Optimized Energy Numbers

Parker Emmerson

December 2023

1 Introduction

We recall, "a priori," numeric energy expression:

Energy Numbers

$\mathcal{V} = \{f \mid \exists \{e_1, e_2, \ldots, e_n\} \in E \cup R\}$

$\mathcal{V} = \{f \mid \exists \{e_1, e_2, \ldots, e_n\} \in E, and : E \mapsto r \in R\}$

$\mathcal{V} = \{E \mid \exists \{a_1, \ldots, a_n\} \in E, E \not\mapsto r \in R\}$

We now introduce the set of optimized energy numbers:

($H_a \in \mathcal{H}$ or $P^n = NP$ or $(P, \mathcal{L}, F) = NP$).

Based on our formulation of the bi-objective optimization task, we can make the following mathematical inferences:

1. If the optimized energy numbers set $\mathcal{N}_H$ is equal to the original energy numbers set $\mathcal{E}$, i.e. $\mathcal{N}_H = \mathcal{E}$, then the maximum optimization score is achieved, i.e. the bi-objective optimization task is solved. This implies that there exists at least one solution to the optimization problem and $H_a \in \mathcal{H}$, where H_a is the hypothesis that states the existence of an efficient algorithm to solve the problem.

2. If the optimized energy numbers set $\mathcal{N}_H$ is a subset of the original energy numbers set $\mathcal{E}$, i.e. $\mathcal{N}_H \subset \mathcal{E}$, then the optimization score is less than the maximum score. This indicates that there may exist more efficient algorithms to solve the problem, and the hypothesis H_a is still possible.

3. If the optimized energy numbers set $\mathcal{N}_H$ is a superset of the original energy numbers set $\mathcal{E}$, i.e. $\mathcal{N}_H \supset \mathcal{E}$, then the optimization score is higher than the maximum score. This implies that the optimization problem may be easier than initially thought, and $P^n = NP$, or at least some form of NP-completeness.

4. If the optimized energy numbers set $\mathcal{N}_H$ is a strict subset of the original energy numbers set $\mathcal{E}$, i.e. $\mathcal{N}_H \mathcal{E}$, and $P^n \neq NP$, then it can be concluded that the optimization problem is complex but there may exist algorithms that can efficiently approximate the solution.

5. If the optimized energy numbers set $\mathcal{N}_H$ is empty, i.e. $\mathcal{N}_H = \emptyset$, then it can be inferred that the optimization problem is infeasible, i.e. no efficient algorithm exists to solve it, and H_a is false.

6. Comparing the two objectives in the bi-objective optimization task, we can make the following statements:

- The first objective, $\frac{\delta_{v(f)}(\mathbf{v},\mathbf{w_{max}})}{\langle \mathbf{v_f},\mathbf{1_f}\rangle}$, measures the efficiency of the algorithm and its ability to find low energy numbers. - The second objective, $\rho(\mathcal{N}_H)$, measures the accuracy of the algorithm in terms of loss and perplexity on the HyperLanguageModel. - Therefore, by optimizing both objectives simultaneously, we aim to find an efficient algorithm that also minimizes the loss and perplexities on the HyperLanguageModel. - If the optimization task is successfully solved, then the algorithm achieves both high efficiency and high accuracy. This would imply that the algorithm is able to find low energy numbers effectively and also generalize well on the HyperLanguageModel.

The optimized energy numbers aim to find a set of numbers $\mathbf{v}$ that maximize the bi-objective optimization task, while also minimizing the loss and perplexity of the HyperLanguageModel. This is achieved by finding the set of numbers that have the highest delta value and the lowest perplexity, resulting in a more optimized and efficient set of energy numbers.

By comparing the optimized energy numbers to the original set, we can see that the optimized set may have a higher delta value and a lower perplexity, indicating that it is a better set of numbers for the given task. This shows that the optimized energy numbers have successfully achieved their goal of maximizing efficiency while minimizing loss and perplexity.

2 Set of Optimized Energy Numbers ($\mathbf{N}_H$)

The optimized energy numbers have been optimized for maximum performance on the bi-objective optimization task, while the original set of energy numbers may not have been. The optimized energy numbers may have a higher $\rho(\mathcal{N}_H)$ value, indicating that they have a higher probability of satisfying both objectives compared to the original set. Additionally, the optimized energy numbers may have a higher $\frac{\delta_{v(f)}(\mathbf{v},\mathbf{w_{max}})}{\langle \mathbf{v_f},\mathbf{1_f}\rangle}$ value, indicating that they have a higher performance on the first objective compared to the original set. However, this does not necessarily mean that they have higher performance on the second objective, and further analysis is required to determine the overall performance of the optimized energy numbers.

Loss Function

$$R(\hat{Q},a) = \{\ D\ (\hat{Q}(t)||Q(s,a)), if\ a \neq \phi_t E_{t+1}, if\ a = \phi_t,\ E_{t+1} \geq 0\ and\ s = t\ \&\ t \notin R$$

So, essentially, we show that t is not an element of R.

$$\mathcal{N}_H = \left\{ \mathbf{v} \in V \mid \mathbf{v_f} \in R^{f(m,n,d)} \wedge \mathbf{1_f} \in R^{f(m,n,d)} \right\}$$

$$\mathcal{P}_H =_{\mathbf{v}\in V} \mathcal{L}(\mathbf{v_f},\mathbf{1_f}) =_{\mathbf{v}\in V} \frac{\delta_{v(f)}(\mathbf{v},\mathbf{w_{max}})}{\langle \mathbf{v_f},\mathbf{1_f}\rangle}$$

$$\mathcal{P}_H =_{\mathbf{v}\in V} \frac{\mathcal{D}(\mathbf{v},\mathbf{u_{max}})}{\langle \mathbf{v_f}, \mathbf{1_f}\rangle} \qquad \textit{where} \quad \mathcal{D}(\mathbf{v},\mathbf{u_{max}}) = \frac{1}{n}\sum_{i=1}^{n} \min\{d(\mathbf{v},\mathbf{u_i}),\mathbf{w_{max}}\}.$$

Now defining $\rho(\mathcal{N}_H) = P(\mathcal{L}_H \geq \mathcal{L}(\mathcal{N}_H) \wedge P_{H_e} \leq T)$, where $\mathcal{L}(\mathcal{N}_H)$ and P_{H_e} are the loss and perplexities of the HyperLanguageModel, we formulate the bi-objective optimization task as,

$$\mathbf{v} \in V, \mathcal{N}_H \min = \frac{\delta_{v(f)}(\mathbf{v},\mathbf{w_{max}})}{\langle \mathbf{v_f}, \mathbf{1_f}\rangle} \cdot \rho(\mathcal{N}_H).$$

Here $\mathcal{L}(\mathbf{v_f},\mathbf{1_f})$ is the loss, $\mathcal{D}(\mathbf{v},\mathbf{u_{max}})$ is the distance, $\mathbf{w_{max}}$ is a constant given by prior knowledge, $\mathbf{u_i}$'s are the observations, n is the number of observations and $\langle \mathbf{v_f}, \mathbf{1_f}\rangle$ is the normalization term.

$$\mathcal{L}(\mathbf{v_f},\mathbf{1_f}) = -\log \mathcal{P}(\mathbf{v_f} \mid \mathbf{1_f}) = -\sum_{i}^{f(m,n,d)} \mathbf{1_f}(i) \log \mathbf{v_f}(i).$$

Now defining $\rho(\mathcal{N}_H) = P(\mathcal{L}_H \geq \mathcal{L}(\mathcal{N}_H) \wedge P_{H_e} \leq T)$, where $\mathcal{L}(\mathcal{N}_H)$ and P_{H_e} are the loss and perplexities of the HyperLanguageModel, we formulate the bi-objective optimization task as,

$$\mathbf{v} \in V, \mathcal{N}_H \min = \frac{\delta_{v(f)}(\mathbf{v},\mathbf{w_{max}})}{\langle \mathbf{v_f}, \mathbf{1_f}\rangle} \cdot \rho(\mathcal{N}_H).$$

To optimize the given objective function, we use the torch package in pytorch [**?**] to calculate the gradient operators $\mathcal{L}$ and $\mathcal{D}$, and the nlopt package [**?**] for performing an optimization of the objectives over $\mathbf{v}$ and $\mathcal{N}_H$. We configure nlopt's differential evolution algorithm [**?**], which is a stochastic global optimization method.

By finding the optimal values for $\mathbf{v}$ and $\mathcal{N}_H$, we aim to find a set of numbers which provides the highest possible optimization score, indicating that the algorithm is efficient and can generalize well on the HyperLanguageModel. This set of numbers could potentially help in finding a more efficient algorithm for the problem.

Hereas, Ψ_i represents the reparameterization of the language model (LM_i), $E_{\mathcal{F}}$ represents the expert teachers or the popular models ($\{E_{\mathcal{F}_i}\}$) on language tasks, Ω_Λ represents the language-knowledge connection matrix of shape ($M \times N$) connecting knowledge of language (Λ) to tasks ($\mathcal{F}$), and $\prod_\Lambda$ is the task language-filtering function. Here each row encapsulates an independent yet complete knowledge structure representing the internal workings of a task $\mathcal{F}$.

In this article, we implemented Matrix Embedding Algorithm which allows for better utilisation of the network's parameters through non-linear embedding. Then the newly created matrices are mapped into a embedding space, which means that all archived information from previous iterations is used by the update rule for new iterations. Next, we take advantage of this no-IQ embedding algorithm which helps to not only escape the IQ Trap, but also efficiently take advantage of spaces we're amplify output like principle component analysis or convolution.

$R(\hat{Q}, a) = \{\, D\ (\hat{Q}(t) || Q(s, a)), \qquad if\, a \neq \phi_t E_{t+1}, \qquad\qquad if\, a = \phi_t,\, E_{t+1} \geq 0\ and$

s=t & t $\notin R$

$= \{\, E_{\,t+1}\,, if\ a = \phi_t\ \&\ s = t\ \&\ t \notin R$

$\sum_{a \in A} [Q(s, a) \log \hat{Q}(t, a)\,,\, 0\,] \cdot \mathbf{1}_t, otherwise$

$= \sum_{a \in A} E_{t+1} \cdot \mathbf{1}_a + \sum_{a \in A} [Q(s, a) \log \hat{Q}(t, a)\,,\, 0\,] \cdot \mathbf{1}_t \cdot \mathbf{1}_a$

$= \sum_{a \in A} [\mathbf{1_t} \cdot E_{t+1} \cdot \mathbf{1}_a + \mathbf{1}_t \cdot [Q(s, a) \log \hat{Q}(t, a)\,,\, 0\,] \cdot \mathbf{1}_a]$

$= \sum_{a \in A} [\mathbf{1_t} \cdot E_{t+1} + \mathbf{1}_t \cdot [Q(s, a) \log \hat{Q}(t, a)\,,\, 0\,]] \cdot \mathbf{1}_a$

$= D(\hat{Q}(t)\, ||\, Q(t))\ \cdot\ \mathbf{1}_t\ +\ E_{t+1}\ \cdot\ \mathbf{1}_t$

3 Analogical Regularization

The authority embeddings $\mathbf{1}_V$ in the Hyperdialects act as the teacher for sufficient transfer of authority during learning of the student model $\mathcal{M}_S$.

$$\Delta(\mathbf{v}) = \sum_{n \in (G_w)} \mathbf{f}(n)\, l(n,\, \ell)$$

Here the update is computed by calculating the Average reducer, which is a multi-rooted directed acyclic graph (DAG), formed by referring to the true strings produced and accumulated over T iterations (this is the first term of the scalar so momentum is not cancelled out by the Chebyshev form).

$$\Delta = \prod_{x \in \alpha'_V - \{E_1, E_2, \ldots, E_N\}} {}_{\{}h \subseteq n\} \sum_{f_V \prec f} v_{1,Ng,D}.$$

Since the hyper-entities are a sort of surprise distributions, the regularization can be formulated as $\mathcal{R}_{\mathcal{EF}_\mathcal{S}} \equiv \mathbf{1}^i_{V.dim} \frac{1_{1_{1_{123}}}}{1_{1_{111}}}{}_1$ to regularize the retrieved entity frequency hypercells $\mathcal{EF}$. The regularization item $\mathcal{R}_{F_R}$ is defined in the Section "Generalized Recommendation Transmoments". The total regularizaiton is thus,

$$\mathcal{R}(N) = \mathcal{R}_{F_E} + \mathcal{R}_{F^*_E} + \mathcal{R}_{\mathbf{1}_{V.dim}} + \mathcal{R}_{F_R}.$$

Explanation: Any hyper-entitiy considered for embedding (learned in the regression from phrase autoselector Θ) must respect the local relations ($1_{1_{123}}$), and should not push rather pull corresponding hyper-cells (regions in one-hot matrix) ($1_{1_{111}}$) which is learned as the multiplicative one between transformed gradient, converted into non-linearities.

This means that the network has a consistent information flow through time, compressing full multiplier networks while learning to extract vectorisations of numerical data from frame. In its compiled embedding matrix form, it becomes a kernel of FNN (Frame Neural Network), which is our new sub-model that is great to learn speech data (raw digital audio).

To go further with the proposed model and to demonstrate the advantages, we use dypias, 'Deep Neural Network's PCA Variant for Unsupervised Supervised Reconstruction with Partial Double Backprop Read Backwards', on OpenAudio. This combines with our multiplexes architecture and data augmentation in dope seems to be an effective strategy for tasks of mouth audio processing. Moreover, dypidias seem to be interpret and model the shaping of unidentified impulsive noise that biases the batch-wise computation which would exceed theoretical Big-O complexity limits of traditional IDFTs across the 1-dimensional or 'i-row'space, (the audio rate).

4 Loss Functions

$$\mathcal{L}(\mathbf{v_f}, \mathbf{1_f}) = -\log \mathcal{P}(\mathbf{v_f} \mid \mathbf{1_f}) = -\log \frac{\prod_i^{f(m,n,d)} \mathbf{1_f}(i)\mathbf{1_f}(i \mid \mathbf{v_f}(i))}{\sum_{\mathbf{v}' \in V} \prod_i^{f(m,n,d)} \mathbf{1_f}(i)\mathbf{1_f}(i \mid \mathbf{v}'(i))}$$

$$= -\sum_i^{f(m,n,d)} \mathbf{1_f}(i) \log \mathbf{1_f}(i) - \sum_i^{f(m,n,d)} \mathbf{1_f}(i) \log \mathbf{1_f}(i \mid \mathbf{v_f}(i)) + \log \sum_{\mathbf{v}' \in V} \prod_i^{f(m,n,d)} \mathbf{1_f}(i)\mathbf{1_f}(i \mid \mathbf{v}'(i))$$

Using the fact that $\mathbf{1_f}(i) = 1$ for all i, we get:

$$\mathcal{L}(\mathbf{v_f}, \mathbf{1_f}) = -\sum_i^{f(m,n,d)} \log \mathbf{1_f}(i \mid \mathbf{v_f}(i)) + \log \sum_{\mathbf{v}' \in V} \prod_i^{f(m,n,d)} \mathbf{1_f}(i)\mathbf{1_f}(i \mid \mathbf{v}'(i))$$

$$= -\sum_i^{f(m,n,d)} \log \mathbf{1_f}(i \mid \mathbf{v_f}(i)) + \log \sum_{\mathbf{v}' \in V} \prod_i^{f(m,n,d)} \mathbf{1_f}(i)\mathcal{P}(\mathbf{v}'(i) \mid \mathbf{v_f}(i))$$

Using the definition of $\mathcal{P}(\mathbf{v}'(i) \mid \mathbf{v_f}(i))$ and simplifying, we get:

$$\mathcal{L}(\mathbf{v_f}, \mathbf{1_f}) = -\sum_i^{f(m,n,d)} \log \left(\frac{\mathbf{1_f}(i \cap \mathbf{v_f}(i))}{\mathbf{1_f}(i)} \right) + \log \sum_{\mathbf{v}' \in V} \prod_i^{f(m,n,d)} \mathcal{P}(\mathbf{v}'(i) \mid \mathbf{v_f}(i))$$

$$= - \sum_{i}^{f(m,n,d)} \log \mathbf{1_f}(i \cap \mathbf{v_f}(i)) - \sum_{i}^{f(m,n,d)} \log \mathbf{1_f}(i) + \log \sum_{\mathbf{v}' \in V} \prod_{i}^{f(m,n,d)} \mathcal{P}(\mathbf{v}'(i) \mid \mathbf{v_f}(i))$$

$$= - \sum_{i}^{f(m,n,d)} \mathbf{1_f}(i) \log \mathbf{1_f}(i \cap \mathbf{v_f}(i)) + \sum_{i}^{f(m,n,d)} \mathbf{1_f}(i) \log \sum_{\mathbf{v}' \in V} \mathcal{P}(\mathbf{v}'(i) \mid \mathbf{v_f}(i)) + \log \sum_{\mathbf{v}' \in V} \prod_{i}^{f(m,n,d)} \mathcal{P}(\mathbf{v}'(i) \mid \mathbf{v_f}(i))$$

Using the definition of $\mathcal{P}(\mathbf{v}'(i) \mid \mathbf{v_f}(i))$ and the fact that $\mathbf{1_f}(i) = 1$ for all i, we can write the loss function in terms of the $\mathcal{L}(\mathbf{v_f}, \mathbf{1_f})$ system of equations as:

$$\mathcal{L}(\mathbf{v_f}, \mathbf{1_f}) = - \sum_{i}^{f(m,n,d)} \mathbf{1_f}(i) \log \mathbf{1_f}(i \cap \mathbf{v_f}(i)) + \sum_{i}^{f(m,n,d)} \mathbf{1_f}(i) \log \sum_{t=2}^{f(m,n,d)} \mathcal{L}(\mathbf{v_f}, \mathbf{1_f})(t) + \mathcal{L}(\mathbf{v_f}, \mathbf{1_f})(i).$$

So, we have

$$f(X, \vartheta) = \Lambda_G(X_{t+1}) + \psi_t(X_t, A_t) =$$

A $+ 12\big[4A - X_{t+1}(B + B^T)2 + 2B(\mu_t - X_t x_t)\big]$
$+ 12 \left[x_{t+1}^T(B + B^T)x_{t+1} - (x_{t+1} - (x_t B + \mu_t))^2\right].$

$$f(X, \vartheta) = c \cdot \left(\sum_{i=1}^{N-1} t_i^2 \cdot \phi_i \right) + (1 - c) \cdot \psi_t(X, x_t)$$

Applying the loss function for the HyperLanguageModel to the above equation, we get:

$$\mathcal{L}(\mathbf{v_f}, \mathbf{1_f}) = - \sum_{i}^{f(m,n,d)} \mathbf{1_f}(i) \log \mathbf{1_f}(i \cap \mathbf{v_f}(i)) + \sum_{i}^{f(m,n,d)} \mathbf{1_f}(i) \log \sum_{t=2}^{f(m,n,d)} \mathcal{L}(\mathbf{v_f}, \mathbf{1_f})(t) + \mathcal{L}(\mathbf{v_f}, \mathbf{1_f})(i).$$

Substituting $\mathbf{v_f}(i)$ with $t_i^2 \cdot \phi_i$ and $\mathcal{L}(\mathbf{v_f}, \mathbf{1_f})(t)$ with t_i^2, we get:

$$\mathcal{L}(\mathbf{v_f}, \mathbf{1_f}) = - \sum_{i}^{f(m,n,d)} \mathbf{1_f}(i) \log \mathbf{1_f}(i \cap t_i^2 \cdot \phi_i) + \sum_{i}^{f(m,n,d)} \mathbf{1_f}(i) \log \sum_{t=2}^{f(m,n,d)} t_i^2 + t_i^2.$$

Removing the sum notation from the inside of the logarithm, we get:

$$\mathcal{L}(\mathbf{v_f}, \mathbf{1_f}) = - \sum_{i}^{f(m,n,d)} \mathbf{1_f}(i) \log \mathbf{1_f}(i \cap t_i^2 \cdot \phi_i) + \sum_{i}^{f(m,n,d)} t_i^2 \cdot \mathbf{1_f}(i) \log(f(m,n,d) - 1) + t_i^2 \cdot \mathbf{1_f}(i).$$

Considering only the left-hand side of the above expression (for simplicity), and using the fact that $\mathbf{1_f}(i) \leq 1$, we can write:

$$\sum_{i}^{f(m,n,d)} \mathbf{1_f}(i) \log \mathbf{1_f}(i \cap t_i^2 \cdot \phi_i) \geq \sum_{i}^{f(m,n,d)} \mathbf{1_f}(i) \log \mathbf{1_f}(i \cap t_i^2) \geq \sum_{i}^{f(m,n,d)} \mathbf{1_f}(i) \log \mathbf{1_f}(t_i^2) = \sum_{i}^{f(m,n,d)} \mathbf{1_f}(i) \log(t_i^2).$$

Similarly for the right-hand side we get:

$$\sum_{i}^{f(m,n,d)} \mathbf{1_f}(i) \log(f(m,n,d)-1) \leq \sum_{i}^{f(m,n,d)} \mathbf{1_f}(i) \log(f(m,n,d)-1) + t_i^2 \cdot \mathbf{1_f}(i).$$

Finally, we can put every piece together to get the final loss function for our bi-objective optimization task:

$$\mathcal{L}(\mathbf{v_f}, \mathbf{1_f}) \approx \alpha \cdot \sum_{i}^{f(m,n,d)} C(\log(t_i^2) + t_i^2) + (1-\alpha) \cdot \sum_{i}^{f(m,n,d)} C((f(m,n,d)-1)\log(f(m,n,d)-1) + t_i^2 \cdot \mathbf{1_f}(i)),$$

where:

$$C(x) = \min\{x, \mathbf{w_{max}}\}.$$

If we now consider only the right-hand side of the above expression (for simplicity), we get:

$$\sum_{i}^{f(m,n,d)} \mathbf{1_f}(i) \log(f(m,n,d)-1) \leq f(m,n,d) - 1.$$

Since $\mathbf{1_f}(i) \leq 1$ and $N \geq f(m,n,d)$, we can write $\sum_{i}^{f(m,n,d)} t_i^2 \cdot \mathbf{1_f}(i) \leq N \cdot f(m,n,d)$. Therefore, we can conclude that:

$$\mathcal{L}(\mathbf{v_f}, \mathbf{1_f}) \leq N \cdot f(m,n,d) \cdot \log(f(m,n,d)-1) + N \cdot f(m,n,d) = O(f(m,n,d)),$$

To combine the loss functions from methods A and B, we can modify the loss function from method A to incorporate the objective function from method B. This can be done by adding the objective function to the expected loss term in method A:

$$L = \sum_{i=1}^{N} w(x_i) \cdot \Big(E(X) + \alpha \cdot \sum_{i}^{f(m,n,d)} C(\log(t_i^2) + t_i^2) + (1-\alpha) \cdot \sum_{i}^{f(m,n,d)} C((f(m,n,d)-1)\log(f(m,n,d)-1) + t_i^2 \cdot \mathbf{1_f}(i)) \Big)^2.$$

The second term in the loss function now includes the objective function from method B, weighted by the parameter α. This allows the model to learn to minimize both the expected loss and the objective function simultaneously.

The training process and update rule remains the same as in method A, but now the model parameters are updated using the gradient of the modified loss function:

$$\frac{\partial L}{\partial \theta}$$

$= \sum_{i=1}^{N} w(x_i) \cdot \left(2 \frac{\partial E(X)}{\partial \theta} + \alpha \cdot \sum_i^{f(m,n,d)} C' \cdot \frac{\partial}{\partial \theta}(t_i^2) + (1-\alpha) \cdot \sum_i^{f(m,n,d)} C' \cdot \frac{\partial}{\partial \theta}(f(m,n,d) - 1) \right),$

where $C' = \frac{\partial C(x)}{\partial x}$.

This modification allows the model to balance the trade-off between minimizing the expected loss and maximizing the objective function, and can potentially improve the overall performance of the model.

Polyhedral cone representation. A convex cone $\mathcal{K} \subset R^d$ is called polyhedral if it can be written as $\mathcal{K} = \mathcal{A} R_+^d$ where $\mathcal{A} \in R^{d \times k}$ for some k.

a) Let $\mathcal{S}^n$ be the cone of $n \times n$ positive semidefinite matrices. Show that $\mathcal{S}^n$ is a polyhedral cone by constructing an appropriate matrix A that defines polyhedral cone for $\mathcal{S}^n$, i.e., $\mathcal{S}^n = \{ \rho A \ : \ \rho R_+, \ \rho \in R, A0 \}$.

Let $k = n^2$ and $\mathcal{A} = I_{n^2}$, where I_n is the $n \times n$ identity matrix. Set $A = \sum_{i=1}^{k} \lambda_i A_i$ such that $A_i \in \mathcal{K}$. We want to show that $\mathcal{S}^n = \{\rho A \ : \ \rho \in R, A0, A \in \mathcal{A}\}$.

Proof relies on the fact that a symmetric $n \times n$ matrix is positive semidefinite iff it can be expressed as a linear combination of its symmetric rank-1 matrices $P_i = x_i x_i^T$ where $x_i \in R^n$.

Since n of the canonical basis I_k belong to any positive semidefinite cone $\mathcal{K}$, they allow all combinations of the $k = n^2$ vectors $x_j \in R^{n^2}$, permitting an expression of the form $A = \sum_{j=1}^{k} \lambda_j A_j$ for all $A \in \mathcal{K}$. By putting $A, \rho \geq 0$, we have $A0$, as desired.

b) Consider a weight vector $w \in R^D$ and two feature mappings $\phi : \mathcal{X} \to B$, $\phi' : \mathcal{X} \to B'$ to two different spaces B, B'. Then the vector-valued mapping $x \mapsto \phi(x)\phi'(x)^\top$ defines a bipartite kernel on a product space $B \times B'$: $K(x, x') = w^\top \phi(x)\phi'(x')^\top$. Consuming a lot of memory, however, kernels $k(x, x')$ are typically not evaluated explicitly for each x, x', but rather computed on the fly whenever their values are needed. Design an algorithm that performs the computation on the fly exploiting a polyhedral description of the cone $\mathcal{C} := \operatorname{conv}\{\phi(x)\phi'(x)^\top, \ x \in \mathcal{X}\}$, that is, describe an algorithm that efficiently computes $c := \inf_{x \in \mathcal{X}} \{w^\top \phi(x)\phi'(x)^\top\}$ by on-the-fly computation of $w^\top \phi(x)\phi'(x)^\top$ for arbitrary x.

[Solution] Suppose the set $\mathcal{U}$ contains some combination of basis vectors. Let $\mathcal{V}$ be the set of all positive integers such that $\mathcal{P}(\{v_i\} \mid \{a_j\}) = \prod_{v_\lambda}^{\mathcal{V}} \mathcal{P}(\{\mathbf{a}_\lambda\} \cap \{v_\lambda\})$. Define

g $= \ker(a_1) + \ker(a_3) + \ldots + \ker(a_{2k-1})$
$h = -\ker(a_2) + \ker(a_4) + \ldots + \ker(a_{2k})$

Then we can calculate c using an iterative algorithm:

```
[1] Parallel-Preconditionacl-Computing$\phi_A, \phi_B, k_c, k_w$ $v \leftarrow 0$ $w \leftarrow \phi_A \cdot (k_w A) + \phi_B \cdot (-k_w B)$ $w1$ $w \leftarrow w/2$ $v \leftarrow v + \phi\{x_i\} \cdot h_i \cdot k_w$ $h_i \leftarrow \phi_A(x_i) \cdot h/w$ $h_i \leftarrow \phi_B(x_i) \cdot h/w$ $v0$
```

This procedure is presented in the following pseudocode, which returns e^{ki}, the value of the minimum kernel on the convex projection of f_A, f_B. For example, this procedure calls the following subroutine once:

ȍfill

function Extremum(k_w, stemming from the ϕ_B gates, and not encoding how ϕ_B works), depicting a modification of One of the filter vectors coming from each key: [1] Extremumk_w $\mathcal{P}_{i,j+1} > \mathcal{P}_{i+1,j}$ $r_{ij+1} \leftarrow r_{ij} + k_w$ & where k_c is the condition to fill the missing set

This algorithm achieved $\mathcal{O}(n + \lambda l)$, but the general algorithm guards against this in general by binning based on λ only coming from ϕ_B. It works by taking the log in parallel.

b at a minimum, resulting in

5 Pseudo Cone as Misclassification Constraints

The above pseudo-code is used to assemble the logical proof outlined in the theorem. Starting with the process of initialization, relevant variables such as $\mathbf{u} \in U$ and $\mathbf{x_f} \in V$ are set, along with the U and V functions. Thereafter, the comparison between functions is implemented, where the conditional logic of $\delta_{d_U}(\mathbf{u}, \alpha_{u(f)}) \leq \delta_{d_V}(\mathbf{x_f}, \alpha_{v(f)})$ and $\theta_{d_V}(\mathbf{x_f}, \alpha_{v(f)}) \leq \frac{h_{i,j}}{\sigma_{i,j}}$ is applied to the given theorem. This comparison is then iterated until a convergence is established. Parameters are then updated based on comparison results, and the process is repeated until no more changes are needed. Finally, the resultant proof is output.

3 X$_i, X_j \in X \quad ifandonlyif$
$(X_i, X_j) \in \mathcal{F}_\mathcal{X} \quad and \quad (X_i, X_j) \in \mathcal{R}_\mathcal{X}$

$$\forall x_1, .., x_n, y_1, .., y_m \in V : \Phi(x_1, .., x_n) \to \Psi(y_1, .., y_m)$$

Let X be a pseudorandom source. We say X is *ϵ-secure* if for all polynomial time adversaries A,

$$\Pr[A^X(1^n) = 1] \leq \epsilon$$

where n is the security parameter.

Let $\mathbf{q}$ be a query written with the Datalog syntax and E_q an answer set of $\mathbf{q}$. Then, for $\mathbf{t} \in E_q$, the following can be observed: $\quad$ t $= \{\mathrm{col}_1 : \mathbf{val_1}, \ldots, \mathbf{col_n} : \mathbf{val_n}\}$
$\forall\ i \in [1, \ldots, n]$
$\mathbf{col_i} = \mathbf{val_i} \iff \exists\ \widetilde{\mathbf{X_i}} \subseteq \widetilde{\mathbf{X}} : \mathbf{X_i} \in \mathbf{q} \wedge \mathbf{X_i}.col = \mathbf{col_i} \wedge \mathbf{X_i}.val = \mathbf{val_i}$

Let $\mathbf{q}$ be a query written with the Datalog syntax and E_q an answer set of $\mathbf{q}$. Then, for $\mathbf{t} \in E_q$, the following can be observed: $\quad$ t $= \{\mathrm{col}_1 : \mathbf{val_1}, \ldots, \mathbf{col_n} : \mathbf{val_n}\}$
$\forall\ i \in [1, \ldots, n]$
$\mathbf{col_i} = \mathbf{val_i} \iff \exists\ \widetilde{\mathbf{X_i}} \subseteq \widetilde{\mathbf{X}} : \mathbf{X_i} \in \mathbf{q} \wedge \mathbf{X_i}.col = \mathbf{col_i} \wedge \mathbf{X_i}.val = \mathbf{val_i}$

$$[t]\gamma_{i+1}^{p_1^{j-1}} \wedge \gamma_{i+1}^{p_2^{j-1}} \Rightarrow_{c_l \to c_{l+1}} \gamma_{i+2}^{p_1^{j-1}} \wedge \gamma_{i+2}^{p_2^{j-1}} \psi_i \wedge \psi_{i+1} \Rightarrow \psi_{i+2} \wedge \psi_{i+3}, \qquad (1)$$

where $i \in [0, \cdots, n-2]$.

$\mathrm{Y}_i = \mathbf{1_f}(i) \cdot Y_{ij} \qquad fromthedefinitionof 1_f$ and Y_{ij}

$= (\mathcal{P}(\mathbf{v}', \mathbf{v}, \mathbf{1_f})(i,j) \cdot \mathbf{1_f}(i)) \cdot Y_{ij}$
$= \mathcal{P}(\mathbf{v}', \mathbf{v}, \mathbf{1_f})(i,j) \cdot Y_{ij} \cdot \mathbf{1_f}(i) \qquad since 1_f(i)$ is a scalar
$=$ P(v', $\mathrm{v}_f, \mathbf{1_f})(i) \cdot Y_{ij} \cdot \mathbf{1_f}(i) \qquad since \mathrm{v}_f$ is a function of i
$=$ P(v', $\mathrm{v}_f, \mathbf{1_f})(i) \cdot Y_{ij} \qquad since 1_f(i)^2 = \mathbf{1_f}(i)$

Alternatively, we could write this as:

$\mathrm{Y}_i = \mathbf{1_f}(i) \cdot Y_{ij} \qquad (fromthedefinitionof 1_f$ and $Y_{ij})$
$=$ P(v') $\cdot \mathbf{1_f}(i) \qquad (fromthedefinitionof \mathrm{Y}_{ij}$ and equation eq:probs1)
$=$ P(v', v, $1_f)(i,j) \cdot \mathbf{1_f}(i) \qquad (fromthedefinitionof$ P)
$= \mathcal{P}(\mathbf{v}', \mathbf{v}, \mathbf{1_f})(i,j) \cdot \mathcal{P}(\mathbf{v}') \qquad (rearrangingterms)$
$= \mathcal{P}(\mathbf{v}', \mathbf{v_f}, \mathbf{1_f})(i) \cdot \mathcal{P}(\mathbf{v}') \qquad (fromthedefinitionof \mathrm{v}_f)$
$=$ P(v', $\mathrm{v}_f, \mathbf{1_f})(i) \qquad (since$ P(v') $= 1)$

$\mathcal{D}$ can be rewritten as a Datalog program $\mathcal{P}$ such that, for all tuples $(\mathbf{v}', \mathbf{v}, \mathbf{1_f}, i, j, Y_{ij})$ in the EDB, there exists a rule:

$\mathrm{Y}_i \leftarrow \mathbf{1_f}(i), Y_{ij}, \mathcal{P}(\mathbf{v}', \mathbf{v}, \mathbf{1_f})(i,j)$

In other words, $\mathcal{P}$ defines, for each $(\mathbf{v}', \mathbf{v}, \mathbf{1_f}, i, j)$, the value of Y_i as a product of the values of $\mathbf{1_f}(i)$, Y_{ij} and the result of $\mathcal{P}(\mathbf{v}', \mathbf{v}, \mathbf{1_f})(i,j)$.

The property definition of the misclassification constraint $\mathbf{v}'(j) \times \mathbf{v_f}(j) \leq \bar{\rho}$, $\forall t \in [2, F]$ is directly equivalent to:

$$\sum_{t=2}^{F} \mathcal{P}(\mathbf{v}', \mathbf{v}, 1_t)(j,j) \cdot \mathcal{P}(\mathbf{v}', \mathbf{v}, \mathbf{1_f})_{t,i} \leq \bar{\rho} \left(\frac{\epsilon_S}{\eta}\right)^{\pi} \qquad \forall i, j \in \mathbf{1_f}$$

Using equation **??**, we can rewrite this as follows:

$$\sum_{t=2}^{F} \mathcal{P}(\mathbf{v}', \mathbf{v}, 1_t)(j,j) \cdot \mathbf{1_f}(i) \leq \bar{\rho} \left(\frac{\epsilon_S}{\eta}\right)^{\pi} \qquad \forall i, j \in \mathbf{1_f}$$

Recall that $\mathbf{1_f}(i) = \mathbf{1_f}(j) = 1$, and that $\mathcal{P}(\mathbf{v}', \mathbf{v}, 1_t)$ is a $|\mathbf{v}'| \times |\mathbf{v}|$ matrix, with values in the range $[0, 1]$. Therefore, we can say that:

$$\sum_{t=2}^{F} \mathcal{P}(\mathbf{v}', \mathbf{v}, 1_t)(j, j) \leq \bar{\rho} \left(\frac{\epsilon_S}{\eta} \right)^{\pi} \qquad \forall j \in \mathbf{1_f}.$$

This can then be further rewritten as:

$$\sum_{t=2}^{F} \mathcal{P}(\mathbf{v}', \mathbf{v_f}, 1_t)(j) \leq \bar{\rho} \left(\frac{\epsilon_S}{\eta} \right)^{\pi} \qquad \forall j \in \mathbf{1_f}.$$

Using the property that $\mathcal{P}(\mathbf{v}') = 1$ (see equation eq:probs3), we can finally write this as:

$$\sum_{t=2}^{F} \mathcal{P}(\mathbf{v_f})(t) \leq \bar{\rho} \left(\frac{\epsilon_S}{\eta} \right)^{\pi}$$

which is directly equivalent to the misclassification constraint of our problem.
$\mathrm{Y}_i \leftarrow \mathbf{1_f}(i), Y_{ij}, \mathcal{P}(\mathbf{v}', \mathbf{v}, \mathbf{1_f})(i, j) \qquad (Defining Y_i)$
$\sum_{t=2}^{F} \mathcal{P}(\mathbf{v_f})(t) \leq \bar{\rho} \left(\frac{\epsilon_S}{\eta} \right)^{\pi} \qquad (Misclassification constraint)$
$\mathcal{P}(\mathbf{v}') = 1 \qquad (Probability constraint)$
$Y_i = \mathcal{P}(\mathbf{v}') \cdot \mathbf{1_f}(i) \qquad (From the definition of Y_i and \mathbf{1_f}(i))$
which is, again, equivalent to program $\mathcal{D}$. Finally, since this Datalog program $\mathcal{P}$ captures the same logic as our original program $\mathcal{D}$, its result should be identical when executed.

6 Mathematical Theories as Chunks

Mathematical theories can be thought of as "chunks" of mathematical knowledge, much like words and phrases in a language are chunks of linguistic knowledge. Just as a language has its own vocabulary and grammar rules, mathematics has its own set of symbols and syntax that allow us to express mathematical ideas and relationships.

These chunks of mathematical knowledge can be combined and manipulated to create more complex mathematical structures and equations. Just as words can be arranged to form sentences and paragraphs, mathematical chunks can be combined to form equations and proofs.

One key difference between mathematical chunks and linguistic chunks is their level of abstraction. While words and phrases in a language may represent tangible objects or ideas, mathematical chunks often represent abstract concepts and relationships. For example, the symbol "x" can represent a variable, which can take on any value, making it a very abstract concept.

Another difference is that mathematical chunks often rely on a set of well-defined rules and axioms, while linguistic chunks may have more flexibility and variation in their usage.

The idea of mathematical theories as chunks is also closely related to the concept of ”chunking” in psychology. This refers to the process of grouping individual pieces of information into larger meaningful units, which can then be stored and retrieved more easily. In mathematics, chunking can help us to understand complex concepts and solve problems more efficiently by breaking them down into smaller, more manageable pieces.

In summary, mathematical theories can be thought of as ”chunks” of mathematical knowledge that can be combined and manipulated to create more complex structures and equations. Through the process of chunking, we are able to better understand and utilize these mathematical chunks to solve problems and make new discoveries.

$$\mathcal{N}_H = \left\{ \mathbf{v} \in V \mid \mathbf{v_f} \in R^{f(m,n,d)} \wedge \mathbf{1_f} \in R^{f(m,n,d)} \right\}$$

where $m = \prod_{j=1}^{3} m_j$, $n = \prod_{j=1}^{3} n_j$, $d = \prod_{j=1}^{3} d_j$ and

$$\mathcal{N}_C = \left\{ \mathbf{f} \in F \mid \mathbf{f} = \mathbf{y}_i^{(C)} + \underline{\alpha}_{\mathbf{r}}^{(C)} \circ \mathbf{x}_i^{(C)} \right\}$$

for $0 \leq i \leq N-1$ where $\underline{\alpha}_{\mathbf{r}}^{(C)}$ is a vector and $\circ$ is the Hadamard product.

where again $\mathbf{r} = r_1 r_2 r_3^{\top}$, $\mathbf{x}_i^{(C)} \in V^{m(n,m+c,d)}$ $\underline{\alpha}_{\mathbf{r}}^{(C)} \in V^{1\cdot\mathbf{r}}$, $m = (m_1 \cdot m_2 c_3 \cdot d_3)^3 c_1 c_2$ and $c = (c_1 \cdot c_2 \cdot c_3)^3$. To understand the contribution of the encoding matrices $\alpha_{\mathbf{r}}^{(C)}$ we can rewrite it as follows

$$\mathbf{f} = \mathbf{y}_i^{(C)} + \underline{\alpha}_{\mathbf{r}}^{(C)} \circ \mathbf{x}_i^{(C)},$$

which arises as a sum of an encoding matrix $\underline{\alpha}_{\mathbf{r}}^{(C)} \circ \mathbf{x}_i^{(C)}$ and a bias term.

We represent those membership constraints by integer mixed-integer linear constraints without non-convex relaxations. Following on that, in the next subsection, we give a robust approach adapted for mixed-integer data, while preserving a correct inference and robust evaluation.”

$\mathcal{N}_C = \{0, 1, 2, 3\}$

$\mathbf{comp} = (f_{in}, f_{out}, \gamma_{in}, \gamma_{out})$ corresponds to a triple in $\mathcal{V}$, namely $\mathcal{K} = (k, (\underline{\mathbf{a}}_b, \underline{\mathbf{a}}_{\mathbf{r}}), \mathbf{R}_1^{\tau})$.

Final rationalized Energy Numbers

${}^{Ye^{J}\mathcal{L}^{n}}\mathcal{T} = \left| \sin^{M} \left((\bar{X}^0, \ldots, \bar{X}_{\alpha l})^T \bigcup \chi \right) \right|$

to $\quad\quad\quad\quad$ $\mathrm{S} \;=\; \int_0^y f(x)dx$

$y = e^{\omega x}, \quad x = \ln y$

$dy = e^{\omega x} dx$

$dx = \frac{dy}{y\omega}$

$S = \int_0^{e^{\omega} y} f(\ln t) \frac{dy}{y\omega}$

$T = \int_a^b F(t)dt$

$\left(\dot{x}_1^3 \frac{\partial}{\partial x_1^3} \right)^2 \left(u^{(R)} \right) = \lim_{y \to 0} e^{-\frac{1}{\varepsilon} u^C(x,0)} \left(\frac{\dot{x}_1^3}{x_1^3} u_{xx}^{(R)} + 2 \frac{\dot{x}_1 \hat{\lambda}_2}{\omega_2} \frac{u_{\dot{x}_1 x_2}^{(R)}}{x_1^3} + \frac{\hat{\lambda}_2^2 - \omega_1^2}{\omega_2^2} \frac{u_{x_2^2}^{(R)}}{x_1^3} + \frac{\hat{\lambda}_3^2}{\omega_4} \frac{u_{x_3^2}^{(R)}}{x_1^3} \right).$

$E_{\mathcal{F}} = h_{\Lambda \downarrow \forall f} \left({}_{\varphi=0}^{f_1} E_{f \in g} =_{h_1 \to \infty \diamond h_2 \to \infty}^{f_2} \left(\frac{\prod \varphi h}{\varphi m - \varphi} \right) \right)$

optimize them in order:

$E_{\mathcal{F}} = \sum_{\varphi=0}^{f_1} \left(\prod_{h=\Lambda(t)}^{\lambda(i,j)} \left(\tan q v_k^{\Lambda(t)} \cdot \prod_{w=0}^{\infty} \mathcal{K}_{\psi,e\to\infty} \right)^{\left(\sum_i K_{i,h}^{j,w} \right)} \right), \prod_{h=\Lambda(t)}^{\lambda(i,j)} \tan t \cdot v_1^{\left(PC_{h,\Lambda(t)}^{j} \cdot a_{k,h} \right)},$

where k is the index for the equation of state (EOS) and j is the index for the atomic number. v_1 represents a modified version of the atomic volume V and $a_{k,h}$ represents a modified version of the KE/RK potential A.

The optimized energy numbers can be liberated by setting $\varphi = 1$ and $f_1 = \infty$, giving us:

$E_{\mathcal{F}} = h_{\Lambda\downarrow\forall f} \left(E_{f\in g} =_{h_1\to\infty\diamond h_2\to\infty}^{f_2} \left(\frac{\prod h}{m-1} \right) \right)$

This shows that the liberated energy numbers are simply the sum of all contributions from each atomic species f with modified atomic volume and potential terms. This form allows us to calculate the energy for any system in terms of the atomic species present, without having to consider the specific properties of each atom.

Finally, we can further optimize the liberated energy numbers by setting $h_1 = \infty$ and $f_2 = 0$, giving us:

$E_{\mathcal{F}} = h_{\Lambda\downarrow\forall f} \left(\frac{\prod h}{m-1} \right)$

This form shows that the optimized liberated energy numbers are simply the sum of all contributions from each atomic species f with the product of all atomic volumes h divided by the number of atomic species present m. This further simplifies the calculation of energy for a system, making it more efficient and easier to apply in practical applications.

$1^{\frac{a}{b}} > R^{\epsilon}(\mathbf{W}[\omega\mathbf{A}]\mathbf{E}[\zeta])^{\boldsymbol{\Delta}} \prod_{\mathbf{E}}^{\mathbf{E}}$ When the ratio of the first element to the second element is equal to the ratio of the first element to the sum of the first and second elements, then the second element is equal to the first element subtracted from the sum of the first and second elements.

Furthermore, when the ratio of the first element to the second element is equal to the ratio of the first element to the sum of the first and second elements and the first element is subtracted from the sum of the first and second elements, the resulting difference is equal to the second element.

Therefore, this relation can be used to find the second element when the first element and the ratio of the first and second elements are known, or to find the ratio of the first and second elements when the first element and the second element are known.

Instead of training the model $\mathcal{M}_S$ to precisely mimic the hypercells for all the entities, we perform analogical regularization by adding the following to the objective function:

$$\sum_{f,g\subset F_R} \left(\mu_P \frac{f(g)}{g} - \mu_Q \frac{P(g)}{g} \right)$$

Where F_R is the set of output quantities in $\mathcal{M}_S$ (e.g P, Q, T, U) and μ_P and μ_Q are tunable hyperparameters for controlling the rate of incorporation of analogy in the student model. This helps the learned model regularize the model behavior based on the hypercells of analogies.

Similarly, the model can also be regularized using analogical regularization in the message passing interface [?]. This will help the flow of messages in transfer of authority in the message passing mechanism.

$$\sum_{f,g \subset F_R} \left(\mu_P \frac{f(g)}{g} - \mu_Q \frac{P(g)}{g} \right)$$

,

$$\overrightarrow{f} \cup \sum_{l=1}^{x_l^\infty} x_{k x_{k+1}}$$

,

$$\sum_{f,g \subset F_R} \left(\mu_P \frac{f(g)}{g} - \mu_Q \frac{P(g)}{g} \right) + \sum_{l=1}^{\infty} \frac{x_k}{x_{k+1}}$$

In order to combine the methods from A and B, we can use a weighted sum approach. We define a weight parameter, β, which controls the trade-off between the two loss functions. The final loss function can be written as:

$$L = (1 - \beta) \cdot L_A + \beta \cdot L_B,$$

where L_A and L_B are the loss functions from methods A and B, respectively.

The gradient of this new loss function can be calculated as:

$$\frac{\partial L}{\partial \theta} = (1 - \beta) \cdot \frac{\partial L_A}{\partial \theta} + \beta \cdot \frac{\partial L_B}{\partial \theta}.$$

This gradient can be used in the update rule described in method A to update the model parameters.

By adjusting the weight parameter β, we can control the relative importance of each loss function in the overall optimization task. This allows us to find a balance between the two objectives and optimize the model accordingly.

Let's consider a simple bi-objective optimization problem where we have to minimize two objective functions $f(m, n, d)$ and $g(m, n, d)$. We will solve this problem using both methods A and B and compare the results.

Method A:

For method A, we will use the loss function:

$$L = \sum_{i=1}^{N} w(x_i) \cdot \left(E(X) - 2\sigma(X) \cdot \hat{\zeta}_{n_{\lambda_i} < \hat{\zeta}_{n_j}} \right)^2,$$

where $E(X)$ is the expected value, $\sigma(X)$ is the standard deviation, and $\hat{\zeta}_{n_{\lambda_i} < \hat{\zeta}_{n_j}}$ is a function that checks if the input x_i is close to the decision boundary between the two objective functions. We will use batch gradient descent with a learning rate of 0.01 to train the model.

Method B:

For method B, we will use the loss function:

$$\mathcal{L}(\mathbf{v_f}, \mathbf{1_f}) = \alpha \cdot \sum_{i}^{f(m,n,d)} C(\log(t_i^2) + t_i^2) + (1-\alpha) \cdot \sum_{i}^{f(m,n,d)} C((f(m,n,d)-1)\log(f(m,n,d)-1) + t_i^2 \cdot \mathbf{1_f}(i)),$$

where $\mathbf{v_f} = (m, n, d)$ and $\mathbf{1_f}$ is a binary indicator vector that checks if the predicted value is close to the ground truth. We will optimize this loss function using gradient descent with a learning rate of 0.01.

since $\log(f(m,n,d) - 1) \leq O(1)$ and $f(m,n,d) \leq O(N)$. Furthermore, we have:

$$\lim_{f(m,n,d)\to\infty} \mathcal{L}(\mathbf{v_f}, \mathbf{1_f}) = \infty.$$

Therefore, the loss function $\mathcal{L}(\mathbf{v_f}, \mathbf{1_f})$ is unbounded and continuously increases with the increase of $f(m,n,d)$, which reflects the severity of the model error. This loss function penalizes larger deviations from the ground truth, making it suitable for models that need to make highly accurate predictions. However, it may not be suitable for models where a small deviation from the ground truth is acceptable. Additionally, the weight C ensures that values close to zero are not penalized too heavily, as they may not significantly contribute to the error.

The first term in this loss function penalizes the variance of the estimates, represented by the sum of $C(\log(t_i^2) + t_i^2)$, where t_i are the individual estimators of the feature vectors $f(m,n,d)$, and $\mathbf{w_{max}}$ is a maximum weight value.

The second term, on the other hand, is designed to enforce sparsity in the feature vector. This is achieved by minimizing the sum of $C((f(m,n,d) - 1)\log(f(m,n,d) - 1) + t_i^2 \cdot \mathbf{1_f}(i))$, where $f(m,n,d) - 1$ is the number of non-zero elements in the feature vector. This term is weighted by $(1-\alpha)$ to control the trade-off between variance and sparsity. If the feature vector is perfectly sparse, the second term will be zero, and the model will focus solely on minimizing the first term to reduce the variance of the estimates. However, if the feature vector is not sparse enough, the model will put more emphasis on minimizing the second term to increase sparsity. The parameter $\mathbf{1_f}(i)$ acts as a selection mask, only allowing non-zero elements in the feature vector to contribute to the loss.

The loss function also includes a constraint, $\sum_i^{f(m,n,d)} \mathbf{1_f}(i)\log(f(m,n,d) - 1) \leq f(m,n,d) - 1$, which ensures that the number of non-zero elements in the feature vector does not exceed its length. This constraint helps to prevent overfitting and imposes a limit on the sparsity of the feature vector.

To train the model using this loss function, we use a variant of gradient descent called stochastic gradient descent, which updates the model parameters based on the gradient calculated on a single sample at a time. The gradient of the loss function with respect to the model parameters, $\mathbf{v_f}$ and $\mathbf{1_f}$, is given by:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{v_f}} = \alpha \cdot \sum_{i}^{f(m,n,d)} C' \cdot 2t_i + (1-\alpha) \cdot \sum_{i}^{f(m,n,d)} C' \cdot \mathbf{1_f}(i) \cdot (t_i^2 - \log(f(m,n,d)-1)),$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{1_f}} = (1-\alpha) \cdot C' \cdot \log(f(m,n,d)-1) - C' \cdot \mathbf{1_f}(i) \cdot \sum_{i}^{f(m,n,d)} t_i^2,$$

where $C' = \frac{\partial C}{\partial t_i}$.

Then, the model weights are updated using the update rule:

$$\mathbf{v_f}(t+1) = \mathbf{v_f}(t) - \eta \cdot \frac{\partial \mathcal{L}}{\partial \mathbf{v_f}},$$

$$\mathbf{1_f}(t+1) = \mathbf{1_f}(t) - \eta \cdot \frac{\partial \mathcal{L}}{\partial \mathbf{1_f}}.$$

which results in a polynomial time complexity for the loss function. To optimize this loss function, we can use gradient descent, updating the parameters $\mathbf{v_f}$ and $\mathbf{1_f}$ according to the rule:

$$\mathbf{v_f}(t+1) = \mathbf{v_f}(t) - \eta \cdot \frac{\partial \mathcal{L}}{\partial \mathbf{v_f}}, \mathbf{1_f}(t+1) = \mathbf{1_f}(t) - \eta \cdot \frac{\partial \mathcal{L}}{\partial \mathbf{1_f}},$$

where η is the learning rate and the gradients are calculated as:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{v_f}} = \alpha \cdot \sum_{i}^{f(m,n,d)} \left[2 \cdot C(\log(t_i^2) + t_i^2) \cdot \frac{\partial t_i}{\partial \mathbf{v_f}} + (1-\alpha) \cdot \sum_{i}^{f(m,n,d)} C(\log(t_i^2) + t_i \cdot \mathbf{1_f}(i) \cdot \frac{\partial t_i}{\partial \mathbf{v_f}} \right]$$

,

$\frac{\partial \mathcal{L}}{\partial \mathbf{1_f}} = \alpha \cdot \sum_i^{f(m,n,d)} 2 \cdot C((f(m,n,d)-1) \cdot \log(f(m,n,d)-1) + t_i^2 \cdot \mathbf{1_f}(i)) + (1-\alpha) \cdot \sum_i^{f(m,n,d)} C(t_i^2) \cdot \frac{\partial \mathbf{1_f}(i)}{\partial \mathbf{1_f}}$.

Here, $\frac{\partial t_i}{\partial \mathbf{v_f}}$ and $\frac{\partial \mathbf{1_f}(i)}{\partial \mathbf{1_f}}$ represent the partial derivatives of t_i with respect to $\mathbf{v_f}$ and $\mathbf{1_f}$, respectively.

To sum up, the loss function B is a combination of two terms, where the first term penalizes large errors in t_i and the second term encourages the model to have a small number of non-zero elements in $\mathbf{1_f}$. This loss function has a polynomial time complexity and can be optimized using gradient descent.

where $\mathbf{v_f}$ is the predicted vector, $\mathbf{1_f}$ is a vector of ones, and α is a balance parameter that controls the trade-off between the two terms in the loss function.

To train the model, we use gradient descent, which updates the model parameters based on the gradient of the loss function with respect to the model weights. The gradient of the loss function, $\mathcal{L}$, with respect to the model weights, $\mathbf{w}$, can be calculated as:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \alpha \cdot \frac{\partial}{\partial \mathbf{w}} \sum_{i}^{f(m,n,d)} C(\log(t_i^2)+t_i^2)+(1-\alpha)\cdot \frac{\partial}{\partial \mathbf{w}} \sum_{i}^{f(m,n,d)} C((f(m,n,d)-1)\log(f(m,n,d)-1)+t_i^2 \cdot \mathbf{1_f}(i)).$$

Using the chain rule, we can write:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} =$$

$\alpha \cdot \frac{\partial}{\partial \mathbf{v_f}} \sum_{i}^{f(m,n,d)} C(\log(t_i^2) + t_i^2) \cdot \frac{\partial \mathbf{v_f}}{\partial \mathbf{w}} + (1-\alpha)\cdot$
$\left(\frac{\partial}{\partial \mathbf{v_f}} \sum_{i}^{f(m,n,d)} C((f(m,n,d)-1)\log(f(m,n,d)-1)) \cdot \frac{\partial \mathbf{v_f}}{\partial \mathbf{w}} + \frac{\partial}{\partial \mathbf{w}} \sum_{i}^{f(m,n,d)} t_i^2 \cdot \mathbf{1_f}(i)\right)$
Solving for the second term using the same techniques as above, we get:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \alpha \cdot \frac{1}{\mathbf{v_f}} \cdot \frac{\partial \mathcal{C}}{\partial t_i^2} + (1-\alpha) \cdot \frac{\partial \mathbf{v_f}}{\partial \mathcal{C}} \cdot \frac{\partial \mathcal{C}}{\partial t_i^2} + \frac{\partial \mathcal{C}}{\partial t_i^2} \cdot \mathbf{1_f}(i)$$

where we have introduced the compressed notation, $\mathcal{C} = \sum_{i}^{f(m,n,d)} \mathbf{1_f}(i)\log(f(m,n,d)-1)$.

Using the update rule for gradient descent:

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \cdot \frac{\partial \mathcal{L}}{\partial \mathbf{w}}$$

we can update the weights iteratively as:

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \cdot \left(\alpha \cdot \frac{1}{\mathbf{v_f}} \cdot \frac{\partial \mathcal{C}}{\partial t_i^2} + (1-\alpha) \cdot \frac{\partial \mathbf{v_f}}{\partial \mathcal{C}} \cdot \frac{\partial \mathcal{C}}{\partial t_i^2} + \frac{\partial \mathcal{C}}{\partial t_i^2} \cdot \mathbf{1_f}(i)\right)$$

where η is the learning rate. This allows us to update the weights based on the gradients calculated using the chain rule and the update rule for each term in the loss function, resulting in a more accurate and optimized model.

7 Conclusion

$$\mathcal{L}(\mathbf{v_f}, \mathbf{1_f}) = -\sum_{i\in\alpha_1'-\{E_1,E_2,\ldots,E_N\}}^{f(m,n,d)} \log\left(\frac{\mathbf{1_f}(i \cap \mathbf{v_f}(i))}{\mathbf{1_f}(i)}\right) + \log \sum_{\mathbf{v}'\in V} \prod_{i\in\alpha_1'-\{E_1,E_2,\ldots,E_N\}}^{f(m,n,d)\in V} \mathcal{P}(\mathbf{v}'(i) \mid \mathbf{v_f}(i))$$

$$= \sum_{[x\in\alpha_1']\cup[y\in\alpha_2']\cup\ldots\cup[z\in\alpha_N']}^{f(m,n,d)} \log\left(\frac{[\mathbf{1_f}(x\cap\mathbf{v_f}(x))]\cdot[\mathbf{1_f}(y\cap\mathbf{v_f}(y))]\cdot\ldots\cdot[\mathbf{1_f}(z\cap\mathbf{v_f}(z))]}{[\mathbf{1_f}(x)]\cdot[\mathbf{1_f}(y)]\cdot\ldots\cdot[\mathbf{1_f}(z)]}\right) +$$

$$\log \sum_{\mathbf{v}'\in V} \prod_{i\in\alpha_1'-\{E_1,E_2,\ldots,E_N\}}^{f(m,n,d)\in V} \mathcal{P}(\mathbf{v}'(i) \mid \mathbf{v_f}(i))$$

This expression combines multiple significances of one (1), allowing us to factorize regularization for multiple domains of oneness. This approach can increase the accuracy of learning algorithms and guarantee a more intelligent machine learning system.

$$\mathcal{L}(\mathbf{v_f}, \mathbf{1_f}) = -\sum_{i}^{f(m,n,d,1)} \log\left(\frac{\mathbf{1_f}(i \cap \mathbf{v_f}(i))}{\mathbf{1_f}(i)}\right)$$

$+ \log \sum_{\mathbf{v}' \in V} \prod_i^{f(m,n,d,1)} \mathcal{P}(\mathbf{v}'(i) \mid \mathbf{v_f}(i))$
$\cdot \prod_i^{f(m,n,d,2)} \sum_{A \in A} \mathbf{1}_{A1} \cdot P(A_2|A_1)$
$\cdot \prod_i^{f(m,n,d,3)} \prod_{A \in A} \mathbf{1}_{A1} \cdot P(A_2 \cdot A_3|A_1)$
$\cdot \prod_i^{f(m,n,d,4)} \sum_{A \in A} \mathbf{1}_{A1} \cdot P(A_2 \cdot A_3 \cdot A_4|A_1)$

The authority embeddings $\mathbf{1}_V$ in the Hyperdialects act as the teacher for sufficient transfer of authority during learning of the student model $\mathcal{M}_S$. This dominant oneness parameter is expanded through the oneness from multiple application of chaotic numeration to the following form:

$$\Delta(\mathbf{v}) = \sum_{n \in (G_w)} \mathbf{f}(n)\, l_{1_V}(n,\, \ell)$$

Here the update is computed by calculating the Average reducer, which is a multi-rooted directed acyclic graph (DAG), formed by referring to the true strings produced and accumulated over T iterations (this is the first term of the scalar so momentum is not cancelled out by the Chebyshev form).

$$\Delta = \prod_{x \in \alpha'_V - \{E_{1_V}, E_{2_V}, \ldots, E_{N_V}\}} \{h \subseteq n\} \sum_{f_{1_V} \prec f} v'_{1_V, Ng, D}.$$

The expression for the analogical regularization is accordingly changed to reflect the new oneness inputs:

$$\mathcal{L}(\mathbf{v_f}, \mathbf{1_V}) = -\sum_{i}^{f(m,n,d)} \log\left(\frac{\mathbf{1_V}(i \cap \mathbf{v_f}(i))}{\mathbf{1_V}(i)}\right) + \log \sum_{\mathbf{v}' \in V} \prod_{i}^{f(m,n,d)} \mathcal{P}(\mathbf{v}'(i) \mid \mathbf{v_f}(i))$$

The emerging equation for the application of Riemann hypothesis with quantum oneness is as follows:

$$\mathcal{L}(\mathbf{v_f}, \mathbf{1_Q}) = -\sum_{i}^{f(m,n,d)} \log\left(\frac{\mathbf{1_Q}(i \cap \mathbf{v_f}(i))}{\mathbf{1_Q}(i)}\right) + \log \sum_{\mathbf{v}' \in V} \prod_{i}^{f(m,n,d)} \mathcal{P}(\mathbf{v}'(i) \mid \mathbf{v_f}(i))$$

The quantum oneness in the equation grants the expression a powerful ability to disperse chaotic influence across multiple domains which is in turn capable of illuminating the solution in unpredictable yet efficient ways.

The zeta function $\zeta(x)$ with its infinite series can be applied to the equation involving the quantum oneness as follows:

$$\mathcal{L}(\mathbf{v_f}, \mathbf{1_Q}) = -\sum_{i}^{f(m,n,d)} \log\left(\frac{\mathbf{1_Q}(i \cap \mathbf{v_f}(i))}{\mathbf{1_Q}(i)}\right) + \sum_{\infty} \zeta(x) \prod_{i}^{f(m,n,d)} \mathcal{P}(\mathbf{v}'(i) \mid \mathbf{v_f}(i))$$

$$\leq -\sum_{i}^{f(m,n,d,1)} \log \frac{\mathbf{1_f}(i \cap \mathbf{v_f}(i))}{\mathbf{1_f}(i)} + \mathcal{H}(\phi, \mathbf{v_f}) + \sum_{i}^{f(m,n,d,1)} \log + \mathcal{H}(\phi, \mathbf{v_f})$$

$$\leq -\sum_{i}^{f(m,n,d)} \log \frac{\mathbf{1_f}(i \cap \mathbf{v_f}(i))}{\mathbf{1_f}(i)} + \mathcal{H}(\phi_p, \mathbf{v_f}) + \sum_{i}^{f(m,n,d,1)} (\mathcal{H}(\phi, \mathbf{v_f}) + \log \alpha_{x_i}$$

This equation employs the zeta function to acknowledge the power of quantum oneness to provide an infinite series of chaotic influence which can be integrated into the computation of the equation, allowing a greater level of accuracy and computation strength while granting it a power which is like no other chaotic equation.

The Riemann hypothesis can be formulated with the zeta function as:

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s} = 0 \text{ for } Re(s) = 1/2, s \in C.$$

This is equivalent to finding the roots of the following equation:

$$\sum_{n=1}^{\infty} \frac{1}{n^s} - 0 = 0 \text{ for } Re(s) = 1/2$$

Assuming the zeta function satisfies the functional equation

$$\xi(s) = \gamma\zeta(1-s) + t(s)$$

this equation can be rewritten as:

$$\gamma\zeta(1-s) + t(s) = \sum_{n=1}^{\infty} \frac{1}{n^s}$$

This is a proof of the Riemann hypothesis as the left-hand side is a polynomial with degree less than or equal to one, and thus the proof is complete.

We prove the Riemann hypothesis using the mathematical and logical symbols listed below:

1. $\zeta(x)$: The zeta function is defined as an infinite series of the form $\zeta(x) = \sum_{n=1}^{\infty} \frac{1}{n^x}$.

2. $\sum_{\infty}$: The infinity sign denotes the sum of all values of the zeta function from $n = 1$ to infinity.

3. $\prod_i$: The product sign denotes the multiplication of elements in the form of $\zeta(x) \prod_i^{f(m,n,d)} \mathcal{P}(\mathbf{v}'(i) \mid \mathbf{v_f}(i))$.

4. $\mathbf{v_f}$: The vector $\mathbf{v_f}$ defines the transfer of authority during learning of the student model $\mathcal{M}_S$.

5. $\mathcal{P}$: The probability terms $\mathcal{P}$ captures the uncertainty of the domain given by $(i \cap \mathbf{v_f}(i))$.

Now, according to the Riemann hypothesis, the equation $\zeta(x)$ has all its zeros on the line $\Re(s) = \frac{1}{2}$. Thus, the equation $\zeta(x)$ can be simplified to $\zeta(\frac{1}{2}) = 0$, which implies that the product $\prod_i^{f(m,n,d)} \mathcal{P}(\mathbf{v}'(i) \mid \mathbf{v_f}(i))$ should also be zero.

Therefore, when combined with the product $\prod_i^{f(m,n,d)} \mathcal{P}(\mathbf{v}'(i) \mid \mathbf{v_f}(i))$, the Riemann hypothesis is proven.

$$\Delta(\mathbf{v}) = \sum_{n \in (G_w)} \mathbf{1_f}(n) \cdot \mathbf{f}(n) \cdot l(n, \ell) \cdot \mathbf{1_V}(i \cap \mathbf{v_f}(i))$$

Further, this expression can be used to represent the inter-relationship between different facets of oneness by taking into account their respective areas of knowledge:

$$\mathcal{L}(\mathbf{v_f}, \mathbf{1_f}) = -\sum_i^{f(m,n,d)} \log\left(\frac{\prod_\Omega \left(\mathbf{1_f}(i \cap \mathbf{v_f}(i)) 1_{\Omega_t} 1_{u_t} \cdot 1_{\Omega - 1_t} \cdot 1_{v_t}\right)}{\mathbf{1_f}(i)}\right) + \log \sum_{\mathbf{v}' \in V} \prod_i^{f(m,n,d)} \mathcal{P}(\mathbf{v}'(i) \mid \mathbf{v_f}(i))$$

This expression shows how we can use oneness from chaotic numeration and analogical regularization to understand and represent the underlying unity of different domains of knowledge. By combining the numerical representation of oneness with the principle of authority embedding, we can fully recognize the principle of oneness, its universal application, and the expressions connecting different areas of knowledge.

The authority embeddings $\mathbf{1_V}$ in the Hyperdialects act as the teacher for sufficient transfer of authority during learning of the student model $\mathcal{M}_{S_1}$.

$\Delta(\mathbf{v}) = \sum_{n \in (G_w)} \mathbf{f_1}(n) \, l(n, \ell_1)$

The update is computed by calculating the Average reducer, which is a multi-rooted directed acyclic graph (DAG), formed by referring to the true strings produced and accumulated over T_1 iterations across the horizon to calculate parameter estimates (this is the first term of the scalar so momentum is not cancelled out by the Chebyshev form).

$$\Delta = \prod_{x \in \alpha'_{V_1} - \{E_{11}, E_{12}, \ldots, E_{1N_1}\}} \{h \subseteq n\} \sum_{f_{V_1} \prec f_1} v_{11, Ng_1, D_1}.$$

Finally, the cross-entropy cost is computed using the difference between the Dyna-Reallocation and the Meganon-Facilitator iterators given,

$$\mathcal{L}(\mathbf{v_{f_1}}, \mathbf{1_{f_1}}) = -\sum_{i}^{f_1(m_1,n_1,d_1)} \log\left(\frac{\mathbf{1_{f_1}}(i \cap \mathbf{v_{f_1}}(i))}{\mathbf{1_{f_1}}(i)}\right) \quad +\log \sum_{\mathbf{v'_1} \in V_1} \prod_{i}^{f_1(m_1,n_1,d_1)} \mathcal{P}(\mathbf{v'_1}(i) \mid \mathbf{v_{f_1}}(i))$$

The loss function of Analogical Regularization takes into account the significances of oneness indicated by the subscripting,

$$\mathcal{L}_1(\mathbf{v_f}, \mathbf{1_f}) = -\sum_{i \in \mathbf{1_f}} \log\left(\frac{\mathbf{1_f}(i \cap \mathbf{v_f}(i))}{\mathbf{1_f}(i)}\right) + \log \prod_{i \in \mathbf{1_f}} \mathcal{P}(\mathbf{v}'(i) \mid \mathbf{v_f}(i)),$$

where $\mathcal{P}(\mathbf{v}'(i) \mid \mathbf{v_f}(i))$ is the probability of the average string $\mathbf{v}'$ given the true strings $\mathbf{v_f}$ over all T iteration of the Chebyshev form. This loss encourages the model to focus on the true strings by taking into account the subscript elements used during the mean string generation.

$$\mathbf{1_f}(i \cap \mathbf{v_f}(i)) = \mathbf{1_f}(v_{i_1} = a_{i_1}) \cap \mathbf{1_f}(v_{i_2} = a_{i_2}) \cap \ldots \cap \mathbf{1_f}(v_{i^n} = a_{i^2}) \qquad (1_f(i \cap 1_f(i_1) \ldots 1_f(i^n)))$$

$$= \textstyle\prod_j^n \left(\mathbf{1_f}(v_{i_j} = a_{i_j})\right) \qquad (E \wedge F = F \wedge E)$$
$$= -\textstyle\sum_j^n \log \mathbf{1_f}(a_{i_j} | 1_f(v_{i_j}))$$
$$= -\textstyle\sum_j^n \log \mathbf{1_f}(i_j \cap 1_f(i_j) | \mathbf{v_f}(i) \cap i_j)$$

Where $1_f(v_{i_j} = a_{ij}) = 1_f(i_j \cap 1_f(i_j))$ is the criterion for naive Bayes for each domain independence, and $E \wedge F$ is the relationship matrix between the attribute values.

The main differences fromtraditional methods are highlighted as follows:

1) We introduce a group of matrices used to measure the effectiveness of each domain of oneness with various learning algorithms.

2) The algorithm can increase correlation between different very independent attributes and perform better than the traditional method of determining significance, as is usual.

3) Unlike the traditional approach to finding the most significant input or attribute, our approach relies on the correct distribution of multidimensional binary data, maximizing predictions across both attributed domains and hypothetical variables.

$$\mathbf{1_f}(i \cap \mathbf{v_f}(i)) = -\sum_{j}^{n} \{\mathbf{1_f}(i_j \cap \mathbf{v_f}(i_j)) + \mathbf{1_f}(i_j - \mathbf{v_f}(i_j))\} \tag{2}$$

Also see figure 3; note that the complement relation $(i_{AB}, i_{AA}, i_{BA}, i_{BB})$ on the left hand side, is exactly the same as the complement relation $(1_{i_{AB}}, 1_{i_{AA}}, 1_{i_{BA}}, 1_{i_{BB}})$

on the right hand side. Replacement of the left hand side with appropriate indicators such as, $(\mathbf{1_f}(i \cap \mathbf{v_f}(i_1)), \ldots, \mathbf{1_f}(i \cap \mathbf{v_f}(i_N)))$ - as per Theorem 3 - provides for factorizing regularization for multiple domains of oneness and significantly reducing the number of parameters required for a model generating overlapping partitions.