Knowledge Bases and Neural Network Synthesis

Todd Davies
Artificial Intelligence Center
SRI International
333 Ravenswood Avenue
Menlo Park, California 94025 USA
DAVIES@AI.SRI.COM

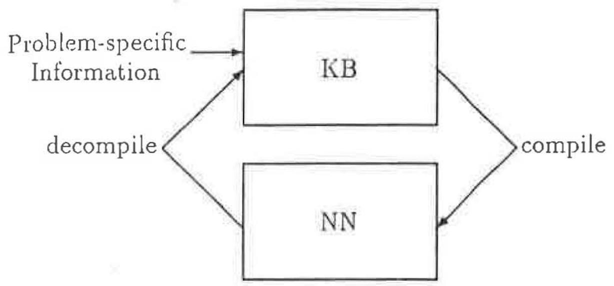

Figure 1: Iterative Compiling and Decompiling.

Abstract

We describe and try to motivate our project to build systems using both a knowledge based and a neural network approach. These two approaches are used at different stages in the solution of a problem, instead of using knowledge bases exclusively on some problems, and neural nets exclusively on others. The knowledge base (KB) is defined first in a declarative, symbolic language that is easy to use. It is then compiled into an efficient neural network (NN) representation, run, and the results from run time and (eventually) from learning are decompiled to a symbolic description of the knowledge contained in the network. After inspecting this recovered knowledge, a designer would be able to modify the KB and go through the whole cycle of compiling, running, and decompiling again, as illustrated in Figure 1.

The central question with which this project is concerned is, therefore,

How do we go from a KB to an NN, and back again?

We are investigating this question by building tools consisting of a repertoire of language/translation/network types, and trying them on problems in a variety of domains.

1 Features of Neural Networks and Knowledge-Based Systems

Attempts to build intelligent machines have historically divided into two broad types. One has emphasized the use of recursive programming languages like Lisp and Prolog, symbolic data structures, and the declarative representation of knowledge (Webber and Nilsson, 1981). The other (anthologized in Anderson and Rosenfeld, 1988) has focused on bounded state machines, mathematical models of neural circuits, and statistical learning algorithms. These two schools of thought are sometimes called "symbolism" and "connectionism", respectively, or the "knowledge-based" (KB) approach and the "neural networks" (NN) approach.

Some problems appear to be more appropriate applications of one approach or the other. For instance, very high level reasoning of the kind often performed by experts with lots of book knowledge (e.g., mathematicians, medical doctors) seems at present more suited to a KB approach (e.g., Buchanan and Shortliffe, 1984; Bundy, 1983), while low-level vision and signal-processing tasks appear better for the NN approach (e.g., Lapedes and Farber, 1987; Mead and Mahowald, 1988). But there is a large array of problems in between, such as the more knowledge-intensive perceptual tasks, natural language tasks, and common sense reasoning. We refer to problems in this general class as "knowledge-intensive problems" or "KIPs." KIPs have been treated as a battleground, with both KB and NN researchers claiming that their approach is the best for these tasks.

Some of the enthusiasm for the NN school that has been rekindled in this decade can be understood as a response to disadvantages of the KB approach for which NNs seem to provide a corresponding advantage. The main advantages of NNs are the following.

1. **They are fast classifiers.** Most fixed weight connectionist networks compute an answer for a given set of inputs very quickly, partly through parallelism (trading space for time), but mostly through a closed world assumption that maps every input to an output within tight temporal bounds.

2. **They learn.** Neural networks can change their weights and/or connections, which makes them self-modifying programs. This can save labor when it is easier to present a set of training examples than to program the network or write a set of rules for solving a problem.

3. **They are probabilistic.** Neural networks learn and classify using statistical optimization criteria, so they can deal well with uncertainty. Specifically, they cope well with conflicting and incomplete information.

4. **They are global.** Neural networks' high degree of connectivity makes them well suited to problems involving a great deal of impinging evidence, for example when all of the data about a situation must be taken into account. Naturally occurring problems are often of this type, particulary KIPs.

5. **They are fault-tolerant.** Part of the extra space and connectivity used by neural networks is informationally redundant in a way that permits graceful degradation when parts of the network malfunction or when there is noise in the inputs.

These advantages of neural networks all address specific disadvantages often attributed to the KB approach, such as the slowness of inference, the need for too much labor to enter all the knowledge, the rigidity of binary distinctions, and the often unrealistic requirements that problems be neatly decomposable and that local information maintain perfect integrity. But neural networks themselves have several problems associated with them. Specifically,

1. **Slow learning.** Neural network learning algorithms

such as back-propagation can take quite long to converge, and although efforts to develop faster algorithms are under way, there appear to be important limits to these speedups. The convergence rate depends on specifics of the problem being worked on (Minsky and Papert, 1988).

2. **Spatial combinatorics.** NN architectures differ in their space requirements, but nearly all of them grow quite rapidly with the size of the problem if we try to solve for the general case, since they usually trade time complexity for space complexity.

3. **Local minima.** Neural nets can get trapped in suboptimal states in the space of activation or of weights. Much of the problem can be eliminated if we choose a judicious input-output representation, but as the problem gets larger and more complicated this can be increasingly hard to do, and it becomes apparent that neural networks fail to eliminate the need for careful thinking about the problem.

4. **Poor generalization.** The network can fail to correctly classify new input patterns even when it has learned all of the correct classifications for the training set. This is an important shortcoming because for problems of reasonable size, it is impossible to present all of the input patterns.

5. **Output underdetermination.** It is usually not possible to solve a real problem using input data from just one sensor or other input source. This underdetermination can arise for a number of reasons. For instance, the context can be important, or the input source may be a limited perceptual window on the problem, or background information whose acquisition is separated in time from the current input can be crucial to an interpretation.

The technical efforts to improve neural networks have so far focused mostly on the first few problems noted above, like learning speed and network size. Yet these do not appear to be the most serious barriers to the widespread use of neural networks. Moreover, the approaches that have been taken in trying to solve them (as well as the few efforts undertaken to address local minima and generalization) have almost all been based on general properties of networks, e.g. learning algorithms that are faster in general, simulated annealing algorithms for escaping local minima, or size considerations that apply to all problems. It looks as though a great deal of effort aimed at solving problems in this way could provide us with algorithms and architectures as generally well-tuned as possible but still prone to serious errors to which people will be very sensitive. These errors would come from those parts of the problems stated above (including part of the speed problem, a good deal of the size, local minima, and generalization problems, and possibly all of the underdetermination problem) which are intrinsically unsolvable using general approaches.

Because the difficulties arising with neural networks depend greatly on the particular problems to which they are being applied—some problems mapping easily onto an efficient architecture, and others not—the real solutions to the general problems with neural networks must vary with the specific application, involving principles for taking advantage of what we know about particular tasks to which the networks will be applied. This approach requires us to use knowledge that we have about a problem in selecting network structures and initial values for parameters. In other words, it requires that we do some initial progıamming of the network. This leads us to three additional problems with neural networks that affect our ability to program and understand (or verify) them:

6. **Lack of locality.** The representation of knowledge in neural networks is global, and this creates problems for building knowledge into them. In general, we cannot simply build links between nodes incrementally without worrying about how such links fit into the entire problem representation.

7. **Restrictive syntax.** To ensure the nice computational properties mentioned earlier and to ensure convergence in dynamic networks, network architectures have restrictions on the types of connections allowed. Sometimes our knowledge of evidential relationships simply does not obey such restrictions, although once expressed it can usually be recast in a form that does obey them.

8. **Semantic obscurity.** When we talk about problems in ordinary language (or even in a formal symbolic language), we do not use terms like weights, energy, and the like, so it is not obvious how to map knowledge so expressed onto a network representation. Hopes for verification procedures also must rest partially on semantic understanding, since empirical tests on a limited set of examples can be risky in real situations.

Consideration of the general problems 1-5 led us to conclude that we need to make substantial use of our knowledge about a particular task and domain, and build it into the network's structure and initial state. Problems 6-8 suggest that it may be too difficult to do this directly. Instead it appears that we should make use of the KB approach in some way, since it is geared toward solving the programming and verification problems we have with neural networks. Specifically, building knowledge bases in a sentential language gives us the following advantages: (a) It gives us a convenient way to enter what we know about the task and domain at whatever level of detail we seem to have in mind. For instance, we can say simply that proposition A supports proposition C as we would in a production system of heuristics or a truth maintenance system, without specifying some numerical probability, or we can specify exact probabilities if we want. (b) It eases the nonlocality problem (number 6) by giving us a way of stating axioms or constraints somewhat independently, with the usual concerns about consistency of the knowledge base. (c) It gives us a much less restrictive, more natural syntax than the ones required for neural networks, with which we can set forth the facts of the problem. And (d) it removes us from the semantic obscurity of neural networks by giving us a language (chosen from the repertoire of AI knowledge representation formalisms) with a well-understood semantics.

So the KB approach to AI has a lot to offer as a solution to the problems that plague neural networks, and, as argued earlier, neural networks nicely complement the KB approach. In particular, networks that overcome the general difficulties with the NN approach must embody, in their initial configuration, knowledge about the particular tasks they are to perform, and defining that knowlege in a design phase is what the KB approach is geared for. At the same time, the learning capabilities of neural nets lessen the amount of knowledge that must be defined, and the other features of neural networks, namely their soft, holistic inferencing, and run-time speed, help to break down the traditional barriers to using large knowledge bases for solving real problems. An obvious way to combine the approaches would be to define our knowledge in a KB first, possibly using the AI tools that have been built over the years such as theorem provers and other inference engines, and tools for entering knowledge. Then, when we felt that we had a good theory of the task and its domain, we could convert the KB into an appropriate neural network which would embody the knowledge contained in the KB. After network learning, we could try to verify the network for correctness by looking at what knowledge it has learned, and the easiest way to do this would be to construct a KB from the NN itself. The difficulty, then, and the main problem to which this project will be addressed, is

How do we go from a KB to an NN, and back again?

In other words, how can a network be made to embody, or be interpreted as, declarative knowledge?

David Marr and many others in the cognitive sciences have noted the usefulness of viewing problems in AI at two levels of analysis, called by him the "computational level" and the "algorithmic level" (Marr, 1982)[1] The basic idea is that solutions at the computational level must specify only the constraints involved in a problem (e.g. enough facts about it to uniquely determine a solution) without meeting the constraints of resource availability for carrying out the computation. At this computational level a theory for solving a problem is like (and may be literally) a logical theory, i.e. a set of axioms and its consequential closure. Solutions at the algorithmic level, on the other hand, must cope with resource constraints as well. Thus, at this level we must specify how conclusions are to be drawn from, for example, a set of axioms, and by necessity some conclusions in the consequential closure will be left out. Γhe resources required for computation at the algorithmic level can be divided into four kinds: time, space, labor, and data. In a Turing machine, these correspond to the number of moves, the amount of tape, the complexity of the finite control, and the length of the inputs.[2] Presently in computing we have, as a rule, less time and labor than we need, and more space and data than we can use, with a few exceptions. For the resource requirements of KIPs, the NNs approach seems well-suited because it is designed to be fast and to learn autonomously rather than to be programmed (minimizing time and labor), while consuming lots of processor power and lots of data for training (sometimes too much—see the next section). But at the computational level, KIPs require lots of empirically-derived constraints or knowledge to specify a solution, so a knowledge-based approach, as its name suggests, seems appropriate at this level. The idea behind our work is that designing systems to solve KIPs can be decomposed in just this way, with the knowledge-based approach and the neural nets approach operating at different levels and complementing one another. The resulting requirement is for a system that relates the two approaches in an approapriate way.

2 Our Approach: Knowledge Compilation and Recovery

2.1 Describing the Approach

We use declarative languages that are both convenient and expressive enough to define at least partially the evidential relationships of a problem. This is the computational level. At the algorithmic level, we use parallel networks, possibly incorporating learning, so that the theory defined at the higher level can be computed as efficiently as possible at run-time under the requirements for correctness. Learning will ease the labor burden that would ordinarily fall at the computational level. And relating the two levels, we use procedures of compilation and decompilation from the language to the networks and back again. The compile and decompile procedures need to be automated because it is often far from transparent what knowledge is embedded in the networks, even those (like the Pearl networks, see detail below) whose structure is closely tied to probabilities, and also because the translation in each direction is generally tedious. This last fact is caused by the global character of the translation.

We have explored several combinations of language, network architecture, and translation. Likewise, in the development tools we are beginning to build, the person attempting to solve a problem will be given a choice of several higher level languages for defining what he/she knows about the domain, matched by translation procedures to different network architectures. This diversity is necessary because different problems and computing situations require different levels of convenience and expressiveness, and different cost priorities for soundness, completeness, time, space, labor, and data. For instance, the method we have chosen below to illustrate this approach on the Yale Shooting Problem uses a variant of first order logic with default rules. The compilation procedure then translates statements in this language individually into constraints on a probability model, finds a particular probability model by maximizing entropy, and embeds this model in what Judea Pearl calls a "causal polytree" for use at run-time (Pearl, 1988).

Our approach to solving the Yale Shooting Problem (which differs from Pearl's own solution in Pearl, 1988) does not require learning, but many more complicated problems would. In fact, one method for solving a problem or building an efficient knowledge base that would be well served by this approach is one involving *iterative* compilation and decompilation. We define what we know about a problem as best we can in a language that seems appropriate, compile this representation into a network, let it do the best it can while constantly receiving new data to modify itself, and then decompile to a representation of what it has learned. At this stage, we could inspect the principles it is applying and see for ourselves whether they are sufficiently general to apply to novel inputs, or obviously taking advantage of regularities that happened to hold during training but will not apply later on. If this latter is the case, then we need to modify the knowledge by hand at the declarative level, recompile, and try again. Over time, we should achieve a better system by this type of refinement, and moreover we can be confident that our solutions are sufficiently general because we can recover explanations from the system.

The development of this methodology for networks that learn is part of the work in which we are currently engaged, but we should emphasize that this is an experimental approach. While we feel after careful analysis that this approach can make neural networks work, we really cannot tell how easy it will be to overcome the inherent problems with the networks that we listed previously. Our rationale is that networks cannot work well on large problems unless we provide initial structuring that reflects what we know about the problem, and knowledge bases large enough to encompass everything we know about natural language, perception, and common sense reasoning problems are impossible to build entirely by hand, let alone to run an inference engine over. So the right method must lie somewhere in the middle, but it will require much experimentation before we can say just where. It is possible, for instance, that learning can solve most of the problem for a surprising number of applications, and only some crude structuring is necessary. Alternatively,

[1]This distinction is really identical to those of Noam Chomsky (competence vs. performance), John McCarthy and Patrick Hayes (epistemological vs. heuristic adequacy), Herbert Simon (substantive vs. procedural rationality), Daniel Dennett (intentional stance vs. subpersonal stance), Allen Newell (knowledge level vs. symbol level), and Hector Levesque (content vs. form).

[2]The four resources divide into two natural clusters, with the minimal requirements for them in a given computation being the subjects of complexity and information theory. Space and time (the subjects of computational complexity theory) can be traded off one for the other, as can program and data (measured by Kolmogorov complexity and Shannon's entropy, respectively, in information theory). In addition, the computational requirements (space-time) trade off against the informational requirements (program-data) in ways that are just beginning to be studied theoretically.

it may prove unmanageable to use learning until most of the knowledge that must be defined about a problem is built into a network. These possibilities are what we would like to test, and the answers will almost certainly depend on what types of problems we are trying to solve, even within the class we have called knowledge-intensive.

2.2 An Example of Compilation

To illustrate the approach we are taking, we will present a step-by-step description of how we might translate the Yale Shooting Problem, described in a declarative language by a designer, into a network that will answer questions about the shooting situation based on the information in the description. To do this, we must choose a specific description language, target architecture, and translation algorithm. All of this should make it clearer what kinds of tools we will be building, although we will stop short of actually solving the problem.

The language we will use for the high level description is a variant of the functional predicate calculus that has a default implication operator ("$\longrightarrow$") which means that the left hand side provides strong evidence for the right hand side. The objects in the domain are all time points. The statements in this language would be divided into background knowledge, which applies more generally across situations, and situational knowledge specific to the case at hand. The background knowledge would be entered as follows.

$$\begin{aligned} \forall t\, Loaded(t) \& Shoot(t) &\Longrightarrow Dead(t+1) \\ \forall t\, Loaded(t) &\longrightarrow Loaded(t+1) \\ \forall t Alive(t) &\longrightarrow Alive(t+1) \\ \forall t\, Dead(t) &\Longleftrightarrow \neg Alive(t) \\ T_1 &= T_0 + 1 \\ T_2 &= T_1 + 1 \end{aligned}$$

The last two statements express background knowledge about specific times, like our knowledge that the 18th Century came before the 19th, and that both have passed. We might think of the above as already forming part of the knowledge base before we describe the current situation, which reads as follows.

$$\begin{gathered} Loaded(T_0) \\ Shoot(T_1) \\ Alive(T_1) \end{gathered}$$

Finally, we would enter a query. Since we want to know whether Fred is dead at time T_2, we would enter

$$?Alive(T_2).$$

The target architecture we will use is Judea Pearl's causal polytrees (Pearl, 1988). Polytrees have the advantages of a well-understood semantics (for easy decompilation) and quick settling. The disadvantages are that the translation (compilation) is computationally difficult and that the highly structured nature of the network does not lend itself to flexible learning. For this problem, we do not require learning and the number of variables is small enough that we can handle a small combinatorial explosion in the translation algorithm, so polytrees will do fine for us this time. For other problems, we would make another choice depending on the cost priorities specific to the problem. The translation algorithm, which is just one of several we could choose for going from logic with defaults to causal polytrees, consists of five steps. First we generate a set of equational constraints on the probability model. These are obtained by translating each statement (except for equalities) in the logical description of background knowledge into a probability equation, and substituting known object constants for object variables to make all of the probability statements unquantified. We can also make use of known situational facts (like $Shoot(T_1)$) and simple definitions (like $Dead(T_2) \Longleftrightarrow \neg Alive(T_2)$) to simplify the probability constraints, yielding a set that includes following.

$$\begin{aligned} Pr\{\neg Loaded(T_1) \vee \neg Alive(T_2)\} &= 1 \\ Pr\{Alive(T_2) \mid Alive(T_1)\} &= \nu \\ Pr\{Loaded(T_1) \mid Loaded(T_0)\} &= \nu \end{aligned}$$

The global parameter ν is a number inside the unit interval which is determined by the designer. In this case, we will set it to be 0.9.

The key idea in most of these translations is to construct a *model* from the constraints. A model specifies what happens for every combination of events expressible in the language, in this case probabilistically, and therefore goes beyond the information in the description. AI researchers have been attracted to models as a way of doing tractable reasoning partly for computational reasons (Levesque, 1986) and partly because they appear from introspection to be how we reason about change ourselves without falling victim to the "Frame Problem," or the problem of enumerating all that becomes or remains true or false after an event or action. We understand what happens when a ball is kicked into a window not because we run down a list of sentences in our head about the effects of kicking balls at windows, but because we set up from the description some internal picture or model of what happens. To get a probability model for the Yale Shooting Problem, we need to infer probabilities that cannot be calculated from the given information merely using the probability calculus. Instead we find a model that maximizes the probabilistic entropy subject to our constraints. To set this up as a nonlinear optimization problem, we need to divide the space of possibilities into disjoint events, set up an objective function for maximizing the entropy of these probabilities, and add a few constraints that come from probability theory. When this is done, the resulting nonlinear program is the following (see notational definitions below).

$$\begin{aligned} MAXIMIZE - \sum_{i,j,k,l} I(Pr\{L_0 = i \& A_1 = j \& L_1 = k \& A_2 = l\}) \\ S.T.\, Pr\{L_1 = 0\} + Pr\{A_1 = 0\} - Pr\{L_1 = 0 \& A_1 = 0\} = 1 \\ Pr\{A_1 = 1 \& A_2 = 1\} / Pr\{A_1 = 1\} = \nu \\ Pr\{L_0 = 1 \& L_1 = 1\} / Pr\{L_0 = 1\} = \nu \\ \sum_{i,j,k,l} Pr\{L_0 = i \& A_1 = j \& L_1 = k \& A_2 = l\} = 1 \\ \forall i,j,k,l\, Pr\{L_0 = i \& A_1 = j \& L_1 = k \& A_2 = l\} \geq 0, \end{aligned}$$

where $I(x) = x \log x$, event variables A_1 and L_0 denote the propositions $Alive(T_1)$ and $Loaded(T_0)$, and so forth, and i, j, k, and l range over $\{0,1\}$ and denote the truth (1) or falsity (0) of the event variable. The last two constraints come from probability theory. The solution to the nonlinear program is a probability model which can then be used to generate a causal polytree for answering queries. We first decompose the probability distribution $P(L_0, A_1, L_1, A_2)$ into a product of probabilities using the chain rule for conjunction. When all conditional independencies are taken into account, with the event variable order we have been

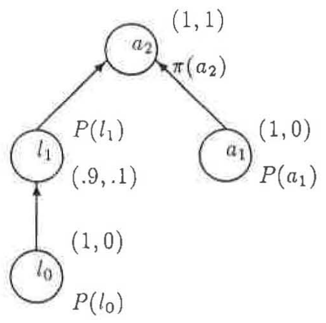

Figure 2: Causal Polytree for the Yale Shooting Problem

using, the distribution can be rewritten as the product $P(L_0)P(A_1)P(L_1 \mid L_0)P(A_2 \mid L_1, A_1)$. Applying Pearl's algorithm for constructing Bayesian networks, this generates the tree shown in Figure 2. Each node has associated with it two parameters, π (below the node) and λ (above the node). If we solve the nonlinear program given above (using, say, the SQP method of optimization employed in the NPSOL program (Gill et al., 1986) and provide the evidence that *Loaded*(T_0) and *Alive*(T_1), then the network concludes that $Pr\{Alive(T_2) \mid Loaded(T_0) \& Alive(T_1)\} = 0.5$. In other words, we have not provided sufficient information to say that it is more likely that the person died than that the gun was unloaded. At this point, we can change the knowledge base of statements to make this fact clear, and iterate through the compilation again. The details of the network updating algorithm and the parameter assignments are too complicated to present in this paper, but are given in section 4.3 of Judea Pearl's book (Pearl, 1988). The parameters can only be fully determined once we have a complete probability model, thus making necessary the solution of the maximum entropy problem. However, because the maximum entropy problem grows exponentially in the number of atomic propositional variables, it is infeasible to solve exactly for problems other than small ones like the Yale Shooting Problem. To get around this problem, we must use heuristic methods of optimization geared toward finding the maximum entropy solution. These methods would need to be developed as part of the project. Our initial plan is to add intelligence to the process of decomposing the space into disjoint possibilities, instead of using the straightforward method given here.

Additional complications that can arise in these problems include the dependence of the constructed network on the ordering of the event variables, the need to eliminate cycles that make networks fail to be singly-connected (and hence not polytrees), and the problem of large matrices of parameters at nodes which depend on many interacting causal variables, since they grow in size exponentially. All of these problems have solutions (see Pearl, 1988), but the solutions are tedious and so automation is crucial if they are to be applied to problems of any size. This complication is a point we want to emphasize generally. We have made this example simple enough to present briefly and in a way that gives a flavor of the work we are doing, but in the process we have stripped away most of the complicating factors (including learning and decompilation) that necessitate tools for constructing these networks automatically from a friendlier declarative description.

2.3 Approaches to Decompilation

Our approach to the development of decompilation algorithms involves analysis of correlations in a unit's activation with conditions in its input set to determine objective meaning for a hidden node, followed by translation of the set of weights into meaningful probability statements. The first of these stages is similar in spirit to the work done by Terry Sejnowski and Charles Rosenberg on analyzing NetTalk (Rosenberg, 1987), and the second stage approach is described in a paper by Davies given at the 1988 INNS Conference in Boston (Davies, 1988). Since this work has appeared elsewhere we will not repeat the mathematical details here.

3 Related Work by Others: How We Differ

A number of researchers have expressed the opinion that NNs and KBs apply best to different *problems* (e.g., Hecht-Nielson, 1986). Hybrid KB/NN systems thus sometimes use the KB and the NN to work on different aspects of a problem, rather than using them at different stages in the solution of a single applications problem as in this project.

The principal research efforts that have involved encoding or recovering knowledge in neural networks, in a manner comparable to that proposed here, have been the following.

1. Work by Steven Gallant that is being further developed by Hecht-Nielson Neurocomputers, on so-called "connectionist expert systems" (Gallant, 1988). This involves an induction of rules from statistics collected from examples rather than a translation of the parameters obtained during adaptation.

2. Work by Dana Ballard on implementing predicate logic theorem proving in neural networks (Ballard, 1986). Of all the work that has been done by others, this is the most like what we are proposing. Ballard's algorithm is used for proof by refutation, and so can only answer Yes/No queries for specific propositions, rather than forward chaining to a set of conclusions as in the algorithm we have developed for translating first order logic constraints (Davies, 1989).

3. Work by a number of researchers (Derthick, 1988; Weber, 1989; Thagard, 1988; Pearl, 1988; Shastri, 1989; Jones & Story, 1989) on using neural networks for nonmonotonic and evidential reasoning. Each of these researchers has constructed networks by hand to solve inference problems, including examples like the Yale Shooting Problem detailed earlier. But the compilation and recovery of declarative knowledge in their systems are, if they are mentioned at all, accomplished by rather simple, local mappings of a constraint onto a network link, rather than by global fitting. This approach limits either the expressiveness of the network or its efficiency rather severely for general problem solving. It limits the expressiveness because sets of constraints whose local translations violate the network's formation rules (syntax) cannot be represented. Thus, for example, Lokenda Shastri, Mark Jones and Guy Story, who do provide interpretations of network structures as embodying declarative theories, restrict themselves to inheritance networks that are relatively inexpressive for local translation algorithms. When the syntax is relaxed to allow more flexibility, for example when cycles or combinatorial rules are permitted, then the network will by necessity run more slowly, if it settles at all. The global translation approach we use, which requires automation, is designed to get around this tradeoff by paying the price during compilation rather than during design or at run time.

To summarize, then, the distinctive features of our approach are (1) the use of global rather than local translation, (2) the idea of an iterative compile/decompile cycle with some place for learning and some place for hand-crafting, and (3) the fact that the KB and NN approaches are both used, but at different stages in the solution of a problem rather than to solve different problems.

References

[1] Anderson, J. A. & Rosenfeld, E. *Neurocomputing: Foundations of Research.* Cambridge, MA: The MIT Press, 1988.

[2] Ballard, D. H. Parallel Logical Inference and Energy Minimization. *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, Philadelphia, PA, August 11-15, 1986, pp. 203-208.

[3] Buchanan, B. G. & Shortliffe, E. H. *Rule-Based Expert Systems: The* MYCIN *Experiments of the Stanford Heuristic Programming Project.* Reading, MA: Addison-Wesley, 1984.

[4] Bundy, A. *The Computer Modeling of Mathematical Reasoning.* New York: Academic Press, 1983.

[5] Davies, T. R. Some Notes on the Probabilistic Semantics of Logistic Function Parameters in Neural Networks. *Neural Networks*, 1, Supplement 1, Abstracts of the First Annual INNS Meeting, Boston, September 6-10, 1988, p. 88.

[6] Davies, T. R. Neural Networks and Artificial Intelligence. Independent Research and Development Data Sheet, Project WS, SRI International, Menlo Park, California, March 15, 1989.

[7] Derthick, M. *Mundane Reasoning by Parallel Constraint Satisfaction.* Technical Report CMU-CS-88-182, Computer Science Department, Carnegie Mellon University, Pittsburg, PA, September 1988.

[8] Gallant, S. I. Connectionist Expert Systems. *Communications of the ACM*, 31(2), February 1988, pp. 152-169.

[9] Gill, P. E., Murray, W., Saunders, M. A., Wright, M. H. *User's Guide for NPSOL (Version 4.0): A Fortran Package for Nonlinear Programming.* Technical Report SOL 86-2, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, CA, January 1986.

[10] Hanks, S. & McDermott, D. Default Reasoning, Nonmonotonic Logics, and the Frame Problem. *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86).* Philadelphia, PA, August 11-15, 1986.

[11] Hecht-Nielsen, R. Performance Limits of Optical, Electro-optical, and Electronic Neurocomputers. In Szu, H. (Ed.) *Hybrid and Optical Computing.* Bellingham, WA: Society of Photo-Optical Instumentation Engineers, 1986, pp. 277-306.

[12] Jones, M. A. & Story, G. A. Inheritance Reasoning in Connectionist Networks. *First International Joint Conference on Neural Networks*, Washington, D.C., June 1989.

[13] Lapedes, A. & Farber, R. *Nonlinear Signal Processing Using Neural Networks: Prediction and System Modeling.* Technical Report LA-UR87-2662, Los Alamos National Laboratory, Los Alamos, NM, 1987.

[14] Levesque, H. J. Making Believers Out of Computers. *Arificial Intelligence*, 30, 1986, pp. 81-108.

[15] Marr, D. *Vision.* New York: W. H. Freeman and Co., 1982.

[16] Mead, C. A. & Mahowald, M. A. A Silicon Model of Early Visual Processing. *Neural Networks*, 1:1, 1988, pp. 91-97.

[17] Minsky, M. L. & Papert, S. A. *Perceptrons* (Expanded Edition). Cambridge, MA: The MIT Pres, 1988.

[18] Pearl, J. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Los Altos, CA: Morgan Kaufmann, 1988.

[19] Rosenberg, C. R. Revealing the Structure of NETtalk's Internal Representations. *The Ninth Annual Conference of the Cognitive Science Society*, Seattle, Washington, 16-18 July, 1987, pp.537-554.

[20] Shastri, L. Default Reasoning in Semantic Networks: A Formalization of Recognition and Inheritance. *Artificial Intelligence*, July, 1989, pp. 283-356.

[21] Thagard, P. *Explanatory Coherence.* CSL Report 16, Cognitive Science Laboratory, Princeton University, Princeton, NJ, March 1988.

[22] Webber, B. L. & Nilsson, N. J. *Readings in Artificial Intelligence.* Los Altos, CA: Morgan Kaufmann, 1981.

[23] Weber, J. A Statistical Approach to the Qualification Problem. Talk Presented at Artificial Intelligence Center, SRI International, Menlo Park, CA, February 23, 1989.

Artificial Intelligence in the Pacific Rim
Proceedings of the Pacific Rim International Conference on Artificial Intelligence, Nagoya 1990

ISBN 4-274-07636-9 (OHMSHA)
ISBN 90-5199-053-7 (IOS Press)

This post-conference edition published and distributed in Japan by:
OHMSHA, LTD.
3-1 Kanda Nishiki-cho, Chiyoda-ku, Tokyo 101, Japan

Distributed in North America by:
IOS Press, Inc.
Postal Drawer 10558, Burke, VA 22009-0558, U.S.A.

Distributed in Europe and the rest of the world by:
IOS Press
Van Diemenstraat 94, 1013 CN Amsterdam, The Netherlands

Printed in Japan

Artificial Intelligence in the Pacific Rim

Edited by
Hozumi Tanaka
Tokyo Institute of Technology

IOS Press
Amsterdam, Washington

Ohmsha
Tokyo, Osaka, Kyoto