weights) with respect to different aggregators and different block sizes.	875
Figure D.18	Kendall's τ for pixel-based distances obtained through the Manhattan metric (using optimized weights) with respect to different aggregators and different block sizes.	876
Figure D.19	Kendall's τ for pixel-based distances obtained through the negated inner product (using optimized weights) with respect to different aggregators and different block sizes.	876
Figure D.20	Coefficient of determination R^2 of a monotone regression for pixel-based distances obtained through the Euclidean metric (using optimized weights) with respect to different aggregators and different block sizes.	877
Figure D.21	Coefficient of determination R^2 of a monotone regression for pixel-based distances obtained through the Manhattan metric (using optimized weights) with respect to different aggregators and different block sizes.	877
Figure D.22	Coefficient of determination R^2 of a monotone regression for pixel-based distances obtained through the negated inner product (using optimized weights) with respect to different aggregators and different block sizes.	878
Figure E.1	Scatter plot for the features FORM and LINES based on the pre-attentive feature scales.	889
Figure E.2	Scatter plot for the features FORM and ORIENTATION based on the pre-attentive feature scales.	890
Figure E.3	Scatter plot for the features FORM and ORIENTATION based on the attentive feature scales.	890

Figure E.4	Scatter plot for the features <code>LINES</code> and <code>ORIENTATION</code> based on the pre-attentive feature scales.	891
Figure E.5	Scatter plot for the features <code>LINES</code> and <code>ORIENTATION</code> based on the attentive feature scales.	891
Figure E.6	Scatter plot comparing pre-attentive and attentive ratings for the <code>FORM</code> feature.	892
Figure E.7	Scatter plot comparing pre-attentive and attentive ratings for the <code>LINES</code> feature.	893
Figure F.1	Kendall's τ between the pixel baseline and the mean matrix as a function of block size and aggregator function, using the Manhattan distance function with uniform weights.	895
Figure F.2	Kendall's τ between the pixel baseline and the mean matrix as a function of block size and aggregator function, using the Manhattan distance function with optimized weights.	896
Figure F.3	Kendall's τ between the pixel baseline and the mean matrix as a function of block size and aggregator function, using the negated inner product with uniform weights as distance function.	896
Figure F.4	Kendall's τ between the pixel baseline and the mean matrix as a function of block size and aggregator function, using the negated inner product with optimized weights as distance function.	897
Figure F.5	Kendall's τ between the pixel baseline and the median matrix as a function of block size and aggregator function, using the Euclidean distance function with uniform weights.	897
Figure F.6	Kendall's τ between the pixel baseline and the median matrix as a function of block size and aggregator function, using the Euclidean distance function with optimized weights.	898
Figure F.7	Kendall's τ between the pixel baseline and the median matrix as a function of block size and aggregator function, using the Manhattan distance function with uniform weights.	898
Figure F.8	Kendall's τ between the pixel baseline and the median matrix as a function of block size and aggregator function, using the Manhattan distance function with optimized weights.	899
Figure F.9	Kendall's τ between the pixel baseline and the median matrix as a function of block size and aggregator function, using the negated inner product with uniform weights as distance function.	899

Figure F.10 Kendall’s τ between the pixel baseline and the median matrix as a function of block size and aggregator function, using the negated inner product with optimized weights as distance function. 900

LIST OF TABLES

Table 1.1	Overview of empirical observations about concept learning and concept usage, and their relation with different psychological theories ("✓" means "predicted", "✗" means "conflicting", and "-" means "neutral").	6
Table 1.2	Overview of empirical observations about concept learning and concept usage, and their relation with different psychological theories, including the conceptual spaces framework ("✓" means "predicted", "✗" means "conflicting", and "-" means "neutral").	18
Table 1.3	Overview of four applications of conceptual spaces in the area of artificial intelligence.	39
Table 2.1	Definitions of several fruit concepts and properties.	77
Table 2.2	Overview of different formalizations of the conceptual spaces framework based on our list of criteria ("✓" means "fulfilled" and "-" means "not fulfilled").	83
Table 2.3	Definitions of conceptual regions of several fruit concepts and properties for comparing different formalizations.	89
Table 2.4	Definitions of domain and dimension weights of several fruit concepts and properties for comparing different formalizations.	89
Table 3.1	Average runtime of the intersection operation (averaged across 1,000 pairs of randomly created single-cuboid concepts).	112
Table 3.2	Average runtime of the intersection operation (averaged across 1,000 pairs of randomly created concepts in a four-dimensional conceptual space).	113
Table 3.3	Definition of the concepts APPLE and PEAR for the intersection example.	114
Table 3.4	Definition of the concepts APPLE and PEAR for the union example.	118
Table 3.5	Definition of the BANANA concept, its projections onto $\Delta_1 = \{\delta_{\text{COLOR}}, \delta_{\text{TASTE}}\}$ and $\Delta_2 = \{\delta_{\text{SHAPE}}\}$ as well as the intersection of these projections.	127
Table 3.6	Definition of the APPLE concept as well as its two sub-concepts \tilde{S}^- and \tilde{S}^+ obtained by applying an axis-parallel cut on d_{HUE}	132
Table 3.7	Definitions of the BANANA concept and the properties GREEN and BLUE for the concept combination example.	138

Table 3.8	Overview of different formalizations of the conceptual spaces framework based on their supported operations on concepts ("✓" means "available", "-" means "not available", and "(−)" means "could be easily added").	142
Table 4.1	Average runtime of the size operation (averaged across 1,000 examples) in comparison to runtime of a numerical approximation (averaged across 100 examples) for randomly generated single-cuboid concepts.	159
Table 4.2	Average runtime of the size operation averaged across 1,000 randomly created concepts in a four-dimensional space.	159
Table 4.3	Definition of several fruit concepts for the size example.	160
Table 4.4	Sizes of all fruit concepts.	161
Table 4.5	Absolute number of cases with $Sub(\tilde{S}_1, \tilde{S}_2) > 1$ for 10,000 randomly generated pairs of concepts for different numbers of cuboids per concept and different dimensionalities of the conceptual space.	166
Table 4.6	Three concepts used to illustrate our subsethood computations.	167
Table 4.7	Definitions of several fruit concepts and properties used to illustrate the implication operation.	169
Table 4.8	Definitions of several fruit concepts for our exemplary similarity computations.	180
Table 4.9	Similarity values based on the degree of subsethood Sim_S for all fruit concepts, rounded to four decimal places.	180
Table 4.10	Similarity values based on the Jaccard index Sim_J for all fruit concepts, rounded to four decimal places.	181
Table 4.11	Average runtime of B_{soft}^{inf} and $B_{soft'}^{int}$ averaged across 100 randomly created triples of single-cuboid concepts, using an average over 20 α -cuts to approximate the integral.	191
Table 4.12	Average runtime of B_{soft}^{inf} and $B_{soft'}^{int}$ averaged across 100 randomly created triples of concepts in a two-dimensional conceptual space, using an average over 20 α -cuts to approximate the intergal.	192
Table 4.13	Average runtime of B_{soft}^{inf} and $B_{soft'}^{int}$ averaged across 100 randomly created triples of single-cuboid concepts in a two-dimensional conceptual space.	193
Table 4.14	Definitions of several fruit concepts for our exemplary betweenness computations.	198

Table 4.15	Betweenness values for selected triples of fruit concepts, rounded to four decimal places. For B_{soft}^{int} , 20 α -cuts were used.	198
Table 4.16	Overview of different formalizations of the conceptual spaces framework based on the relations between concepts they define ("✓" means "available", "-" means "not available", and "(-)" means "could be easily added").	203
Table 5.1	Confusion matrix for a binary classification task	216
Table 8.1	Classification of psychological methods for eliciting dissimilarities.	436
Table 8.2	Example dissimilarities for four fruit stimuli. .	458
Table 8.3	Stress values for the initial configuration from Section 8.2.4 and for the solution provided by classical MDS.	468
Table 8.4	Stress values for the initial configuration and the first update provided by Kruskal's algorithm.	479
Table 8.5	Stress values for the initial configuration and the first update provided by absolute and ordinal SMACOF, respectively.	489
Table 9.1	Overview of the stimuli in the NOUN dataset [199].	515
Table 9.2	Performance comparison of pixel baseline and ANN baseline for uniform and optimized weights.	524
Table 9.3	Silhouette coefficient for the different feature spaces. Values achieved on a shuffled assignment of images to feature vectors are shown in parentheses for comparison. Best values for each of the feature spaces are highlighted in boldface.	540
Table 9.4	Performance of different baselines on the four-dimensional space by Horst and Hout [199]. Best values in each column are highlighted in boldface.	541
Table 9.5	Performance of different regressors for different feature spaces and correct vs. shuffled targets, with respect to the four-dimensional target space by Horst and Hout [199]. The best results for each combination of column and feature space are highlighted in boldface.	543
Table 9.6	Performance of the lasso regressor for different values of β with respect to the four-dimensional target space by Horst and Hout [199]. The best results for each combination of column and feature space are highlighted in boldface. . . .	546
Table 9.7	Performance of the zero baseline on the four-dimensional target spaces derived with different MDS algorithms.	548

Table 9.8	Test set performance of the linear regression on ANN-based features with respect to the different four-dimensional target similarity spaces, along with the relative degree of overfitting. Best results in each of the columns are highlighted in boldface.	549
Table 9.9	Test set performance of the random forest regression on the pixel-based features with respect to the different four-dimensional target similarity spaces, along with the relative degree of overfitting. Best results in each of the columns are highlighted in boldface.	550
Table 9.10	Test set performance of the lasso regression on the ANN-based features with respect to the different four-dimensional target spaces. Best results in each column are highlighted in boldface.	552
Table 9.11	Performance of the zero baseline, the linear regression, and the lasso regression on the ANN-based features and with respect to target spaces of different dimensionality n derived with non-metric SMACOF, along with the relative degree of overfitting. Best values for each column are highlighted for each of the regressors.	554
Table 10.1	Overview of the 60 stimuli in our dataset, based on their categories.	592
Table 10.2	Examples for item pairs with different relations of visual and conceptual similarity.	611
Table 11.1	Kendall's τ for different variations of the feature baseline.	633
Table 11.2	Kendall's τ for different variations of the ANN baseline.	635
Table 11.3	Kendall's τ for different variations of the pixel baseline.	636
Table 11.4	Average pairwise Cosine similarity of the different candidate directions for the three psychological features and similarity spaces of different dimensionality.	659
Table 12.1	Comparison of the three sketch datasets.	670
Table 12.2	Augmentation factors used for the individual data sources.	676
Table 12.3	Distribution of pixel intensities in the augmented dataset, split according to the data source.	681
Table 12.4	Creation of minibatches based on the four data sources for different training settings.	682

Table 12.5	Results of our initial sketch classification experiments using the default setup and variations with respect to a single hyperparameter. For each hyperparameter, the default setup is shown in italics, and the best values for each of the three evaluation criteria are shown in boldface. . . .	686
Table 12.6	Results of our grid search for the sketch classification network. The best values for each of the three evaluation criteria are highlighted in boldface.	688
Table 12.7	Results of our initial autoencoder experiments using the default setup and variations with respect to a single hyperparameter. For each hyperparameter, the default setup is shown in italics, and the best values for each of the three evaluation criteria are highlighted in boldface.	691
Table 12.8	Results of our grid search for the autoencoder. The best values for each of the two evaluation criteria are highlighted in boldface.	693
Table 12.9	Results for the zero baseline and a linear regression into the four-dimensional mean space, using different noise levels. Best results on the test set are highlighted in boldface.	696
Table 12.10	Results for the zero baseline and a linear regression into the four-dimensional median space, using different noise levels. Best results on the test set are highlighted in boldface.	696
Table 12.11	Test set results for the lasso regressor with respect to the four-dimensional target spaces, using 10% salt and pepper noise in during training and no noise during testing.	698
Table 12.12	Comparison between the results from our NOUN study (Section 9.2.4) and our results on the shapes data.	699
Table 12.13	Results of our transfer learning experiments based on the sketch classification network. The best values for each of the three evaluation criteria on the test set are highlighted in boldface.	700
Table 12.14	Dependence of the linear regression results on the noise regime during training and testing. Best test set results for each feature space are highlighted in boldface.	702
Table 12.15	Cluster analysis of the augmented images in the individual classification-based feature spaces using the Silhouette coefficient [339] (averaged across all folds, if applicable).	702

Table 12.16	Results of our transfer learning experiments with the autoencoder (R_{DEFAULT} and R_{BEST}) in comparison to the best results from Sections 12.4.1 ($C_{\text{INCEPTION}}$) and 12.4.2 (C_{SMALL}). The best values for each of the three evaluation criteria on the test set are highlighted in boldface for each type of pretraining objective.	704
Table 12.17	Cluster analysis of the augmented images in both classification-based and reconstruction-based feature spaces (averaged across all folds, if applicable) using the Silhouette coefficient [339].	705
Table 12.18	Test set results of the multitask learning experiments for the sketch-based classification setup. Best values for each of the evaluation metrics are highlighted in boldface.	708
Table 12.19	Test set results of the multitask learning experiments for the sketch-based reconstruction setup. Best values of the reconstruction approach for each of the evaluation metrics are highlighted in boldface.	710
Table 12.20	Training time and classification accuracy of the multitask learner for target spaces of different dimensionality.	714
Table 12.21	Training time and reconstruction error of the multitask learner for target spaces of different dimensionality.	715
Table 12.22	Comparison of best mapping performance in the generalization experiments. The best value for each evaluation metric is highlighted in boldface.	716
Table D.1	Performance of different regressors for different feature spaces and correct vs. shuffled targets, with respect to the four-dimensional target space based on classical MDS. The best results for each combination of column and feature space are highlighted in boldface.	878
Table D.2	Performance of different regressors for different feature spaces and correct vs. shuffled targets, with respect to the four-dimensional target space based on Kruskal's MDS algorithm. The best results for each combination of column and feature space are highlighted in boldface.	879
Table D.3	Performance of different regressors for different feature spaces and correct vs. shuffled targets, with respect to the four-dimensional target space based on metric SMACOF. The best results for each combination of column and feature space are highlighted in boldface.	879

Table D.4	Performance of different regressors for different feature spaces and correct vs. shuffled targets, with respect to the four-dimensional target space based on nonmetric SMACOF. The best results for each combination of column and feature space are highlighted in boldface.	880
Table D.5	Performance of the lasso regressor for different values of β with respect to the four-dimensional target space based on classical MDS. The best results for each combination of column and feature space are highlighted in boldface.	880
Table D.6	Performance of the lasso regressor for different values of β with respect to the four-dimensional target space based on Kruskal's MDS algorithm. The best results for each combination of column and feature space are highlighted in boldface.	881
Table D.7	Performance of the lasso regressor for different values of β with respect to the four-dimensional target space based on metric SMACOF. The best results for each combination of column and feature space are highlighted in boldface.	881
Table D.8	Performance of the lasso regressor for different values of β with respect to the four-dimensional target space based on nonmetric SMACOF. The best results for each combination of column and feature space are highlighted in boldface.	882
Table D.9	Performance of the lasso regressor for different values of β with respect to the one-dimensional target space based on nonmetric SMACOF. The best results for each combination of column and feature space are highlighted in boldface.	883
Table D.10	Performance of the lasso regressor for different values of β with respect to the two-dimensional target space based on nonmetric SMACOF. The best results for each combination of column and feature space are highlighted in boldface.	884
Table D.11	Performance of the lasso regressor for different values of β with respect to the three-dimensional target space based on nonmetric SMACOF. The best results for each combination of column and feature space are highlighted in boldface.	884
Table D.12	Performance of the lasso regressor for different values of β with respect to the five-dimensional target space based on nonmetric SMACOF. The best results for each combination of column and feature space are highlighted in boldface.	885

Table D.13	Performance of the lasso regressor for different values of β with respect to the six-dimensional target space based on nonmetric SMACOF. The best results for each combination of column and feature space are highlighted in boldface. . . .	885
Table D.14	Performance of the lasso regressor for different values of β with respect to the seven-dimensional target space based on nonmetric SMACOF. The best results for each combination of column and feature space are highlighted in boldface. . . .	886
Table D.15	Performance of the lasso regressor for different values of β with respect to the eight-dimensional target space based on nonmetric SMACOF. The best results for each combination of column and feature space are highlighted in boldface. . . .	886
Table D.16	Performance of the lasso regressor for different values of β with respect to the nine-dimensional target space based on nonmetric SMACOF. The best results for each combination of column and feature space are highlighted in boldface. . . .	887
Table D.17	Performance of the lasso regressor for different values of β with respect to the ten-dimensional target space based on nonmetric SMACOF. The best results for each combination of column and feature space are highlighted in boldface. . . .	887
Table G.1	List of the 98 common categories shared by TU Berlin [143] and Sketchy [348].	902
Table G.2	List of the categories, which are contained in TU Berlin [143], but not in Sketchy [348].	903
Table G.3	List of the categories, which are contained in Sketchy [348], but not in TU Berlin [143].	904
Table G.4	Results for the lasso regressor based on C_{DEFAULT} with respect to the four-dimensional mean target space, using 10% salt and pepper noise in the inputs during training, and no noise during testing. Best test set results are highlighted in boldface.	904
Table G.5	Results for the lasso regressor based on C_{LARGE} with respect to the four-dimensional mean target space, using 10% salt and pepper noise in the inputs during training, and no noise during testing. Best test set results are highlighted in boldface.	905
Table G.6	Results for the lasso regressor based on $C_{\text{CORRELATION}}$ with respect to the four-dimensional mean target space, using 10% salt and pepper noise in the inputs during training, and no noise during testing. Best test set results are highlighted in boldface.	905

Table G.7	Results for the lasso regressor based on C_{SMALL} with respect to the four-dimensional mean target space, using 10% salt and pepper noise in the inputs during training, and no noise during testing. Best test set results are highlighted in boldface.	906
Table G.8	Results for the lasso regressor based on R_{DEFAULT} with respect to the four-dimensional mean target space, using 10% salt and pepper noise in the inputs during training, and no noise during testing. Best test set results are highlighted in boldface.	906
Table G.9	Results for the lasso regressor based on R_{BEST} with respect to the four-dimensional mean target space, using 10% salt and pepper noise in the inputs during training, and no noise during testing. Best test set results are highlighted in boldface.	907
Table G.10	Test set results for the multitask learner based on C_{DEFAULT} with respect to the four-dimensional mean target space, using 10% salt and pepper noise in the inputs during training, and no noise during testing. Best results for all evaluation metrics are highlighted in boldface.	907
Table G.11	Test set results for the multitask learner based on C_{SMALL} with respect to the four-dimensional mean target space, using 10% salt and pepper noise in the inputs during training, and no noise during testing. Best results for all evaluation metrics are highlighted in boldface.	908
Table G.12	Test set results for the multitask learner based on $C_{\text{CORRELATION}}$ with respect to the four-dimensional mean target space, using 10% salt and pepper noise in the inputs during training, and no noise during testing. Best results for all evaluation metrics are highlighted in boldface.	908
Table G.13	Test set results for the multitask learner based on R_{DEFAULT} with respect to the four-dimensional mean target space, using 10% salt and pepper noise in the inputs during training, and no noise during testing. Best results for all evaluation metrics are highlighted in boldface.	909
Table G.14	Test set results for the multitask learner based on R_{BEST} with respect to the four-dimensional mean target space, using 10% salt and pepper noise in the inputs during training, and no noise during testing. Best results for all evaluation metrics are highlighted in boldface.	909

Table G.15	Test set results for the zero baseline with respect to the mean target spaces of different dimensionality, using 10% salt and pepper noise in the inputs during training, and no noise during testing.	910
Table G.16	Test set results for the transfer learner with $\beta = 0.005$ based on the $C_{\text{INCEPTION}}$ feature space with respect to the mean target spaces of different dimensionality, using 10% salt and pepper noise in the inputs during training, and no noise during testing.	910
Table G.17	Test set results for the transfer learner with $\beta = 0.02$ based on the C_{SMALL} feature space with respect to the mean target spaces of different dimensionality, using 10% salt and pepper noise in the inputs during training, and no noise during testing.	911
Table G.18	Test set results for the multitask learner with $\lambda_3 = 0.0625$ based on the C_{DEFAULT} configuration with respect to the mean target spaces of different dimensionality, using 10% salt and pepper noise in the inputs during training, and no noise during testing.	911
Table G.19	Test set results for the transfer learner with $\beta = 0.02$ based on the R_{BEST} feature space with respect to the mean target spaces of different dimensionality, using 10% salt and pepper noise in the inputs during training, and no noise during testing.	912
Table G.20	Test set results for the multitask learner with $\lambda_3 = 0.0625$ based on the R_{BEST} configuration with respect to the mean target spaces of different dimensionality, using 10% salt and pepper noise in the inputs during training, and no noise during testing.	912

LIST OF ALGORITHMS

Algorithm 3.1	Finding the highest non-empty α -cut of two fuzzified cuboids.	110
Algorithm 5.1	General gradient descent procedure.	235
Algorithm 7.1	Control strategy of the COBWEB algorithm [159].	403
Algorithm 8.1	Classical MDS algorithm as described by Borg and Groenen [71, Chapter 12].	465
Algorithm 8.2	General gradient descent procedure.	469
Algorithm 8.3	Monotone regression as described in [234]. . .	472
Algorithm 8.4	Kruskal's MDS algorithm as described in [234].	476
Algorithm 8.5	General iterative majorization algorithm.	481
Algorithm 8.6	The SMACOF algorithm.	487

LIST OF DEFINITIONS,
 LEMMATA,
 PROPOSITIONS, AND
 COROLLARIES

2.1	Definition (Weighted Minkowski Metric)	55
2.2	Definition (Betweenness)	56
2.3	Definition (Convexity)	58
2.4	Definition (Star-Shapedness)	58
2.5	Definition (Criterion P)	59
2.6	Definition (Criterion C)	59
2.7	Definition (Combined Metric)	66
2.1	Lemma (Metric Properties of d_C^Δ)	67
2.2	Lemma (Union of Convex Sets)	67
2.8	Definition (Axis-Parallel Cuboid)	68
2.3	Lemma (Cuboids are Convex)	68
2.9	Definition (Core of a Conceptual Region)	68
2.10	Definition (Fuzzy Set)	70
2.11	Definition (α -Cut)	70
2.12	Definition (Quasi-Concavity)	71
2.13	Definition (Fuzzy Star-Shapedness)	71
2.14	Definition (Concept as Fuzzy Region)	71
2.4	Lemma (α -Cut is an ϵ -Neighborhood)	72
2.1	Proposition (Concepts are Fuzzy Star-Shaped)	72
3.1	Lemma (Intersection of Cores is Union of Cuboids)	103
3.2	Lemma (Sufficient Condition for Star-Shaped Intersection Result)	103
3.1	Corollary (Intersection of Orthogonal Cores)	103
3.1	Definition (Modified Intersection of Cores)	106
3.2	Definition (Modified Intersection of Concepts)	107
3.3	Definition (Fuzzy Intersection)	108
3.4	Definition (Fuzzified Cuboid)	109
3.5	Definition (Modified Union of Cores)	116
3.6	Definition (Modified Union of Concepts)	117
3.7	Definition (Fuzzy Union)	117
3.1	Proposition (Modified Union as Superset of Standard Union)	118
3.8	Definition (Projection of a Cuboid)	125
3.9	Definition (Projection of a Core)	125
3.2	Proposition (Intersection of Projections of Cores)	125
3.10	Definition (Projection of a Concept)	125
3.11	Definition (Fuzzy Projection)	126

3.3 Proposition (Modified Projection as a Subset of Standard Projection) 126

3.4 Proposition (Intersection of Projections of Concepts) 126

3.12 Definition (Axis-Parallel Cut of a Core) 131

3.5 Proposition (Cut of a Core Results in Cores) 131

3.13 Definition (Axis-Parallel Cut of a Concept) 131

4.1 Proposition (Volume of a Hyperball without Salience Weights) 154

4.2 Proposition (Volume of a Hyperball with Salience Weights) 155

4.3 Proposition (Size of an α -Cut) 155

4.1 Definition (Measure) 156

4.4 Proposition (Size of a Fuzzified Cuboid) 156

4.2 Definition (Size of a Concept) 156

4.3 Definition (Subsethood for Crisp Sets) 162

4.4 Definition (Subsethood for Fuzzy Sets) 162

4.5 Proposition (Conditions for Crisp Subsethood) 162

4.5 Definition (Degree of Subsethood for Fuzzy Sets) 163

4.6 Definition (Modified Degree of Subsethood for Concepts) 163

4.7 Definition (Degree of Implications for Concepts) 169

4.8 Definition (Similarity Function) 178

4.6 Proposition (Sim_S is a Similarity Function) 179

4.7 Proposition (Sim_J is a Similarity Function) 179

4.9 Definition (Infimum-Based Betweenness) 188

4.8 Proposition (Problematic Case for B_{soft}^{inf}) 189

4.10 Definition (Integral-Based Betweenness) 190

4.1 Lemma (B_{soft}^{int} is Bounded by B_{soft}^{inf}) 190

4.1 Corollary (Properties of B_{soft}^{int}) 190

5.1 Definition (Machine Learning) 212

13.1 Definition (Natural Multi-Domain Concept) 730

A.1 Lemma (Weighted Euclidean Distance) 825

C.1 Lemma (Simplification of an Integral over Angles) 843

C.2 Lemma (Simplification of an Integral over a Single Radius) 844

C.3 Lemma (Simplification of an Integral over Multiple Radii) 845

C.4 Lemma (Simplification of an Integral over an Exponentiated Logarithm) 853

Part V

APPENDIX

The following appendices provide additional supplementary information for Parts [I](#) and [III](#) of this dissertation. Appendices [A](#), [B](#), and [C](#) contain formal mathematical proofs for the lemmata and propositions from Chapters [2](#), [3](#), and [4](#), respectively. The appendices [E](#), [F](#), and [G](#) provide additional information and illustrations for our experimental studies from Chapters [10](#), [11](#) and [12](#).

A

DEFINING CONCEPTUAL REGIONS

In this appendix, we give proofs for the lemmata and propositions from Chapter 2, where we introduced our formalization of concepts as fuzzy star-shaped regions.

A.1 AN ARGUMENT AGAINST CONVEXITY

In the following lemma, we show that the weighted Euclidean distance of weighted Euclidean distances can be expressed as a single weighted Euclidean distance. This was used in Section 2.2.2 as an argument against replacing the Manhattan distance with the Euclidean distance in the conceptual spaces framework. Please recall that the weighted Euclidean metric is a special case of the weighted Minkowski metric for $r = 2$:

Euclidean distance of Euclidean distances is a Euclidean distance

Definition 2.1 (Weighted Minkowski Metric)

Let $x, y \in CS$. Their distance according to the weighted Minkowski metric of order $r > 0$ with weights w_i is defined as follows:

$$d_r(x, y) = \left(\sum_{i=1}^n w_i \cdot |x_i - y_i|^r \right)^{\frac{1}{r}}$$

Lemma A.1 (Weighted Euclidean Distance)

Let D be the set of dimensions d spanning a conceptual space CS . Let Δ be a partition of these dimensions into domains $\delta \subseteq D$. Let moreover $\forall \delta \in \Delta : d_\delta(x, y) = \sqrt{\sum_{d \in \delta} w_d \cdot |x_d - y_d|^2}$ be the weighted Euclidean distance of two points x, y within domain δ and let $d(x, y) = \sqrt{\sum_{\delta \in \Delta} w_\delta \cdot (d_\delta(x, y))^2}$ be the weighted Euclidean distance between x and y of these intra-domain distances. Then, we can write this overall distance as $d(x, y) = \sqrt{\sum_{d \in D} w'_d \cdot |x_d - y_d|^2}$.

Proof. We can prove this lemma by making the following straightforward transformations:

$$\begin{aligned} d(x, y) &= \sqrt{\sum_{\delta \in \Delta} w_\delta \cdot (d_\delta(x, y))^2} \\ &= \sqrt{\sum_{\delta \in \Delta} w_\delta \cdot \left(\sqrt{\sum_{d \in \delta} w_d \cdot |x_d - y_d|^2} \right)^2} \\ &= \sqrt{\sum_{\delta \in \Delta} w_\delta \cdot \sum_{d \in \delta} w_d \cdot |x_d - y_d|^2} \end{aligned}$$

$$= \sqrt{\sum_{\delta \in \Delta} \sum_{d \in \delta} w_{\delta} \cdot w_d \cdot |x_d - y_d|^2}$$

Now define $w'_d = w_{\delta} \cdot w_d$ and replace $\sum_{\delta \in \Delta} \sum_{d \in \delta}$ by $\sum_{d \in D}$ (the latter is possible since Δ is a partition of D). We can then write $d(x, y)$ as a weighted Euclidean distance:

$$d(x, y) = \sqrt{\sum_{d \in D} w'_d \cdot |x_d - y_d|^2} \quad \square$$

A.2 FORMALIZING CONCEPTUAL SIMILARITY SPACES

In Section 2.3.1, we introduced the combined metric d_C^{Δ} with $W = \langle W_{\Delta}, \{W_{\delta}\}_{\delta \in \Delta} \rangle$ containing the salience weights for both domains and dimensions. Lemma 2.1 states that d_C^{Δ} is a metric if W is kept fixed.

Definition 2.7 (Combined Metric)

Let CS be a conceptual space based on dimensions $d \in D$, which are partitioned into domains $D \supseteq \delta \in \Delta$. Let W_{Δ} be the set of positive domain weights w_{δ} for all $\delta \in \Delta$ with $\sum_{\delta \in \Delta} w_{\delta} = |\Delta|$. Let moreover W_{δ} be the set of positive domain weights w_d for all $d \in \delta$ with $\sum_{d \in \delta} w_d = 1$. Let furthermore $x, y \in CS$. Their distance according to the combined metric is defined as follows, where $W = \langle W_{\Delta}, \{W_{\delta}\}_{\delta \in \Delta} \rangle$:

$$d_C^{\Delta}(x, y, W) = \sum_{\delta \in \Delta} w_{\delta} \cdot d_E^{\delta}(x, y, W_{\delta}) = \sum_{\delta \in \Delta} \left(w_{\delta} \cdot \sqrt{\sum_{d \in \delta} w_d \cdot |x_d - y_d|^2} \right)$$

Lemma 2.1 (Metric Properties of d_C^{Δ})

d_C^{Δ} with a fixed parameter W is a metric.

Proof. Let $x, y, z \in CS$ be arbitrary, but fixed. As W is fixed, we write $d_C^{\Delta}(x, y)$ instead of $d_C^{\Delta}(x, y, W)$. We now show that d_C^{Δ} fulfills all four properties of a metric, i.e., non-negativity, identity of indiscernibles, symmetry, and the triangle inequality.

Non-negativity

(i) $d_C^{\Delta}(x, y) \geq 0$:

We know that $\forall d \in D : |x_d - y_d| \geq 0, w_d > 0$ and that $\forall \delta \in \Delta : w_{\delta} > 0$. Moreover, the operations of squaring and computing the square root always return nonnegative results. It therefore follows that $d_C^{\Delta}(x, y) \geq 0$.

Identity of
indiscernibles

(ii) $d_C^{\Delta}(x, y) = 0 \Leftrightarrow x = y$:

One can make the following transformations:

$$\begin{aligned} (x = y) &\Leftrightarrow (\forall d \in D : x_d = y_d) \Leftrightarrow (\forall d \in D : |x_d - y_d| = 0) \\ &\Leftrightarrow (d_C^{\Delta}(x, y) = 0) \end{aligned}$$

The last equivalence holds also in direction \Leftarrow , because all weights w_{δ} and w_d are positive.

(iii) $d_C^\Delta(x, y) = d_C^\Delta(y, x)$:

Symmetry

Since $|x_d - y_d| = |y_d - x_d|$, this property can be easily shown:

$$\begin{aligned} d_C^\Delta(x, y) &= \sum_{\delta \in \Delta} \left(w_\delta \cdot \sqrt{\sum_{d \in \delta} w_d \cdot |x_d - y_d|^2} \right) \\ &= \sum_{\delta \in \Delta} \left(w_\delta \cdot \sqrt{\sum_{d \in \delta} w_d \cdot |y_d - x_d|^2} \right) = d_C^\Delta(y, x) \end{aligned}$$

(iv) $d_C^\Delta(x, z) \leq d_C^\Delta(x, y) + d_C^\Delta(y, z)$:

Triangle inequality

$$\begin{aligned} d_C^\Delta(x, y) + d_C^\Delta(y, z) &= \sum_{\delta \in \Delta} (w_\delta \cdot d_E^\delta(x, y)) + \sum_{\delta \in \Delta} (w_\delta \cdot d_E^\delta(y, z)) \\ &= \sum_{\delta \in \Delta} w_\delta \cdot (d_E^\delta(x, y) + d_E^\delta(y, z)) \end{aligned}$$

Since d_E^δ is a metric, the following triangle inequality holds:

$$d_E^\delta(x, y) + d_E^\delta(y, z) \geq d_E^\delta(x, z)$$

Therefore, we can write:

$$\begin{aligned} d_C^\Delta(x, y) + d_C^\Delta(y, z) &= \sum_{\delta \in \Delta} w_\delta \cdot (d_E^\delta(x, y) + d_E^\delta(y, z)) \\ &\geq \sum_{\delta \in \Delta} w_\delta \cdot d_E^\delta(x, z) = d_C^\Delta(x, z) \end{aligned}$$

Since d_C^Δ fulfills all properties of a metric, d_C^Δ is a metric. \square

A.3 CRISP CONCEPTUAL REGIONS

In Section 2.3.2, we defined the core of a conceptual region as a union of convex regions with a non-empty intersection. This was based on Lemma 2.2, which relates the notions of convexity and star-shapedness with each other. Both of them are defined based on the betweenness of points $x, y, z \in CS$ under a metric d :

*Constructing
star-shaped regions
based on convex sets*

Definition 2.2 (Betweenness)

Let $x, y, z \in CS$ and d be a metric on CS . The point y is said to lie between x and z (denoted as $B_d(x, y, z)$) if and only if $d(x, y) + d(y, z) = d(x, z)$.

Definition 2.3 (Convexity)

A set $C \subseteq CS$ in a conceptual space CS is convex under a metric $d \Leftrightarrow$

$$\forall x \in C, z \in C, y \in CS : (B_d(x, y, z) \rightarrow y \in C)$$

Definition 2.4 (Star-Shapedness)

A set $S \in CS$ in a conceptual space CS is star-shaped under a metric d with respect to a set $P \subseteq S \Leftrightarrow$

$$\forall p \in P, z \in S, y \in CS : (B_d(p, y, z) \rightarrow y \in S)$$

Lemma 2.2 (Union of Convex Sets)

Let C_1, \dots, C_m be convex sets under some metric d and let $P = \bigcap_{i=1}^m C_i$. If $P \neq \emptyset$, then $S = \bigcup_{i=1}^m C_i$ is star-shaped under d with respect to P .

Proof. Assume $P = \bigcap_{i=1}^m C_i \neq \emptyset$ and let $x \in P$. Obviously, $\forall i \in \{1, \dots, m\} : x \in C_i$. Pick any point $z \in S = \bigcup_{i=1}^m C_i$. We know that $\exists i \in \{1, \dots, m\} : z \in C_i$. As both $x, z \in C_i$ and as C_i is convex under d , we know that $\forall y \in CS : (B_d(x, y, z) \rightarrow y \in C_i)$. Since $C_i \subseteq S$, this also implies that $y \in S$. This holds for all $x \in P$ and $z \in S$, therefore S is star-shaped under d with respect to P . \square

Cuboids as building
blocks

We then based our cores on axis-parallel cuboids, which turn out to be convex under the combined metric d_C^Δ (see Lemma 2.3):

Definition 2.8 (Axis-Parallel Cuboid)

We describe an axis-parallel cuboid¹ C as a triple $\langle \Delta_C, p^-, p^+ \rangle$. C is defined on the domains $\Delta_C \subseteq \Delta$, i.e., on the dimensions $D_C = \bigcup_{\delta \in \Delta_C} \delta$. We call p^-, p^+ the support points of C and require that $p^+, p^- \in D_C$. This means that p^+ and p^- have an entry p_d^+, p_d^- for each dimension $d \in D_C$. For all $d \in D \setminus D_C$, we can for convenience assume that $p_d^+ = +\infty$ and $p_d^- = -\infty$. If for any $d \in D_C$ we have $p_d^- > p_d^+$, then the cuboid C is empty, i.e., $C = \emptyset$.

Then, we define the cuboid C in the following way:

$$C = \{x \in CS \mid \forall d \in D : p_d^- \leq x_d \leq p_d^+\}$$

Lemma 2.3 (Cuboids are Convex)

A cuboid C is convex under $d_C^{\Delta_C}$, given a fixed set of weights W .

Cuboids are convex
under d_C

Proof. Let C be a cuboid and let $x, z \in C$. This means that $\forall d \in D_C : p_d^- \leq x_d, z_d \leq p_d^+$. Now let $y \in CS$ such that $B_{d_C^{\Delta_C}}(x, y, z)$.

$$\begin{aligned} B_{d_C^{\Delta_C}}(x, y, z) &\Leftrightarrow \left(d_C^{\Delta_C}(x, y) + d_C^{\Delta_C}(y, z) = d_C^{\Delta_C}(x, z) \right) \\ &\Leftrightarrow \left(\sum_{\delta \in \Delta_C} w_\delta \cdot d_E^\delta(x, y) + \sum_{\delta \in \Delta_C} w_\delta \cdot d_E^\delta(y, z) \right. \\ &\qquad \qquad \qquad \left. = \sum_{\delta \in \Delta_C} w_\delta \cdot d_E^\delta(x, z) \right) \\ &\Leftrightarrow \left(\sum_{\delta \in \Delta_C} w_\delta \cdot (d_E^\delta(x, y) + d_E^\delta(y, z)) \right. \\ &\qquad \qquad \qquad \left. = \sum_{\delta \in \Delta_C} w_\delta \cdot d_E^\delta(x, z) \right) \end{aligned}$$

If for some domain $\delta \in \Delta_C$ we observed $d_E^\delta(x, y) + d_E^\delta(y, z) > d_E^\delta(x, z)$, we would need to find some other domain $\delta' \in \Delta_C$ with $d_E^{\delta'}(x, y) + d_E^{\delta'}(y, z) < d_E^{\delta'}(x, z)$ if we want the equation from above

¹ We will drop the modifier "axis-parallel" from now on.

to hold. However, this would violate the triangle equality for $d_E^{\delta'}$. Therefore, we can conclude that $d_E^{\delta}(x, y) + d_E^{\delta}(y, z) = d_E^{\delta}(x, z)$ for all $\delta \in \Delta_C$. Moreover, since betweenness in the Euclidean sense refers to all points on the line segment between the two end points x and z , we can rewrite this by using a parameter $t \in [0, 1]$:

$$\begin{aligned} B_{d_C^{\Delta_C}}(x, y, z) \\ \Leftrightarrow \forall \delta \in \Delta_C : d_E^{\delta}(x, y) + d_E^{\delta}(y, z) &= d_E^{\delta}(x, z) \\ \Leftrightarrow \forall \delta \in \Delta_C : \exists t \in [0, 1] : \forall d \in \delta : y_d &= t \cdot x_d + (1 - t) \cdot z_d \end{aligned}$$

We already know that $\forall d \in D_C : x_d, z_d \in [p_d^-, p_d^+]$. Because y_d is a convex combination of x_d and z_d , one can easily see that $\forall d \in D_C : y_d \in [p_d^-, p_d^+]$. This means that $y \in C$. Therefore, C is convex under $d_C^{\Delta_C}$. \square

This leads to our definition of a core, which will play a role again in the proofs of Section A.4:

Cores as union of intersecting cuboids

Definition 2.9 (Core of a Conceptual Region)

We describe a core S as a tuple $\langle \Delta_S, \{C_1, \dots, C_m\} \rangle$ where $\Delta_S \subseteq \Delta$ is a set of domains on which the cuboids $\{C_1, \dots, C_m\}$ (and thus also S) are defined. Again, we can derive $D_S = \bigcup_{\delta \in \Delta_S} \delta$, i.e., the set of all dimensions on which S is defined. We further require that the central region $P = \bigcap_{i=1}^m C_i \neq \emptyset$. Then the core S is defined as follows:

$$S = \bigcup_{i=1}^m C_i$$

A.4 FUZZY CONCEPTUAL REGIONS

In Section 2.3.3, we have defined concepts as fuzzy sets based on a given core and a similarity-based membership function. The proofs below are based on the following definitions:

Definition 2.10 (Fuzzy Set)

A fuzzy set \tilde{A} on CS is defined by its membership function $\mu_{\tilde{A}} : CS \rightarrow [0, 1]$. For each $x \in CS$, $\mu_{\tilde{A}}(x)$ is interpreted as degree of membership of x in \tilde{A} , with $\mu_{\tilde{A}}(x) = 1$ indicating full membership and $\mu_{\tilde{A}}(x) = 0$ indicating complete non-membership.

Definition 2.11 (α -Cut)

Given a fuzzy set \tilde{A} on CS , its α -cut \tilde{A}^α for $\alpha \in [0, 1]$ is defined as follows:

$$\tilde{A}^\alpha = \{x \in CS \mid \mu_{\tilde{A}}(x) \geq \alpha\}$$

The special case of \tilde{A}^1 is called the core of \tilde{A} .

Definition 2.13 (Fuzzy Star-Shapedness)

A fuzzy set \tilde{A} on CS is called star-shaped under a metric d with respect to a crisp set $P \Leftrightarrow$

$$\forall \alpha \in [0, 1] : \left(\tilde{A}^\alpha = \emptyset \text{ or } \tilde{A}^\alpha \text{ is star-shaped under } d \text{ with respect to } P \right).$$

Definition 2.14 (Concept as Fuzzy Region)

A concept \tilde{S} is described by a quadruple $\langle S, \mu_0, c, W \rangle$. The components of this quadruple are the following:

- $S = \langle \Delta_S, \{C_1, \dots, C_m\} \rangle$ is a non-empty core as described in Definition 2.9.
- The parameter $\mu_0 \in (0, 1]$ controls the highest possible membership to \tilde{S} and is usually set to 1.
- The sensitivity parameter $c > 0$ controls the rate of the exponential decay in the similarity function and thus the overall fuzziness of \tilde{S} .
- Finally, $W = \langle W_{\Delta_S}, \{W_\delta\}_{\delta \in \Delta_S} \rangle$ contains positive weights for all domains in Δ_S and all dimensions within these domains. These weights are used when computing the combined metric $d_C^{\Delta_S}$ and reflect the relative importance of the respective domains and dimensions. As argued in Section 2.3.1, we assume that $\sum_{\delta \in \Delta_S} w_\delta = |\Delta_S|$ and that $\forall \delta \in \Delta_S : \sum_{d \in \delta} w_d = 1$.

The membership function of \tilde{S} is then defined as follows:

$$\mu_{\tilde{S}}(x) = \mu_0 \cdot \max_{y \in S} \left(e^{-c \cdot d_C^{\Delta_S}(x, y, W)} \right)$$

α -cuts are
 ϵ -neighborhoods

Showing that concepts are fuzzy star-shaped regions becomes easier by first demonstrating that each α -cut of the concept corresponds to an ϵ -neighborhood of its core:

Lemma 2.4 (α -Cut is an ϵ -Neighborhood)

Let $\tilde{S} = \langle S, \mu_0, c, W \rangle$ be a concept and let $\alpha \leq \mu_0$. Then, the α -cut \tilde{S}^α is equivalent to an ϵ -neighborhood of S with $\epsilon = -\frac{1}{c} \cdot \ln \left(\frac{\alpha}{\mu_0} \right)$.

Proof. This lemma can be shown with the following straightforward transformations:

$$\begin{aligned} x \in \tilde{S}^\alpha &\Leftrightarrow \left(\mu_{\tilde{S}}(x) = \mu_0 \cdot \max_{y \in S} \left(e^{-c \cdot d_C^{\Delta_S}(x, y, W)} \right) \geq \alpha \right) \\ &\Leftrightarrow \left(e^{-c \cdot \min_{y \in S} \left(d_C^{\Delta_S}(x, y, W) \right)} \geq \frac{\alpha}{\mu_0} \right) \\ &\Leftrightarrow \left(-c \cdot \min_{y \in S} d_C^{\Delta_S}(x, y, W) \geq \ln \left(\frac{\alpha}{\mu_0} \right) \right) \\ &\Leftrightarrow \left(\min_{y \in S} d_C^{\Delta_S}(x, y, W) \leq -\frac{1}{c} \cdot \ln \left(\frac{\alpha}{\mu_0} \right) \right) \quad \square \end{aligned}$$

Concepts are fuzzy
star-shaped regions

We can now finally show that fuzzy concepts as defined in Definition 2.14 are star-shaped according to Definition 2.13:

Proposition 2.1 (Concepts are Fuzzy Star-Shaped)

Any concept $\tilde{S} = \langle S, \mu_0, c, W \rangle$ is star-shaped with respect to $P = \bigcap_{i=1}^m C_i$ under $d_C^{\Delta_S}$.

Proof. For $\alpha \leq \mu_0$, \tilde{S}^α is an ϵ -neighborhood of S (Lemma 2.4). It therefore suffices to consider ϵ -neighborhoods of S . We can define the ϵ -neighborhood of a single cuboid C_i under $d_C^{\Delta S}$ as follows:

The ϵ -neighborhood of a cuboid

$$C_i^\epsilon = \{z \in CS \mid \forall d \in D_S : p_{id}^- - u_d \leq z_d \leq p_{id}^+ + u_d\}$$

Here, the vector \vec{u} represents the largest possible difference between $x \in C_i$ and $z \in C_i^\epsilon$. Thus, \vec{u} must fulfill the following constraints:

$$\left(\sum_{\delta \in \Delta_S} w_\delta \cdot \sqrt{\sum_{d \in \delta} w_d \cdot (u_d)^2} \leq \epsilon \right) \quad \wedge \quad (\forall d \in D_S : u_d \geq 0)$$

Let now $x \in C_i$ be a point in the cuboid C_i and $z \in C_i^\epsilon$ be a point in this cuboid's ϵ -neighborhood. We can write their coordinates as follows:

End points for betweenness considerations

$$\begin{aligned} \forall d \in D_S : x_d = p_{id}^- + a_d \text{ with } a_d \in [0, p_{id}^+ - p_{id}^-] \\ z_d = p_{id}^- + b_d \text{ with } b_d \in [-u_d, p_{id}^+ - p_{id}^- + u_d] \end{aligned}$$

We know that a point $y \in CS$ is between x and z with respect to $d_C^{\Delta S}$ if the following condition is true:

Considering intermediate points

$$\begin{aligned} & \left(d_C^{\Delta S}(x, y, W) + d_C^{\Delta S}(y, z, W) = d_C^{\Delta S}(x, z, W) \right) \\ & \Leftrightarrow \left(\forall \delta \in \Delta_S : d_E^\delta(x, y, W_\delta) + d_E^\delta(y, z, W_\delta) = d_E^\delta(x, z, W_\delta) \right) \\ & \Leftrightarrow (\forall \delta \in \Delta_S : \exists t \in [0, 1] : \forall d \in \delta : y_d = t \cdot x_d + (1 - t) \cdot z_d) \end{aligned}$$

The first equivalence holds because $d_C^{\Delta S}$ is a weighted sum of Euclidean metrics d_E^δ . The second equivalence has already been seen in the proof of Lemma 2.3. We can thus re-write the components of y :

$$\begin{aligned} \forall d \in D_S : \exists t \in [0, 1] : y_d &= t \cdot x_d + (1 - t) \cdot z_d \\ &= t \cdot (p_{id}^- + a_d) + (1 - t) \cdot (p_{id}^- + b_d) \\ &= p_{id}^- + \underbrace{t \cdot a_d + (1 - t) \cdot b_d}_{=c_d} \end{aligned}$$

Because c_d is a convex combination of a_d and b_d , we know that $c_d \in [-u_d, p_{id}^+ - p_{id}^- + u_d]$. Therefore $y \in C_i^\epsilon$. So C_i^ϵ is star-shaped with respect to C_i under $d_C^{\Delta S}$.

Since $P \subseteq C_i$, we also know that C_i^ϵ is star-shaped with respect to P under $d_C^{\Delta S}$. Therefore, $S^\epsilon = \bigcup_{i=1}^m C_i^\epsilon$ is star-shaped under $d_C^{\Delta S}$ with respect to P . Thus, all \tilde{S}^α with $\alpha \leq \mu_0$ are star-shaped under $d_C^{\Delta S}$ with respect to P . It is obvious that $\tilde{S}^\alpha = \emptyset$ if $\alpha > \mu_0$, so \tilde{S} is star-shaped according to Definition 2.13. \square

Every ϵ -neighborhood is star-shaped

B | OPERATIONS FOR COMBINING CONCEPTS

In this appendix, we give proofs for the lemmata and propositions from Chapter 3, where we defined various operations for creating new concepts based on additional ones.

B.1 INTERSECTION

The following two lemmata from Section 3.1.1 focus on the intersection of cores and show under which circumstances the result of a naive set intersection is again a valid core. For additional clarity, we re-print the definitions of cuboids and cores:

*Definitions of cuboid
and core*

Definition 2.8 (Axis-Parallel Cuboid)

We describe an axis-parallel cuboid¹ C as a triple $\langle \Delta_C, p^-, p^+ \rangle$. C is defined on the domains $\Delta_C \subseteq \Delta$, i.e., on the dimensions $D_C = \bigcup_{\delta \in \Delta_C} \delta$. We call p^-, p^+ the support points of C and require that $p^+, p^- \in D_C$. This means that p^+ and p^- have an entry p_d^+, p_d^- for each dimension $d \in D_C$. For all $d \in D \setminus D_C$, we can for convenience assume that $p_d^+ = +\infty$ and $p_d^- = -\infty$. If for any $d \in D_C$ we have $p_d^- > p_d^+$, then the cuboid C is empty, i.e., $C = \emptyset$.

Then, we define the cuboid C in the following way:

$$C = \{x \in CS \mid \forall d \in D : p_d^- \leq x_d \leq p_d^+\}$$

Definition 2.9 (Core of a Conceptual Region)

We describe a core S as a tuple $\langle \Delta_S, \{C_1, \dots, C_m\} \rangle$ where $\Delta_S \subseteq \Delta$ is a set of domains on which the cuboids $\{C_1, \dots, C_m\}$ (and thus also S) are defined. Again, we can derive $D_S = \bigcup_{\delta \in \Delta_S} \delta$, i.e., the set of all dimensions on which S is defined. We further require that the central region $P = \bigcap_{i=1}^m C_i \neq \emptyset$. Then the core S is defined as follows:

$$S = \bigcup_{i=1}^m C_i$$

Lemma 3.1 (Intersection of Cores is Union of Cuboids)

Let $S_1 = \langle \Delta_{S_1}, \{C_1^{(1)}, \dots, C_{m_1}^{(1)}\} \rangle$ and $S_2 = \langle \Delta_{S_2}, \{C_1^{(2)}, \dots, C_{m_2}^{(2)}\} \rangle$ be two cores. Then $S = S_1 \cap S_2$ can be written as union of cuboids, namely, $S = \bigcup_{i \in I} C_i$.

¹ We will drop the modifier "axis-parallel" from now on.

Conditions for set membership

Proof. Let us first note that the resulting set S will be defined on the domains $\Delta_S = \Delta_{S_1} \cup \Delta_{S_2}$ and thus on the dimensions $D_S = \bigcup_{\delta \in \Delta_S} \delta = D_{S_1} \cup D_{S_2}$. We can write S as follows, using the definition of cuboids:

$$\begin{aligned}
S &= S_1 \cap S_2 = \bigcup_{i_1=1}^{m_1} C_{i_1}^{(1)} \cap \bigcup_{i_2=1}^{m_2} C_{i_2}^{(2)} \\
&= \{x \in CS \mid \exists i_1 \in \{1, \dots, m_1\}, i_2 \in \{1, \dots, m_2\} : \\
&\quad x \in C_{i_1}^{(1)} \wedge x \in C_{i_2}^{(2)}\} \\
&= \{x \in CS \mid \exists i_1 \in \{1, \dots, m_1\}, i_2 \in \{1, \dots, m_2\} : \\
&\quad \left(\forall d \in D : p_{i_1 d}^{-(1)} \leq x_d \leq p_{i_1 d}^{+(1)} \right) \\
&\quad \wedge \left(\forall d \in D : p_{i_2 d}^{-(2)} \leq x_d \leq p_{i_2 d}^{+(2)} \right)\} \\
&= \{x \in CS \mid \exists i_1 \in \{1, \dots, m_1\}, i_2 \in \{1, \dots, m_2\} : \forall d \in D_S : \\
&\quad \max(p_{i_1 d}^{-(1)}, p_{i_2 d}^{-(2)}) \leq x_d \leq \min(p_{i_1 d}^{+(1)}, p_{i_2 d}^{+(2)})\}
\end{aligned}$$

Non-overlapping dimensions

Please recall from Definition 2.8 that if a cuboid is not defined on a dimension $d \in D$, then $p_d^- = -\infty$ and $p_d^+ = +\infty$. For instance, if $d \notin D_{S_1}$, we know that $p_{i_1 d}^{-(1)} = -\infty$ and $p_{i_1 d}^{+(1)} = +\infty$. We can therefore state the following:

$$\begin{aligned}
\forall d \in D_S \setminus D_{S_1} : \max(p_{i_1 d}^{-(1)}, p_{i_2 d}^{-(2)}) &= p_{i_2 d}^{-(2)} \\
\forall d \in D_S \setminus D_{S_1} : \min(p_{i_1 d}^{+(1)}, p_{i_2 d}^{+(2)}) &= p_{i_2 d}^{+(2)} \\
\forall d \in D_S \setminus D_{S_2} : \max(p_{i_1 d}^{-(1)}, p_{i_2 d}^{-(2)}) &= p_{i_1 d}^{-(1)} \\
\forall d \in D_S \setminus D_{S_2} : \min(p_{i_1 d}^{+(1)}, p_{i_2 d}^{+(2)}) &= p_{i_1 d}^{+(1)}
\end{aligned}$$

Pairwise intersection of cuboids

Let us now consider for all $i_1 \in \{1, \dots, m_1\}$ and $i_2 \in \{1, \dots, m_2\}$ the intersection of $C_{i_1}^{(1)}$ and $C_{i_2}^{(2)}$ and label it as $C_{i_1 \cdot m_2 + i_2}$:

$$\begin{aligned}
C_{i_1 \cdot m_2 + i_2} &= C_{i_1}^{(1)} \cap C_{i_2}^{(2)} \\
&= \{x \in CS \mid \forall d \in D_S : \\
&\quad \max(p_{i_1 d}^{-(1)}, p_{i_2 d}^{-(2)}) \leq x_d \leq \min(p_{i_1 d}^{+(1)}, p_{i_2 d}^{+(2)})\}
\end{aligned}$$

If we furthermore define $i = i_1 \cdot m_2 + i_2$, then we can write $S = S_1 \cap S_2 = \bigcup_{i=1}^{m_1 \cdot m_2} C_i$. Obviously, each C_i is a cuboid, thus S can be represented as a union of cuboids.

Removing empty cuboids

For mathematical convenience, it makes sense to remove all cuboids that are empty from our definition of S : We therefore keep a set I of indices such that $\forall i \in I : C_i \neq \emptyset$. This does not change the set S itself, because removing an empty set from a union has no effects: $S = \bigcup_{i \in I} C_i = \bigcup_{i=1}^{m_1 \cdot m_2} C_i$. We can now define a central region $P = \bigcap_{i \in I} C_i$ for the intersection result, which may, however, be empty. \square

Sufficient condition for star-shaped intersection result

In the following, we show a straightforward sufficient condition, which ensures that the intersection result is a valid core:

Lemma 3.2 (Sufficient Condition for Star-Shaped Intersection Result)

Let $S_1 = \langle \Delta_{S_1}, \{C_1^{(1)}, \dots, C_{m_1}^{(1)}\} \rangle$ and $S_2 = \langle \Delta_{S_2}, \{C_1^{(2)}, \dots, C_{m_2}^{(2)}\} \rangle$ be two cores with central regions P_1 and P_2 , respectively. Let furthermore $S = S_1 \cap S_2 = \bigcup_{i \in I} C_i$, where C_i are the cuboids of the intersection result. If $P_1 \cap P_2 \neq \emptyset$, then $P = \bigcap_{i \in I} C_i \neq \emptyset$

Proof. One can easily see the following equivalence:

$$\begin{aligned} (P_1 \cap P_2 \neq \emptyset) &\Leftrightarrow (\exists x \in CS : x \in P_1 \wedge x \in P_2) \\ &\Leftrightarrow (\exists x \in CS : (\forall i_1 \in \{1, \dots, m_1\} : x \in C_{i_1}^{(1)}) \\ &\quad \wedge (\forall i_2 \in \{1, \dots, m_2\} : x \in C_{i_2}^{(2)})) \end{aligned}$$

Now take any $i \in I$. We know that there exist i_1, i_2 , such that $C_i = C_{i_1}^{(1)} \cap C_{i_2}^{(2)}$ based on how C_i was constructed in the proof of Lemma 3.1. Since x is both in $C_{i_1}^{(1)}$ and in $C_{i_2}^{(2)}$, it is also in C_i . This holds for any $i \in I$, therefore $x \in P = \bigcap_{i \in I} C_i$, which shows that P is not empty and S is a valid core. \square

B.2 UNION

In Section 3.2.1, we introduced our modified union operation for cores and concepts:

Definitions of the union operation

Definition 3.5 (Modified Union of Cores)

Let $S_1 = \langle \Delta_{S_1}, \{C_1^{(1)}, \dots, C_{m_1}^{(1)}\} \rangle$ and $S_2 = \langle \Delta_{S_2}, \{C_1^{(2)}, \dots, C_{m_2}^{(2)}\} \rangle$ be two cores. Their modified union $U(S_1, S_2) = \langle \Delta_{S_1} \cup \Delta_{S_2}, \{C'_1, \dots, C'_{m'}\} \rangle$ is based on the cuboids C'_i , which have been obtained from the corresponding cuboids C_i from S_1 and S_2 , using the heuristic from Section 3.1.1 to obtain a central point p :

$$p = \frac{1}{m'} \cdot \sum_{i=1}^m \frac{(p_i^- + p_i^+)}{2}$$

$$\forall d \in D : p_{id}^- = \min(p_{id}^-, p_d), \quad p_{id}^+ = \max(p_{id}^+, p_d)$$

Definition 3.6 (Modified Union of Concepts)

Let $\tilde{S}_1 = \langle S_1, \mu_0^{(1)}, c^{(1)}, W^{(1)} \rangle$ and $\tilde{S}_2 = \langle S_2, \mu_0^{(2)}, c^{(2)}, W^{(2)} \rangle$ be two concepts. We define their modified union as $\tilde{S}' = U(\tilde{S}_1, \tilde{S}_2) = \langle S', \mu'_0, c', W' \rangle$ with the following components:

- $\mu'_0 = \max(\mu_0^{(1)}, \mu_0^{(2)})$
- $S' = U(S_1, S_2)$
- $c' = \min(c^{(1)}, c^{(2)})$

- W' with weights w'_δ, w'_d defined as follows (using interpolation factors $s, t \in [0, 1]$):

$$\begin{aligned} \forall \delta \in \Delta_{S_1} \cap \Delta_{S_2} : & \left((w'_\delta = s \cdot w_\delta^{(1)} + (1-s) \cdot w_\delta^{(2)}) \right. \\ & \left. \wedge \forall d \in \delta : (w'_d = t \cdot w_d^{(1)} + (1-t) \cdot w_d^{(2)}) \right) \\ \forall \delta \in \Delta_{S_1} \setminus \Delta_{S_2} : & \left((w'_\delta = w_\delta^{(1)}) \wedge \forall d \in \delta : (w'_d = w_d^{(1)}) \right) \\ \forall \delta \in \Delta_{S_2} \setminus \Delta_{S_1} : & \left((w'_\delta = w_\delta^{(2)}) \wedge \forall d \in \delta : (w'_d = w_d^{(2)}) \right) \end{aligned}$$

Definition 3.7 (Fuzzy Union)

Let \tilde{A}, \tilde{B} be two fuzzy sets on CS . Then, the membership function of their fuzzy union is defined as follows:

$$\forall x \in CS : \mu_{\tilde{A} \cup \tilde{B}}(x) = \max(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x))$$

Relation of our modified union to the standard union of fuzzy sets

The following proposition shows that the standard fuzzy set union from Definition 3.7 is under certain circumstances a fuzzy subset of our modified union operation from Definition 3.6:

Proposition 3.1 (Modified Union as Superset of Standard Union)

Let $\tilde{S}_1 = \langle S_1, \mu_0^{(1)}, c^{(1)}, W^{(1)} \rangle$ and $\tilde{S}_2 = \langle S_2, \mu_0^{(2)}, c^{(2)}, W^{(2)} \rangle$ be two concepts. If we assume that $\Delta_{S_1} = \Delta_{S_2}$ and $W^{(1)} = W^{(2)}$, then $\tilde{S}_1 \cup \tilde{S}_2 \subseteq U(\tilde{S}_1, \tilde{S}_2) = \tilde{S}'$, i.e., $\forall x \in CS : \mu_{\tilde{S}_1 \cup \tilde{S}_2}(x) \leq \mu_{U(\tilde{S}_1, \tilde{S}_2)}(x)$.

Preliminary observations

Proof. We know that both original cores are contained in the new core, because we use overextension as a repair mechanism (cf. Definition 3.5), i.e., $S_1 \cup S_2 \subseteq S'$. Moreover, because $\Delta_{S_1} = \Delta_{S_2}$ and $W^{(1)} = W^{(2)}$, we get that $\Delta_{S'} = \Delta_{S_1} = \Delta_{S_2}$ (cf. Definition 3.5) and that $W' = W^{(1)} = W^{(2)}$ (cf. Definition 3.6). Therefore, the distance functions used in the membership function of the three concepts are identical:

$$\forall x, y \in CS : d_C^{\Delta_{S_1}}(x, y, W^{(1)}) = d_C^{\Delta_{S_2}}(x, y, W^{(2)}) = d_C^{\Delta_{S'}}(x, y, W')$$

Rewriting the membership function

In order to simplify our notation, we write $d(x, y)$ to refer to this distance function. We can now rewrite $\mu_{\tilde{S}_1 \cup \tilde{S}_2}(x)$ as follows:

$$\begin{aligned} \mu_{\tilde{S}_1 \cup \tilde{S}_2}(x) &= \max(\mu_{\tilde{S}_1}(x), \mu_{\tilde{S}_2}(x)) \\ &= \max\left(\mu_0^{(1)} \cdot \max_{y \in S_1} e^{-c^{(1)} \cdot d_C^{\Delta_{S_1}}(x, y, W^{(1)})}, \right. \\ &\quad \left. \mu_0^{(2)} \cdot \max_{y \in S_2} e^{-c^{(2)} \cdot d_C^{\Delta_{S_2}}(x, y, W^{(2)})}\right) \\ &\stackrel{(i)}{=} \max\left(\mu_0^{(1)} \cdot e^{-c^{(1)} \cdot \min_{y \in S_1} d(x, y)}, \right. \\ &\quad \left. \mu_0^{(2)} \cdot e^{-c^{(2)} \cdot \min_{y \in S_2} d(x, y)}\right) \\ &\stackrel{(ii)}{\leq} \mu'_0 \cdot \max\left(e^{-c^{(1)} \cdot \min_{y \in S_1} d(x, y)}, e^{-c^{(2)} \cdot \min_{y \in S_2} d(x, y)}\right) \\ &\stackrel{(iii)}{\leq} \mu'_0 \cdot e^{-c' \cdot \min(\min_{y \in S_1} d(x, y), \min_{y \in S_2} d(x, y))} \end{aligned}$$

$$\begin{aligned}
 \stackrel{(iv)}{\leq} \mu'_0 \cdot e^{-c' \cdot \min_{y \in S'} d(x,y)} &= \mu'_0 \cdot \max_{y \in S'} e^{-c' \cdot d_C^{\Delta_{S'}}(x,y,W')} \\
 &= \mu_{\tilde{S}'}(x)
 \end{aligned}$$

In step (i), we use the equality of the distance functions. Steps (ii) and (iii) are based on the fact that according to Definition 3.6 $\mu'_0 = \max(\mu_0^{(1)}, \mu_0^{(2)})$ and $c' = \min(c^{(1)}, c^{(2)})$, respectively. Finally, step (iv) applies our knowledge that $S_1 \cup S_2 \subseteq S'$. Overall, we thus get that $\mu_{\tilde{S}_1 \cup \tilde{S}_2}(x) \leq \mu_{\tilde{S}'}(x)$, i.e., $\tilde{S}_1 \cup \tilde{S}_2 \subseteq U(\tilde{S}_1, \tilde{S}_2)$. \square

Explanation of the transformation steps

B.3 SUBSPACE PROJECTION

In Section 3.4.1, we have defined the projection of a core onto a given subset of domains based on the projections of its cuboids:

Definitions of the projection operation for cuboids and cores

Definition 3.8 (Projection of a Cuboid)

Let $C = \langle \Delta_C, p^-, p^+ \rangle$ be a cuboid defined on Δ_C . The projection $P(C, \Delta_{C'})$ of this cuboid onto a subspace $\Delta_{C'} \subseteq \Delta_C$ is defined as the cuboid $C' = \langle \Delta_{C'}, p^{-'}, p^{+'} \rangle$, whose support points are defined as follows (using $D_C = \bigcup_{\delta \in \Delta_C} \delta$ and $D_{C'} = \bigcup_{\delta \in \Delta_{C'}} \delta$):

$$\begin{aligned}
 \forall d \in D_{C'} : p_d^{-'} &= p_d^- \wedge p_d^+ = p_d^+ \\
 \forall d \in D \setminus D_{C'} : p_d^{-'} &= -\infty \wedge p_d^+ = +\infty
 \end{aligned}$$

Definition 3.9 (Projection of a Core)

Let $S = \langle \Delta_S, \{C_1, \dots, C_m\} \rangle$ be a core. Let $C'_i = P(C_i, \Delta_{S'})$ be the projection of C_i onto the domains $\Delta_{S'} \subseteq \Delta_S$. Then, $S' = P(S, \Delta_{S'}) = \langle \Delta_{S'}, \{C'_1, \dots, C'_m\} \rangle$ is the projected version of S .

In the following proposition, we show that if a core is projected onto two complementary subspaces, the intersection of these projections is a superset of the original core. We will use Corollary 3.1 as part of this proof, so we reprint here as well:

Intersecting complementary projections of cores

Corollary 3.1 (Intersection of Orthogonal Cores)

If two cores S_1 and S_2 are defined on completely different domains (i.e., $\Delta_{S_1} \cap \Delta_{S_2} = \emptyset$ and therefore $D_{S_1} \cap D_{S_2} = \emptyset$), then $P_1 \cap P_2 \neq \emptyset$ and $S_1 \cap S_2$ is a core.

Proposition 3.2 (Intersection of Projections of Cores)

Let $S = \langle \Delta_S, \{C_1, \dots, C_m\} \rangle$ be a core, and let $S_1 = P(S, \Delta_1)$ and $S_2 = P(S, \Delta_2)$ be its projections, where $\Delta_1 \cup \Delta_2 = \Delta_S$ and $\Delta_1 \cap \Delta_2 = \emptyset$. Then, $S \subseteq S' = I(S_1, S_2)$.

Proof. Since S , S_1 , and S_2 are cores, we can write them as unions of cuboids:

Writing cores as unions of cuboids

$$\begin{aligned}
 S &= \bigcup_{i=1}^m C_i & S_1 &= \bigcup_{i=1}^m P(C_i, \Delta_1) = \bigcup_{i=1}^m C_i^{(1)} \\
 & & S_2 &= \bigcup_{i=1}^m P(C_i, \Delta_2) = \bigcup_{i=1}^m C_i^{(2)}
 \end{aligned}$$

As seen before in Lemma 3.1, we can also write the intersection $S' = I(S_1, S_2)$ as a union of cuboids:

$$S' = \bigcup_{i=1}^{m^2} C'_i \text{ with } C'_i = C_{\lfloor \frac{i}{m} \rfloor}^{(1)} \cap C_{i \bmod m}^{(2)}$$

Please note that we use $i = m_2 \cdot i_1 + i_2$ in order to write $i_1 = \lfloor \frac{i}{m} \rfloor$ and $i_2 = i \bmod m$, using the fact that $m_1 = m_2 = m$. So let C_i from S be projected to $C_i^{(1)}$ and $C_i^{(2)}$:

$$\begin{aligned} C_i &= \{x \in CS \mid \forall d \in D_S : p_{id}^- \leq x_d \leq p_{id}^+\} \\ C_i^{(1)} &= \{x \in CS \mid \forall d \in D_1 : p_{id}^- \leq x_d \leq p_{id}^+\} \\ C_i^{(2)} &= \{x \in CS \mid \forall d \in D_2 : p_{id}^- \leq x_d \leq p_{id}^+\} \end{aligned}$$

Considering pairwise intersections of cuboids

Since $D_1 \cap D_2 = \emptyset$ and $D_1 \cup D_2 = D_S$ and $C'_{i \cdot m + i} = C_i^{(1)} \cap C_i^{(2)} = C_i$ can be defined by combining the constraints from $C_i^{(1)}$ and $C_i^{(2)}$. We hence re-create all the original C_i s when defining S' by combining their respective projections. However, we also generate more cuboids C'_i as there are $m^2 \geq m$ combinations of $C_{i_1}^{(1)}$ and $C_{i_2}^{(2)}$. Since $\Delta_1 \cap \Delta_2 = \emptyset$, Corollary 3.1 tells us that $S' = S_1 \cap S_2$ is a valid core, i.e., that its central region $P = \bigcap_{i=1}^{m^2} C'_i$ is not empty. Overall, we therefore get that $S = \bigcup_{i=1}^m C_i \subseteq \bigcup_{i=1}^{m^2} C'_i = S'$. \square

Definition of the projection operation for concepts

In Section 3.4.1, we then generalized the modified projection from cores to concepts as follows:

Definition 3.10 (Projection of a Concept)

Let $\tilde{S} = \langle S, \mu_0, c, W \rangle$ be a concept and $\Delta_{S'} \subseteq \Delta_S$ a subset of its domains. The projection of \tilde{S} on $\Delta_{S'}$ is defined as $P(\tilde{S}, \Delta_{S'}) = \langle S', \mu_0, c, W' \rangle$ with $S' = P(S, \Delta_{S'})$ and $W' = \left\langle \left\{ |\Delta_{S'}| \cdot \frac{w_\delta}{\sum_{\delta' \in \Delta_{S'}} w_{\delta'}} \right\}_{\delta \in \Delta_{S'}}, \{W_\delta\}_{\delta \in \Delta_{S'}} \right\rangle$.

Relation of our projection to the standard fuzzy set projection

In Proposition 3.3, we show that this definition is a fuzzy subset of the standard projection of fuzzy sets:

Definition 3.11 (Fuzzy Projection)

Let \tilde{A} be a concept defined on a space $d_1 \times \dots \times d_n$. Its projection $\tilde{A} \downarrow (d_1 \times \dots \times d_l)$ to a subspace $d_1 \times \dots \times d_l$ (with $l < n$) is defined as follows:

$$\mu_{\tilde{A} \downarrow (d_1 \times \dots \times d_l)}(x_1, \dots, x_l) = \max_{\substack{(x_{l+1}, \dots, x_n) \\ \in d_{l+1} \times \dots \times d_n}} \mu_{\tilde{A}}(x_1, \dots, x_l, x_{l+1}, \dots, x_n)$$

Proposition 3.3 (Modified Projection as a Subset of Standard Projection)

Let $\tilde{S} = \langle S, \mu_0, c, W \rangle$ be a concept. Let $\Delta_{S'} \subseteq \Delta_S$ and let $P(\tilde{S}, \Delta_{S'}) = \langle S', \mu_0, c, W' \rangle$ be the projection of \tilde{S} onto $\Delta_{S'}$. Let furthermore $\tilde{S}_{\downarrow \Delta_{S'}}$ be the standard fuzzy set projection of \tilde{S} onto the domains $\Delta_{S'}$ as defined above. Then, $P(\tilde{S}, \Delta_{S'}) \subseteq \tilde{S}_{\downarrow \Delta_{S'}}$, i.e., $\forall x \in CS : \mu_{P(\tilde{S}, \Delta_{S'})}(x) \leq \mu_{\tilde{S}_{\downarrow \Delta_{S'}}}(x)$.

Proof. For notational convenience, let $\tilde{S}' = P(\tilde{S}, \Delta_{S'})$ and $\tilde{S}_1 = \tilde{S}_{\downarrow \Delta_{S'}}$. We consider $D_{S'} = \bigcup_{\delta \in \Delta_{S'}} \delta$, i.e., the set of all dimensions contained in the domains of $\Delta_{S'}$. Let furthermore $\{d_1, \dots, d_l\} = D_S \setminus D_{S'}$ and $\{d_{l+1}, \dots, d_n\} = D_{S'}$ in order to establish a notation convention bridging Definitions 3.10 and 3.11.

Notation

Using $\bar{x} = (x_{d_1}, \dots, x_{d_l}, x_{d_{l+1}}, \dots, x_{d_n})$, we can write the standard projection $\mu_{\tilde{S}_1}(x) = \mu_{\tilde{S}_{\downarrow \Delta_{S'}}}(x)$ as follows (cf. Definition 3.11):

Rewriting the standard projection

$$\begin{aligned}
 \mu_{\tilde{S}_1}(x) &= \mu_{\tilde{S}_1}(x_{d_{l+1}}, \dots, x_{d_n}) \\
 &= \max_{\substack{(x_{d_1}, \dots, x_{d_l}) \\ \in d_1 \times \dots \times d_l}} \mu_{\tilde{S}}(x_{d_1}, \dots, x_{d_l}, x_{d_{l+1}}, \dots, x_{d_n}) \\
 &= \max_{\substack{(x_{d_1}, \dots, x_{d_l}) \\ \in d_1 \times \dots \times d_l}} \mu_{\tilde{S}}(\bar{x}) = \max_{\substack{(x_{d_1}, \dots, x_{d_l}) \\ \in d_1 \times \dots \times d_l}} \mu_0 \cdot \max_{y \in S} e^{-c \cdot d_C^{\Delta_S}(\bar{x}, y, W)} \\
 &= \mu_0 \cdot e^{-c \cdot \min_{\substack{(x_{d_1}, \dots, x_{d_l}) \\ \in d_1 \times \dots \times d_l}} \min_{y \in S} d_C^{\Delta_S}(\bar{x}, y, W)} \\
 &= \mu_0 \cdot e^{-c \cdot \min_{y \in S} \min_{\substack{(x_{d_1}, \dots, x_{d_l}) \\ \in d_1 \times \dots \times d_l}} d_C^{\Delta_S}(\bar{x}, y, W)} \tag{B.1}
 \end{aligned}$$

On the other hand, the membership function of $\tilde{S}' = P(\tilde{S}, \Delta_{S'})$ can be written as follows:

Rewriting our projection

$$\begin{aligned}
 \mu_{\tilde{S}'}(x) &= \mu_{\tilde{S}'}(x_{d_{l+1}}, \dots, x_{d_n}) = \mu_0 \cdot \max_{y \in S'} e^{-c \cdot d_C^{\Delta_{S'}}(x, y, W')} \\
 &= \mu_0 \cdot e^{-c \cdot \min_{y \in S'} d_C^{\Delta_{S'}}(x, y, W')} \tag{B.2}
 \end{aligned}$$

Based on Equations B.1 and B.2, it is easy to see that the following equivalence holds:

The condition to show

$$\begin{aligned}
 & \left(\mu_{\tilde{S}'}(x) \leq \mu_{\tilde{S}_1}(x) \right) \\
 & \Leftrightarrow \left(\min_{y \in S'} d_C^{\Delta_{S'}}(x, y, W') \geq \min_{y \in S} \min_{\substack{(x_{d_1}, \dots, x_{d_l}) \\ \in d_1 \times \dots \times d_l}} d_C^{\Delta_S}(\bar{x}, y, W) \right)
 \end{aligned}$$

Based on the definition of the combined metric $d_C^{\Delta_S}$, we moreover know the following:

Irrelevance of the removed dimensions

$$\min_{y \in S} \min_{\substack{(x_{d_1}, \dots, x_{d_l}) \\ \in d_1 \times \dots \times d_l}} d_C^{\Delta_S}(\bar{x}, y, W) = \min_{y \in S} \min_{\substack{(x_{d_1}, \dots, x_{d_l}) \\ \in d_1 \times \dots \times d_l}} \sum_{\delta \in \Delta_S} w_\delta \cdot \sqrt{\sum_{d \in \delta} w_d \cdot |x_d - y_d|^2}$$

One can easily see that we can minimize this sum over $(x_{d_1}, \dots, x_{d_l})$ for a given y by using the following values:

$$\forall i \in \{1, \dots, l\} : x_{d_i} = y_{d_i}$$

This way, we ensure that $\forall i \in \{1, \dots, l\} : |x_{d_i} - y_{d_i}| = 0$. The dimensions d_1, \dots, d_l thus do not play any role for the computation of $\mu_{\tilde{S}_1}(x)$. Moreover, they are also not involved in computing $\mu_{\tilde{S}'}(x)$, since they do not belong to $\Delta_{S'}$.

Differences in distances must be based on weights

Furthermore, the values of x on all other dimensions x_{l+1}, \dots, x_n are equal for both $\mu_{\tilde{S}_1}(x)$ and $\mu_{\tilde{S}'}(x)$. The only difference that can arise between $\mu_{\tilde{S}_1}(x)$ and $\mu_{\tilde{S}'}(x)$ therefore stems from the weights. The normalization procedure from Definition 3.10 guarantees for the domain weights that $\forall \delta \in \Delta_{S'} : w'_\delta \geq w_\delta$. Moreover, the dimension weights are left unchanged, i.e., $\forall d \in D_{S'} : w'_d = w_d$. Since all weights for potentially relevant domains and dimensions are not smaller in W' than in W , and because $S' = P(S, \Delta_{S'})$, we can conclude:

$$\forall x \in CS : \min_{y \in S'} d_C^{\Delta_{S'}}(x, y, W') \geq \min_{y \in S} \min_{\substack{(x_{d_1}, \dots, x_{d_l}) \\ \in d_1 \times \dots \times d_l}} d_C^{\Delta_S}(\bar{x}, y, W)$$

As stated above, this is equivalent to $\tilde{S}' \subseteq \tilde{S}_1$. \square

Intersecting complementary projections of concepts

Finally, we can generalize Proposition 3.2 from cores to concepts under one additional constraint:

Proposition 3.4 (Intersection of Projections of Concepts)

Let $\tilde{S} = \langle S, \mu_0, c, W \rangle$ be a concept, and let $\tilde{S}_1 = P(\tilde{S}, \Delta_1)$ and $\tilde{S}_2 = P(\tilde{S}, \Delta_2)$ be its projections with $\Delta_1 \cup \Delta_2 = \Delta_S$ and $\Delta_1 \cap \Delta_2 = \emptyset$. Let furthermore $\tilde{S}' = I(\tilde{S}_1, \tilde{S}_2)$ as defined in Definition 3.2. If $\sum_{\delta \in \Delta_1} w_\delta = |\Delta_1|$ and $\sum_{\delta \in \Delta_2} w_\delta = |\Delta_2|$, then $\tilde{S} \subseteq \tilde{S}' = I(\tilde{S}_1, \tilde{S}_2)$, i.e., $\forall x \in CS : \mu_{\tilde{S}}(x) \leq \mu_{I(\tilde{S}_1, \tilde{S}_2)}(x)$.

Transformation of the membership function

Proof. We can transform $\mu_{\tilde{S}}(x)$ as follows:

$$\begin{aligned} \mu_{\tilde{S}}(x) &= \mu_0 \cdot \max_{y \in S} e^{-c \cdot d_C^{\Delta_S}(x, y, W)} \stackrel{(i)}{=} \mu'_0 \cdot \max_{y \in S} e^{-c' \cdot d_C^{\Delta_S}(x, y, W)} \\ &\stackrel{(ii)}{\leq} \mu'_0 \cdot \max_{y \in S'} e^{-c' \cdot d_C^{\Delta_S}(x, y, W)} \stackrel{(iii)}{=} \mu'_0 \cdot \max_{y \in S'} e^{-c' \cdot d_C^{\Delta_S}(x, y, W')} \\ &= \mu_{\tilde{S}'}(x) \end{aligned}$$

Explanation of the transformation steps

In step (i), we use that $\mu'_0 = \mu_0$ and $c' = c$, because both parameters are not changed during projection and intersection (cf. Definitions 3.10 and 3.2, respectively). With respect to step (ii), we already know from Proposition 3.2 that $S \subseteq I(P(S, \Delta_1), P(S, \Delta_2)) = S'$. Finally, $W^{(1)}$ only contains weights for Δ_1 , whereas $W^{(2)}$ only contains weights for Δ_2 . Since $\sum_{\delta \in \Delta_i} w_\delta = |\Delta_i|$ (for $i \in \{1, 2\}$), the weights are not changed during the projection. As $\Delta_1 \cap \Delta_2 = \emptyset$, they are also not changed during the intersection, so $W' = W$. This allows us to make the transformation of step (iii). \square

B.4 AXIS-PARALLEL CUT

Definition of the cut operation for cores

In Section 3.5.1, we have defined the axis-parallel cut of a core as follows:

Definition 3.12 (Axis-Parallel Cut of a Core)

Let $S = \langle \Delta_S, \{C_1, \dots, C_m\} \rangle$ be a core, $d^* \in D_S$ be any dimension and v be any value on this dimension. The axis-parallel cut of S based on a threshold value v for a dimension d^* is given by $S^- = \{x \in S \mid x_{d^*} \leq v\}$ and $S^+ = \{x \in S \mid x_{d^*} \geq v\}$.

We now show that this cut always results in two valid cores.

Proposition 3.5 (Cut of a Core Results in Cores)

Let S be a core and $S^-, S^+ = C(S, d^*, v)$ its cut at a threshold value v on a dimension $d^* \in D_S$. Then, both S^- and S^+ are valid cores.

Proof. We can prove this by distinguishing different five cases:

1. $v \geq \max_{i \in \{1, \dots, m\}} (p_{id^*}^+)$:

In this case, $S^+ = \emptyset$ and $S^- = S$. Both S^+ and S^- are obviously star-shaped and can be represented by a union of cuboids.

Cut above the concept

2. $v \leq \min_{i \in \{1, \dots, m\}} (p_{id^*}^-)$:

In this case, $S^- = \emptyset$ and $S^+ = S$. Both S^+ and S^- are obviously star-shaped and can be represented by a union of cuboids.

Cut below the concept

3. $\max_{i \in \{1, \dots, m\}} (p_{id^*}^-) \leq v \leq \min_{i \in \{1, \dots, m\}} (p_{id^*}^+)$:

In this case, we dissect the central region $P = \bigcap_{i \in \{1, \dots, m\}} C_i$. We therefore get $P^- = \{x \in P \mid x_{d^*} \leq v\}$ and $P^+ = \{x \in P \mid x_{d^*} \geq v\}$. Obviously, $P^+ \subseteq S^+$ and $P^- \subseteq S^-$. All the cuboids $C_i \in S$ are split into two parts $C_i^{(+)} = \{x \in C_i \mid x_{d^*} \geq v\}$ and $C_i^{(-)} = \{x \in C_i \mid x_{d^*} \leq v\}$. One can easily see that both $C_i^{(+)}$ and $C_i^{(-)}$ are valid cuboids.² Moreover, $P^+ = \bigcap_{i \in \{1, \dots, m\}} C_i^{(+)}$ and $P^- = \bigcap_{i \in \{1, \dots, m\}} C_i^{(-)}$, so both S^+ and S^- are valid cores.

Cut through the central region

4. $\min_{i \in \{1, \dots, m\}} (p_{id^*}^+) < v < \max_{i \in \{1, \dots, m\}} (p_{id^*}^-)$:

In this case, the cut occurs "above" the central region P and therefore does not affect all cuboids. This is illustrated in Figure B.1. Let $T^- = \{C_i \mid p_{id^*}^+ \leq v\}$ and $T^+ = \{C_i \mid p_{id^*}^- > v\}$. Both sets are not empty. All cuboids from T^- completely belong to S^- and do not play any role for S^+ . All cuboids from T^+ are split at v . Their top part $C_i^{(+)}$ belongs to S^+ , their bottom part $C_i^{(-)}$ to S^- . It is obvious that $P \subseteq S^-$ and one can easily see that S^- is star-shaped with respect to P .

Cut above the central region

S^+ only contains $C_i^{(+)}$ based on $C_i \in T^+$. All original C_i intersected in P with respect to all dimensions $d \in D_S$. This also holds if we only consider the C_i from T^+ and if we only look at dimensions $d \in D_S \setminus \{d^*\}$. Moreover, the $C_i^{(+)}$ agree with the C_i from T^+ on $D_S \setminus \{d^*\}$. One can easily see that the $C_i^{(+)}$ intersect also with respect to d^* , as $C_i^{(+)} = \{x \in C_i \mid x_{d^*} \geq v\}$. This means that with respect to d^* , at least v is contained in each $C_i^{(+)}$. Therefore, all cuboids $C_i^{(+)}$ intersect on all dimensions $d \in D_S$ and their intersection $P^+ = \bigcap_{i \in \{1, \dots, m\}} C_i^{(+)}$ is not empty. Thus, also S^+ is star-shaped.

5. $\min_{i \in \{1, \dots, m\}} (p_{id^*}^-) < v < \max_{i \in \{1, \dots, m\}} (p_{id^*}^+)$:

Analogous to 4. □

Cut below the central region

² A strict inequality in the definition of $C_i^{(+)}$ or $C_i^{(-)}$ would not yield a cuboid.

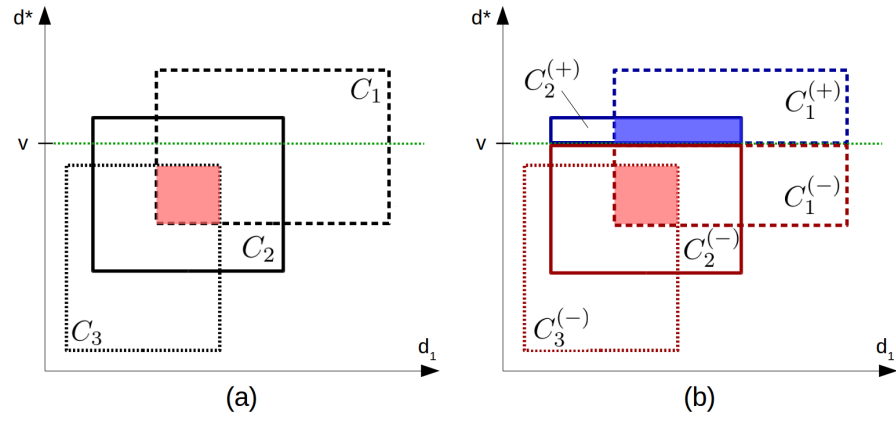


Figure B.1: Cut above the central region. (a) Original core $S = \langle \Delta_S, \{C_1, C_2, C_3\} \rangle$. (b) Resulting cores $S^- = \langle \Delta_S, \{C_1^{(-)}, C_2^{(-)}, C_3^{(-)}\} \rangle$ and $S^+ = \langle \Delta_S, \{C_1^{(+)}, C_2^{(+)}\} \rangle$

C

MEASURING RELATIONS BETWEEN CONCEPTS

In this appendix, we give proofs for the lemmata and propositions from Chapter 4, where we defined various relations between conceptual regions. Moreover, some additional helper lemmata are included to facilitate the proofs.

C.1 SIZE

In Section 4.1, we introduced a way of measuring the size of a conceptual region. This was mainly based on the size of a fuzzified cuboid's α -cut as motivate in Section 4.1.1. Most of the heavy lifting is done in Proposition 4.1, which provides a closed formula for the size of a hyperball under the unweighted combined metric d_C^Δ .¹

Size of a fuzzified cuboid's α -cut

In order to provide a better structure to the proof of Proposition 4.1, we will now show three helper lemmata, that will allow us to considerably simplify some mathematical terms and equations. They make use of Euler's gamma function $\Gamma(z) = \int_0^\infty x^{z-1} e^{-x} dx$ (for which $\forall n \in \mathbb{N} : \Gamma(n) = (n-1)!$), the beta function $B(x, y) = \int_0^1 t^{x-1} (1-t)^{y-1} dt$, and their relation $B(x, y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$.

Helper lemmata for simplifying nested integrals

In the proof of Proposition 4.1, we will compute a nested integral over spherical coordinates. Lemma C.1 provides us with a way of simplifying the part of this nested integral that concerns the angles $\phi_1, \dots, \phi_{n-1}$:

Nested integral over angles

Lemma C.1 (Simplification of an Integral over Angles)

Let $n \in \mathbb{N}$ and $\phi_1, \dots, \phi_{n-1} \in \mathbb{R}$. With $\Gamma(\cdot)$ being Euler's gamma function, the following equality holds:

$$\int_0^{2\pi} \int_0^\pi \int_0^\pi \dots \int_0^\pi \sin^{n-2}(\phi_1) \sin^{n-3}(\phi_2) \dots \sin(\phi_{n-2}) d\phi_1 \dots d\phi_{n-1} = 2 \cdot \frac{\pi^{\frac{n}{2}}}{\Gamma\left(\frac{n}{2}\right)}$$

Proof. We start by restructuring the overall integral:

Restructuring the integral

$$\begin{aligned} I &= \int_0^{2\pi} \int_0^\pi \int_0^\pi \dots \int_0^\pi \sin^{n-2}(\phi_1) \sin^{n-3}(\phi_2) \dots \sin(\phi_{n-2}) d\phi_1 \dots d\phi_{n-1} \\ &= \left(\int_0^{2\pi} 1 d\phi_{n-1} \right) \left(\int_0^\pi \sin(\phi_{n-2}) d\phi_{n-2} \right) \dots \left(\int_0^\pi \sin^{n-2}(\phi_1) d\phi_1 \right) \end{aligned}$$

¹ The proofs contained in this section have been published in [36] as a preprint.

$$= \left(4 \cdot \int_0^{\frac{\pi}{2}} 1 \, d\phi_{n-1} \right) \left(2 \cdot \int_0^{\frac{\pi}{2}} \sin(\phi_{n-2}) \, d\phi_{n-2} \right) \cdots \left(2 \cdot \int_0^{\frac{\pi}{2}} \sin^{n-2}(\phi_1) \, d\phi_1 \right)$$

Applying the beta function

We now apply the following, alternative definition of the beta function:

$$B(x, y) = 2 \cdot \int_0^{\frac{\pi}{2}} \sin^{2x-1}(\phi) \cos^{2y-1}(\phi) \, d\phi$$

Using $y = \frac{1}{2}$, we get:

$$I = \left(4 \cdot \int_0^{\frac{\pi}{2}} 1 \, d\phi_{n-1} \right) \left(2 \cdot \int_0^{\frac{\pi}{2}} \sin(\phi_{n-2}) \, d\phi_{n-2} \right) \cdots \left(2 \cdot \int_0^{\frac{\pi}{2}} \sin^{n-2}(\phi_1) \, d\phi_1 \right) \\ = 2 \cdot B\left(\frac{1}{2}, \frac{1}{2}\right) \cdot B\left(1, \frac{1}{2}\right) \cdots B\left(\frac{n-2}{2}, \frac{1}{2}\right) \cdot B\left(\frac{n-1}{2}, \frac{1}{2}\right)$$

Simplification with the gamma function

Next, we use the identity $B(x, y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$ with Euler's gamma function Γ and the fact that $\Gamma(\frac{1}{2}) = \sqrt{\pi}$. Since most of the terms cancel out, we arrive at our desired result:

$$I = 2 \cdot \frac{\Gamma(\frac{1}{2})\Gamma(\frac{1}{2})}{\Gamma(1)} \cdot \frac{\Gamma(1)\Gamma(\frac{1}{2})}{\Gamma(\frac{3}{2})} \cdots \frac{\Gamma(\frac{n-2}{2})\Gamma(\frac{1}{2})}{\Gamma(\frac{n-1}{2})} \cdot \frac{\Gamma(\frac{n-1}{2})\Gamma(\frac{1}{2})}{\Gamma(\frac{n}{2})} \\ = 2 \cdot \frac{\Gamma(\frac{1}{2})^n}{\Gamma(\frac{n}{2})} = 2 \cdot \frac{\sqrt{\pi}^n}{\Gamma(\frac{n}{2})} = 2 \cdot \frac{\pi^{\frac{n}{2}}}{\Gamma(\frac{n}{2})} \quad \square$$

Nested integral over radii

In addition to integrating over angles, we will also need to integrate over radii r_δ within the different domains $\delta \in \Delta$ in our conceptual space. Lemma C.2 provides us with an intermediate step for simplifying a nested integral over these radii in Lemma C.3.²

Lemma C.2 (Simplification of an Integral over a Single Radius)

Let $n, j \in \mathbb{N}$ with $n > j$, $a, b \in \mathbb{R}$, and $r_1, \dots, r_n \in \mathbb{R}^+$. For $r = \sum_{i=1}^n r_i$, the following equation holds:

$$\int_0^{r - \sum_{i=1}^{j-1} r_i} r_j^{a-1} \cdot \left(r - \sum_{i=1}^j r_i \right)^b \, dr_j = B(a, b+1) \cdot \left(r - \sum_{i=1}^{j-1} r_i \right)^{a+b}$$

Variable change

Proof. It is easy to see that $r_j = r - \sum_{i=1}^{j-1} r_i > 0$. We can therefore make

² I am indebted to the user Andreas (user:317854) on StackExchange, who has provided me with the key insight for proving these lemmata, see <https://math.stackexchange.com/questions/2222064/proof-of-equation-with-nested-dependent-integrals>.

a variable change from r_j to z by defining $r_j = \left(r - \sum_{i=1}^{j-1} r_i\right) \cdot z$ which gives $dr_j = \left(r - \sum_{i=1}^{j-1} r_i\right) \cdot dz$:

$$\begin{aligned} I &= \int_0^{r - \sum_{i=1}^{j-1} r_i} r_j^{a-1} \cdot \left(r - \sum_{i=1}^j r_i\right)^b dr_j \\ &= \int_0^{r - \sum_{i=1}^{j-1} r_i} r_j^{a-1} \cdot \left(r - \sum_{i=1}^{j-1} r_i - r_j\right)^b dr_j \\ &= \int_0^1 \left(r - \sum_{i=1}^{j-1} r_i\right)^{a-1} \cdot z^{a-1} \cdot \left(r - \sum_{i=1}^{j-1} r_i - \left(r - \sum_{i=1}^{j-1} r_i\right) \cdot z\right)^b \\ &\quad \cdot \left(r - \sum_{i=1}^{j-1} r_i\right) dz \end{aligned}$$

By factoring out $\left(r - \sum_{i=1}^j r_i\right)$, this can be further simplified:

Factoring out r_j

$$\begin{aligned} I &= \int_0^1 \left(r - \sum_{i=1}^{j-1} r_i\right)^{a-1} \cdot z^{a-1} \cdot \left(r - \sum_{i=1}^{j-1} r_i - \left(r - \sum_{i=1}^{j-1} r_i\right) \cdot z\right)^b \\ &\quad \cdot \left(r - \sum_{i=1}^{j-1} r_i\right) dz \\ &= \int_0^1 \left(r - \sum_{i=1}^{j-1} r_i\right)^a \cdot z^{a-1} \cdot \left(\left(r - \sum_{i=1}^{j-1} r_i\right) (1 - z)\right)^b dz \\ &= \left(r - \sum_{i=1}^{j-1} r_i\right)^{a+b} \int_0^1 z^{a-1} (1 - z)^b dz \end{aligned}$$

Since $B(x, y) = \int_0^1 t^{x-1} (1 - t)^{y-1} dt$, this can be written as follows:

Using the beta function

$$\begin{aligned} I &= \left(r - \sum_{i=1}^{j-1} r_i\right)^{a+b} \int_0^1 z^{a-1} (1 - z)^b dz \\ &= \left(r - \sum_{i=1}^{j-1} r_i\right)^{a+b} \cdot B(a, b + 1) \quad \square \end{aligned}$$

Lemma C.3 (Simplification of an Integral over Multiple Radii)

For any $k \in \mathbb{N}$, any $r_1, \dots, r_k, n_1, \dots, n_k \in \mathbb{R}^+$, $n = \sum_{i=0}^k n_i$, and $r = \sum_{i=1}^k r_i$, the following equation holds:

$$\begin{aligned} \int_0^r r_1^{n_1-1} \int_0^{r-r_1} r_2^{n_2-1} \dots \int_0^{r - \sum_{i=1}^{k-1} r_i} r_k^{n_k-1} dr_k \dots dr_1 \\ = \frac{r^n}{\Gamma(n+1)} \prod_{i=1}^k \Gamma(n_i) \end{aligned}$$

Simplifying the innermost integral

Proof. The innermost integral $\int_0^{r-\sum_{i=1}^{k-1} r_i} r_i$ can be simplified using Lemma C.2 with $j = k, a = n_k, b = 0$:

$$\begin{aligned} I &= \int_0^r r_1^{n_1-1} \int_0^{r-r_1} r_2^{n_2-1} \dots \int_0^{r-\sum_{i=1}^{k-1} r_i} r_k^{n_k-1} dr_k \dots dr_1 \\ &= \int_0^r r_1^{n_1-1} \int_0^{r-r_1} r_2^{n_2-1} \dots \int_0^{r-\sum_{i=1}^{k-2} r_i} r_{k-1}^{n_{k-1}-1} \cdot B(n_k, 1) \\ &\quad \cdot \left(r - \sum_{i=1}^{k-1} r_i \right)^{n_k} dr_{k-1} \dots dr_1 \\ &= B(n_k, 1) \cdot \int_0^r r_1^{n_1-1} \dots \int_0^{r-\sum_{i=1}^{k-2} r_i} r_{k-1}^{n_{k-1}-1} \cdot \left(r - \sum_{i=1}^{k-1} r_i \right)^{n_k} \\ &\quad dr_{k-1} \dots dr_1 \end{aligned}$$

Next simplification step

As one can see, we can again apply Lemma C.2 to the innermost integral (this time using $j = k - 1, a = n_{k-1}, b = n_k$):

$$\begin{aligned} I &= B(n_k, 1) \cdot \int_0^r r_1^{n_1-1} \dots \int_0^{r-\sum_{i=1}^{k-2} r_i} r_{k-1}^{n_{k-1}-1} \cdot \left(r - \sum_{i=1}^{k-1} r_i \right)^{n_k} \\ &\quad dr_{k-1} \dots dr_1 \\ &= B(n_k, 1) \cdot B(n_{k-1}, n_k + 1) \cdot \\ &\quad \int_0^r r_1^{n_1-1} \dots \int_0^{r-\sum_{i=1}^{k-3} r_i} r_{k-2}^{n_{k-2}-1} \cdot \left(r - \sum_{i=1}^{k-2} r_i \right)^{n_k+n_{k-1}} \\ &\quad dr_{k-1} \dots dr_1 \end{aligned}$$

Repeated simplifications

We can further simplify this expression by repeatedly applying Lemma C.2. In the last application of Lemma C.2, we use $j = 1, a = n_1$, and $b = n_2 + \dots + n_k$. Hence, $\left(r - \sum_{i=1}^{j-1} r_i \right)^{a+b}$ corresponds to $r^{n_1+\dots+n_k}$. Overall, we thus get the following equation:

$$I = B(n_k, 1) \cdot B(n_{k-1}, n_k + 1) \cdot \dots \cdot B(n_1, n_2 + \dots + n_k + 1) \cdot r^{n_1+\dots+n_k}$$

Using the gamma function

We can now use $B(x, y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$ in order to rewrite this equation:

$$\begin{aligned} I &= r^{n_1+\dots+n_k} \cdot \frac{\Gamma(n_k)\Gamma(1)}{\Gamma(n_k+1)} \cdot \frac{\Gamma(n_{k-1})\Gamma(n_k+1)}{\Gamma(n_{k-1}+n_k+1)} \\ &\quad \dots \frac{\Gamma(n_1)\Gamma(n_2+\dots+n_k+1)}{\Gamma(n_1+n_2+\dots+n_k+1)} \end{aligned}$$

Because $\Gamma(1) = 1$ and $n = \sum_{i=1}^k n_i$, and because most of the terms cancel out, this reduces to:

$$\begin{aligned} I &= r^{n_1+\dots+n_k} \cdot \Gamma(n_k) \cdot \dots \cdot \Gamma(n_1) \cdot \frac{1}{\Gamma(n_1+\dots+n_k+1)} \\ &= \frac{r^n}{\Gamma(n+1)} \prod_{i=1}^k \Gamma(n_i) \end{aligned} \quad \square$$

Using Lemmata C.1 and C.3, we can now prove Proposition 4.1:

Proposition 4.1 (Volume of a Hyperball without Saliency Weights)
The size of a hyperball with radius r in a space with the combined metric d_C^Δ , the domain structure Δ , and without domain and dimension weights can be computed in the following way, where n is the overall number of dimensions, n_δ is the number of dimensions in domain δ , and $\Gamma(\cdot)$ is Euler's gamma function:

Hyperball volume under d_C^Δ without saliency weights

$$V(r, \Delta) = \frac{r^n}{n!} \prod_{\delta \in \Delta} \left(n_\delta! \frac{\pi^{\frac{n_\delta}{2}}}{\Gamma(\frac{n_\delta}{2} + 1)} \right)$$

Proof. A hyperball of radius r can be defined as the set of all points that have a distance of at most r to its center. Without loss of generality, we assume that this center is identical to the origin. Since all w_δ and all w_d are assumed to be one, this can be written as follows:

Hyperballs

$$H = \left\{ x \in CS \mid d_C^\Delta(x, 0) = \sum_{\delta \in \Delta} \sqrt{\sum_{d \in \delta} x_d^2} \leq r \right\}$$

If we define $\forall \delta \in \Delta : r_\delta = \sqrt{\sum_{d \in \delta} x_d^2}$, we can easily see that $\sum_{\delta \in \Delta} r_\delta \leq r$. The term r_δ can be interpreted as the distance between x and the origin within the domain δ . The constraint $\sum_{\delta \in \Delta} r_\delta \leq r$ then simply means that the sum of domain-wise distances is less than the given radius. One can thus interpret r_δ as the radius within domain δ .

Domain-based radii

The size (or hypervolume) of the hyperball H can be computed with the following integral:

Obtaining the size through integration

$$V(r, \Delta) = \int \cdots \int_H 1 \, dH \quad (\text{C.1})$$

This integration becomes much easier if we use spherical coordinates instead of the Cartesian coordinates provided by our conceptual space.

Let us first consider the case of a single domain δ with n_δ dimensions. A single domain corresponds to a standard Euclidean space, therefore we can use the standard procedure of changing to spherical coordinates, following [121]. Let us index the dimensions of δ as d_1, \dots, d_{n_δ} . The coordinate change within the domain δ then looks like this:

Spherical coordinates within a domain

$$\begin{aligned} x_1 &= r_\delta \cdot \cos(\phi_1) \\ x_2 &= r_\delta \cdot \sin(\phi_1) \cdot \cos(\phi_2) \\ &\vdots \\ x_{n_\delta-1} &= r_\delta \cdot \sin(\phi_1) \cdots \sin(\phi_{n_\delta-2}) \cdot \cos(\phi_{n_\delta-1}) \\ x_{n_\delta} &= r_\delta \cdot \sin(\phi_1) \cdots \sin(\phi_{n_\delta-2}) \cdot \sin(\phi_{n_\delta-1}) \end{aligned}$$

This means, that we can express each coordinate x_i based on the distance r_δ to the origin and the rotation angles $\phi_1, \dots, \phi_{n_\delta-1}$.

Using the Jacobian matrix to determine the volume element

In order to switch the integral to spherical coordinates, we need to calculate the so-called *volume element*, which will replace the constant value 1 in Equation C.1 when the variables to integrate over are changed to $r_\delta, \phi_1, \dots, \phi_{n_\delta-1}$. The volume element can be found by looking at the determinant of the transformation's Jacobian matrix, which contains the partial derivatives of all original coordinates (i.e., x_1, \dots, x_{n_δ}) with respect to all new coordinates (i.e., $r_\delta, \phi_1, \dots, \phi_{n_\delta-1}$). The Jacobian matrix of the transformation of a single domain δ can be written as follows:

$$J_\delta = \begin{bmatrix} \frac{\delta x_1}{\delta r_\delta} & \frac{\delta x_1}{\delta \phi_1} & \cdots & \frac{\delta x_1}{\delta \phi_{n_\delta-1}} \\ \frac{\delta x_2}{\delta r_\delta} & \frac{\delta x_2}{\delta \phi_1} & \cdots & \frac{\delta x_2}{\delta \phi_{n_\delta-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\delta x_{n_\delta}}{\delta r_\delta} & \frac{\delta x_{n_\delta}}{\delta \phi_1} & \cdots & \frac{\delta x_{n_\delta}}{\delta \phi_{n_\delta-1}} \end{bmatrix} = \begin{bmatrix} \cos(\phi_1) & -r_\delta \sin(\phi_1) & 0 & 0 & \cdots & 0 \\ \sin(\phi_1) \cos(\phi_2) & r_\delta \cos(\phi_1) \cos(\phi_2) & -r_\delta \sin(\phi_1) \sin(\phi_2) & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \sin(\phi_1) \cdots \sin(\phi_{n_\delta-2}) \cos(\phi_{n_\delta-1}) & \cdots & \cdots & \cdots & \cdots & -r_\delta \sin(\phi_1) \cdots \sin(\phi_{n_\delta-2}) \sin(\phi_{n_\delta-1}) \\ \sin(\phi_1) \cdots \sin(\phi_{n_\delta-2}) \sin(\phi_{n_\delta-1}) & \cdots & \cdots & \cdots & \cdots & r_\delta \sin(\phi_1) \cdots \sin(\phi_{n_\delta-2}) \cos(\phi_{n_\delta-1}) \end{bmatrix}$$

The determinant of this matrix can be computed as follows:

$$\det(J_\delta) = r_\delta^{n_\delta-1} \cdot \sin^{n_\delta-2}(\phi_1) \cdot \sin^{n_\delta-3}(\phi_2) \cdots \sin(\phi_{n_\delta-2})$$

Spherical coordinates for all domains

We can now translate from Cartesian to spherical coordinates for the overall conceptual space by performing this coordinate change for each domain individually. Let us index the Cartesian coordinates of a point x in domain δ by $x_{\delta,1}, \dots, x_{\delta,n_\delta}$. Let us further index the spherical coordinates of domain δ by r_δ and $\phi_{\delta,1}, \dots, \phi_{\delta,n_\delta-1}$. Let $k = |\Delta|$ denote the total number of domains.

Because $x_{\delta,j}$ is defined independently from $r_{\delta'}$ and $\phi_{\delta',j'}$ for different domains $\delta \neq \delta'$, any derivative $\frac{\delta x_{\delta,j}}{\delta r_{\delta'}}$ or $\frac{\delta x_{\delta,j}}{\delta \phi_{\delta',j'}}$ will be zero. If we apply the coordinate change to all domains at once, the Jacobian matrix of the overall transformation has therefore the structure of a block matrix:

$$J = \begin{bmatrix} J_1 & 0 & \cdots & 0 \\ 0 & J_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & J_k \end{bmatrix}$$

Obtaining the overall volume element

The blocks on the diagonal are the Jacobian matrices of the individual domains as defined above, and all other blocks are filled with zeros because all cross-domain derivatives are zero. Since the overall J is a block matrix, we get that $\det(J) = \prod_{\delta \in \Delta} \det(J_\delta)$ (cf. [372]). Our overall volume element can thus be computed as follows:

$$\begin{aligned} \det(J) &= \prod_{\delta \in \Delta} \det(J_\delta) \\ &= \prod_{\delta \in \Delta} r_\delta^{n_\delta-1} \sin^{n_\delta-2}(\phi_{\delta,1}) \sin^{n_\delta-3}(\phi_{\delta,2}) \cdots \sin(\phi_{\delta,n_\delta-2}) \end{aligned}$$

The limits of the angle integrals for each domain are $[0, 2\pi]$ for the outermost and $[0, \pi]$ for all other integrals. Based on our constraint $\sum_{\delta \in \Delta} r_\delta \leq r$, we can derive the limits for the integrals over the r_δ as follows, assuming an arbitrarily ordered indexing $\delta_1, \dots, \delta_k$ of the domains:

Limits of the integrals

$$\begin{aligned} r_1 &\in [0, r] \\ r_2 &\in [0, r - r_1] \\ r_3 &\in [0, r - r_1 - r_2] \\ &\vdots \\ r_k &\in [0, r - \sum_{i=1}^{k-1} r_i] \end{aligned}$$

In general, we can write $\forall j \in \{1, \dots, k\} : r_j \in [0, r - \sum_{i=1}^{j-1} r_i]$. The overall coordinate change therefore looks like this:

Applying the overall coordinate change

$$\begin{aligned} V(r, \Delta) &= \int \cdots \int_H 1 \, dH \\ &= \underbrace{\int_{\phi_{1,n_1-1}=0}^{2\pi} \int_{\phi_{1,n_1-2}=0}^{\pi} \cdots \int_{\phi_{1,1}=0}^{\pi} \int_{r_1=0}^r}_{\delta=1} \\ &\quad \cdots \underbrace{\int_{\phi_{k,n_k-1}=0}^{2\pi} \int_{\phi_{k,n_k-2}=0}^{\pi} \cdots \int_{\phi_{k,1}=0}^{\pi} \int_{r_k=0}^{r - \sum_{i=1}^{k-1} r_i}}_{\delta=k} \\ &\quad \underbrace{r_1^{n_1-1} \sin^{n_1-2}(\phi_{1,1}) \cdots \sin(\phi_{1,n_1-2})}_{\delta=1} \\ &\quad \cdots \underbrace{r_k^{n_k-1} \sin^{n_k-2}(\phi_{k,1}) \cdots \sin(\phi_{k,n_k-2})}_{\delta=k} \\ &\quad \underbrace{dr_k d\phi_{k,1} \cdots d\phi_{k,n_k-1}}_{\delta=k} \cdots \underbrace{dr_1 d\phi_{1,1} \cdots d\phi_{1,n_1-1}}_{\delta=1} \\ &= \underbrace{\int_0^{2\pi} \int_0^\pi \cdots \int_0^\pi \sin^{n_1-2}(\phi_{1,1}) \cdots \sin(\phi_{1,n_1-2}) \, d\phi_{1,1} \cdots d\phi_{1,n_1-1}}_{\delta=1} \\ &\quad \cdots \underbrace{\int_0^{2\pi} \int_0^\pi \cdots \int_0^\pi \sin^{n_k-2}(\phi_{k,1}) \cdots \sin(\phi_{k,n_k-2}) \, d\phi_{k,1} \cdots d\phi_{k,n_k-1}}_{\delta=k} \end{aligned}$$

$$\int_0^r r_1^{n_1-1} \cdots \int_0^{r-\sum_{i=1}^{k-1} r_i} r_k^{n_k-1} dr_1 \cdots dr_k$$

Applying the helper
lemmata

By applying Lemma C.1 and Lemma C.3, we can derive a considerably more compact expression:

$$\begin{aligned} V(r, \Delta) &= \left(2 \cdot \frac{\pi^{\frac{n_1}{2}}}{\Gamma(\frac{n_1}{2})}\right) \cdots \left(2 \cdot \frac{\pi^{\frac{n_k}{2}}}{\Gamma(\frac{n_k}{2})}\right) \cdot \frac{r^n}{\Gamma(n+1)} \prod_{i=1}^k \Gamma(n_i) \\ &= \frac{r^n}{\Gamma(n+1)} \cdot \prod_{i=1}^k \left(2 \cdot \pi^{\frac{n_i}{2}} \cdot \frac{\Gamma(n_i)}{\Gamma(\frac{n_i}{2})}\right) \end{aligned}$$

Further
simplifications

We can simplify this formula further by using the identities $\forall n \in \mathbb{N} : \Gamma(n+1) = n!$ (step (i)), $\forall n \in \mathbb{N} : n! = n \cdot (n-1)!$ (step (ii)), and $\forall x \in \mathbb{R}^+ : \Gamma(x) \cdot x = \Gamma(x+1)$ (step (iii)). Moreover, we can change our notation by using $\prod_{i=0}^k \hat{=} \prod_{\delta \in \Delta}$. We therefore get the following result:

$$\begin{aligned} V(r, \Delta) &= \frac{r^n}{\Gamma(n+1)} \cdot \prod_{i=1}^k \left(2 \cdot \pi^{\frac{n_i}{2}} \cdot \frac{\Gamma(n_i)}{\Gamma(\frac{n_i}{2})}\right) \\ &\stackrel{(i)}{=} \frac{r^n}{n!} \cdot \prod_{\delta \in \Delta} \left(2 \cdot \pi^{\frac{n_\delta}{2}} \cdot \frac{(n_\delta - 1)!}{\Gamma(\frac{n_\delta}{2})}\right) \\ &\stackrel{(ii)}{=} \frac{r^n}{n!} \cdot \prod_{\delta \in \Delta} \left(2 \cdot \frac{n_\delta!}{n_\delta} \cdot \frac{\pi^{\frac{n_\delta}{2}}}{\Gamma(\frac{n_\delta}{2})}\right) \\ &= \frac{r^n}{n!} \cdot \prod_{\delta \in \Delta} \left(n_\delta! \cdot \frac{\pi^{\frac{n_\delta}{2}}}{\frac{n_\delta}{2} \cdot \Gamma(\frac{n_\delta}{2})}\right) \\ &\stackrel{(iii)}{=} \frac{r^n}{n!} \cdot \prod_{\delta \in \Delta} \left(n_\delta! \cdot \frac{\pi^{\frac{n_\delta}{2}}}{\Gamma(\frac{n_\delta}{2} + 1)}\right) \quad \square \end{aligned}$$

Adding salience
weights

Now that the heavy lifting has been done, we can generalize the result from Proposition 4.1 to the *weighted* combined distance d_C^Δ :

Proposition 4.2 (Volume of a Hyperball with Saliency Weights)

The size of a hyperball with radius r in a space with the weighted combined metric d_C , the domain structure Δ , and the set of weights W can be computed in the following way, where n is the overall number of dimensions, n_δ is the number of dimensions in domain δ , $\Gamma(\cdot)$ is Euler's gamma function, and $\delta(d)$ is the unique $\delta \in \Delta$ with $d \in \delta$:

$$V(r, \Delta, W) = \frac{1}{\prod_{d \in D} w_{\delta(d)} \sqrt{w_d}} \cdot \frac{r^n}{n!} \cdot \prod_{\delta \in \Delta} \left(n_\delta! \cdot \frac{\pi^{\frac{n_\delta}{2}}}{\Gamma(\frac{n_\delta}{2} + 1)}\right)$$

Weighting as
stretching

Proof. As Gärdenfors has already argued in [179, Section 4.7.2], putting weights on dimensions in a conceptual space is equivalent to stretching each dimension of the unweighted space by the weight assigned to it. This has also been illustrated for the two-dimensional case in Figure 4.2 in Section 4.1.1 of this dissertation.

If the overall radius of a ball is r , and some dimension has the weight w , then the farthest away any point x can be from the origin on this dimension must satisfy $w \cdot x = r$, i.e., $x = \frac{r}{w}$. That is, the ball needs to be stretched by a factor $\frac{1}{w}$ in the given dimension, thus its size also changes by a factor of $\frac{1}{w}$. A hyperball under the weighted metric is thus equivalent to a hyperellipse under the unweighted metric. In our case, the weight for any dimension d within a domain δ corresponds to $w_\delta \cdot \sqrt{w_d}$.

Computing the scaling factor

Let us denote by $\delta(d)$ the domain to which dimension d belongs. If we look at a point x with coordinates $(0, \dots, 0, x_d, 0, \dots, 0)$, then $d_C^{Delta}(0, x) = w_{\delta(d)} \cdot \sqrt{w_d \cdot x_d^2} = w_{\delta(d)} \cdot \sqrt{w_d} \cdot |x_d|$. If we multiply the size of the hyperball by $\frac{1}{w_{\delta(d)} \cdot \sqrt{w_d}}$ for each dimension d , we get the following result:

Stretching the hyperball

$$\begin{aligned} V(r, \Delta, W) &= \frac{1}{\prod_{d \in D} w_{\delta(d)} \sqrt{w_d}} \cdot V(r, \Delta) \\ &= \frac{1}{\prod_{d \in D} w_{\delta(d)} \sqrt{w_d}} \cdot \frac{r^n}{n!} \cdot \prod_{\delta \in \Delta} \left(n_\delta! \cdot \frac{\pi^{\frac{n_\delta}{2}}}{\Gamma(\frac{n_\delta}{2} + 1)} \right) \quad \square \end{aligned}$$

Now that we have a way of computing the size of a hyperball under the weighted combined metric, we can use this to derive the size of an entire α -cut. This requires us to insert the formula from Proposition 4.2 into Equation 4.7 from Section 4.1.1, which is re-printed below:

The size of an α -cut

$$\begin{aligned} V(\tilde{C}^\alpha) &= \sum_{i=0}^n \left(\sum_{\substack{\{d_1, \dots, d_i\} \\ \subseteq D}} \left(\prod_{d \in D \setminus \{d_1, \dots, d_i\}} b_d \right) \right. \\ &\quad \left. \cdot V\left(-\frac{1}{c} \cdot \ln\left(\frac{\alpha}{\mu_0}\right), \Delta_{\{d_1, \dots, d_i\}}, W\right) \right) \quad (4.7) \end{aligned}$$

Proposition 4.3 (Size of an α -Cut)

For a given fuzzified cuboid $\tilde{C} = \langle C, \mu_0, c, W \rangle$ and $\alpha \in [0, \mu_0]$, we can describe the size of its α -cut as follows, using $a_d = w_{\delta(d)} \cdot \sqrt{w_d} \cdot (p_d^+ - p_d^-) \cdot c$:

$$\begin{aligned} V(\tilde{C}^\alpha) &= \frac{1}{c^n \prod_{d \in D} w_{\delta(d)} \sqrt{w_d}} \\ &\quad \sum_{i=0}^n \left(\frac{(-1)^i \cdot \ln\left(\frac{\alpha}{\mu_0}\right)^i}{i!} \cdot \sum_{\substack{\{d_1, \dots, d_i\} \\ \subseteq D}} \left(\prod_{d \in D \setminus \{d_1, \dots, d_i\}} a_d \right) \cdot \right. \\ &\quad \left. \prod_{\substack{\delta \in \Delta \\ \Delta_{\{d_1, \dots, d_i\}}} \left(n_\delta! \cdot \frac{\pi^{\frac{n_\delta}{2}}}{\Gamma(\frac{n_\delta}{2} + 1)} \right) \right) \end{aligned}$$

Proof. We start by applying Proposition 4.2 to Equation 4.7:

Combining the two formulas

$$\begin{aligned}
V(\tilde{C}^\alpha) &= \sum_{i=0}^n \left(\sum_{\substack{\{d_1, \dots, d_i\} \\ \subseteq D}} \left(\prod_{d \in D \setminus \{d_1, \dots, d_i\}} b_d \right) \right. \\
&\quad \left. \cdot V \left(-\frac{1}{c} \cdot \ln \left(\frac{\alpha}{\mu_0} \right), W, \Delta_{\{d_1, \dots, d_i\}} \right) \right) \\
&= \sum_{i=0}^n \left(\sum_{\substack{\{d_1, \dots, d_i\} \\ \subseteq D}} \left(\prod_{d \in D \setminus \{d_1, \dots, d_i\}} b_d \right) \cdot \frac{1}{\prod_{d \in \{d_1, \dots, d_i\}} w_{\delta(d)} \sqrt{w_d}} \right. \\
&\quad \left. \cdot \frac{\left(-\frac{1}{c} \cdot \ln \left(\frac{\alpha}{\mu_0} \right) \right)^i}{i!} \cdot \prod_{\delta \in \Delta_{\{d_1, \dots, d_i\}}} \left(n_{\delta}! \cdot \frac{\pi^{\frac{n_{\delta}}{2}}}{\Gamma(\frac{n_{\delta}}{2} + 1)} \right) \right) \\
&= \sum_{i=0}^n \left(\sum_{\substack{\{d_1, \dots, d_i\} \\ \subseteq D}} \frac{\prod_{d \in D \setminus \{d_1, \dots, d_i\}} b_d}{\prod_{d \in \{d_1, \dots, d_i\}} w_{\delta(d)} \sqrt{w_d}} \right. \\
&\quad \left. \cdot \frac{(-1)^i \cdot \ln \left(\frac{\alpha}{\mu_0} \right)^i}{c^i \cdot i!} \cdot \prod_{\delta \in \Delta_{\{d_1, \dots, d_i\}}} \left(n_{\delta}! \cdot \frac{\pi^{\frac{n_{\delta}}{2}}}{\Gamma(\frac{n_{\delta}}{2} + 1)} \right) \right)
\end{aligned}$$

Some simplifications

A simple extension of both numerators and denominators in the inner sum with c^{n-i} and $\prod_{d \in D \setminus \{d_1, \dots, d_i\}} w_{\delta(d)} \sqrt{w_d}$ can be used to simplify this:

$$\begin{aligned}
V(\tilde{C}^\alpha) &= \frac{c^{n-i}}{c^{n-i}} \sum_{i=0}^n \left(\sum_{\substack{\{d_1, \dots, d_i\} \\ \subseteq D}} \frac{\prod_{d \in D \setminus \{d_1, \dots, d_i\}} w_{\delta(d)} \sqrt{w_d}}{\prod_{d \in D \setminus \{d_1, \dots, d_i\}} w_{\delta(d)} \sqrt{w_d}} \right. \\
&\quad \cdot \frac{\prod_{d \in D \setminus \{d_1, \dots, d_i\}} b_d}{\prod_{d \in \{d_1, \dots, d_i\}} w_{\delta(d)} \sqrt{w_d}} \\
&\quad \cdot \frac{c^{n-i} \cdot (-1)^i \cdot \ln \left(\frac{\alpha}{\mu_0} \right)^i}{c^{n-i} \cdot c^i \cdot i!} \\
&\quad \left. \cdot \prod_{\delta \in \Delta_{\{d_1, \dots, d_i\}}} \left(n_{\delta}! \cdot \frac{\pi^{\frac{n_{\delta}}{2}}}{\Gamma(\frac{n_{\delta}}{2} + 1)} \right) \right) \\
&= \sum_{i=0}^n \left(\sum_{\substack{\{d_1, \dots, d_i\} \\ \subseteq D}} \frac{c^{n-i} \cdot (-1)^i \cdot \ln \left(\frac{\alpha}{\mu_0} \right)^i}{c^{n-i} \cdot c^i \cdot i!} \right. \\
&\quad \cdot \frac{\left(\prod_{d \in D \setminus \{d_1, \dots, d_i\}} w_{\delta(d)} \sqrt{w_d} \right) \left(\prod_{d \in D \setminus \{d_1, \dots, d_i\}} b_d \right)}{\left(\prod_{d \in D \setminus \{d_1, \dots, d_i\}} w_{\delta(d)} \sqrt{w_d} \right) \left(\prod_{d \in \{d_1, \dots, d_i\}} w_{\delta(d)} \sqrt{w_d} \right)}
\end{aligned}$$

$$\begin{aligned}
 & \cdot \prod_{\substack{\delta \in \\ \Delta_{\{d_1, \dots, d_i\}}} \left(n_\delta! \cdot \frac{\pi^{\frac{n_\delta}{2}}}{\Gamma(\frac{n_\delta}{2} + 1)} \right) \\
 = & \sum_{i=0}^n \left(\sum_{\substack{\{d_1, \dots, d_i\} \\ \subseteq D}} \frac{(-1)^i \cdot \ln\left(\frac{\alpha}{\mu_0}\right)^i}{i!} \right. \\
 & \cdot \frac{\prod_{d \in D \setminus \{d_1, \dots, d_i\}} w_{\delta(d)} \cdot \sqrt{w_d} \cdot b_d \cdot c}{c^n \prod_{d \in D} w_{\delta(d)} \sqrt{w_d}} \\
 & \left. \cdot \prod_{\substack{\delta \in \\ \Delta_{\{d_1, \dots, d_i\}}} \left(n_\delta! \cdot \frac{\pi^{\frac{n_\delta}{2}}}{\Gamma(\frac{n_\delta}{2} + 1)} \right) \right)
 \end{aligned}$$

Finally, we can use the shorthand notation $a_d = w_{\delta(d)} \cdot \sqrt{w_d} \cdot (p_d^+ - p_d^-) \cdot c$ and move some terms in front of the sum:

Slight notational changes

$$\begin{aligned}
 V(\tilde{C}^\alpha) = & \frac{1}{c^n \prod_{d \in D} w_{\delta(d)} \sqrt{w_d}} \sum_{i=0}^n \left(\frac{(-1)^i \cdot \ln\left(\frac{\alpha}{\mu_0}\right)^i}{i!} \right. \\
 & \cdot \sum_{\substack{\{d_1, \dots, d_i\} \\ \subseteq D}} \left(\prod_{d \in D \setminus \{d_1, \dots, d_i\}} a_d \right) \\
 & \left. \cdot \prod_{\substack{\delta \in \\ \Delta_{\{d_1, \dots, d_i\}}} \left(n_\delta! \cdot \frac{\pi^{\frac{n_\delta}{2}}}{\Gamma(\frac{n_\delta}{2} + 1)} \right) \right) \square
 \end{aligned}$$

In Section 4.1.2, we have used the insights gained so far in order to compute the overall size of a fuzzified cuboid. This can be done by combining Proposition 4.3 with Equation 4.8 from Section 4.1.2, which is re-printed below:

The size of a fuzzified cuboid

$$M(\tilde{A}) = \int_0^1 V(\tilde{A}^\alpha) d\alpha \tag{4.8}$$

Moreover, we will use the following helper lemma in order to simplify the resulting equation:

Lemma C.4 (Simplification of an Integral over an Exponentiated Logarithm)

Let $n \in \mathbb{N}$. Then, $\int_0^1 \ln(x)^n dx = (-1)^n \cdot n!$

Proof. Let us make the following substitution: $x = e^t$ (implying $dx = e^t dt$ and changing the limits of the integral from 0 and 1 to $-\infty$ and 0, respectively). This results in the following transformation:

First substitution

$$I = \int_0^1 \ln(x)^n dx = \int_{-\infty}^0 \ln(e^t)^n \cdot e^t dt = \int_{-\infty}^0 t^n \cdot e^t dt$$

Second substitution Another substitution of $s = -t$ yields:

$$\begin{aligned} I &= \int_{-\infty}^0 t^n \cdot e^t dt = \int_0^{\infty} (-s)^n \cdot e^{-s} ds \\ &= \int_0^{\infty} (-1)^n \cdot s^n \cdot e^{-s} ds = (-1)^n \cdot \int_0^{\infty} s^n \cdot e^{-s} ds \end{aligned}$$

Using the gamma function We can now use the definition of the gamma function $\Gamma(z) = \int_0^{\infty} x^{z-1} \cdot e^{-x} dx$ and the fact that for positive integers, $\Gamma(z) = (z - 1)!$

$$I = (-1)^n \cdot \int_0^{\infty} s^n \cdot e^{-s} ds = (-1)^n \cdot \Gamma(n + 1) = (-1)^n \cdot n! \quad \square$$

Proposition 4.4 (Size of a Fuzzified Cuboid)

The size of a fuzzified cuboid \tilde{C} can be computed as follows:

$$\begin{aligned} M(\tilde{C}) &= \frac{\mu_0}{c^n \prod_{d \in D} w_{\delta(d)} \sqrt{w_d}} \sum_{i=0}^n \left(\sum_{\substack{\{d_1, \dots, d_i\} \\ \subseteq D}} \left(\prod_{d \in D \setminus \{d_1, \dots, d_i\}} a_d \right) \right. \\ &\quad \left. \cdot \prod_{\substack{\delta \in \Delta_{\{d_1, \dots, d_i\}}} \left(n_{\delta}! \cdot \frac{\pi^{\frac{n_{\delta}}{2}}}{\Gamma(\frac{n_{\delta}}{2} + 1)} \right) \right) \end{aligned}$$

Empty α -cuts and a substitution

Proof. We know that $\forall \alpha > \mu_0 : \tilde{C}^{\alpha} = \emptyset$. Therefore, $\int_0^1 V(\tilde{C}^{\alpha}) d\alpha = \int_0^{\mu_0} V(\tilde{C}^{\alpha}) d\alpha$. By substituting $x = \frac{\alpha}{\mu_0}$ (implying $d\alpha = \mu_0 \cdot dx$), we get the following intermediate result:

$$M(\tilde{C}) = \int_0^{\mu_0} V(\tilde{C}^{\alpha}) d\alpha = \int_0^1 V(\tilde{C}^{\mu_0 \cdot x}) \cdot \mu_0 dx = \mu_0 \cdot \int_0^1 V(\tilde{C}^{\mu_0 \cdot x}) dx$$

Applying Proposition 4.3

Simply inserting our formula for $V(\tilde{C}^{\alpha})$ from Proposition 4.3 yields the following expression:

$$\begin{aligned} M(\tilde{C}) &= \mu_0 \cdot \int_0^1 \frac{1}{c^n \prod_{d \in D} w_{\delta(d)} \sqrt{w_d}} \\ &\quad \sum_{i=0}^n \left(\frac{(-1)^i \cdot \ln\left(\frac{\mu_0 \cdot x}{\mu_0}\right)^i}{i!} \cdot \sum_{\substack{\{d_1, \dots, d_i\} \\ \subseteq D}} \left(\prod_{d \in D \setminus \{d_1, \dots, d_i\}} a_d \right) \right. \\ &\quad \left. \cdot \prod_{\substack{\delta \in \Delta_{\{d_1, \dots, d_i\}}} \left(n_{\delta}! \cdot \frac{\pi^{\frac{n_{\delta}}{2}}}{\Gamma(\frac{n_{\delta}}{2} + 1)} \right) \right) dx \end{aligned}$$

Moving the integral sign

Since the integral only concerns $\ln\left(\frac{\mu_0 \cdot x}{\mu_0}\right)^i$, we can move it inside the outer sum:

$$M(\tilde{C}) = \frac{\mu_0}{c^n \prod_{d \in D} w_{\delta(d)} \sqrt{w_d}} \sum_{i=0}^n \left(\sum_{\substack{\{d_1, \dots, d_i\} \\ \subseteq D}} \left(\prod_{\substack{d \in \\ D \setminus \{d_1, \dots, d_i\}}} a_d \right) \right. \\ \cdot \prod_{\substack{\delta \in \\ \Delta_{\{d_1, \dots, d_i\}}} \left(n_{\delta}! \cdot \frac{\pi^{\frac{n_{\delta}}{2}}}{\Gamma(\frac{n_{\delta}}{2} + 1)} \right) \\ \cdot \int_0^1 \frac{(-1)^i \cdot \ln(x)^i}{i!} dx$$

We can now apply Lemma C.4 to the integral, which allows us to make the following simplification:

Applying Lemma C.4

$$\int_0^1 \frac{(-1)^i \cdot \ln(x)^i}{i!} dx = \frac{(-1)^i \cdot (-1)^i \cdot i!}{i!} = (-1)^{2i} = ((-1)^2)^i = 1^i = 1$$

Inserting this into $M(\tilde{C})$ results in the following formula:

$$M(\tilde{C}) = \frac{\mu_0}{c^n \prod_{d \in D} w_{\delta(d)} \sqrt{w_d}} \sum_{i=0}^n \left(\sum_{\substack{\{d_1, \dots, d_i\} \\ \subseteq D}} \left(\prod_{\substack{d \in \\ D \setminus \{d_1, \dots, d_i\}}} a_d \right) \right. \\ \cdot \prod_{\substack{\delta \in \\ \Delta_{\{d_1, \dots, d_i\}}} \left(n_{\delta}! \cdot \frac{\pi^{\frac{n_{\delta}}{2}}}{\Gamma(\frac{n_{\delta}}{2} + 1)} \right)$$

□

C.2 SUBSETHOOD

In Section 4.2.1, we have considered a formal notion of crisp subsethood for both crisp and fuzzy conceptual regions:

Crisp subsethood

Definition 4.3 (Subsethood for Crisp Sets)

Let S_1 and S_2 be two crisp sets in a conceptual space CS . We say that S_1 is a subset of S_2 ($S_1 \subseteq S_2$) if and only if $\forall x \in CS : (x \in S_1 \Rightarrow x \in S_2)$.

Definition 4.4 (Subsethood for Fuzzy Sets)

Let \tilde{S}_1 and \tilde{S}_2 be two fuzzy sets in a conceptual space CS . We say that \tilde{S}_1 is a subset of \tilde{S}_2 ($\tilde{S}_1 \subseteq \tilde{S}_2$) if and only if $\forall x \in CS : \mu_{\tilde{S}_1}(x) \leq \mu_{\tilde{S}_2}(x)$. This is equivalent to requiring subsethood according to Definition 4.3 for all α -cuts \tilde{S}_1^α and \tilde{S}_2^α .

The following proposition gives us a list of necessary and jointly sufficient conditions for crisp subsethood of two concepts.

Conditions for subsethood

Proposition 4.5 (Conditions for Crisp Subsethood)

Let $\tilde{S}_1 = \langle S_1, \mu_0^{(1)}, c^{(1)}, W^{(1)} \rangle$ and $\tilde{S}_2 = \langle S_2, \mu_0^{(2)}, c^{(2)}, W^{(2)} \rangle$ be two concepts. Then, $\tilde{S}_1 \subseteq \tilde{S}_2$ if and only if all of the following conditions are fulfilled:

1. $\mu_0^{(1)} \leq \mu_0^{(2)}$
2. $S_1 \subseteq \tilde{S}_2^{\mu_0^{(1)}}$
3. $\Delta_{S_2} \subseteq \Delta_{S_1}$
4. $\forall d \in D_{S_2} : c^{(1)} \cdot w_{\delta(d)}^{(1)} \cdot \sqrt{w_d^{(1)}} \geq c^{(2)} \cdot w_{\delta(d)}^{(2)} \cdot \sqrt{w_d^{(2)}}$

Proof. We prove the two directions of the equivalence separately.

Necessity of the conditions

$\tilde{S}_1 \subseteq \tilde{S}_2 \Rightarrow$ **conditions**

Let $\tilde{S}_1 \subseteq \tilde{S}_2$, i.e., $\forall x \in CS : \mu_{\tilde{S}_1}(x) \leq \mu_{\tilde{S}_2}(x)$. We now show that each of the four conditions is fulfilled.

First condition

Condition 1 (μ_0): If $\mu_0^{(1)} > \mu_0^{(2)}$, then $\exists x \in S_1 : \mu_{\tilde{S}_1}(x) = \mu_0^{(1)} > \mu_0^{(2)} \geq \mu_{\tilde{S}_2}(x)$, which is a contradiction to $\tilde{S}_1 \subseteq \tilde{S}_2$. Therefore, $\mu_0^{(1)} \leq \mu_0^{(2)}$.

Second condition

Condition 2 (S): If $S_1 \not\subseteq \tilde{S}_2^{\mu_0^{(1)}}$, then $\exists x \in S_1 : \mu_{\tilde{S}_1}(x) = \mu_0^{(1)} > \mu_{\tilde{S}_2}(x)$, which is a contradiction to $\tilde{S}_1 \subseteq \tilde{S}_2$. Therefore, $S_1 \subseteq \tilde{S}_2^{\mu_0^{(1)}}$.

Third condition

Condition 3 (Δ): If $\Delta_{S_2} \not\subseteq \Delta_{S_1}$, then $\exists \delta \in \Delta_{S_2} : \delta \notin \Delta_{S_1}$. Pick any $d \in \delta$ and any $x \in CS$ with $x_d \gg p_d^+$ for all $C \in S_2$. Thus, even for small weights w_δ and w_d , we get $\min_{y \in S_2} d_C^\Delta(x, y, W^{(2)}) \gg 0$. A large value of x_d leads to a large value of $\min_{y \in S_2} d_C^{\Delta_{S_2}}(x, y, W^{(2)})$, i.e., a small membership $\mu_{\tilde{S}_2}(x)$. On the other hand, as $\delta \notin \Delta_{S_1}$, the value of x_d does not affect $\mu_{\tilde{S}_1}(x)$. If x_d is chosen large enough, we can thus construct an example with $\mu_{\tilde{S}_1}(x) > \mu_{\tilde{S}_2}(x)$, which is a contradiction to $\tilde{S}_1 \subseteq \tilde{S}_2$. Therefore, $\Delta_{S_2} \subseteq \Delta_{S_1}$.

Fourth condition

Condition 4 (c and W): Assume for some $d^* \in D_2$ that $c^{(1)} \cdot w_{\delta(d^*)}^{(1)} \cdot \sqrt{w_{d^*}^{(1)}} < c^{(2)} \cdot w_{\delta(d^*)}^{(2)} \cdot \sqrt{w_{d^*}^{(2)}}$. In the most simple case, $S_1 = S_2$. Pick $y \in S_1 = S_2$ on the upper border of S_1 with respect to d^* (i.e., such that $y_{d^*} = \max_{i \in \{1, \dots, m\}} p_{id}^+$) and define a point $x \in CS$ by setting $\forall d \in D \setminus \{d^*\} : x_d = y_d$ and $x_{d^*} > y_{d^*}$. The membership of x in \tilde{S}_2 can be written as follows:

$$\begin{aligned}
 \mu_{\tilde{S}_2}(x) &= \mu_0^{(2)} \cdot e^{-c^{(2)} \cdot \min_{y \in S_2} \sum_{\delta \in \Delta_2} w_\delta^{(2)} \cdot \sqrt{\sum_{d \in \delta} w_d^{(2)} \cdot |x_d - y_d|^2}} \\
 &= \mu_0^{(2)} \cdot e^{-\min_{y \in S_2} \sum_{\delta \in \Delta_2} \sqrt{\sum_{d \in \delta} \left(c^{(2)} \cdot w_\delta^{(2)} \cdot \sqrt{w_d^{(2)}} \right)^2 \cdot |x_d - y_d|^2}} \\
 &\stackrel{(i)}{=} \mu_0^{(2)} \cdot e^{-\left(c^{(2)} \cdot w_{\delta(d^*)}^{(2)} \cdot \sqrt{w_{d^*}^{(2)}} \right) \cdot |x_{d^*} - y_{d^*}|} \\
 &\stackrel{(ii)}{=} \mu_0^{(2)} \cdot e^{-\left(c^{(1)} \cdot w_{\delta(d^*)}^{(1)} \cdot \sqrt{w_{d^*}^{(1)}} + \epsilon \right) \cdot |x_{d^*} - y_{d^*}|} \\
 &= \mu_0^{(2)} \cdot e^{-c^{(1)} \cdot w_{\delta(d^*)}^{(1)} \cdot \sqrt{w_{d^*}^{(1)}} \cdot |x_{d^*} - y_{d^*}|} \cdot e^{-\epsilon \cdot |x_{d^*} - y_{d^*}|} \tag{C.2}
 \end{aligned}$$

In step (i), we use that $\forall d \in D \setminus \{d^*\} : x_d = y_d$, and hence $|x_d - y_d| = 0$. In step (ii), we use our assumption that $c^{(1)} \cdot w_{\delta(d^*)}^{(1)} \cdot \sqrt{w_{d^*}^{(1)}} < c^{(2)} \cdot w_{\delta(d^*)}^{(2)} \cdot \sqrt{w_{d^*}^{(2)}}$, where $\epsilon > 0$ denotes their difference.

The insight that $\forall d \in D \setminus \{d^*\} : |x_d - y_d| = 0$ allows us to express $\mu_{\tilde{S}_1}(x)$ in a similar way:

$$\mu_{\tilde{S}_1}(x) = \mu_0^{(1)} \cdot e^{-c^{(1)} \cdot w_{\delta(d^*)}^{(1)} \cdot \sqrt{w_{d^*}^{(1)}} \cdot |x_{d^*} - y_{d^*}|} \quad (\text{C.3})$$

We can see based on equations C.2 and C.3 that $\mu_{\tilde{S}_2}(x)$ is smaller than $\mu_{\tilde{S}_1}(x)$ if and only if $\mu_0^{(1)} \cdot e^{-\epsilon \cdot |x_{d^*} - y_{d^*}|} < \mu_0^{(2)}$, which happens if $|x_{d^*} - y_{d^*}| > -\frac{1}{\epsilon} \ln \left(\frac{\mu_0^{(2)}}{\mu_0^{(1)}} \right)$. Thus, if we pick x_{d^*} large enough, then $\mu_{\tilde{S}_1}(x) > \mu_{\tilde{S}_2}(x)$, which is a contradiction to $\tilde{S}_1 \subseteq \tilde{S}_2$.

Let us now remove the simplifying assumption of $S_1 = S_2$. If we construct x based on $y^{(1)} \in S_1$ then we still know that $\forall d \in D \setminus \{d^*\} : |x_d - y_d^{(1)}| = 0$. However, for the optimal $y^{(2)} \in S_2$ that minimizes $d_C^{\Delta_2}(x, y^{(2)})$, we only know that $|x_d - y_d^{(2)}| \geq 0 = |x_d - y_d^{(1)}|$. This means that the distance from x to the closest $y^{(2)} \in S_2$ with respect to all dimensions $d \in D \setminus \{d^*\}$ is not smaller than to the closest $y^{(1)} \in S_1$. Since $y^{(1)} \neq y^{(2)}$, we now need to consider cases where $|x_{d^*} - y_{d^*}^{(1)}| \geq |x_{d^*} - y_{d^*}^{(2)}|$. In this case, we know that $\mu_{\tilde{S}_2}(x)$ is smaller than $\mu_{\tilde{S}_1}(x)$ under the following condition:

Considering the general case

$$\begin{aligned} & \left[c^{(1)} w_{\delta(d^*)}^{(1)} \sqrt{w_{d^*}^{(1)}} |x_{d^*} - y_{d^*}^{(1)}| < c^{(2)} w_{\delta(d^*)}^{(2)} \sqrt{w_{d^*}^{(2)}} |x_{d^*} - y_{d^*}^{(2)}| \right. \\ & \quad \left. = \left(c^{(1)} w_{\delta(d^*)}^{(1)} \sqrt{w_{d^*}^{(1)}} + \epsilon \right) |x_{d^*} - y_{d^*}^{(2)}| \right] \\ \Leftrightarrow & \left[\epsilon |x_{d^*} - y_{d^*}^{(2)}| > c^{(1)} w_{\delta(d^*)}^{(1)} \sqrt{w_{d^*}^{(1)}} \underbrace{\left(|x_{d^*} - y_{d^*}^{(1)}| - |x_{d^*} - y_{d^*}^{(2)}| \right)}_{=a} \right] \end{aligned}$$

One can easily see that a is a constant independent of x_{d^*} . Since also $c^{(1)} w_{\delta(d^*)}^{(1)} \sqrt{w_{d^*}^{(1)}}$ is a constant, we can find a large enough x_{d^*} to make the above inequality true. Then, $\forall d \in D_{S_2} : c^{(1)} \cdot w_{\delta(d)}^{(1)} \cdot \sqrt{w_d^{(1)}} \geq c^{(2)} \cdot w_{\delta(d)}^{(2)} \cdot \sqrt{w_d^{(2)}}$, which is a contradiction to $\tilde{S}_1 \subseteq \tilde{S}_2$.

Conditions $\Rightarrow (\tilde{S}_1 \subseteq \tilde{S}_2)$

Let \tilde{S}_1, \tilde{S}_2 be two concepts which fulfill all four conditions, and let $x \in CS$ be arbitrary but fixed. Let us transform $\mu_{\tilde{S}_1}(x)$ as follows:

Joint sufficiency of the conditions

$$\begin{aligned} \mu_{\tilde{S}_1}(x) &= \mu_0^{(1)} \cdot e^{-c^{(1)} \cdot \min_{y \in S_1} \sum_{\delta \in \Delta_1} w_{\delta}^{(1)} \cdot \sqrt{\sum_{d \in \delta} w_d^{(1)} \cdot |x_d - y_d|^2}} \\ &\stackrel{(i)}{\leq} \mu_0^{(1)} \cdot e^{-c^{(1)} \cdot \min_{y \in \tilde{S}_2^{\mu_0^{(1)}}} \sum_{\delta \in \Delta_1} w_{\delta}^{(1)} \cdot \sqrt{\sum_{d \in \delta} w_d^{(1)} \cdot |x_d - y_d|^2}} \end{aligned}$$

$$\begin{aligned}
 & \stackrel{(ii)}{\leq} \mu_0^{(1)} \cdot e^{-c^{(1)} \cdot \min_{y \in \tilde{S}_2^{\mu_0^{(1)}}} \sum_{\delta \in \Delta_2} w_\delta^{(1)} \cdot \sqrt{\sum_{d \in \delta} w_d^{(1)} \cdot |x_d - y_d|^2}} \\
 & = \mu_0^{(1)} \cdot e^{-\min_{y \in \tilde{S}_2^{\mu_0^{(1)}}} \sum_{\delta \in \Delta_2} \sqrt{\sum_{d \in \delta} \left(c^{(1)} \cdot w_\delta^{(1)} \cdot \sqrt{w_d^{(1)}} \right)^2 \cdot |x_d - y_d|^2}} \\
 & \stackrel{(iii)}{\leq} \mu_0^{(1)} \cdot e^{-\min_{y \in \tilde{S}_2^{\mu_0^{(1)}}} \sum_{\delta \in \Delta_2} \sqrt{\sum_{d \in \delta} \left(c^{(2)} \cdot w_\delta^{(2)} \cdot \sqrt{w_d^{(2)}} \right)^2 \cdot |x_d - y_d|^2}} \\
 & = \mu_0^{(1)} \cdot e^{-c^{(2)} \cdot \min_{y \in \tilde{S}_2^{\mu_0^{(1)}}} \sum_{\delta \in \Delta_2} w_\delta^{(2)} \cdot \sqrt{\sum_{d \in \delta} w_d^{(2)} \cdot |x_d - y_d|^2}} \\
 & = f(x)
 \end{aligned}$$

In step (i), we applied $S_1 \subseteq \tilde{S}_2^{\mu_0^{(1)}}$ (condition 2), which also implicitly uses that $\mu_0^{(1)} \leq \mu_0^{(2)}$ (condition 1). In step (ii), we then used that $\Delta_{S_2} \subseteq \Delta_{S_1}$ (condition 3). Finally, in step (iii), we used that $\forall d \in D_{S_2} : c^{(1)} \cdot w_{\delta(d)}^{(1)} \cdot \sqrt{w_d^{(1)}} \geq c^{(2)} \cdot w_{\delta(d)}^{(2)} \cdot \sqrt{w_d^{(2)}}$ (condition 4).

One can easily see that $\forall x \in \tilde{S}_2^{\mu_0^{(1)}} : f(x) = \mu_0^{(1)} \leq \mu_{\tilde{S}_2}(x)$. Thus, we only need to consider $x \notin \tilde{S}_2^{\mu_0^{(1)}}$. Now for each $x \notin \tilde{S}_2^{\mu_0^{(1)}}$ (i.e., with $\mu_{\tilde{S}_2}(x) < \mu_0^{(1)}$) there is an $z \in CS$ with $B_{d_C^{\Delta_2}}(x, z, y)$ for the closest $y \in S_2$ and $\mu_{\tilde{S}_2}(z) = \mu_0^{(1)}$. This is true because $\mu_{\tilde{S}_2}(x) = \mu_0^{(1)} \leq \mu_0^{(2)} = \mu_{\widetilde{widetilde{S}_2}}(y)$ and since $\mu_{\tilde{S}_2}(x)$ continuously decreases between y and x . We can now split $\mu_{\tilde{S}_2}(x)$ into two parts:

$$\begin{aligned}
 \mu_{\tilde{S}_2}(x) & = \mu_0^{(2)} \cdot e^{-c^{(2)} \cdot d_C^{\Delta_{S_2}}(x, y, W^{(2)})} \\
 & \stackrel{(i)}{=} \mu_0^{(2)} \cdot e^{-c^{(2)} \cdot \left(d_C^{\Delta_{S_2}}(x, z, W^{(2)}) + d_C^{\Delta_{S_2}}(z, y, W^{(2)}) \right)} \\
 & = \mu_0^{(2)} \cdot e^{-c^{(2)} \cdot d_C^{\Delta_{S_2}}(z, y, W^{(2)})} \cdot e^{-c^{(2)} \cdot d_C^{\Delta_{S_2}}(x, z, W^{(2)})} \\
 & \stackrel{(ii)}{=} \mu_0^{(1)} \cdot e^{-c^{(2)} \cdot d_C^{\Delta_{S_2}}(x, z, W^{(2)})}
 \end{aligned}$$

Step (i) is based on the definition of betweenness (cf. Definition 2.2) and step (ii) uses that $\mu_{\tilde{S}_2}(z) = \mu_0^{(1)}$. So instead of calculating $\mu_{\tilde{S}_2}(x)$ based on the distance between x and S_2 , we can calculate $\mu_{\tilde{S}_2}(x)$ based on the distance between x and $\tilde{S}_2^{\mu_0^{(1)}}$ and rescale the result, such that it is at most $\mu_0^{(1)}$. Obviously, $z \in \tilde{S}_2^{\mu_0^{(1)}}$, and since $B_{d_C^{\Delta_2}}(x, z, y)$, it is also the point in $\tilde{S}_2^{\mu_0^{(1)}}$ which minimizes the distance to x . Therefore, $f(x) = \mu_{\tilde{S}_2}(x)$ for $x \notin \tilde{S}_2^{\mu_0^{(1)}}$. Thus $\mu_{\tilde{S}_1}(x) \leq f(x) \leq \mu_{\tilde{S}_2}(x)$ for all $x \in CS$, which means that $\tilde{S}_1 \subseteq \tilde{S}_2$. \square

C.3 SIMILARITY

In Section 4.4.2, we introduced two possible definitions for the similarity of concepts, based on subsethood and the Jaccard index, respectively. For clarity, we re-print our definition of a similarity function.

Criteria for similarity functions

Definition 4.8 (Similarity Function)

A function $Sim(\tilde{S}_1, \tilde{S}_2) \in [0, 1]$ is called a similarity function, if it fulfills the following criteria for all concepts \tilde{S}_1, \tilde{S}_2 :

1. $(Sim(\tilde{S}_1, \tilde{S}_2) = 1) \Rightarrow (Sub(\tilde{S}_1, \tilde{S}_2) = 1)$
2. $(\tilde{S}_1 = \tilde{S}_2) \Rightarrow (Sim(\tilde{S}_1, \tilde{S}_2) = 1)$
3. $(\tilde{S}_1 \subseteq \tilde{S}_2) \Rightarrow (Sim(\tilde{S}_1, \tilde{S}_2) \geq Sim(\tilde{S}_2, \tilde{S}_1))$

Let us first consider the option of re-using the definition of conceptual subsethood from Section 4.2.1.

Similarity as subsethood

Proposition 4.6 (Sim_S is a Similarity Function)

$Sim_S(\tilde{S}_1, \tilde{S}_2) = Sub(\tilde{S}_1, \tilde{S}_2)$ is a similarity function according to Definition 4.8.

Proof. Let us consider each property from Definition 4.8 individually.

1. Trivial, since $Sim_S(\tilde{S}_1, \tilde{S}_2) = Sub(\tilde{S}_1, \tilde{S}_2)$.
2. Let $\tilde{S}_1 = \tilde{S}_2$. Then also $\tilde{S}_1 \subseteq \tilde{S}_2$ and therefore $Sim_S(\tilde{S}_1, \tilde{S}_2) = Sub(\tilde{S}_1, \tilde{S}_2) = 1$.
3. $\tilde{S}_1 \subseteq \tilde{S}_2$ implies that $Sub(\tilde{S}_1, \tilde{S}_2) = 1$ which implies by definition that $Sim_S(\tilde{S}_1, \tilde{S}_2) = 1$. Because $Sim_S(\tilde{S}_2, \tilde{S}_1) \leq 1$, we get that $Sim_S(\tilde{S}_1, \tilde{S}_2) \geq Sim_S(\tilde{S}_2, \tilde{S}_1)$. \square

Let us now consider an alternative definition for the similarity of concepts based on the Jaccard index:

Similarity based on the Jaccard index

Proposition 4.7 (Sim_J is a Similarity Function)

$Sim_J(\tilde{S}_1, \tilde{S}_2) = \frac{M(I(\tilde{S}_1, \tilde{S}_2))}{M(U(\tilde{S}_1, \tilde{S}_2))}$ is a similarity function according to Definition 4.8.

Proof. Let us consider each property from Definition 4.8 individually.

1. Let $\tilde{S}_I = I(\tilde{S}_1, \tilde{S}_2)$ and $\tilde{S}_U = U(\tilde{S}_1, \tilde{S}_2)$. As we know from Definitions 3.2 and 3.6, both intersection and union use the same way of estimating updated weights and sensitivity parameters, hence $c^{(I)} = c^{(U)}$ and $W^{(I)} = W^{(U)}$. It is easy to see that $\mu_0^{(I)} \leq \min(\mu_0^{(1)}, \mu_0^{(2)}) \leq \max(\mu_0^{(1)}, \mu_0^{(2)}) = \mu_0^{(U)}$. Moreover, $|S_I| \leq |S_U|$, because S_U is the star-shaped closure of all cuboids from S_1 and S_2 , while S_I is based on the highest intersecting α -cut and only contains points between the two original concepts.. Therefore, $Sim_J(\tilde{S}_1, \tilde{S}_2) = 1$ can only happen if

$\tilde{S}_I = \tilde{S}_U$, which implies that also $\mu_0^{(I)} = \mu_0^{(U)}$. The observation that $\mu_0^{(I)} = \mu_0^{(U)}$ implies that $\mu_0^{(1)} = \mu_0^{(2)}$ (as $\mu_0^{(I)} \leq \min(\mu_0^{(1)}, \mu_0^{(2)})$ and $\mu_0^{(U)} = \max(\mu_0^{(1)}, \mu_0^{(2)})$) and that $S_1 \cap S_2 \neq \emptyset$. If $\tilde{S}_I = \tilde{S}_U$, then also $S_I = S_U$. Then, $I(S_1, S_2) = S_I$ (because $S_1 \cap S_2 \neq \emptyset$ and $\mu_0^{(1)} = \mu_0^{(2)}$) and $U(S_1, S_2) = S_U$ (by definition of the union). Thus, $I(S_1, S_2) = U(S_1, S_2)$. This is only possible if $S_1 = S_2$. Then, $Sub(\tilde{S}_1, \tilde{S}_2) = \frac{M(I(\tilde{S}_1, \tilde{S}_2))_2}{M(\tilde{S}_1)_2} = \frac{M(\tilde{S}_1)_2}{M(\tilde{S}_1)_2} = 1$ (where we use $M(\tilde{S})_i$ to denote that $W^{(i)}$ and $c^{(i)}$ are used for computing the size).

2. Let $\tilde{S}_1 = \tilde{S}_2$. Obviously, $I(\tilde{S}_1, \tilde{S}_2) = \tilde{S}_1 = U(\tilde{S}_1, \tilde{S}_2)$ and therefore $Sim_J(\tilde{S}_1, \tilde{S}_2) = \frac{M(I(\tilde{S}_1, \tilde{S}_2))}{M(U(\tilde{S}_1, \tilde{S}_2))} = \frac{M(\tilde{S}_1)}{\tilde{S}_1} = 1$
3. Because $I(\tilde{S}_1, \tilde{S}_2) = I(\tilde{S}_2, \tilde{S}_1)$ and $U(\tilde{S}_1, \tilde{S}_2) = U(\tilde{S}_2, \tilde{S}_1)$, we get that $Sim_J(\tilde{S}_1, \tilde{S}_2) = Sim_J(\tilde{S}_2, \tilde{S}_1)$ for all \tilde{S}_1, \tilde{S}_2 . Thus, the consequent of the implication is always true. \square

C.4 BETWEENNESS

*Infimum-based
degree of betweenness*

In Section 4.5.2, we have developed two definitions for a soft degree of fuzzy betweenness. Let us first consider B_{soft}^{inf} , which is defined based on an infimum over α -cuts:

Definition 4.9 (Infimum-Based Betweenness)

Let \tilde{S}_1, \tilde{S}_2 , and \tilde{S}_3 be three concepts defined on the same set of domains Δ_S . The soft degree to which \tilde{S}_2 lies between \tilde{S}_1 and \tilde{S}_3 can be computed as follows:

$$B_{soft}^{inf}(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3) = \inf_{\alpha \in [0,1]} \inf_{y \in \tilde{S}_2^\alpha} \sup_{x \in \tilde{S}_1^\alpha} \sup_{z \in \tilde{S}_3^\alpha} \frac{d_C^{\Delta_S}(x, z, W^{(2)})}{d_C^{\Delta_S}(x, y, W^{(2)}) + d_C^{\Delta_S}(y, z, W^{(2)})}$$

*Pathological failures
of B_{soft}^{inf}*

We also illustrated in Section 4.5.2 that B_{soft}^{inf} yields some rather unintuitive results under certain circumstances. One of these cases is now treated more formally in Proposition 4.8:

Proposition 4.8 (Problematic Case for B_{soft}^{inf})

Let \tilde{S}_1, \tilde{S}_2 , and \tilde{S}_3 be three concepts. Assume we find a dimension $d^* \in \delta^* \in \Delta_S$ for which the following is true:

$$c^{(2)} w_{\delta^*}^{(2)} \sqrt{w_{d^*}^{(2)}} < \min \left(c^{(1)} w_{\delta^*}^{(1)} \sqrt{w_{d^*}^{(1)}}, c^{(3)} w_{\delta^*}^{(3)} \sqrt{w_{d^*}^{(3)}} \right)$$

Then, $B_{soft}^{inf}(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3) = 0$.

*α -cuts as
 ϵ -neighborhoods*

Proof. We know from Lemma 2.4 that each α -cut of a given concept $\tilde{S} = \langle S, \mu_0, c, W \rangle$ is equivalent to an ϵ -neighborhood of its core S :

$$\tilde{S}^\alpha = \left\{ x \in CS \mid \min_{y \in S} d_C^{\Delta_S}(x, y, W) \leq -\frac{1}{c} \cdot \ln \left(\frac{\alpha}{\mu_0} \right) \right\}$$

Let us for now only consider points $x \in CS$ that differ from their closest point $y \in S$ only with respect to the dimension $d^* \in \delta^*$ (i.e.,

$\forall d \in D \setminus \{d^*\} : x_d = y_d$). Without loss of generality, assume that $x_{d^*} > y_{d^*}$. The weighted combined metric between x and y can then be computed as follows:

$$\begin{aligned} \min_{y \in S} d_C^{\Delta S}(x, y, W) &= \min_{y \in S} \sum_{\delta \in \Delta_S} w_\delta \cdot \sqrt{\sum_{d \in \delta} w_d \cdot |x_d - y_d|^2} \\ &= w_{\delta^*} \sqrt{w_{d^*} (x_{d^*} - y_{d^*})^2} \\ &= w_{\delta^*} \sqrt{w_{d^*}} (x_{d^*} - y_{d^*}) \end{aligned}$$

The largest value of x_{d^*} such that $x \in \tilde{S}^\alpha$ can thus be obtained with the following transformations:

*Upper border of the α -cut with respect to d^**

$$\begin{aligned} \left(\min_{y \in S} d_C^{\Delta S}(x, y, W) = -\frac{1}{c} \cdot \ln \left(\frac{\alpha}{\mu_0} \right) \right) \\ \Leftrightarrow \left(w_{\delta^*} \sqrt{w_{d^*}} (x_{d^*} - y_{d^*}) = -\frac{1}{c} \cdot \ln \left(\frac{\alpha}{\mu_0} \right) \right) \\ \Leftrightarrow \left(x_{d^*} - y_{d^*} = -\frac{1}{c w_{\delta^*} \sqrt{w_{d^*}}} \cdot (\ln(\alpha) - \ln(\mu_0)) \right) \\ \Leftrightarrow \left(x_{d^*} = y_{d^*} - \frac{\ln(\alpha)}{c w_{\delta^*} \sqrt{w_{d^*}}} + \frac{\ln(\mu_0)}{c w_{\delta^*} \sqrt{w_{d^*}}} \right) \quad (C.4) \end{aligned}$$

The coordinate x_{d^*} thus increases logarithmically with a decreasing value of α , mediated by a factor of $\frac{1}{c w_{\delta^*} \sqrt{w_{d^*}}}$. The other two terms are independent of α and are thus constants in our considerations.

The condition of this proposition assumes that for $d^* \in \delta^*$, the following is true with respect to the three concepts \tilde{S}_1 , \tilde{S}_2 , and \tilde{S}_3 :

Growth rates of the three α -cuts

$$c^{(2)} w_{\delta^*}^{(2)} \sqrt{w_{d^*}^{(2)}} < \min \left(c^{(1)} w_{\delta^*}^{(1)} \sqrt{w_{d^*}^{(1)}}, c^{(3)} w_{\delta^*}^{(3)} \sqrt{w_{d^*}^{(3)}} \right)$$

This implies the following relation:

$$\frac{1}{c^{(2)} w_{\delta^*}^{(2)} \sqrt{w_{d^*}^{(2)}}} > \max \left(\frac{1}{c^{(1)} w_{\delta^*}^{(1)} \sqrt{w_{d^*}^{(1)}}}, \frac{1}{c^{(3)} w_{\delta^*}^{(3)} \sqrt{w_{d^*}^{(3)}}} \right)$$

This means that with decreasing value of α , $x_{d^*}^{(2)}$ grows faster than both $x_{d^*}^{(1)}$ and $x_{d^*}^{(3)}$. Even if it is smaller for large values of α (due to the location of S_2 , which influences y_{d^*} , or due to the value of $\mu_0^{(2)}$), it will eventually outgrow them as $\alpha \rightarrow 0$. Figure C.1 illustrates this case, where $x_{d^*}^{(2)} > \max(x_{d^*}^{(1)}, x_{d^*}^{(3)})$ for a given fixed value of α .

Recall from Definition 4.9 that B_{soft}^{inf} is defined based on the pairwise distances between $x^{(1)}$, $x^{(2)}$, and $x^{(3)}$. From Figure C.1, we can see that these distances can be computed as follows, where we use the

Pairwise distances

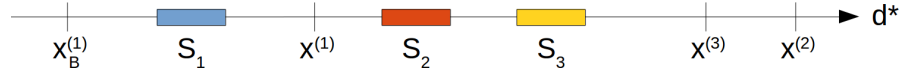


Figure C.1: Illustration of distance computations with respect to d^* .

shorthand notations $d(x^{(i)}, S_i) = \min_{y^{(i)} \in S_i} d_C^{\Delta S}(x^{(i)}, y^{(i)}, W^{(2)})$, and $d(S_i, S_j) = \max_{y^{(i)} \in S_i} \min_{y^{(j)} \in S_j} d_C^{\Delta S}(y^{(i)}, y^{(j)}, W^{(2)})$:

$$d_C^{\Delta S}(x^{(1)}, x^{(2)}, W^{(2)}) = d(x^{(2)}, S_2) + d(S_2, S_1) - d(x^{(1)}, S_1) \quad (\text{C.5})$$

$$d_C^{\Delta S}(x^{(1)}, x^{(3)}, W^{(2)}) = d(x^{(3)}, S_3) + d(S_3, S_1) - d(x^{(1)}, S_1) \quad (\text{C.6})$$

$$d_C^{\Delta S}(x^{(2)}, x^{(3)}, W^{(2)}) = d(x^{(2)}, S_2) + d(S_3, S_2) - d(x^{(3)}, S_3) \quad (\text{C.7})$$

Distance of a point to
its core

We can furthermore write $d(x^{(i)}, S_i)$ (i.e., the distance between $x^{(i)}$ and the closest point in S_i , where the distance is computed with the weights $W^{(2)}$ instead of $W^{(i)}$) as follows, using that $x_d^{(i)} = y_d^{(i)}$ for all $d \in D \setminus \{d^*\}$ (step (i)), that $x_{d^*}^{(i)} > y_{d^*}^{(i)}$ (step (ii), and Equation C.4 from above for $x_{d^*}^{(i)}$ (step (iii)):

$$\begin{aligned} d(x^{(i)}, S_i) &= d_C^{\Delta S}(x^{(i)}, y^{(i)}, W^{(2)}) = \sum_{\delta \in \Delta} w_{\delta}^{(2)} \cdot \sqrt{\sum_{d \in \delta} w_d^{(2)} \cdot |x_d^{(i)} - y_d^{(i)}|^2} \\ &\stackrel{(i)}{=} w_{\delta^*}^{(2)} \cdot \sqrt{w_{d^*}^{(2)} \cdot |x_{d^*}^{(i)} - y_{d^*}^{(i)}|^2} \\ &\stackrel{(ii)}{=} w_{\delta^*}^{(2)} \sqrt{w_{d^*}^{(2)}} \left(x_{d^*}^{(i)} - y_{d^*}^{(i)} \right) \\ &\stackrel{(iii)}{=} w_{\delta^*}^{(2)} \sqrt{w_{d^*}^{(2)}} \left(y_{d^*}^{(i)} - \frac{\ln(\alpha)}{c^{(i)} w_{\delta^*}^{(i)} \sqrt{w_{d^*}^{(i)}}} + \frac{\ln(\mu_0^{(i)})}{c^{(i)} w_{\delta^*}^{(i)} \sqrt{w_{d^*}^{(i)}}} - y_{d^*}^{(i)} \right) \\ &= -\frac{w_{\delta^*}^{(2)} \sqrt{w_{d^*}^{(2)}}}{c^{(i)} w_{\delta^*}^{(i)} \sqrt{w_{d^*}^{(i)}}} \left(\ln(\alpha) - \ln(\mu_0^{(i)}) \right) \\ &= -\frac{1}{c^{(2)}} \left(\ln(\alpha) - \ln(\mu_0^{(i)}) \right) \frac{c^{(2)} w_{\delta^*}^{(2)} \sqrt{w_{d^*}^{(2)}}}{c^{(i)} w_{\delta^*}^{(i)} \sqrt{w_{d^*}^{(i)}}} \end{aligned} \quad (\text{C.8})$$

Soft betweenness of
points

Let us first consider $B_{soft}(x^{(1)}, x^{(2)}, x^{(3)})$ for a fixed value of α . We know that $B_{soft}(x^{(1)}, x^{(2)}, x^{(3)})$ is based on the pairwise distances between $x^{(1)}$, $x^{(2)}$, and $x^{(3)}$:

$$B_{soft}(x^{(1)}, x^{(2)}, x^{(3)}) = \frac{d_C^{\Delta S}(x^{(1)}, x^{(3)}, W^{(2)})}{d_C^{\Delta S}(x^{(1)}, x^{(2)}, W^{(2)}) + d_C^{\Delta S}(x^{(2)}, x^{(3)}, W^{(2)})} \quad (\text{C.9})$$

Rewriting the
numerator

The numerator of this fraction can be expressed as follows, based on

our insights from above, namely, Equation C.6 for step (i) and Equation C.8 for step (ii):

$$\begin{aligned}
& d_C^{\Delta_S}(x^{(1)}, x^{(3)}, W^{(2)}) \\
& \stackrel{(i)}{=} d(x^{(3)}, S_3) + d(S_3, S_1) - d(x^{(1)}, S_1) \\
& \stackrel{(ii)}{=} -\frac{1}{c^{(2)}} \left(\ln(\alpha) - \ln(\mu_0^{(3)}) \right) \frac{c^{(2)} w_{\delta^*}^{(2)} \sqrt{w_{d^*}^{(2)}}}{c^{(3)} w_{\delta^*}^{(3)} \sqrt{w_{d^*}^{(3)}}} + d(S_3, S_1) \\
& \quad + \frac{1}{c^{(2)}} \left(\ln(\alpha) + \ln(\mu_0^{(1)}) \right) \frac{c^{(2)} w_{\delta^*}^{(2)} \sqrt{w_{d^*}^{(2)}}}{c^{(1)} w_{\delta^*}^{(1)} \sqrt{w_{d^*}^{(1)}}} \\
& = -\frac{1}{c^{(2)}} \cdot \ln(\alpha) \cdot \left[\frac{c^{(2)} w_{\delta^*}^{(2)} \sqrt{w_{d^*}^{(2)}}}{c^{(3)} w_{\delta^*}^{(3)} \sqrt{w_{d^*}^{(3)}}} - \frac{c^{(2)} w_{\delta^*}^{(2)} \sqrt{w_{d^*}^{(2)}}}{c^{(1)} w_{\delta^*}^{(1)} \sqrt{w_{d^*}^{(1)}}} \right] \\
& \quad + \frac{1}{c^{(2)}} \ln(\mu_0^{(3)}) \frac{c^{(2)} w_{\delta^*}^{(2)} \sqrt{w_{d^*}^{(2)}}}{c^{(3)} w_{\delta^*}^{(3)} \sqrt{w_{d^*}^{(3)}}} \\
& \quad - \frac{1}{c^{(2)}} \ln(\mu_0^{(1)}) \frac{c^{(2)} w_{\delta^*}^{(2)} \sqrt{w_{d^*}^{(2)}}}{c^{(1)} w_{\delta^*}^{(1)} \sqrt{w_{d^*}^{(1)}}} + d(S_3, S_1) \quad (C.10)
\end{aligned}$$

Also the denominator of Equation C.9 can be rewritten based on Equations C.5 and C.7 (step (i)), and Equation C.8 (step (ii)), yielding the following result:

Rewriting the denominator

$$\begin{aligned}
& d_C^{\Delta_S}(x^{(1)}, x^{(2)}, W^{(2)}) + d_C^{\Delta_S}(x^{(2)}, x^{(3)}, W^{(2)}) \\
& \stackrel{(i)}{=} d(x^{(2)}, S_2) + d(S_2, S_1) - d(x^{(1)}, S_1) \\
& \quad + d(x^{(2)}, S_2) - d(S_3, S_2) - d(x^{(3)}, S_3) \\
& \stackrel{(ii)}{=} -\frac{1}{c^{(2)}} \left(\ln(\alpha) - \ln(\mu_0^{(2)}) \right) + d(S_2, S_1) \\
& \quad + \frac{1}{c^{(2)}} \left(\ln(\alpha) - \ln(\mu_0^{(1)}) \right) \frac{c^{(2)} w_{\delta^*}^{(2)} \sqrt{w_{d^*}^{(2)}}}{c^{(1)} w_{\delta^*}^{(1)} \sqrt{w_{d^*}^{(1)}}} \\
& \quad - \frac{1}{c^{(2)}} \left(\ln(\alpha) - \ln(\mu_0^{(2)}) \right) - d(S_3, S_2) \\
& \quad + \frac{1}{c^{(2)}} \left(\ln(\alpha) - \ln(\mu_0^{(3)}) \right) \frac{c^{(2)} w_{\delta^*}^{(2)} \sqrt{w_{d^*}^{(2)}}}{c^{(3)} w_{\delta^*}^{(3)} \sqrt{w_{d^*}^{(3)}}} \\
& = -\frac{1}{c^{(2)}} \cdot \ln(\alpha) \cdot \left[1 - \frac{c^{(2)} w_{\delta^*}^{(2)} \sqrt{w_{d^*}^{(2)}}}{c^{(1)} w_{\delta^*}^{(1)} \sqrt{w_{d^*}^{(1)}}} + 1 - \frac{c^{(2)} w_{\delta^*}^{(2)} \sqrt{w_{d^*}^{(2)}}}{c^{(3)} w_{\delta^*}^{(3)} \sqrt{w_{d^*}^{(3)}}} \right] \\
& \quad + \frac{1}{c^{(2)}} \cdot \ln(\mu_0^{(2)}) + d(S_2, S_1)
\end{aligned}$$

$$\begin{aligned}
& -\frac{1}{c^{(2)}} \ln\left(\mu_0^{(1)}\right) \frac{c^{(2)} w_{\delta^*}^{(2)} \sqrt{w_{d^*}^{(2)}}}{c^{(1)} w_{\delta^*}^{(1)} \sqrt{w_{d^*}^{(1)}}} \\
& + \frac{1}{c^{(2)}} \cdot \ln\left(\mu_0^{(2)}\right) - d(S_3, S_2) \\
& - \frac{1}{c^{(2)}} \ln\left(\mu_0^{(3)}\right) \frac{c^{(2)} w_{\delta^*}^{(2)} \sqrt{w_{d^*}^{(2)}}}{c^{(3)} w_{\delta^*}^{(3)} \sqrt{w_{d^*}^{(3)}}} \tag{C.11}
\end{aligned}$$

The denominator
grows faster than the
numerator

Since $d(S_i, S_j)$ operates on the crisp cores, it is independent of α and can therefore be ignored for our analysis. In order to show that $B_{soft}^{inf}(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3) = 0$, we now show that $B_{soft}(x^{(1)}, x^{(2)}, x^{(3)}) \xrightarrow{\alpha \rightarrow 0} 0$ by proving that the denominator (Equation C.11) grows faster than the numerator (Equation C.10) for decreasing values of α . We use the fact that $c^{(2)} w_{\delta^*}^{(2)} \sqrt{w_{d^*}^{(2)}} < c^{(3)} w_{\delta^*}^{(3)} \sqrt{w_{d^*}^{(3)}}$, which implies that the fraction $\frac{c^{(2)} w_{\delta^*}^{(2)} \sqrt{w_{d^*}^{(2)}}}{c^{(3)} w_{\delta^*}^{(3)} \sqrt{w_{d^*}^{(3)}}}$ is smaller than one:

$$\begin{aligned}
& 1 - \frac{c^{(2)} w_{\delta^*}^{(2)} \sqrt{w_{d^*}^{(2)}}}{c^{(1)} w_{\delta^*}^{(1)} \sqrt{w_{d^*}^{(1)}}} + 1 - \frac{c^{(2)} w_{\delta^*}^{(2)} \sqrt{w_{d^*}^{(2)}}}{c^{(3)} w_{\delta^*}^{(3)} \sqrt{w_{d^*}^{(3)}}} \\
& > 1 - \frac{c^{(2)} w_{\delta^*}^{(2)} \sqrt{w_{d^*}^{(2)}}}{c^{(1)} w_{\delta^*}^{(1)} \sqrt{w_{d^*}^{(1)}}} + 1 - 1 \\
& = 1 - \frac{c^{(2)} w_{\delta^*}^{(2)} \sqrt{w_{d^*}^{(2)}}}{c^{(1)} w_{\delta^*}^{(1)} \sqrt{w_{d^*}^{(1)}}} \\
& > \frac{c^{(2)} w_{\delta^*}^{(2)} \sqrt{w_{d^*}^{(2)}}}{c^{(3)} w_{\delta^*}^{(3)} \sqrt{w_{d^*}^{(3)}}} - \frac{c^{(2)} w_{\delta^*}^{(2)} \sqrt{w_{d^*}^{(2)}}}{c^{(1)} w_{\delta^*}^{(1)} \sqrt{w_{d^*}^{(1)}}}
\end{aligned}$$

Since the factor associated with $-\ln(\alpha)$ is thus greater in the denominator than in the numerator, it is easy to see that $B_{soft}(x^{(1)}, x^{(2)}, x^{(3)}) \xrightarrow{\alpha \rightarrow 0} 0$. Please note that this result is independent of any differences between the cores with respect to other dimensions of the conceptual space: These differences on other domains do influence the values of $d_C^{\Delta S}(x^{(i)}, x^{(j)}, W^{(2)})$, but they do not depend on α and are therefore constants in our considerations. They thus do not influence the overall limit value analysis. Based on the definition of $B_{soft}^{inf}(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3)$, we thus get the desired result:

$$B_{soft}^{inf}(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3) = \inf_{\alpha \in [0,1]} \min_{x^{(2)} \in \tilde{S}_2^\alpha} \max_{x^{(1)} \in \tilde{S}_1^\alpha} \max_{x^{(3)} \in \tilde{S}_3^\alpha} B_{soft}(x^{(1)}, x^{(2)}, x^{(3)}) = 0$$

Choice of $x^{(1)}$ does
not affect results

One may argue that if we for example chose in Figure C.1 the point $x_B^{(1)}$ instead of $x^{(1)}$, for which both $d_C^{\Delta S}(x_B^{(1)}, x^{(3)}, W^{(2)}) > d_C^{\Delta S}(x^{(1)}, x^{(3)}, W^{(2)})$

and $d_C^\Delta(x_B^{(1)}, x^{(2)}, W^{(2)}) > d_C^{\Delta_S}(x^{(1)}, x^{(2)}, W^{(2)})$, the desired result may no longer hold. However, \tilde{S}_2^α grows faster than both \tilde{S}_1^α and \tilde{S}_3^α for decreasing α . Thus, $x^{(2)}$ will always be able to "outrun" $x^{(1)}$ and $x^{(3)}$, causing $d_C^{\Delta_S}(x^{(1)}, x^{(2)}, W^{(2)})$ and $d_C^{\Delta_S}(x^{(2)}, x^{(3)}, W^{(2)})$ to grow faster than $d_C^{\Delta_S}(x^{(1)}, x^{(3)}, W^{(2)})$. Analogously to the case investigated above, one can show that also $B_{soft}(x_B^{(1)}, x^{(2)}, x^{(3)}) \xrightarrow{\alpha \rightarrow 0} 0$. Thus, if the condition with respect to the sensitivity parameters and the weights is fulfilled, $B_{soft}^{inf}(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3) = 0$. \square

In order to circumvent these pathological cases, we have then introduced an integral-based soft notion of betweenness:

Integral-based degree of betweenness

Definition 4.10 (Integral-Based Betweenness)

Let \tilde{S}_1, \tilde{S}_2 , and \tilde{S}_3 be three concepts defined on the same set of domains Δ_S . The integral-based soft degree to which \tilde{S}_2 lies between \tilde{S}_1 and \tilde{S}_3 can be computed as follows:

$$B_{soft}^{int}(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3) = \int_0^1 \min_{y \in \tilde{S}_2^\alpha} \max_{x \in \tilde{S}_1^\alpha} \max_{z \in \tilde{S}_3^\alpha} Btw_3(x, y, z) d\alpha$$

One can easily see that B_{soft}^{inf} cannot be larger than B_{soft}^{int} :

Relation between the two definitions

Lemma 4.1 (B_{soft}^{int} is Bounded by B_{soft}^{inf})

Let \tilde{S}_1, \tilde{S}_2 , and \tilde{S}_3 be three concepts. Then, the following inequation holds:

$$B_{soft}^{inf}(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3) \leq B_{soft}^{int}(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3)$$

Proof. Both B_{soft}^{inf} and B_{soft}^{int} use the same way to compute betweenness for a given α -cut:

$$B_{soft}(\tilde{S}_1^\alpha, \tilde{S}_2^\alpha, \tilde{S}_3^\alpha) = \min_{y \in \tilde{S}_2^\alpha} \max_{x \in \tilde{S}_1^\alpha} \max_{z \in \tilde{S}_3^\alpha} B_{soft}(x, y, z)$$

The only difference between B_{soft}^{inf} and B_{soft}^{int} is the way in which they aggregate across different values of α , namely with the infimum and the integral, respectively.

We can bound the integral $\int_0^1 B_{soft}(\tilde{S}_1^\alpha, \tilde{S}_2^\alpha, \tilde{S}_3^\alpha) d\alpha$ from below by using the lower Riemann sum for a fixed number n_α of α -cuts:

$$\int_0^1 B_{soft}(\tilde{S}_1^\alpha, \tilde{S}_2^\alpha, \tilde{S}_3^\alpha) d\alpha \geq \sum_{i=1}^{n_\alpha} \inf_{\alpha \in [\frac{i-1}{n_\alpha}, \frac{i}{n_\alpha}]} B_{soft}(\tilde{S}_1^\alpha, \tilde{S}_2^\alpha, \tilde{S}_3^\alpha) \cdot \frac{1}{n_\alpha}$$

For computing this bound, we thus split the interval $[0, 1]$, over which α is optimized, into n_α equally sized parts. For each of these parts, we compute the infimum of $B_{soft}(\tilde{S}_1^\alpha, \tilde{S}_2^\alpha, \tilde{S}_3^\alpha)$ and multiply it with the size of this part. By summing over all parts, we obtain an estimate of the overall integral which is guaranteed to be not larger than the actual

integral. For $n_\alpha \rightarrow \infty$, both sides of the equation become identical. Moreover, one can easily see that the following inequality is true:

$$\begin{aligned} & \sum_{i=1}^{n_\alpha} \inf_{\alpha \in \left[\frac{i-1}{n_\alpha}, \frac{i}{n_\alpha}\right]} B_{soft}(\tilde{S}_1^\alpha, \tilde{S}_2^\alpha, \tilde{S}_3^\alpha) \cdot \frac{1}{n_\alpha} \\ & \geq \sum_{i=1}^{n_\alpha} \inf_{\alpha \in [0,1]} B_{soft}(\tilde{S}_1^\alpha, \tilde{S}_2^\alpha, \tilde{S}_3^\alpha) \cdot \frac{1}{n_\alpha} = \inf_{\alpha \in [0,1]} B_{soft}(\tilde{S}_1^\alpha, \tilde{S}_2^\alpha, \tilde{S}_3^\alpha) \end{aligned}$$

We therefore get the expected result:

$$\begin{aligned} B_{soft}^{int}(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3) &= \int_0^1 B_{soft}(\tilde{S}_1^\alpha, \tilde{S}_2^\alpha, \tilde{S}_3^\alpha) d\alpha \\ &\geq \inf_{\alpha \in [0,1]} B_{soft}(\tilde{S}_1^\alpha, \tilde{S}_2^\alpha, \tilde{S}_3^\alpha) = B_{soft}^{inf}(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3) \end{aligned}$$

□

D | FEASIBILITY STUDY ON NOVEL OBJECTS

Overview

In this appendix, we provide further visualizations and raw results with respect to our study from Chapter 9, where we applied our hybrid approach to the NOUN dataset [199]. In Sections D.1 and D.2, we provide additional visualizations of the pixel baseline (supplementing the visualizations from Section 9.1.3) for different distances and correlation metrics, and with respect to uniform and optimized dimension weights, respectively. Sections D.3 and D.4 contain additional tables with results for our second machine learning experiment from Section 9.2.4, namely, for unregularized and regularized regressions, respectively, with respect to target spaces from different MDS algorithms. Finally, Section D.5 provides detailed results for the lasso regression with respect to target spaces of different dimensionality (cf. Section 9.2.5).

D.1 PIXEL BASELINE WITH UNIFORM WEIGHTS

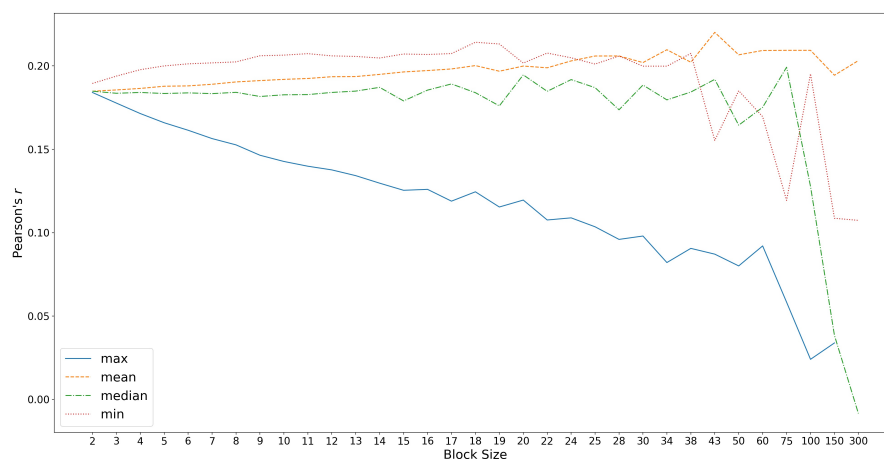


Figure D.1: Pearson's r for pixel-based distances obtained through the Euclidean metric (using uniform weights) with respect to different aggregators and different block sizes.

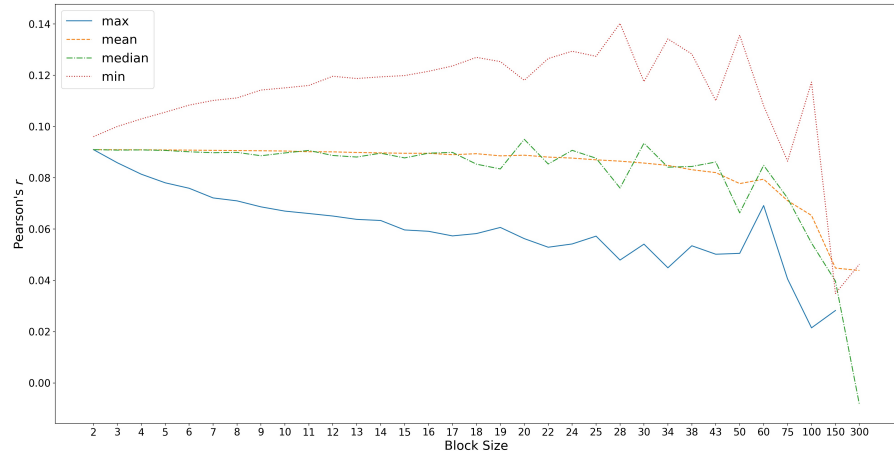


Figure D.2: Pearson's r for pixel-based distances obtained through the negated inner product (using uniform weights) with respect to different aggregators and different block sizes.

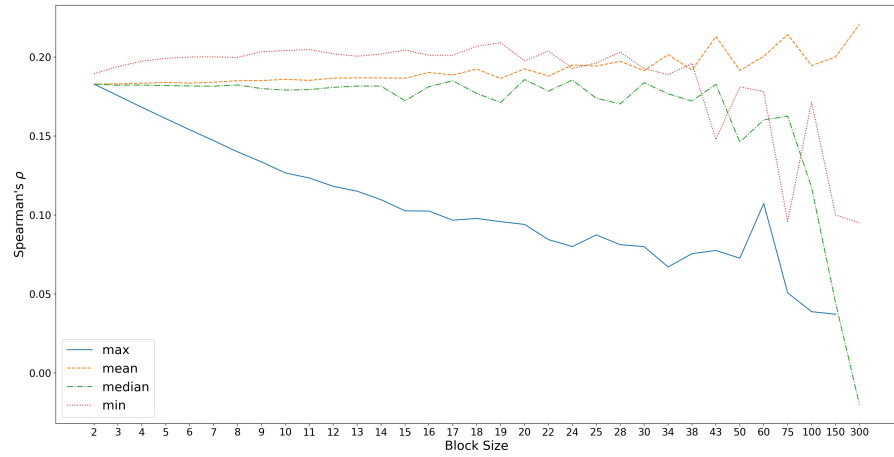


Figure D.3: Spearman's ρ for pixel-based distances obtained through the Euclidean metric (using uniform weights) with respect to different aggregators and different block sizes.

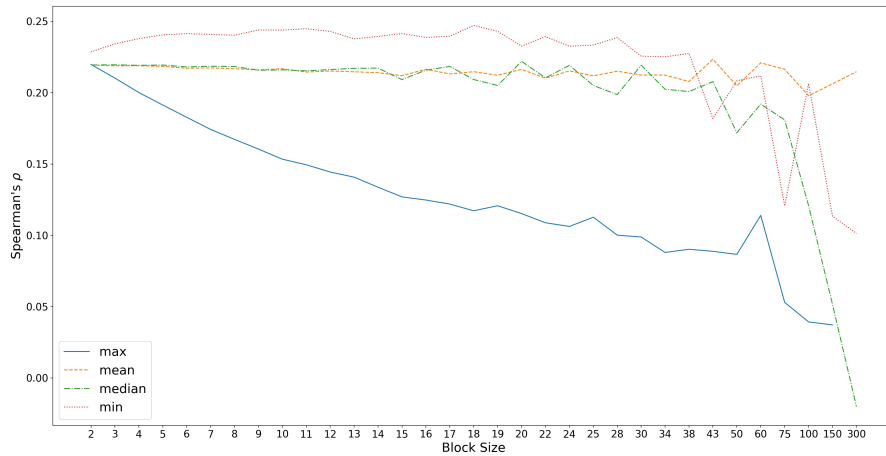


Figure D.4: Spearman's ρ for pixel-based distances obtained through the Manhattan metric (using uniform weights) with respect to different aggregators and different block sizes.

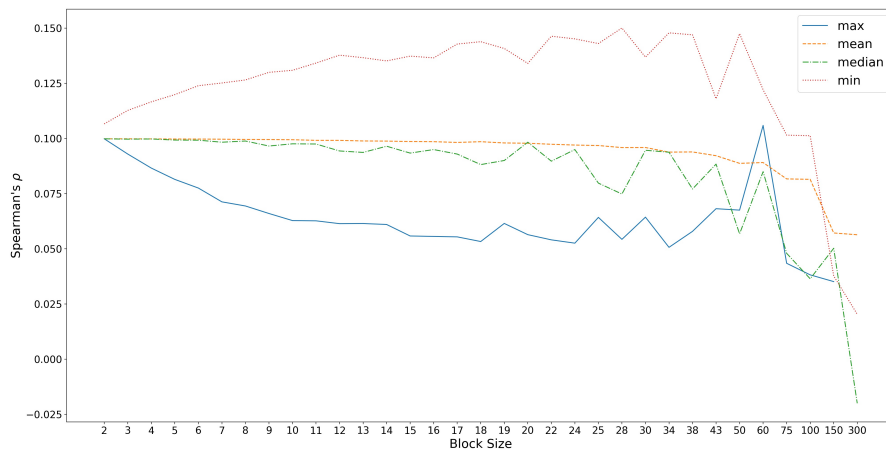


Figure D.5: Spearman's ρ for pixel-based distances obtained through the negated inner product (using uniform weights) with respect to different aggregators and different block sizes.

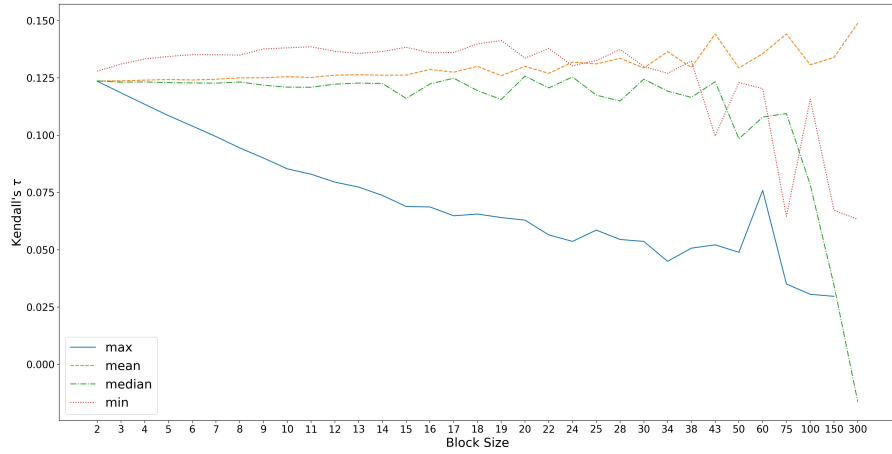


Figure D.6: Kendall's τ for pixel-based distances obtained through the Euclidean metric (using uniform weights) with respect to different aggregators and different block sizes.

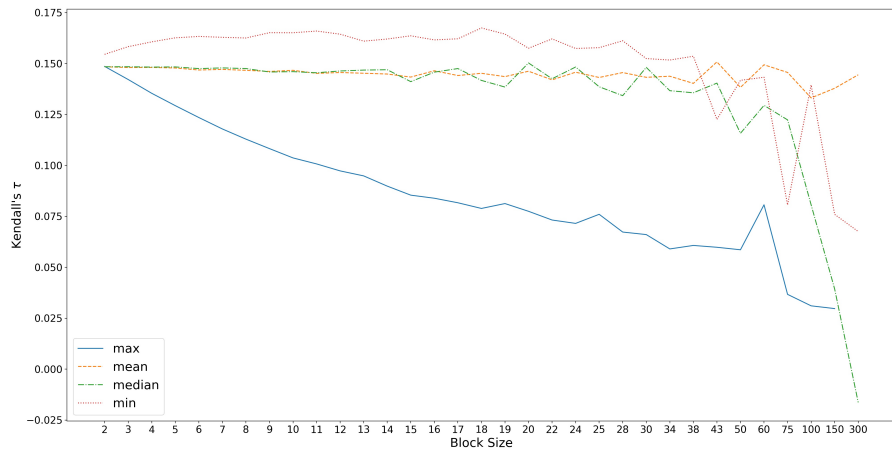


Figure D.7: Kendall's τ for pixel-based distances obtained through the Manhattan metric (using uniform weights) with respect to different aggregators and different block sizes.

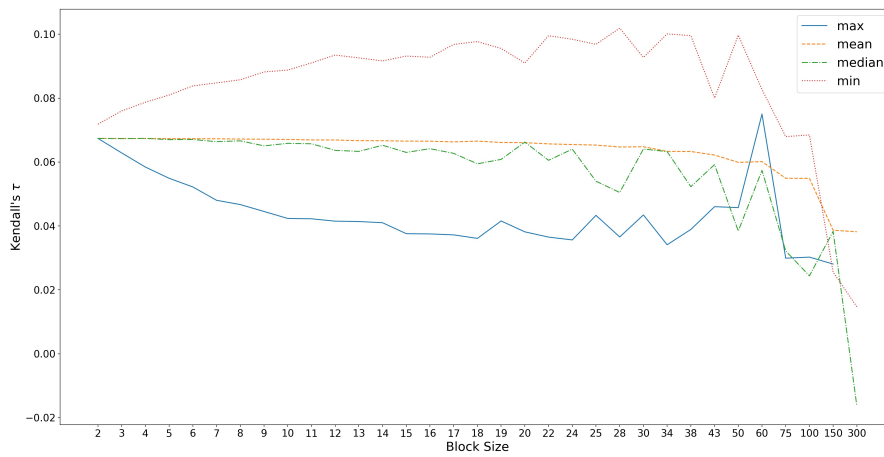


Figure D.8: Kendall's τ for pixel-based distances obtained through the negated inner product (using uniform weights) with respect to different aggregators and different block sizes.

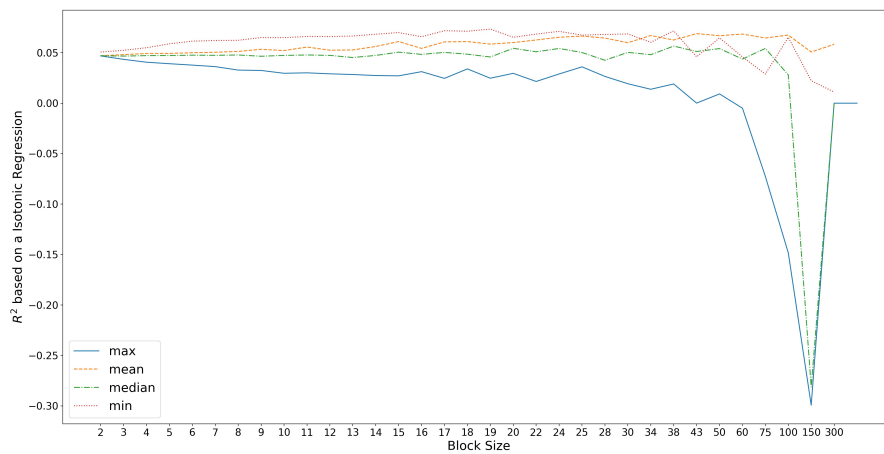


Figure D.9: Coefficient of determination R^2 of a monotone regression for pixel-based distances obtained through the Euclidean metric (using uniform weights) with respect to different aggregators and different block sizes.

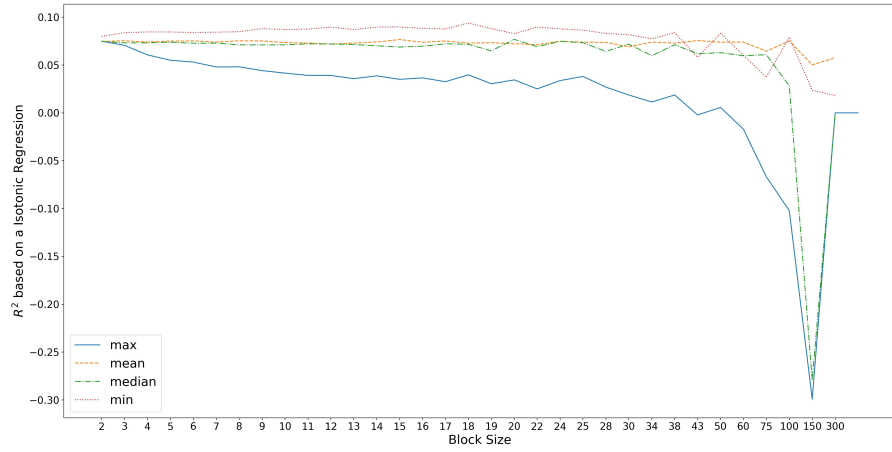


Figure D.10: Coefficient of determination R^2 of a monotone regression for pixel-based distances obtained through the Manhattan metric (using uniform weights) with respect to different aggregators and different block sizes.

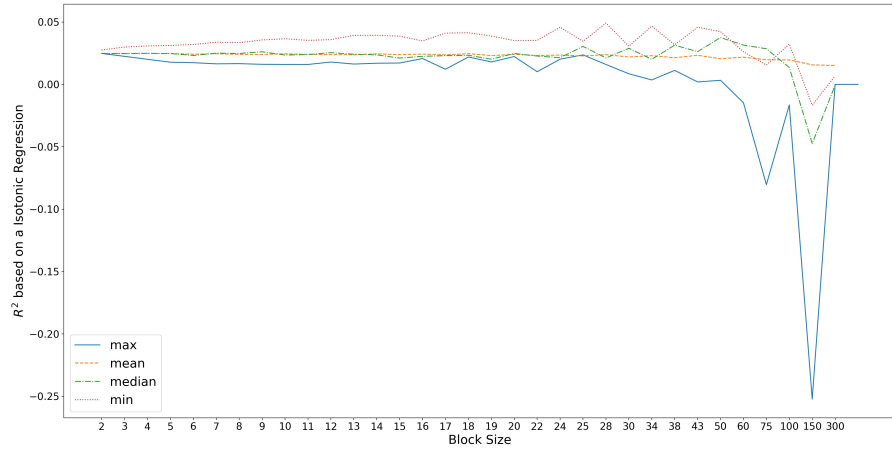


Figure D.11: Coefficient of determination R^2 of a monotone regression for pixel-based distances obtained through the negated inner product (using uniform weights) with respect to different aggregators and different block sizes.

D.2 PIXEL BASELINE WITH OPTIMIZED WEIGHTS

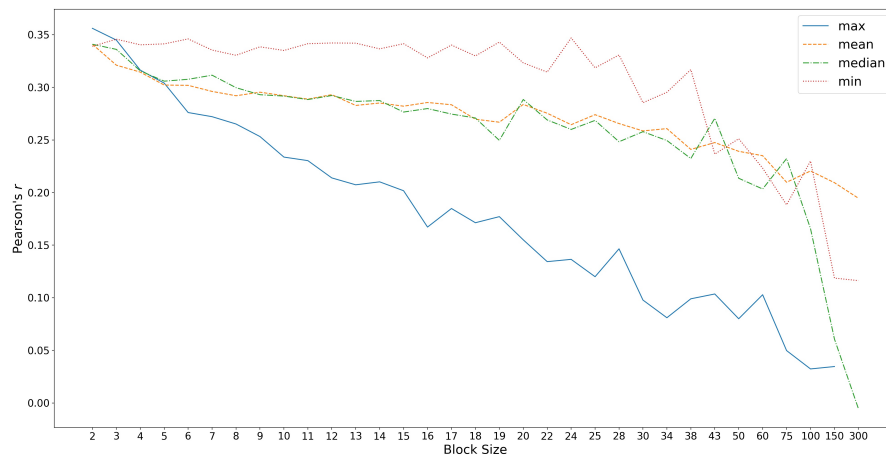


Figure D.12: Pearson's r for pixel-based distances obtained through the Manhattan metric (using optimized weights) with respect to different aggregators and different block sizes.

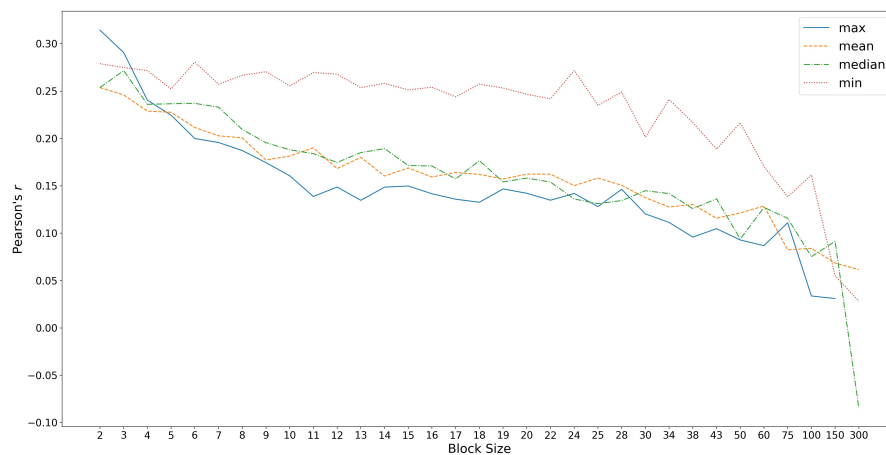


Figure D.13: Pearson's r for pixel-based distances obtained through the negated inner product (using optimized weights) with respect to different aggregators and different block sizes.

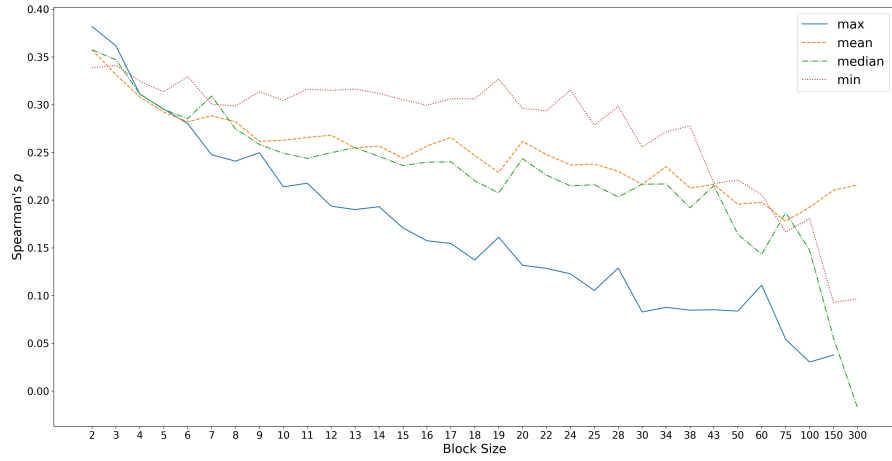


Figure D.14: Spearman's ρ for pixel-based distances obtained through the Euclidean metric (using optimized weights) with respect to different aggregators and different block sizes.

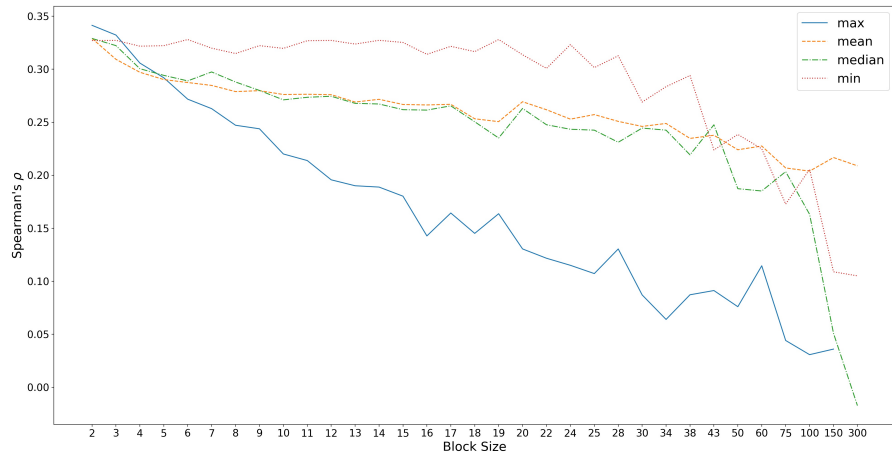


Figure D.15: Spearman's ρ for pixel-based distances obtained through the Manhattan metric (using optimized weights) with respect to different aggregators and different block sizes.

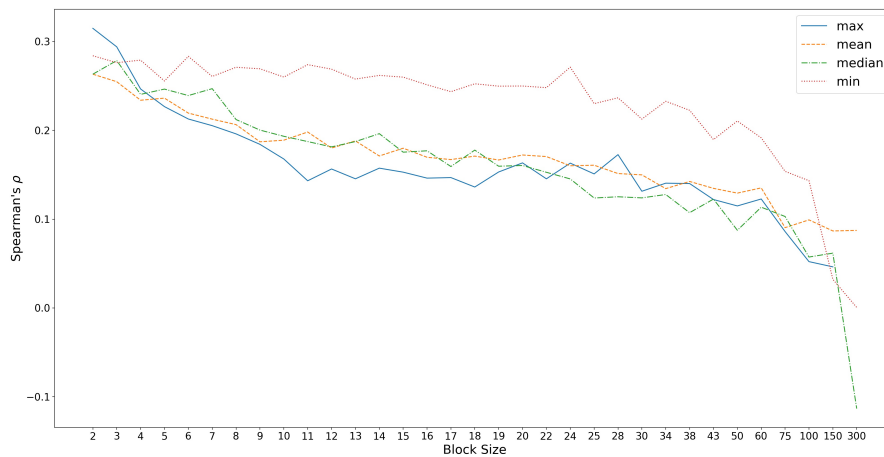


Figure D.16: Spearman's ρ for pixel-based distances obtained through the negated inner product (using optimized weights) with respect to different aggregators and different block sizes.

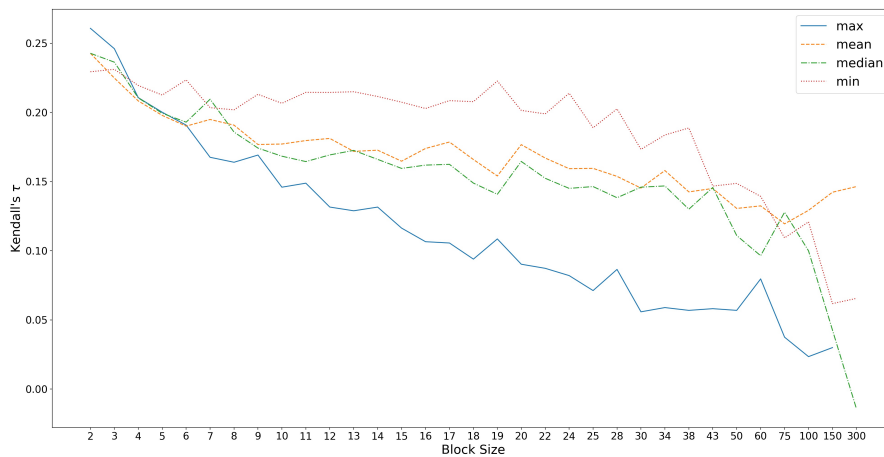


Figure D.17: Kendall's τ for pixel-based distances obtained through the Euclidean metric (using optimized weights) with respect to different aggregators and different block sizes.

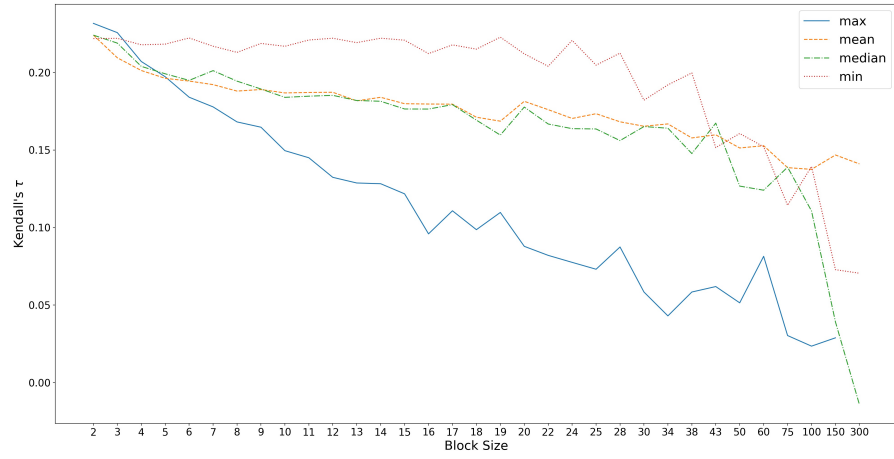


Figure D.18: Kendall's τ for pixel-based distances obtained through the Manhattan metric (using optimized weights) with respect to different aggregators and different block sizes.

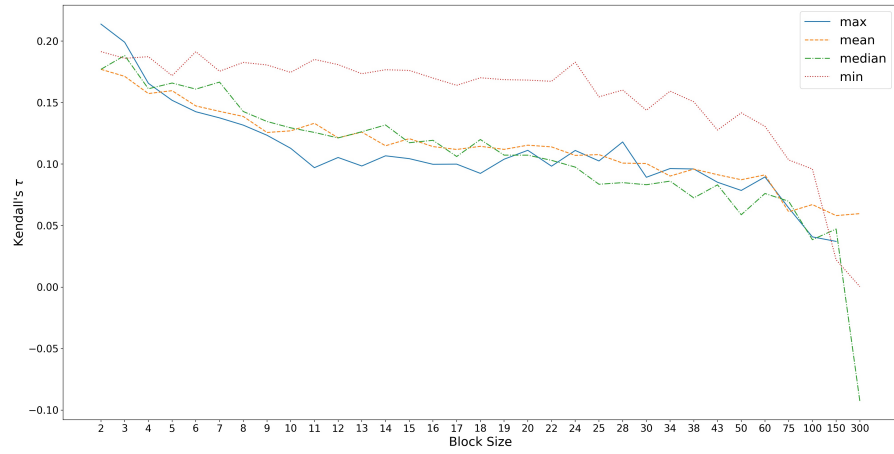


Figure D.19: Kendall's τ for pixel-based distances obtained through the negated inner product (using optimized weights) with respect to different aggregators and different block sizes.

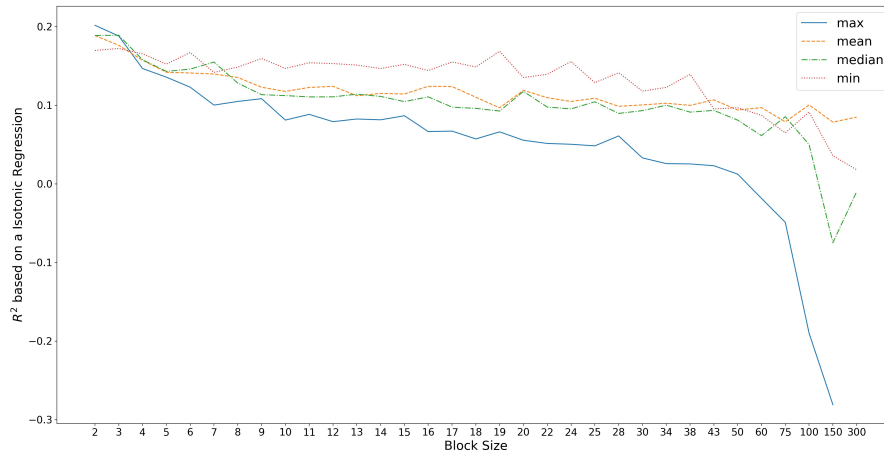


Figure D.20: Coefficient of determination R^2 of a monotone regression for pixel-based distances obtained through the Euclidean metric (using optimized weights) with respect to different aggregators and different block sizes.

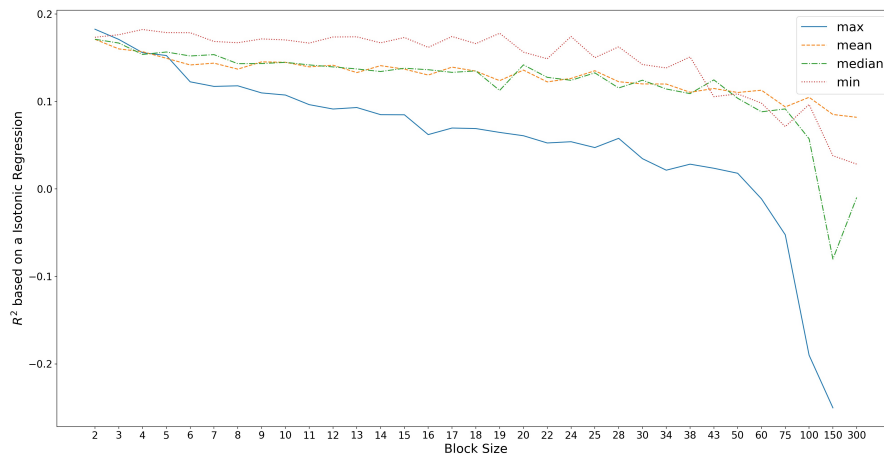


Figure D.21: Coefficient of determination R^2 of a monotone regression for pixel-based distances obtained through the Manhattan metric (using optimized weights) with respect to different aggregators and different block sizes.

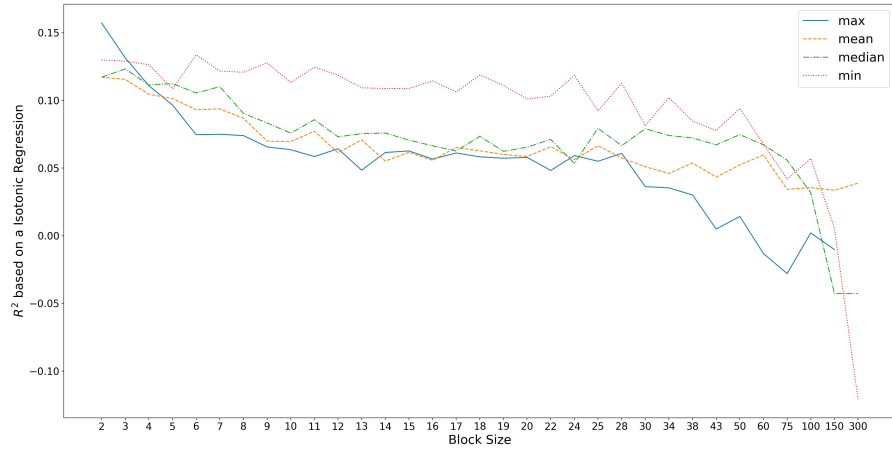


Figure D.22: Coefficient of determination R^2 of a monotone regression for pixel-based distances obtained through the negated inner product (using optimized weights) with respect to different aggregators and different block sizes.

D.3 UNREGULARIZED REGRESSION FOR DIFFERENT MDS ALGORITHMS

Feature Space	Regressor	Targets	Training			Test		
			MSE	MED	R^2	MED	MED	R^2
Any	Zero Baseline	Any	1.0000	0.9962	0.0000	1.0000	0.9962	0.0000
ANN (2048)	Linear	Correct	0.0132	0.1048	0.9862	0.5537	0.7063	0.4029
		Shuffled	0.0204	0.1296	0.9797	1.1510	1.0323	-0.1346
	Random Forest	Correct	0.0017	0.0158	0.9983	0.7504	0.8257	0.2091
		Shuffled	0.0025	0.0248	0.9975	1.1322	1.0236	-0.1109
Pixel (1875)	Linear	Correct	0.4844	0.6460	0.5022	1.2302	1.0449	-0.2613
		Shuffled	0.6171	0.7292	0.3856	1.6067	1.1989	-0.6058
	Random Forest	Correct	0.0058	0.0463	0.9939	0.8556	0.8721	0.1174
		Shuffled	0.0071	0.0539	0.9930	1.1766	1.0344	-0.1477

Table D.1: Performance of different regressors for different feature spaces and correct vs. shuffled targets, with respect to the four-dimensional target space based on classical MDS. The best results for each combination of column and feature space are highlighted in boldface.

Feature Space	Regressor	Targets	Training			Test		
			MSE	MED	R^2	MED	MED	R^2
Any	Zero Baseline	Any	1.0000	0.9962	0.0000	1.0000	0.9962	0.0000
ANN (2048)	Linear	Correct	0.0144	0.1095	0.9856	0.6097	0.7460	0.3903
		Shuffled	0.0209	0.1315	0.9791	1.2400	1.0894	-0.2406
	Random Forest	Correct	0.0019	0.0178	0.9981	0.8036	0.8676	0.1959
		Shuffled	0.0029	0.0261	0.9971	1.1900	1.0763	-0.1882
Pixel (1875)	Linear	Correct	0.5052	0.6668	0.4935	1.3384	1.0947	-0.3415
		Shuffled	0.5997	0.7292	0.3996	1.6110	1.2113	-0.6099
	Random Forest	Correct	0.0060	0.0473	0.9937	0.9002	0.9111	0.0972
		Shuffled	0.0073	0.0547	0.9927	1.2151	1.0729	-0.2156

Table D.2: Performance of different regressors for different feature spaces and correct vs. shuffled targets, with respect to the four-dimensional target space based on Kruskal's MDS algorithm. The best results for each combination of column and feature space are highlighted in boldface.

Feature Space	Regressor	Targets	Training			Test		
			MSE	MED	R^2	MED	MED	R^2
Any	Zero Baseline	Any	1.0000	0.9962	0.0000	1.0000	0.9962	0.0000
ANN (2048)	Linear	Correct	0.0146	0.1102	0.9854	0.6172	0.7560	0.3766
		Shuffled	0.0206	0.1308	0.9794	1.2227	1.0838	-0.2240
	Random Forest	Correct	0.0020	0.0183	0.9980	0.8325	0.8881	0.1604
		Shuffled	0.0027	0.0254	0.9973	1.1691	1.0702	-0.1693
Pixel (1875)	Linear	Correct	0.5135	0.6718	0.4871	1.3178	1.0886	-0.3188
		Shuffled	0.6017	0.7315	0.4006	1.6061	1.2116	-0.5981
	Random Forest	Correct	0.0059	0.0473	0.9941	0.8913	0.9118	0.1062
		Shuffled	0.0073	0.0554	0.9927	1.2253	1.0817	-0.2159

Table D.3: Performance of different regressors for different feature spaces and correct vs. shuffled targets, with respect to the four-dimensional target space based on metric SMACOF. The best results for each combination of column and feature space are highlighted in boldface.

Feature Space	Regressor	Targets	Training			Test		
			MSE	MED	R^2	MED	MED	R^2
Any	Zero Baseline	Any	1.0000	0.9962	0.0000	1.0000	0.9962	0.0000
ANN (2048)	Linear	Correct	0.0143	0.1092	0.9853	0.6086	0.7461	0.3706
		Shuffled	0.0208	0.1312	0.9792	1.2340	1.0869	-0.2366
	Random Forest	Correct	0.0020	0.0180	0.9980	0.7926	0.8619	0.1899
		Shuffled	0.0027	0.0258	0.9973	1.1680	1.0673	-0.1626
Pixel (1875)	Linear	Correct	0.5117	0.6704	0.4769	1.3305	1.0921	-0.3587
		Shuffled	0.6009	0.7300	0.4013	1.6071	1.2087	-0.5973
	Random Forest	Correct	0.0062	0.0477	0.9936	0.9033	0.9120	0.0732
		Shuffled	0.0071	0.0550	0.9930	1.2278	1.0785	-0.2150

Table D.4: Performance of different regressors for different feature spaces and correct vs. shuffled targets, with respect to the four-dimensional target space based on nonmetric SMACOF. The best results for each combination of column and feature space are highlighted in boldface.

D.4 LASSO REGRESSION FOR DIFFERENT MDS ALGORITHMS

β	Training			Test		
	MSE	MED	R^2	MSE	MED	R^2
Zero Baseline	1.0000	0.9636	0.0000	1.0000	0.9636	0.0000
0	0.0132	0.1048	0.9862	0.5537	0.7063	0.4029
0.001	0.0144	0.1089	0.9849	0.5362	0.6934	0.4241
0.002	0.0190	0.1249	0.9801	0.5272	0.6856	0.4317
0.005	0.0158	0.1140	0.9835	0.5324	0.6896	0.4274
0.01	0.0227	0.1365	0.9762	0.5315	0.6889	0.4248
0.02	0.0282	0.1521	0.9703	0.5467	0.6993	0.4083
0.05	0.0410	0.1836	0.9565	0.5681	0.7151	0.3841
0.1	0.0596	0.2228	0.9363	0.5756	0.7203	0.3746
0.2	0.0968	0.2883	0.8956	0.5874	0.7283	0.3619
0.5	0.2339	0.4614	0.7450	0.6245	0.7572	0.3273
1	0.4746	0.6653	0.4832	0.6730	0.7887	0.2793
2	0.8300	0.8798	0.1287	0.8951	0.9110	0.0696
≥ 5	0.9972	0.9623	0.0026	1.0414	0.9822	-0.0389

Table D.5: Performance of the lasso regressor for different values of β with respect to the four-dimensional target space based on classical MDS. The best results for each combination of column and feature space are highlighted in boldface.

β	Training			Test		
	MSE	MED	R^2	MSE	MED	R^2
Zero Baseline	1.0000	0.9957	0.0000	1.0000	0.9957	0.0000
0	0.0144	0.1095	0.9856	0.6097	0.7460	0.3903
0.001	0.0156	0.1135	0.9844	0.6070	0.7420	0.3932
0.002	0.0171	0.1186	0.9830	0.6097	0.7427	0.3906
0.005	0.0204	0.1297	0.9796	0.6162	0.7456	0.3839
0.01	0.0245	0.1419	0.9755	0.6294	0.7527	0.3707
0.02	0.0305	0.1580	0.9696	0.6424	0.7601	0.3577
0.05	0.0441	0.1906	0.9559	0.6679	0.7746	0.3321
0.1	0.0642	0.2314	0.9358	0.6759	0.7787	0.3244
0.2	0.1056	0.3011	0.8944	0.6856	0.7868	0.3142
0.5	0.2476	0.4764	0.7524	0.7182	0.8175	0.2809
1	0.5200	0.7093	0.4797	0.8045	0.8825	0.1951
2	0.9646	0.9776	0.0354	1.0202	1.0050	-0.0201
≥ 5	0.9977	0.9944	0.0023	1.0341	1.0119	-0.0342

Table D.6: Performance of the lasso regressor for different values of β with respect to the four-dimensional target space based on Kruskal's MDS algorithm. The best results for each combination of column and feature space are highlighted in boldface.

β	Training			Test		
	MSE	MED	R^2	MSE	MED	R^2
Zero Baseline	1.0000	0.9981	0.0000	1.0000	0.9981	0.0000
0	0.0146	0.1102	0.9854	0.6172	0.7560	0.3766
0.001	0.0158	0.1143	0.9842	0.6086	0.7485	0.3848
0.002	0.0173	0.1194	0.9827	0.6052	0.7458	0.3880
0.005	0.0207	0.1307	0.9792	0.6056	0.7459	0.3873
0.01	0.0249	0.1432	0.9750	0.6151	0.7507	0.3775
0.02	0.0310	0.1597	0.9689	0.6298	0.7585	0.3622
0.05	0.0447	0.1923	0.9551	0.6611	0.7755	0.3294
0.1	0.0645	0.2327	0.9351	0.6762	0.7852	0.3128
0.2	0.1048	0.3014	0.8944	0.6837	0.7922	0.3038
0.5	0.2483	0.4802	0.7491	0.7294	0.8311	0.2547
1	0.5324	0.7212	0.4610	0.8389	0.9066	0.1481
2	0.9519	0.9731	0.0443	0.9996	0.9971	-0.0020
≥ 5	0.9980	0.9969	0.0019	1.0293	1.0122	-0.0290

Table D.7: Performance of the lasso regressor for different values of β with respect to the four-dimensional target space based on metric SMA-COF. The best results for each combination of column and feature space are highlighted in boldface.

β	Training			Test		
	MSE	MED	R^2	MSE	MED	R^2
Zero Baseline	1.0000	0.9956	0.0000	1.0000	0.9956	0.0000
0	0.0143	0.1092	0.9853	0.6086	0.7461	0.3706
0.001	0.0155	0.1133	0.9841	0.5987	0.7373	0.3812
0.002	0.0170	0.1183	0.9826	0.5965	0.7348	0.3833
0.005	0.0204	0.1295	0.9791	0.5938	0.7316	0.3853
0.01	0.0245	0.1419	0.9748	0.5980	0.7331	0.3804
0.02	0.0439	0.1901	0.9548	0.6267	0.7495	0.3501
0.05	0.0305	0.1582	0.9687	0.6063	0.7375	0.3717
0.1	0.1044	0.2995	0.8921	0.6573	0.7715	0.3145
0.2	0.0637	0.2306	0.9343	0.6528	0.7660	0.3207
0.5	0.2476	0.4763	0.7433	0.6666	0.7868	0.3044
1	0.5206	0.7077	0.4591	0.7629	0.8581	0.2087
2	0.9073	0.9465	0.0771	0.9579	0.9718	0.0287
≥ 5	0.9977	0.9943	0.0022	1.0338	1.0117	-0.0329

Table D.8: Performance of the lasso regressor for different values of β with respect to the four-dimensional target space based on nonmetric SMACOF. The best results for each combination of column and feature space are highlighted in boldface.

D.5 LASSO REGRESSION FOR DIFFERENT DIMENSIONALITY

β	Training			Test		
	MSE	MED	R^2	MSE	MED	R^2
Zero Baseline	1.0000	0.8664	0.0000	1.0000	0.8664	0.0000
0	0.0195	0.1084	0.9805	1.1499	0.9046	-0.1499
0.001	0.0199	0.1095	0.9801	1.1477	0.9038	-0.1477
0.002	0.0208	0.1117	0.9792	1.1474	0.9039	-0.1474
0.005	0.0234	0.1185	0.9766	1.1710	0.9129	-0.1710
0.01	0.0270	0.1273	0.9730	1.1955	0.9216	-0.1955
0.02	0.0326	0.1400	0.9674	1.1849	0.9153	-0.1849
0.05	0.0446	0.1638	0.9554	1.1392	0.8956	-0.1392
0.1	0.0605	0.1910	0.9395	1.1443	0.8942	-0.1443
0.2	0.0882	0.2320	0.9118	1.1095	0.8830	-0.1095
0.5	0.1774	0.3373	0.8226	1.0376	0.8492	-0.0376
1	0.3535	0.4910	0.6465	0.9968	0.8368	0.0032
2	0.7258	0.7307	0.2742	0.9911	0.8477	0.0088
≥ 5	0.9980	0.8647	0.0020	1.0306	0.8826	-0.0306

Table D.9: Performance of the lasso regressor for different values of β with respect to the one-dimensional target space based on nonmetric SMACOF. The best results for each combination of column and feature space are highlighted in boldface.

β	Training			Test		
	MSE	MED	R^2	MSE	MED	R^2
Zero Baseline	1.0000	0.9580	0.0000	1.0000	0.9580	0.0000
0	0.0128	0.0976	0.9872	0.4995	0.6370	0.5002
0.001	0.0136	0.1000	0.9864	0.5036	0.6350	0.4962
0.002	0.0146	0.1037	0.9854	0.5031	0.6319	0.4967
0.005	0.0172	0.1123	0.9828	0.4869	0.6174	0.5129
0.01	0.0203	0.1219	0.9797	0.4770	0.6079	0.5228
0.02	0.0247	0.1342	0.9753	0.4728	0.6052	0.5271
0.05	0.0339	0.1572	0.9661	0.4773	0.6113	0.5225
0.1	0.0464	0.1850	0.9536	0.4907	0.6213	0.5091
0.2	0.0699	0.2299	0.9301	0.4918	0.6209	0.5079
0.5	0.1518	0.3492	0.8481	0.5163	0.6458	0.4829
1	0.3139	0.5168	0.6859	0.5838	0.7043	0.4153
2	0.6626	0.7744	0.3370	0.8074	0.8525	0.1920
≥ 5	0.9973	0.9564	0.0027	1.0408	0.9760	-0.0408

Table D.10: Performance of the lasso regressor for different values of β with respect to the two-dimensional target space based on nonmetric SMACOF. The best results for each combination of column and feature space are highlighted in boldface.

β	Training			Test		
	MSE	MED	R^2	MSE	MED	R^2
Zero Baseline	1.0000	0.9848	0.0000	1.0000	0.9848	0.0000
0	0.0134	0.1036	0.9865	0.5554	0.69789	0.4435
0.001	0.0144	0.1069	0.9856	0.5509	0.6927	0.4486
0.002	0.0156	0.1113	0.9843	0.5443	0.6880	0.4556
0.005	0.0186	0.1214	0.9813	0.5343	0.6794	0.4677
0.01	0.0223	0.1325	0.9776	0.5337	0.6750	0.4700
0.02	0.0274	0.1469	0.9725	0.5322	0.6719	0.4722
0.05	0.0384	0.1741	0.9615	0.5489	0.6805	0.4543
0.1	0.0541	0.2081	0.9456	0.5816	0.7007	0.4205
0.2	0.0862	0.2666	0.9132	0.6093	0.7198	0.3937
0.5	0.1972	0.4159	0.8016	0.6104	0.7348	0.3918
1	0.4103	0.6156	0.5871	0.6593	0.7804	0.3401
2	0.7937	0.8746	0.2039	0.8840	0.9210	0.1164
≥ 5	0.9977	0.9834	0.0022	1.0351	1.0010	-0.03365

Table D.11: Performance of the lasso regressor for different values of β with respect to the three-dimensional target space based on nonmetric SMACOF. The best results for each combination of column and feature space are highlighted in boldface.

β	Training			Test		
	MSE	MED	R^2	MSE	MED	R^2
Zero Baseline	1.0000	0.9966	0.0000	1.0000	0.9966	0.0000
0	0.0146	0.1114	0.9853	0.6333	0.7692	0.3595
0.001	0.0160	0.1162	0.9839	0.6193	0.7588	0.3743
0.002	0.0176	0.1219	0.9822	0.6180	0.7576	0.3755
0.005	0.0213	0.1339	0.9785	0.6212	0.7578	0.3724
0.01	0.0256	0.1469	0.9741	0.6300	0.7610	0.3636
0.02	0.0321	0.1645	0.9675	0.6472	0.7699	0.3465
0.05	0.0466	0.1987	0.9529	0.6909	0.7946	0.3028
0.1	0.0686	0.2429	0.9306	0.7180	0.8115	0.2755
0.2	0.1157	0.3206	0.8828	0.7371	0.8261	0.2548
0.5	0.2842	0.5176	0.7118	0.7536	0.8482	0.2374
1	0.5966	0.7637	0.3963	0.8484	0.9111	0.1442
2	0.9723	0.9823	0.0254	1.0207	1.0060	-0.0217
≥ 5	0.9975	0.9951	0.0025	1.0373	1.0144	-0.0370

Table D.12: Performance of the lasso regressor for different values of β with respect to the five-dimensional target space based on nonmetric SMACOF. The best results for each combination of column and feature space are highlighted in boldface.

β	Training			Test		
	MSE	MED	R^2	MSE	MED	R^2
Zero Baseline	1.0000	0.9973	0.0000	1.0000	0.9973	0.0000
0	0.0146	0.1125	0.9852	0.6359	0.7734	0.3469
0.001	0.0162	0.1178	0.9836	0.6277	0.7662	0.3548
0.002	0.0179	0.1238	0.9819	0.6274	0.7651	0.3543
0.005	0.0218	0.1366	0.9779	0.6298	0.7655	0.3504
0.01	0.0264	0.1504	0.9732	0.6438	0.7732	0.3356
0.02	0.0334	0.1691	0.9660	0.6674	0.7864	0.3111
0.05	0.0499	0.2073	0.9493	0.6931	0.8004	0.2830
0.1	0.0743	0.2551	0.9243	0.7040	0.8069	0.2709
0.2	0.1262	0.3377	0.8711	0.7301	0.8254	0.2444
0.5	0.3160	0.5489	0.6762	0.7828	0.8673	0.1952
1	0.6563	0.8034	0.3292	0.8597	0.9205	0.1259
2	0.9781	0.9861	0.0183	1.0220	1.0077	-0.0238
≥ 5	0.9977	0.9960	0.0022	1.0341	1.0138	-0.0337

Table D.13: Performance of the lasso regressor for different values of β with respect to the six-dimensional target space based on nonmetric SMACOF. The best results for each combination of column and feature space are highlighted in boldface.

β	Training			Test		
	MSE	MED	R^2	MSE	MED	R^2
Zero Baseline	1.0000	0.9978	0.0000	1.0000	0.9978	0.0000
0	0.0149	0.1143	0.9850	0.6675	0.7956	0.3204
0.001	0.0166	0.1201	0.9832	0.6589	0.7889	0.3280
0.002	0.0185	0.1266	0.9813	0.6597	0.7880	0.3254
0.005	0.0228	0.1403	0.9770	0.6609	0.7864	0.3221
0.01	0.0278	0.1549	0.9720	0.6598	0.7839	0.3214
0.02	0.0351	0.1743	0.9645	0.6710	0.7899	0.3091
0.05	0.0523	0.2136	0.9470	0.7032	0.8093	0.2774
0.1	0.0790	0.2649	0.9198	0.7259	0.8243	0.2579
0.2	0.1369	0.3537	0.8608	0.7402	0.8360	0.2466
0.5	0.3446	0.5748	0.6487	0.7902	0.8747	0.2015
1	0.7204	0.8430	0.2681	0.9260	0.9568	0.0701
≥ 2	0.9976	0.9965	0.0023	1.0354	1.0148	-0.0349

Table D.14: Performance of the lasso regressor for different values of β with respect to the seven-dimensional target space based on nonmetric SMACOF. The best results for each combination of column and feature space are highlighted in boldface.

β	Training			Test		
	MSE	MED	R^2	MSE	MED	R^2
Zero Baseline	1.0000	0.9980	0.0000	1.0000	0.9980	0.0000
0	0.0152	0.1158	0.9846	0.6846	0.8094	0.3033
0.001	0.0171	0.1222	0.9827	0.6752	0.8022	0.3117
0.002	0.0191	0.1290	0.9806	0.6793	0.8038	0.3061
0.005	0.0235	0.1431	0.9761	0.6929	0.8104	0.2897
0.01	0.0288	0.1583	0.9707	0.7008	0.8140	0.2815
0.02	0.0368	0.1788	0.9625	0.7131	0.8203	0.2702
0.05	0.0555	0.2207	0.9433	0.7377	0.8343	0.2474
0.1	0.0843	0.2746	0.9136	0.7593	0.8477	0.2268
0.2	0.1488	0.3705	0.8474	0.7821	0.8649	0.2054
0.5	0.3884	0.6125	0.6010	0.8307	0.8993	0.1557
1	0.7702	0.8729	0.2146	0.9315	0.9606	0.0607
≥ 2	0.9978	0.9968	0.0022	1.0330	1.0140	-0.0324

Table D.15: Performance of the lasso regressor for different values of β with respect to the eight-dimensional target space based on nonmetric SMACOF. The best results for each combination of column and feature space are highlighted in boldface.

β	Training			Test		
	MSE	MED	R^2	MSE	MED	R^2
Zero Baseline	1.0000	0.9982	0.0000	1.0000	0.9982	0.0000
0	0.0152	0.1161	0.9845	0.6810	0.8078	0.2983
0.001	0.0172	0.1229	0.9825	0.6680	0.7988	0.3108
0.002	0.0193	0.1301	0.9803	0.6687	0.7980	0.3099
0.005	0.0239	0.1447	0.9756	0.6772	0.8010	0.3010
0.01	0.02931	0.1601	0.9700	0.6827	0.8034	0.2944
0.02	0.03735	0.1808	0.9617	0.6878	0.8059	0.2899
0.05	0.05665	0.2238	0.9417	0.7058	0.8170	0.2725
0.1	0.0871	0.2804	0.9101	0.7178	0.8259	0.2630
0.2	0.1556	0.3808	0.8389	0.7413	0.8429	0.2426
0.5	0.4081	0.6304	0.5770	0.8102	0.8902	0.1724
1	0.8222	0.9028	0.1535	0.9566	0.9746	0.0266
2	0.9931	0.9947	0.0052	1.0293	1.0124	-0.0297
≥ 5	0.9978	0.9971	0.0021	1.0329	1.0142	-0.0321

Table D.16: Performance of the lasso regressor for different values of β with respect to the nine-dimensional target space based on nonmetric SMACOF. The best results for each combination of column and feature space are highlighted in boldface.

β	Training			Test		
	MSE	MED	R^2	MSE	MED	R^2
Zero Baseline	1.0000	0.9984	0.0000	1.0000	0.9984	0.0000
0	0.0154	0.1173	0.9844	0.7107	0.8259	0.2807
0.001	0.0176	0.1246	0.9823	0.7029	0.8202	0.2889
0.002	0.0198	0.1320	0.9800	0.6993	0.8172	0.2924
0.005	0.0246	0.1470	0.9752	0.7010	0.8166	0.2911
0.01	0.0302	0.1630	0.9695	0.7055	0.8183	0.2882
0.02	0.0387	0.1846	0.9609	0.7164	0.8241	0.2792
0.05	0.0594	0.2298	0.9401	0.7462	0.8415	0.2503
0.1	0.0929	0.2905	0.9063	0.7675	0.8546	0.2299
0.2	0.1672	0.3959	0.8312	0.7862	0.8685	0.2123
0.5	0.4274	0.6463	0.5681	0.8501	0.9131	0.1477
1	0.8516	0.9200	0.1415	0.9891	0.9916	0.0083
≥ 2	0.9976	0.9971	0.0024	1.0358	1.0158	-0.0360

Table D.17: Performance of the lasso regressor for different values of β with respect to the ten-dimensional target space based on nonmetric SMACOF. The best results for each combination of column and feature space are highlighted in boldface.

E | A DATASET ON SHAPE PERCEPTION

In Chapter 10, we have presented and analyzed a new dataset on shape perception. In this appendix, we provide additional scatter plots, which further illustrate our analysis of feature ratings from Section 10.3.3: Section E.1 contains scatter plots for two different features rated with the same procedure, while Section E.2 provides scatter plots for attentive vs. pre-attentive ratings of individual features.

Overview

E.1 CORRELATIONS BETWEEN FEATURES

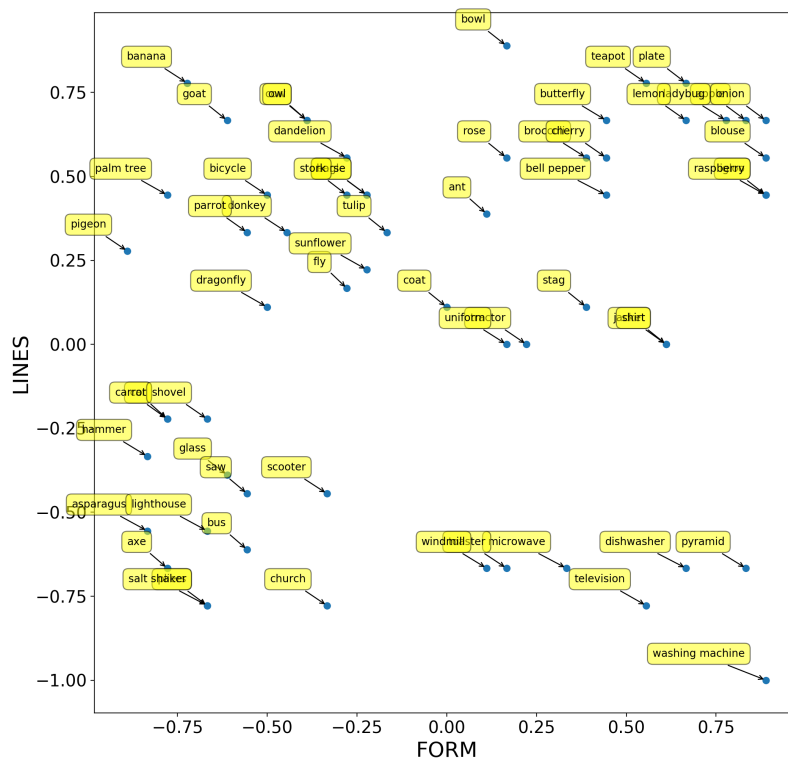


Figure E.1: Scatter plot for the features FORM and LINES based on the pre-attentive feature scales.

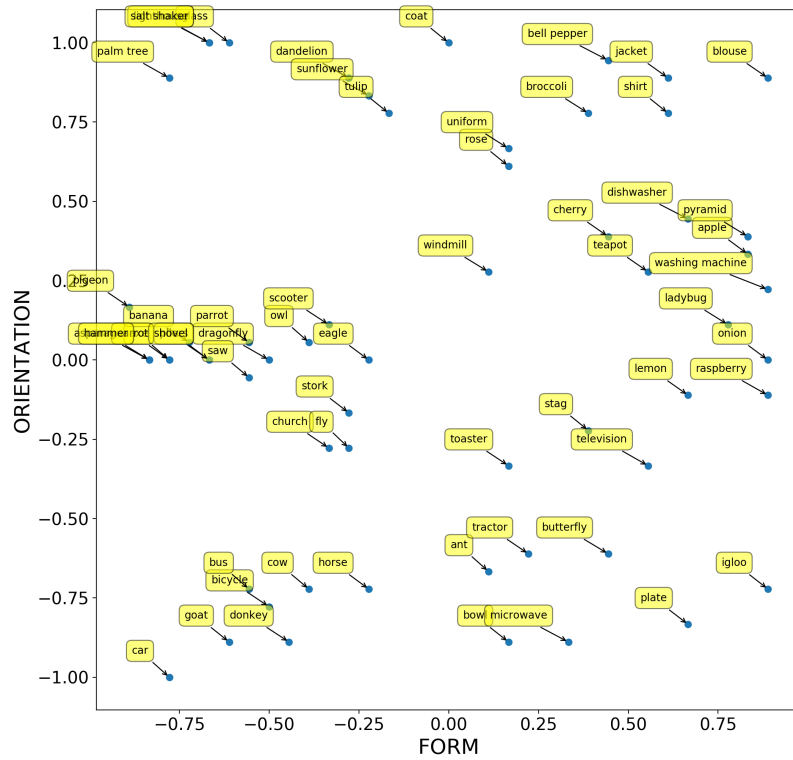


Figure E.2: Scatter plot for the features FORM and ORIENTATION based on the pre-attentive feature scales.

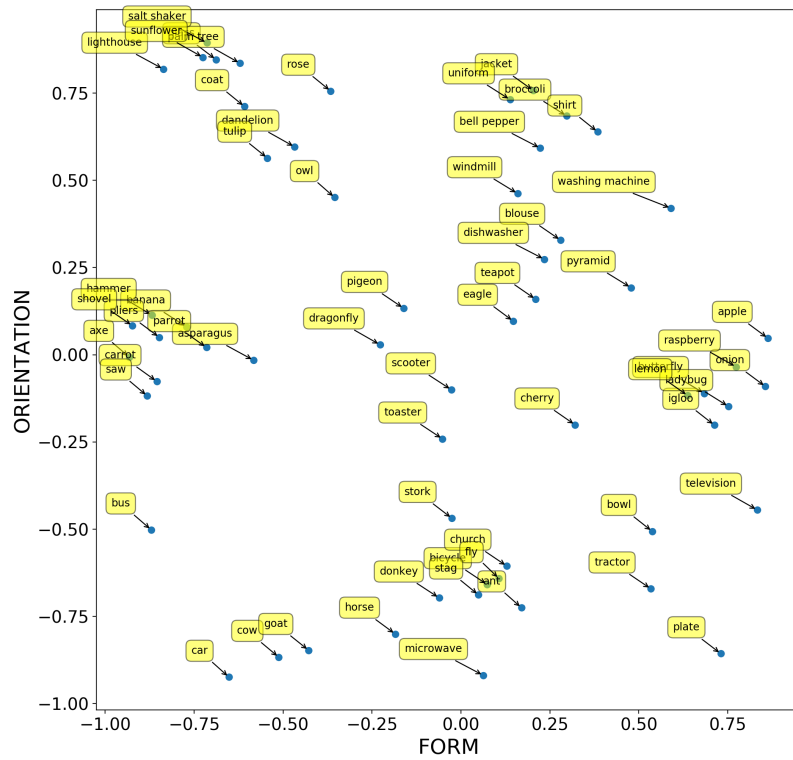


Figure E.3: Scatter plot for the features FORM and ORIENTATION based on the attentive feature scales.

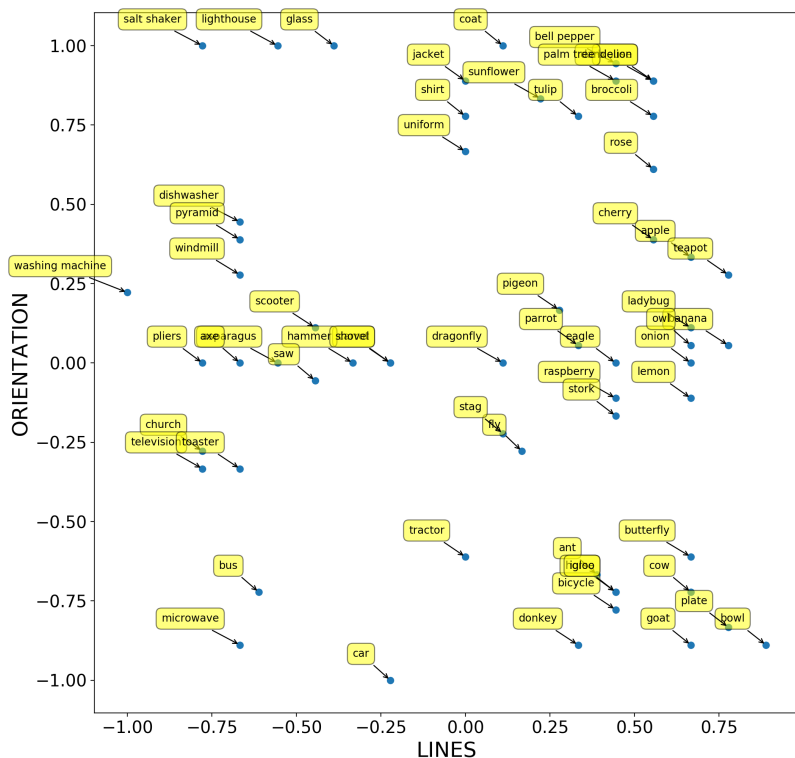


Figure E.4: Scatter plot for the features LINES and ORIENTATION based on the pre-attentive feature scales.

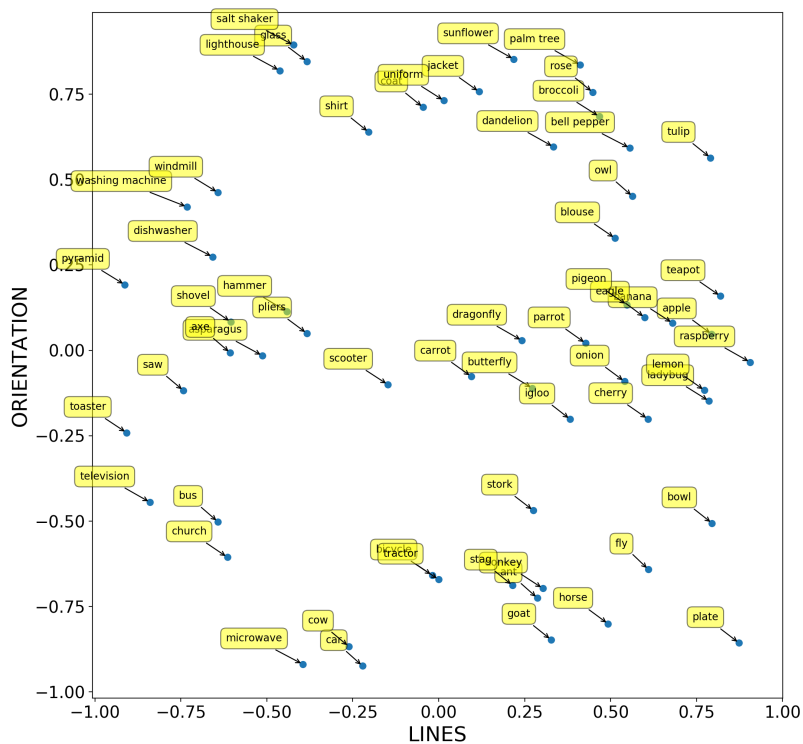


Figure E.5: Scatter plot for the features LINES and ORIENTATION based on the attentive feature scales.

E.2 ATTENTIVE AND PRE-ATTENTIVE RATINGS

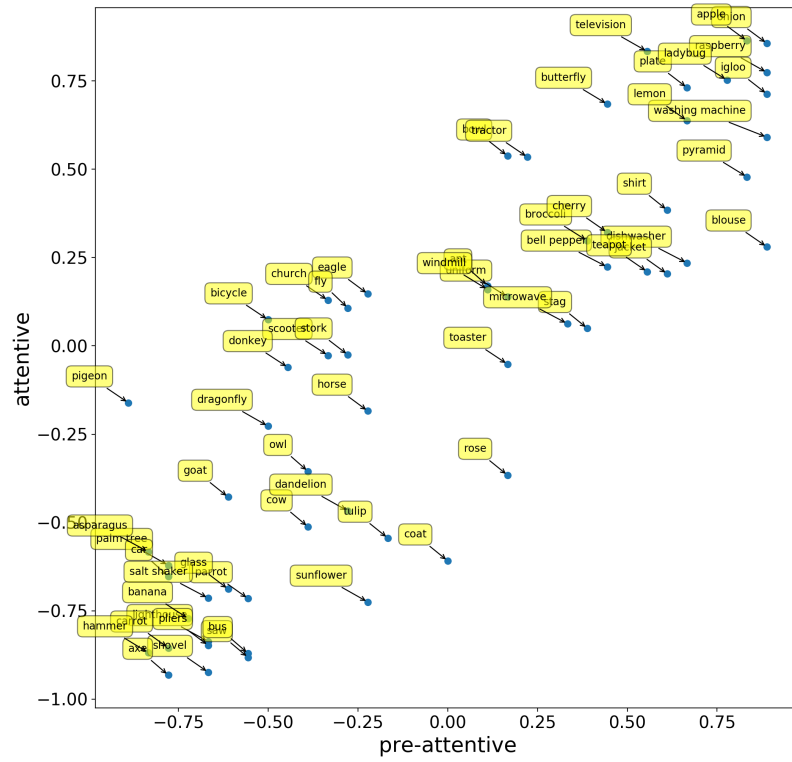


Figure E.6: Scatter plot comparing pre-attentive and attentive ratings for the FORM feature.

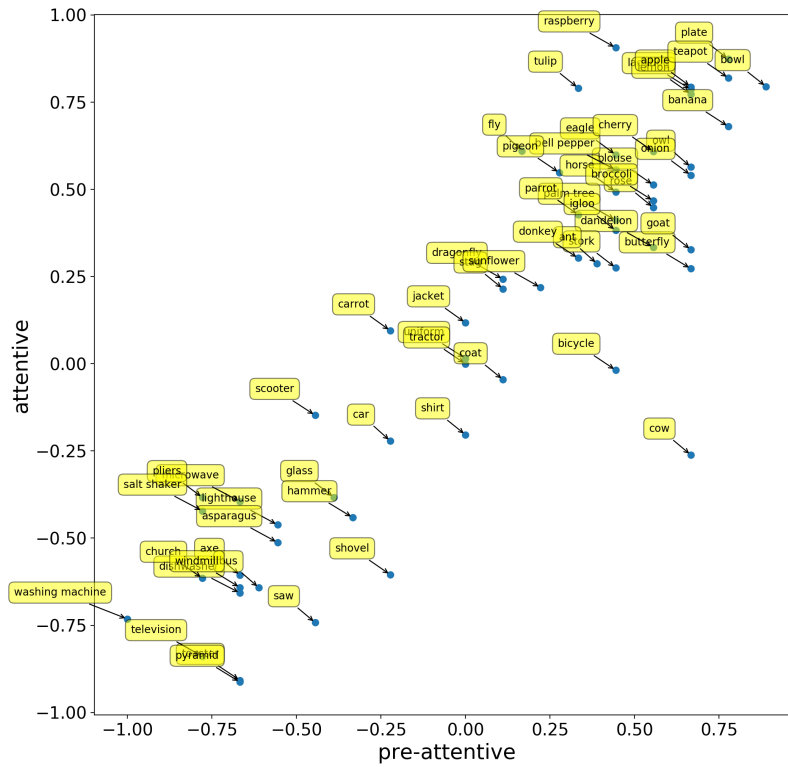


Figure E.7: Scatter plot comparing pre-attentive and attentive ratings for the LINES feature.

F

A PSYCHOLOGICAL SIMILARITY SPACE FOR SHAPES

Overview

This chapter provides further illustrations for our analysis of our shapes dataset with the usage of conceptual spaces in Chapter 11. In Section 11.2.2, we have tried to predict the entries of the overall dissimilarity matrix with three baselines. For the pixel baseline (which considers the distance of down-scaled images), we provide additional visualizations with respect to both the mean and the median matrices in Sections F.1 and F.2. These additional visualizations show the performance of the pixel baseline as a function of aggregator function and block size, considering different distance metrics.

F.1 PIXEL BASELINE FOR THE MEAN MATRIX

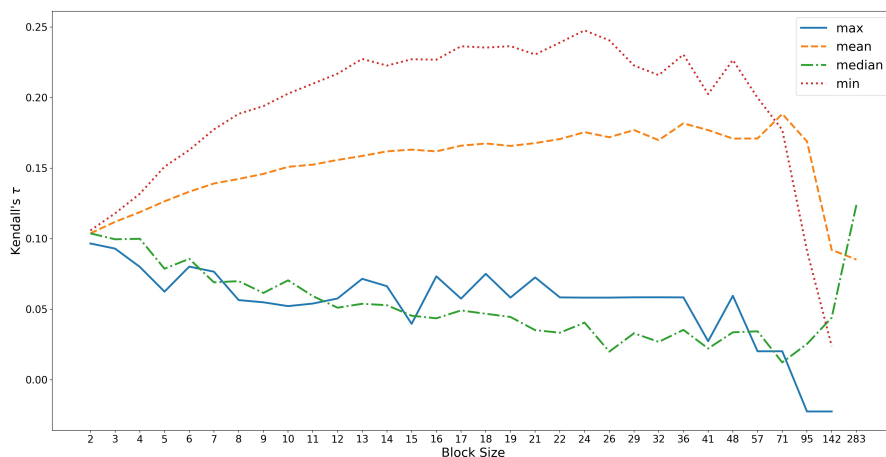


Figure F.1: Kendall's τ between the pixel baseline and the mean matrix as a function of block size and aggregator function, using the Manhattan distance function with uniform weights.

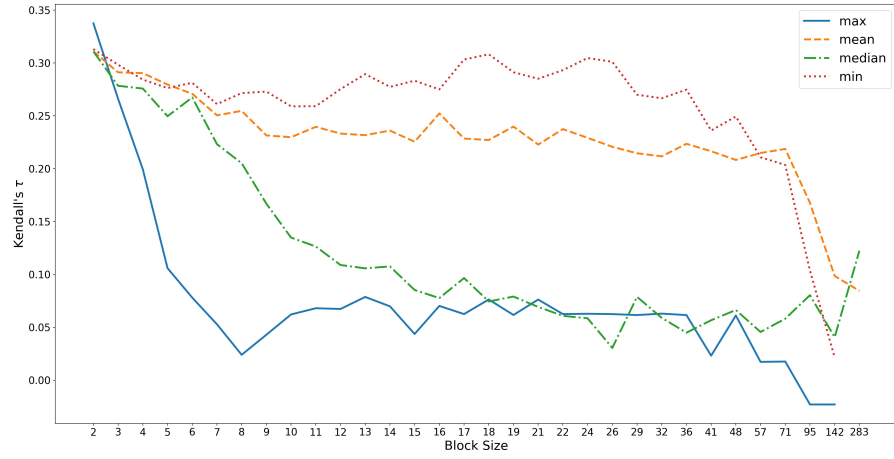


Figure F.2: Kendall's τ between the pixel baseline and the mean matrix as a function of block size and aggregator function, using the Manhattan distance function with optimized weights.

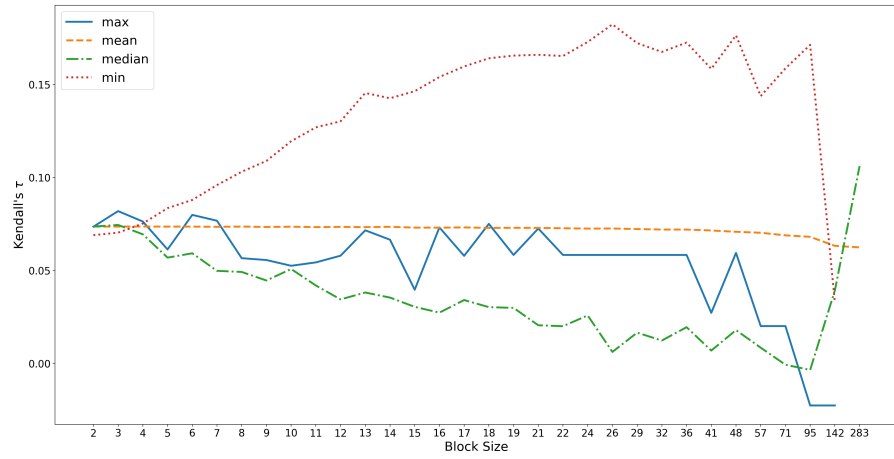


Figure F.3: Kendall's τ between the pixel baseline and the mean matrix as a function of block size and aggregator function, using the negated inner product with uniform weights as distance function.

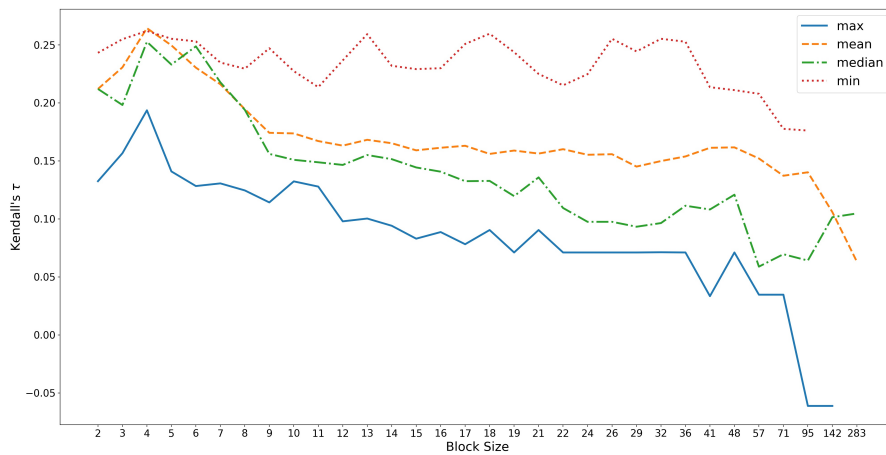


Figure F.4: Kendall's τ between the pixel baseline and the mean matrix as a function of block size and aggregator function, using the negated inner product with optimized weights as distance function.

F.2 PIXEL BASELINE FOR THE MEDIAN MATRIX

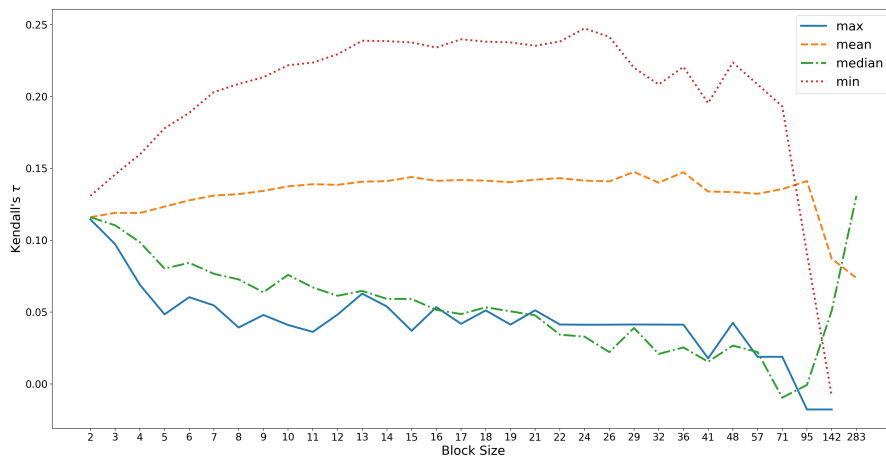


Figure F.5: Kendall's τ between the pixel baseline and the median matrix as a function of block size and aggregator function, using the Euclidean distance function with uniform weights.

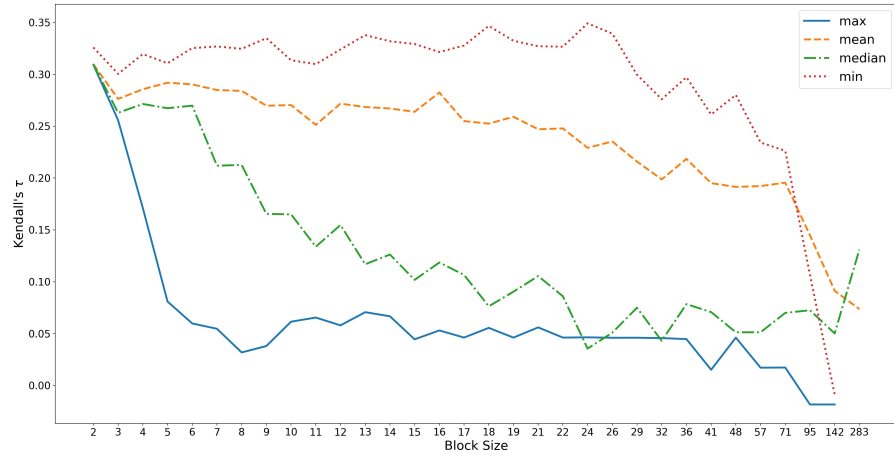


Figure F.6: Kendall's τ between the pixel baseline and the median matrix as a function of block size and aggregator function, using the Euclidean distance function with optimized weights.

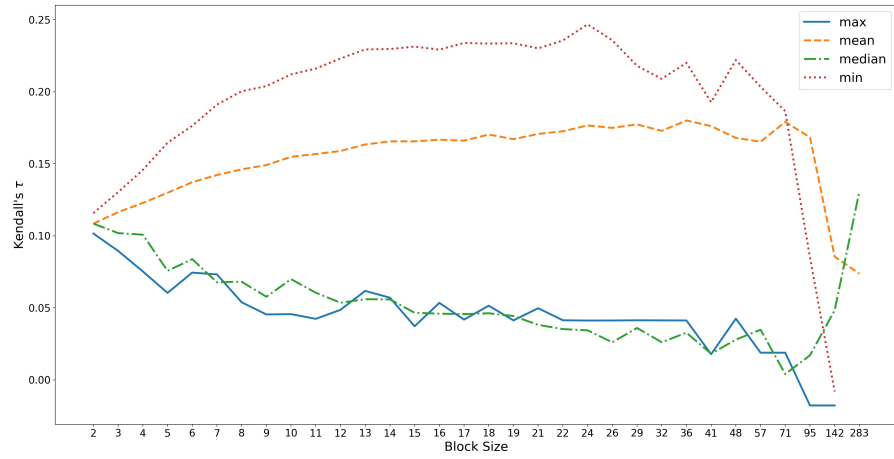


Figure F.7: Kendall's τ between the pixel baseline and the median matrix as a function of block size and aggregator function, using the Manhattan distance function with uniform weights.

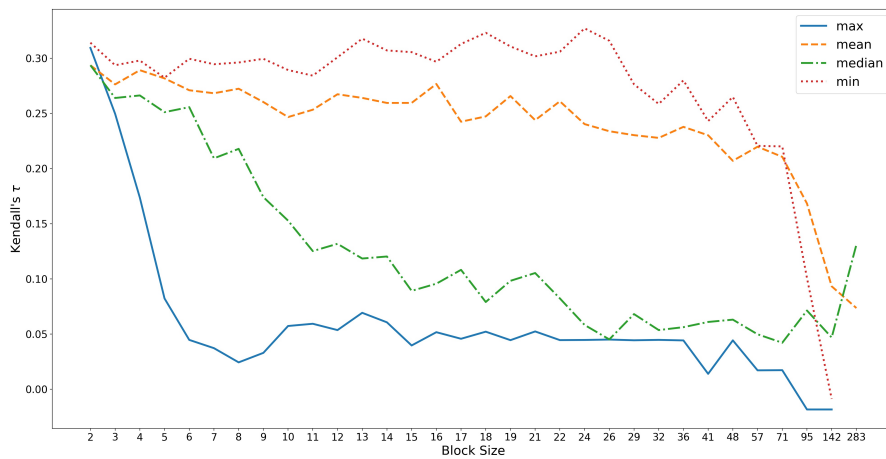


Figure F.8: Kendall's τ between the pixel baseline and the median matrix as a function of block size and aggregator function, using the Manhattan distance function with optimized weights.

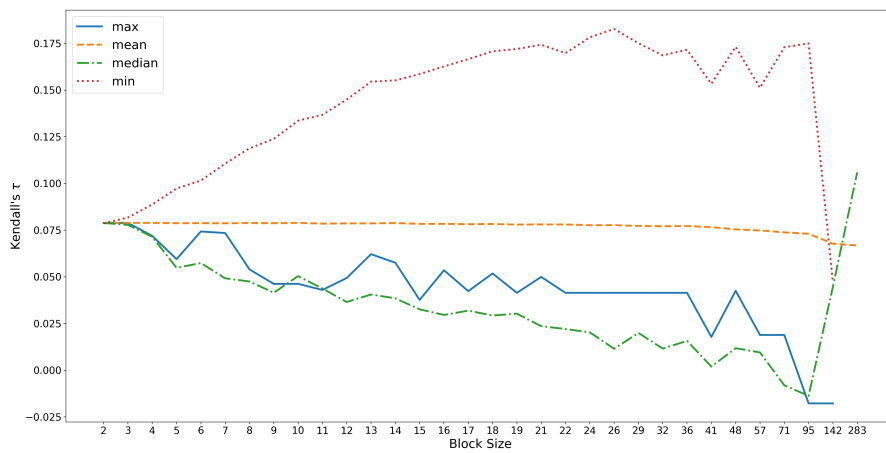


Figure F.9: Kendall's τ between the pixel baseline and the median matrix as a function of block size and aggregator function, using the negated inner product with uniform weights as distance function.

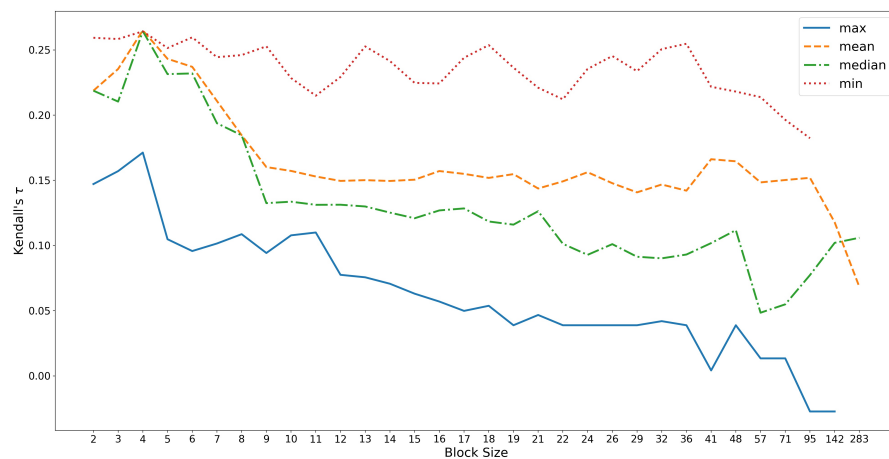


Figure F.10: Kendall's τ between the pixel baseline and the median matrix as a function of block size and aggregator function, using the negated inner product with optimized weights as distance function.

G

LEARNING A MAPPING INTO SHAPE SPACE

In Chapter 12, we trained different variants of convolutional neural networks (CNNs, cf. Section 6.2.2) on the task of mapping raw pixel input onto coordinates in similarity spaces for the `SHAPE` domain. This appendix contains some additional pieces of information with respect to our machine learning experiments. In Section G.1, we list the classes contained in the datasets TU Berlin [143] and Sketchy [348]. In Sections 12.4.2 and 12.4.3, we investigated the performance of transfer learning with classification-based and reconstruction-based networks, which were pretrained on sketch data, respectively. Sections G.2 and G.3 provide the results of the respective lasso regressors for different regularization strengths. In Sections 12.5.1 and 12.5.2, we then conducted some multitask learning experiments, whose raw results can be found in Sections G.4 and G.5, respectively. Finally, Sections 12.6.1 and 12.6.2 contained the results of some generalization experiments, where we fixed the hyperparameter settings of the machine learning architecture, and varied the dimensionality of the target similarity space. The raw results of these experiments can be found in Sections G.6 and G.7.

Overview

G.1 CLASS STRUCTURE OF SKETCH DATASETS

Common (98)			
airplane	alarm clock	ant	apple
axe	banana	bear	bee
bell	bench	bicycle	blimp
bread	butterfly	camel	candle
cannon	car	castle	cat
chair	church	couch	cow
crab	crocodile	cup	dog
dolphin	door	duck	elephant
eyeglasses	fan	fish	flower
frog	giraffe	guitar	hamburger
hammer	harp	hat	hedgehog
helicopter	horse	hot air balloon	hotdog
hourglass	kangaroo	knife	lion
lobster	mouse	motorcycle	mushroom
owl	parrot	pear	penguin
piano	pickup truck	pig	pineapple
pistol	pizza	pretzel	rabbit
rifle	sailboat	saxophone	scissors
scorpion	sea turtle	seagull	shark
sheep	shoe	skyscraper	snail
snake	spider	spoon	squirrel
strawberry	swan	sword	table
teapot	teddy bear	tiger	tree
trumpet	umbrella	violin	windmill
wine bottle	zebra		

Table G.1: List of the 98 common categories shared by TU Berlin [143] and Sketchy [348].

TU Berlin only (152)			
angel	arm	armchair	ashtray
backpack	barn	baseball bat	basket
bathhtub	bed	beer mug	binoculars
book	bookshelf	boomerang	bottle opener
bowl	brain	bridge	bulldozer
bus	bush	cabinet	cactus
cake	calculator	camera	canoe
carrot	cell phone	chandelier	cigarette
cloud	comb	computer monitor	computer mouse
crane	crown	diamond	donut
door handle	dragon	ear	envelope
eye	face	feather	fire hydrant
flashlight	floor lamp	flying bird	flying saucer
foot	fork	frying pan	grapes
grenade	hand	head	head phones
helmet	house	human skeleton	ice cream cone
ipod	key	keyboard	ladder
laptop	leaf	lightbulb	lighter
loudspeaker	mailbox	megaphone	mermaid
microphone	microscope	monkey	moon
mosquito	mouth	mug	nose
octopus	palm tree	panda	paper clip
parachute	parking meter	pen	person sitting
person walking	pigeon	pipe	potted plant
power outlet	present	pumpkin	purse
race car	radio	rainbow	rollerblades
rooster	santa clause	sattelite	sattelite dish
screwdriver	ship	shovel	skateboard
skull	snowboard	snowman	socks
space shuttle	speed boat	sponge bob	standing bird
stapler	streetlight	submarine	suitcase
sun	suv	syringe	t-shirt
tablelamp	teacup	telephone	tennis racket
tent	tire	toilet	tomato
tooth	toothbrush	tractor	traffic light
train	trombone	trousers	truck
tv	van	vase	walkie talkie
wheel	wheelbarrow	wineglass	wrist watch

Table G.2: List of the categories, which are contained in TU Berlin [143], but not in Sketchy [348].

Sketchy only (27)			
ape	armor	bat	beetle
cabin	chicken	deer	geyser
hermit crab	jack o lantern	jellyfish	lizard
raccoon	racket	ray	rhinoceros
rocket	saw	seal	songbird
starfish	tank	turtle	volcano
wading bird	wheelchair	window	

Table G.3: List of the categories, which are contained in Sketchy [348], but not in TU Berlin [143].

G.2 TRANSFER LEARNING ON SKETCH CLASSIFICATION

β	Training			Test		
	MSE	MED	R ²	MSE	MED	R ²
0	0.0479	0.2004	0.9520	0.5567	0.6879	0.4409
0.001	0.0748	0.2512	0.9249	0.5576	0.6846	0.4381
0.002	0.0819	0.2632	0.9176	0.5583	0.6840	0.4374
0.005	0.0960	0.2856	0.9035	0.5537	0.6797	0.4426
0.01	0.1116	0.3084	0.8879	0.5384	0.6704	0.4589
0.02	0.1350	0.3399	0.8642	0.5138	0.6574	0.4846
0.05	0.1911	0.4062	0.8080	0.4775	0.6419	0.5216
0.1	0.2754	0.4948	0.7237	0.4865	0.6586	0.5133
0.2	0.4475	0.6481	0.5519	0.5696	0.7309	0.4310
0.5	0.9356	0.9615	0.0642	0.9583	0.9728	0.0415
≥ 1	0.9984	0.9930	0.0016	1.0146	1.0010	-0.0147

Table G.4: Results for the lasso regressor based on C_{DEFAULT} with respect to the four-dimensional mean target space, using 10% salt and pepper noise in the inputs during training, and no noise during testing. Best test set results are highlighted in boldface.

β	Training			Test		
	MSE	MED	R ²	MSE	MED	R ²
0	0.0513	0.2073	0.9486	0.554	0.6848	0.4451
0.001	0.0557	0.2156	0.9442	0.5485	0.6806	0.4501
0.002	0.0598	0.2235	0.9400	0.5539	0.6830	0.4442
0.005	0.0689	0.2401	0.9309	0.5656	0.6876	0.4320
0.01	0.0783	0.2561	0.9214	0.5709	0.6899	0.4262
0.02	0.0900	0.2748	0.9097	0.5705	0.6898	0.4262
0.05	0.1123	0.3076	0.8872	0.5538	0.6806	0.4431
0.1	0.1395	0.3439	0.8597	0.5343	0.6718	0.4635
0.2	0.1851	0.3983	0.8139	0.5065	0.6602	0.4929
0.5	0.3121	0.5285	0.6871	0.5120	0.6783	0.4892
1	0.5238	0.7064	0.4763	0.6164	0.7654	0.3851
2	0.9233	0.9552	0.0770	0.9456	0.9664	0.0547
≥ 5	0.9984	0.9930	0.0016	1.0146	1.0010	-0.0147

Table G.5: Results for the lasso regressor based on C_{LARGE} with respect to the four-dimensional mean target space, using 10% salt and pepper noise in the inputs during training, and no noise during testing. Best test set results are highlighted in boldface.

β	Training			Test		
	MSE	MED	R ²	MSE	MED	R ²
0	0.0563	0.2145	0.9433	0.7307	0.7825	0.2624
0.001	0.0642	0.2287	0.9353	0.6723	0.7475	0.3219
0.002	0.0712	0.2408	0.9283	0.6502	0.7347	0.3444
0.005	0.0880	0.2682	0.9114	0.6275	0.7213	0.3682
0.01	0.1094	0.2996	0.8898	0.6038	0.7080	0.3925
0.02	0.1427	0.3433	0.8562	0.5778	0.6946	0.4190
0.05	0.2184	0.4299	0.7801	0.5478	0.6815	0.4505
0.1	0.3186	0.5313	0.6797	0.5584	0.6994	0.4406
0.2	0.5005	0.6874	0.4981	0.6471	0.7758	0.3521
0.5	0.9874	0.9875	0.0126	1.0084	0.9978	-0.0084
≥ 1	0.9984	0.9930	0.0016	1.0146	1.0010	-0.0147

Table G.6: Results for the lasso regressor based on $C_{\text{CORRELATION}}$ with respect to the four-dimensional mean target space, using 10% salt and pepper noise in the inputs during training, and no noise during testing. Best test set results are highlighted in boldface.

β	Training			Test		
	MSE	MED	R ²	MSE	MED	R ²
0	0.0563	0.2145	0.9433	0.7307	0.7825	0.2624
0.001	0.0642	0.2287	0.9353	0.6723	0.7475	0.3219
0.002	0.0712	0.2408	0.9283	0.6502	0.7347	0.3444
0.005	0.0880	0.2682	0.9114	0.6275	0.7213	0.3682
0.01	0.1094	0.2996	0.8898	0.6038	0.7080	0.3925
0.02	0.1427	0.3433	0.8562	0.5778	0.6946	0.4190
0.05	0.2184	0.4299	0.7801	0.5478	0.6815	0.4505
0.1	0.3186	0.5313	0.6797	0.5584	0.6994	0.4406
0.2	0.5005	0.6874	0.4981	0.6471	0.7758	0.3521
0.5	0.9874	0.9875	0.0126	1.0084	0.9978	-0.0084
≥ 1	0.9984	0.9930	0.0016	1.0146	1.0010	-0.0147

Table G.7: Results for the lasso regressor based on C_{SMALL} with respect to the four-dimensional mean target space, using 10% salt and pepper noise in the inputs during training, and no noise during testing. Best test set results are highlighted in boldface.

G.3 TRANSFER LEARNING ON SKETCH RECONSTRUCTION

β	Training			Test		
	MSE	MED	R ²	MSE	MED	R ²
0	0.2206	0.4288	0.7773	0.9709	0.9054	0.0168
0.001	0.3873	0.579	0.6098	0.8580	0.8664	0.1327
0.002	0.4480	0.6266	0.5488	0.8663	0.8743	0.1246
0.005	0.5179	0.6784	0.4787	0.8649	0.8784	0.1267
0.01	0.5532	0.7045	0.4436	0.8518	0.8760	0.1406
0.02	0.5845	0.7283	0.4125	0.8384	0.8739	0.1551
0.05	0.6378	0.7695	0.3591	0.8315	0.8797	0.1631
0.1	0.7058	0.8188	0.2907	0.8501	0.8988	0.1452
0.2	0.8220	0.8948	0.1742	0.9015	0.9367	0.0947
0.5	0.9981	0.9929	0.0019	1.0144	1.0009	-0.0145
≥ 1	0.9984	0.9930	0.0016	1.0146	1.0010	-0.0147

Table G.8: Results for the lasso regressor based on R_{DEFAULT} with respect to the four-dimensional mean target space, using 10% salt and pepper noise in the inputs during training, and no noise during testing. Best test set results are highlighted in boldface.

β	Training			Test		
	MSE	MED	R^2	MSE	MED	R^2
0	0.1410	0.3383	0.8587	1.0791	0.9362	-0.0886
0.001	0.1698	0.3710	0.8298	0.9160	0.8755	0.0767
0.002	0.1898	0.3931	0.8097	0.8697	0.8555	0.1245
0.005	0.2309	0.4361	0.7686	0.8002	0.8261	0.1958
0.01	0.2826	0.4873	0.7169	0.7571	0.8103	0.2400
0.02	0.3610	0.5598	0.6383	0.7376	0.8102	0.2605
0.05	0.5177	0.6894	0.4809	0.7536	0.8363	0.2443
0.1	0.6839	0.8083	0.3139	0.8268	0.8902	0.1708
0.2	0.8976	0.9384	0.1001	0.9518	0.9669	0.0465
≥ 0.5	0.9984	0.993	0.0016	1.0146	1.0010	-0.0147

Table G.9: Results for the lasso regressor based on R_{BEST} with respect to the four-dimensional mean target space, using 10% salt and pepper noise in the inputs during training, and no noise during testing. Best test set results are highlighted in boldface.

G.4 MULTITASK LEARNING ON SKETCH CLASSIFICATION

λ_3	Epochs	Corr.	Acc. TU Berlin	Acc. Sketchy	MSE	MED	R^2
0	188.6	0.2743	0.6320	0.7933	–	–	–
0.0625	76.4	0.4141	0.5640	0.7510	0.4041	0.5920	0.5775
0.125	101.8	0.4240	0.6113	0.7835	0.4316	0.6162	0.5481
0.25	66.8	0.4210	0.5570	0.7504	0.4733	0.6428	0.5054
0.5	55.6	0.4346	0.5034	0.7172	0.4668	0.6268	0.5114
1	106.2	0.4505	0.6063	0.7747	0.4521	0.6348	0.5321
2	68.4	0.4557	0.5333	0.7390	0.4819	0.6518	0.5014

Table G.10: Test set results for the multitask learner based on C_{DEFAULT} with respect to the four-dimensional mean target space, using 10% salt and pepper noise in the inputs during training, and no noise during testing. Best results for all evaluation metrics are highlighted in boldface.

λ_3	Epochs	Corr.	Acc. TU Berlin	Acc. Sketchy	MSE	MED	R^2
0	178.4	0.2777	0.6274	0.7908	–	–	–
0.0625	103.0	0.4053	0.6059	0.7802	0.4513	0.6299	0.5208
0.125	123.2	0.4118	0.6145	0.7818	0.4182	0.6020	0.5567
0.25	106.6	0.4252	0.5938	0.7697	0.4641	0.6350	0.5111
0.5	108.6	0.4422	0.6115	0.7785	0.4318	0.6156	0.5471
1	111.8	0.4482	0.6111	0.7780	0.4664	0.6418	0.5157
2	67.2	0.4426	0.5754	0.7587	0.4771	0.6492	0.4999

Table G.11: Test set results for the multitask learner based on C_{SMALL} with respect to the four-dimensional mean target space, using 10% salt and pepper noise in the inputs during training, and no noise during testing. Best results for all evaluation metrics are highlighted in boldface.

λ_3	Epochs	Corr.	Acc. TU Berlin	Acc. Sketchy	MSE	MED	R^2
0	5.6	0.3292	0.3636	0.6146	–	–	–
0.0625	72.2	0.4138	0.5058	0.6842	0.6121	0.7099	0.3683
0.125	95.2	0.4302	0.5160	0.6874	0.4912	0.6353	0.4913
0.25	115.2	0.4333	0.5153	0.6927	0.4879	0.6338	0.4905
0.5	82.2	0.4474	0.5131	0.6934	0.5086	0.6451	0.4727
1	23.0	0.4407	0.5113	0.6836	0.5572	0.6767	0.4160
2	96.6	0.4534	0.5196	0.7018	0.4513	0.6115	0.5201

Table G.12: Test set results for the multitask learner based on $C_{\text{CORRELATION}}$ with respect to the four-dimensional mean target space, using 10% salt and pepper noise in the inputs during training, and no noise during testing. Best results for all evaluation metrics are highlighted in boldface.

G.5 MULTITASK LEARNING ON SKETCH RECONSTRUCTION

λ_3	Epochs	Corr.	Rec. Error	MSE	MED	R^2
0	56.2	0.2280	0.1303	–	–	–
0.0625	25.2	0.3895	0.1336	0.6712	0.7471	0.2831
0.125	41.4	0.3887	0.1332	0.6705	0.7470	0.2903
0.25	33.0	0.3833	0.1355	0.6692	0.7556	0.2944
0.5	42.4	0.3790	0.1356	0.6471	0.7402	0.3166
1	35.2	0.3837	0.1363	0.6384	0.7423	0.3179
2	42.4	0.3783	0.1391	0.6211	0.7297	0.3369

Table G.13: Test set results for the multitask learner based on R_{DEFAULT} with respect to the four-dimensional mean target space, using 10% salt and pepper noise in the inputs during training, and no noise during testing. Best results for all evaluation metrics are highlighted in boldface.

λ_3	Epochs	Corr.	Rec. Error	MSE	MED	R^2
0	192.4	0.3019	0.0828	–	–	–
0.0625	9.8	0.3893	0.1023	0.5504	0.6851	0.4144
0.125	4.0	0.4049	0.1135	0.5587	0.6870	0.4036
0.25	4.6	0.4033	0.1114	0.5506	0.6855	0.4213
0.5	4.8	0.3987	0.1129	0.5494	0.6860	0.4205
1	6.4	0.3788	0.1876	0.6827	0.7507	0.2605
2	6.4	0.3948	0.1153	0.5520	0.6846	0.4141

Table G.14: Test set results for the multitask learner based on R_{BEST} with respect to the four-dimensional mean target space, using 10% salt and pepper noise in the inputs during training, and no noise during testing. Best results for all evaluation metrics are highlighted in boldface.

G.6 CLASSIFICATION-BASED GENERALIZATION EXPERIMENTS

Number of Dimensions	MSE	MED	R^2
1	1.0000	0.8628	0.0000
2	1.0000	0.9642	0.0000
3	1.0000	0.9904	0.0000
4	1.0000	0.9940	0.0000
5	1.0000	0.9957	0.0000
6	1.0000	0.9964	0.0000
7	1.0000	0.9971	0.0000
8	1.0000	0.9975	0.0000
9	1.0000	0.9977	0.0000
10	1.0000	0.9980	0.0000

Table G.15: Test set results for the zero baseline with respect to the mean target spaces of different dimensionality, using 10% salt and pepper noise in the inputs during training, and no noise during testing.

Number of Dimensions	MSE	MED	R^2
1	0.5618	0.5983	0.4382
2	0.4667	0.6236	0.5277
3	0.4875	0.6585	0.5088
4	0.5090	0.6828	0.4924
5	0.5427	0.7138	0.4515
6	0.5549	0.7246	0.4395
7	0.5715	0.7355	0.4100
8	0.5974	0.7527	0.3934
9	0.6282	0.7760	0.3588
10	0.6528	0.7906	0.3415

Table G.16: Test set results for the transfer learner with $\beta = 0.005$ based on the $C_{\text{INCEPTION}}$ feature space with respect to the mean target spaces of different dimensionality, using 10% salt and pepper noise in the inputs during training, and no noise during testing.

Number of Dimensions	MSE	MED	R^2
1	0.5072	0.5396	0.4928
2	0.3893	0.5472	0.607
3	0.4353	0.5996	0.5585
4	0.4612	0.6337	0.5374
5	0.4666	0.6440	0.5288
6	0.5151	0.6826	0.4812
7	0.5305	0.6957	0.4519
8	0.5601	0.7188	0.4298
9	0.6037	0.7503	0.3652
10	0.6145	0.7594	0.3759

Table G.17: Test set results for the transfer learner with $\beta = 0.02$ based on the C_{SMALL} feature space with respect to the mean target spaces of different dimensionality, using 10% salt and pepper noise in the inputs during training, and no noise during testing.

Number of Dimensions	MSE	MED	R^2
1	0.6356	0.5766	-130.4124
2	0.4178	0.5537	0.4525
3	0.4553	0.6064	0.5266
4	0.4041	0.5920	0.5775
5	0.4499	0.6332	0.5323
6	0.4560	0.6403	0.5273
7	0.5130	0.6839	0.4691
8	0.5376	0.7063	0.4488
9	0.5442	0.7132	0.4397
10	0.5895	0.7399	0.3978

Table G.18: Test set results for the multitask learner with $\lambda_3 = 0.0625$ based on the C_{DEFAULT} configuration with respect to the mean target spaces of different dimensionality, using 10% salt and pepper noise in the inputs during training, and no noise during testing.

G.7 RECONSTRUCTION-BASED GENERALIZATION EXPERIMENTS

Number of Dimensions	MSE	MED	R^2
1	0.8255	0.7113	0.1745
2	0.6637	0.7159	0.3337
3	0.7145	0.7822	0.2790
4	0.7376	0.8102	0.2605
5	0.6982	0.7971	0.3049
6	0.7283	0.8204	0.2733
7	0.7672	0.8482	0.2249
8	0.7680	0.8522	0.2179
9	0.7926	0.8690	0.1846
10	0.8032	0.8771	0.1869

Table G.19: Test set results for the transfer learner with $\beta = 0.02$ based on the R_{BEST} feature space with respect to the mean target spaces of different dimensionality, using 10% salt and pepper noise in the inputs during training, and no noise during testing.

Number of Dimensions	MSE	MED	R^2
1	0.6126	0.5980	-62.8045
2	0.3868	0.5263	0.5454
3	0.4997	0.6456	0.4641
4	0.5504	0.6851	0.4144
5	0.5765	0.7099	0.3868
6	0.5634	0.7115	0.3977
7	0.5968	0.7354	0.3659
8	0.6421	0.7622	0.3211
9	0.7020	0.8022	0.2588
10	0.7011	0.8032	0.2584

Table G.20: Test set results for the multitask learner with $\lambda_3 = 0.0625$ based on the R_{BEST} configuration with respect to the mean target spaces of different dimensionality, using 10% salt and pepper noise in the inputs during training, and no noise during testing.

ERKLÄRUNG ÜBER DIE EIGENSTÄNDIGKEIT DER ERBRACHTEN WISSENSCHAFTLICHEN LEISTUNG

Ich erkläre hiermit, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Bei der Auswahl und Auswertung folgenden Materials haben mir die nachstehend aufgeführten Personen in der jeweils beschriebenen Weise entgeltlich/ unentgeltlich geholfen.

1. *Mingya Liu* und *Ulf Krumnack* haben (unentgeltlich) gemeinsam mit mir den Antragstext für die Summer School "Concepts in Action: Representation, Learning, and Application" geschrieben. Dieser Antragstext wurde von Mingya Liu und mir teilweise im Einleitungskapitel [50] zum gleichnamigen Sammelband [49] wiederverwendet. Einige Textstellen daraus habe ich mit kleinen Änderungen im Einleitungstext zu Kapitel 1 sowie in Abschnitt 1.1 verwendet.
2. *Elektra Kypridemou* hat (unentgeltlich) gemeinsam mit mir die ursprüngliche Version der NOUN-Studie (Kapitel 9) durchgeführt. Wie jeweils zu Beginn der Kapitel 8 und 9 bereits dargelegt, hat Elektra Kypridemou bei unserer gemeinsamen Publikation [40] den Hintergrund zu Ähnlichkeitsbewertungen und Multidimensional Scaling beschrieben, sowie die dort betrachteten Ähnlichkeitsräume aus den vorhandenen Ähnlichkeitsbewertungen extrahiert. Der in Abschnitt 8.7 beschriebene hybride Ansatz sowie die Machine Learning Experimente (siehe Abschnitt 9.2) wurden von mir entwickelt. Die in Abschnitten 8.1, 8.2 und 8.7 dargestellten Inhalte basieren teilweise auf unserer gemeinsamen Veröffentlichung, wurden von mir aber noch deutlich detaillierter ausgearbeitet. Auch die in Kapitel 9 besprochene Studie auf dem NOUN Datenset wurde von mir nach unserer Kollaboration nochmals in verbesserter und vertiefter Weise wiederholt.

3. *Margit Scheibel* hat mir freundlicherweise (unentgeltlich) Daten aus einer ihrer Studien [349] bezüglich der visuellen Ähnlichkeit von Strichzeichnungen zur Verfügung gestellt. In der daraus erwachsenen (unentgeltlichen) Kooperation hat Margit Scheibel die psychologischen Studien (siehe Abschnitt 10.2) geplant und durchgeführt, sowie eine statistische Auswertung der Rohdaten (siehe Abschnitt 10.3) vorgenommen. Für die gegenwärtig noch in Vorbereitung befindliche Publikation [54] hat sie außerdem relevante Quellen für den Literaturüberblick aus dem Bereich (Neuro-)Psychologie gesichtet. Die in Abschnitten 10.2 und 10.3 enthaltenen Texte basieren auf unserem Entwurf für einen gemeinsamen Artikel und wurden maßgeblich von Margit Scheibel geschrieben, für den Kontext dieser Dissertation allerdings von mir noch angepasst. Da die ursprünglichen Skripte zur statistischen Analyse nicht mehr verfügbar waren, habe ich die statistische Analyse von Abschnitt 10.3 nochmals basierend auf ihrer Beschreibung reimplementiert und selbst durchgeführt. Für den Literaturüberblick in Abschnitt 10.1 habe ich mich an den von Margit Scheibel ausgewählten Autoren und Quellen orientiert und diese mit eigenen Rechercheergebnissen (vor allem in Bezug auf Conceptual Spaces, Computer Vision und CNNs) ergänzt und vertieft. Die in Kapitel 11 beschriebene Analyse der Ähnlichkeitsräume wurde komplett von mir durchgeführt, ist aber auch Teil der geplanten gemeinsamen Veröffentlichung.

Weitere Personen waren an der inhaltlichen materiellen Erstellung der vorliegenden Arbeit nicht beteiligt. Insbesondere habe ich hierfür nicht die entgeltliche Hilfe von Vermittlungs- bzw. Beratungsdiensten (Promotionsberater oder andere Personen) in Anspruch genommen. Niemand hat von mir unmittelbar oder mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen.

Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form einer anderen Prüfungsbehörde vorgelegt.

St. Leon-Rot, 30.07.2022

(Ort, Datum)

(Unterschrift)

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede and Ivo Pletikosić. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both L^AT_EX and L^yX:

<https://bitbucket.org/amiede/classicthesis/>

Final Version as of October 20, 2023 (`classicthesis` v4.6).