Learning in Parallel Distributed Processing Networks: Computational Complexity and Information Content[1]

JOHN F. KOLEN[2]

ASHOK K. GOEL[3]

August 7, 1990

To appear in IEEE Systems, Man, and Cybernetics.

Keywords: Back propagation, computational complexity, information content, learning, neural networks, parallel and distributed processing.

[1] Manuscript received . . .. This paper is an expanded and revised version of work presented at the Fourth Annual Aerospace Applications of Artificial Intelligence Conference, Dayton, Ohio, October 1988, and at the Second IEEE International Conference on Neural Networks, San Diego, California, July 1988. This research has been supported by the Office of Naval Research, grant N00014-89-J-1200.

[2] J. F. Kolen is with the Laboratory for Artificial Intelligence Research, Department of Computer and Information Science, The Ohio State University, Columbus, Ohio 43210, USA.

[3] A. K. Goel is with the Artificial Intelligence and Cognitive Science Group, College of Computing, Georgia Institute of Technology, Atlanta, Georgia 30332-0280, USA.

Learning in Parallel Distributed Processing Networks: Computational Complexity and Information Content

JOHN F. KOLEN and ASHOK K. GOEL

Abstract

Neural networks offer an intriguing set of techniques for learning based on the adjustment of weights of connections between processing units. However, the power and limitations of connectionist methods for learning, such as the method of back propagation in parallel distributed processing networks, are not yet entirely clear. We report on a set of experiments that more precisely identify the power and limitations of the method of back propagation. The experiment on learning to compute the exclusive-OR function suggests that the computational efficiency of learning by the method of back propagation depends on the initial weights in the network. The experiment on learning to play *Tic-Tac-Toe* suggests that the information content of what is learned by the back propagation method is dependent on the initial abstractions in the network. It also suggests that these abstractions are a major source of power for learning in parallel distributed processing networks. In addition, we show that the learning task addressed by connectionist methods, including the back propagation method, is computationally intractable. These experimental and theoretical results strongly indicate that current connectionist methods may be too limited for the complex task of learning they seek to solve. We propose that the power of neural networks may be enhanced by developing task-specific connectionist methods.

Introduction

Machine learning has long been an important issue in Artificial Intelligence research. As early as the mid-sixties, Samuel [38] identified three central issues concerning the development of information processing systems capable of learning. First, the system must be able to internally represent what it knows and what is to be learned: *the representation problem*. Second, once incorrect performance has been detected, the system must be able to identify which part of the system is responsible for the incorrect performance (and make modifications to achieve correct performance): *the credit assignment problem*. Third, the system must be capable of generalizing over existing abstractions in its memory: *the generalization problem*. In the late sixties, Minsky and Papert [30] emphasized the importance of a fourth constraint, namely, the system must be capable of learning within a reasonable amount of time: *the computational complexity problem*.

Since then, several attempts have been made at building systems that learn. Until recently, most of these efforts were based on algorithmic processes operating on discrete symbolic representations (e.g., [2]). This research has led to the development of several techniques for learning such as learning from examples, from analogies, from explanations, etc. While this line of research has led to the building of small-scale systems that learn more or less well in relatively narrow domains, computationally feasible solutions to the general problem of learning are yet to be discovered.

Recently, research on neural networks has led to the development of a different set of techniques for learning based on the adjustment of weights of connections between processing units in a "continuous" space. Again, these techniques have been used with some success at building small-scale systems capable of learning relatively narrow domains. However, the power and limitations of connectionist methods for learning such as the method of *back propagation* of corrective feedback are not yet entirely clear. If, for instance, a network merely memorizes the correct solutions to the specific exemplars on which it is trained, then there would be little *a priori* reason to believe that it will behave appropriately when presented with a novel situation. An adequate general solution to the learning problem must enable the system to generalize over the abstractions stored in its memory.

Overview

We have conducted a set of experiments on learning that more precisely identify the power and limitations of the method of back propagation in parallel distributed processing (PDP) networks. Elsewhere [16], we provided a preliminary account of the experimental results concerning the credit assignment problem of learning. In this paper, we describe some of the experiments in detail, and discuss the results in relation to the general problem of learning. First, we present a brief review of connectionist methods for learning, specifically, the method of back propagation of corrective feedback. Next, we report on a set of experiments on learning to compute the exclusive-OR function. The goal of these experiments was to explore the computational feasibility of learning by the method of back propagation. We found that the computational efficiency of learning depends on the *initial weights* in the network, i.e., on the weights of the connections between the processing units prior to learning.

We then report on a second set of experiments on learning to play the game of *Tic-tac-toe*. This experiment had two aims. First, to identify the information content of what is learned by the method of back propagation.[4] Second, to identify the sources of power for learning in PDP networks. We find that the content of what is learned is dependent on the *initial abstractions* in the network, i.e., on the domain concepts stored in the network memory prior to learning. We also find that these abstractions are a major source of power for learning in PDP networks.

Next, we consider the computational complexity of the learning task addressed by the connectionist methods of learning, including, but not limited to, the back propagation method. We formally show that this learning task is computationally intractable. Finally, we discuss potential solutions to some of the limitations of connectionist methods for learning relative to the learning task they seek to solve. Specifically, we propose the development of *task-specific* connectionist methods.

Learning in Parallel Distributed Processing Networks

Neural networks come in a number of different varieties [13]. For example, *connectionist networks* [9], *parallel distributed processing* (PDP) networks [36], *constraint optimization* networks [20], and *adaptive resonance* networks [3]. Within the framework of PDP networks, a number of techniques can be used for learning. For example, the family of *back propagation* methods [35], the family of methods based on *simulated annealing* [19], and the family of methods based on *harmony theory* [39]. In this paper, we confine our attention to the method of back propagation of learning in PDP networks. However, much of our analysis is relevant to other connectionist learning schemes as well.[5]

Parallel Distributed Processing

The processing units in PDP networks are organized in input, output, and one or more hidden layers. The units in the various layers are connected in a layered configuration. Figure 1 shows units i and j, and the connection between the units with a weight of w_{ij} in a PDP network. The output of a unit is computed by means of a semi-linear threshold function. A commonly used function of this kind is the classic logistic function [35]:

$$o_j = \frac{1}{1 + e^{-(\sum_i w_{ij} o_i - \theta)/T}} \tag{1}$$

where θ is the threshold of the unit, and T, the temperature, regulates the slope of the function within the critical region.

Insert Figure 1 about here.

It has been demonstrated that, at least in principle, it is possible to design neural networks that are Turing-Universal [27, 33, 11]. [6] However, in practice, designing a PDP network to compute a given function can be very difficult. The critical choices are the number of layers, the number of units in each layer, and the connectivity between the layers and the units.

To illustrate the difficulty of designing PDP networks, let us consider the design of a n input, 1 output network capable of learning an arbitrary binary function. Assuming binary inputs and outputs, the network must be able to compute any one of 2^{2^n} functions, each requiring a different weight configuration. If m is the number of connections in the network, and W is the set of possible weight values for each connection in the network, then there are $|W|^m$ weight configurations, where $|W|$ is the size of W. To cover the entire function space, W must contain at least $2^{2^n/m}$ distinguishable weight configurations.[7]

Back Propagation of Corrective Feedback

Learning in PDP networks takes the form of adjusting the weights of connections between the processing units in the network. The network is subjected to training during which

> a specific exemplar of a given task is presented as input to the network and the incorrect output of the network is detected,
>
> a corrective feedback is supplied by the trainer and back propagated to the individual connections in the network, and
>
> the connection weight changes are accumulated until all (or a suitable subset) of exemplars have been presented, at which time the weights are updated in the direction of steepest gradient descent in error space.

This procedure, called *back-propagation of corrective feedback*, is often expressed as the *generalized delta rule* [35], which calculates error signals at each unit of the form

$$\Delta w_{ij} = \eta \delta_j o_i$$

with

$$\delta_j = t_j - o_j$$

for the output units, and

$$\delta_j = \sum_k \delta_k w_{kj}$$

for the hidden units, where Δw_{ij} is the change in the weight w_{ij} of the connection from unit i to unit j, o_i is the output of the unit i, t_j is the correct output supplied by the trainer, o_j is the output of the network, and η is a constant of proportionality called the learning rate. This procedure is executed repeatedly until some performance criterion is met, when the network is said to converge to the correct solution for specific examples of the task. In this manner, the network is trained to correctly perform the task on a set of specific exemplars selected by the trainer.

Learning to Compute XOR

We conducted two sets of experiments on learning in PDP networks by the method of back propagation of corrective feedback. The first set of experiments focused on learning to compute the Boolean function of two-input exclusive-OR (XOR). The XOR task is a simple version of the more general *parity problem* [30] with input patterns of size two. The goal of these experiments was to explore the computational feasibility of learning by the method of back propagation, specifically, to identify some of the conditions under which the back-propagation method can lead to computationally efficient learning.

The Original Experiment

Rumelhart, Hinton and Williams [35] have earlier reported on a PDP network that learns to compute the XOR function by the method of back propagation. Their network consists of two units for the two inputs, an output unit for the one output, and one hidden unit. The network learned to correctly compute XOR in a few hundred (558) sweeps through the four stimulus patterns. This implies that starting from an initially random set of the connection weights, it took a few thousand ($558 \times 4 = 2232$) training sessions to adjust the connection weights before the network learned to compute XOR correctly.

Minsky and Papert [30] have pointed out that it is rather difficult to analyze the meaning of these results for the general parity problem without knowing how well the learning scheme performs on input patterns of size greater than two. In fact, it is difficult to evaluate the computational efficiency of the learning scheme even for input patterns of size two since Rumelhart *et al* do not provide much of the information needed for such an evaluation. While the learning rule is reported to be the generalized delta rule with a learning rate of $\eta = 0.5$, the range in which the initial weights of the connections can vary is not specified. Specifying the range of initial weights for a deterministic learning procedure is important since it defines the set of possible trajectories through the parameter space. Further, no mention is made of whether the biases of the processing units in the network are fixed or can vary. In sum, while one may conclude from their experiment that the network learned to compute the XOR function correctly, it is not possible to draw any definitive conclusions about the computational feasibility of learning by the method of back propagation.

A Simulation of the Original Experiment

In order to more precisely determine the conditions under which the method of back propagation of corrective feedback can lead to computationally efficient learning of the XOR function, we repeated the experiment of Rumelhart *et al.*. In our simulation, we used the same network architecture as in the original experiment; a schematic of the network is shown in Figure 2. The learning rule (the generalized delta rule) and the learning rate ($\eta = 0.5$) were also kept the same. The range of the initial weights and the biases were 0.0 to 5.0. The simulation was performed a hundred times, the results of which are shown in Figure 3. The x coordinate in the figure identifies the number of sessions that were required to train the network, while the y coordinate identifies the number of simulations (out of 100) for which a given number of training sessions were required. The simulations that took more than two hundred thousand training sessions have been collapsed giving the spike at two hundred thousand sessions.

Insert Figure 2 about here.

Insert Figure 3 about here.

In each of the 100 simulations, the networks started with a randomly generated initial set of connection weights, and many eventually learned the set of weights for correctly computing XOR. We note that in a few of the simulations we were able to reproduce the results reported by Rumelhart *et al*. However, for a vast majority of simulations it took the network substantially more training sessions before it learned to compute XOR correctly. In fact, for nearly half the simulations, more than two hundred thousand training sessions had passed, and still the network could not compute XOR correctly.

The important issue here is the difference between conditions under which the network learned to correctly compute XOR in only a few thousand training sessions and those under which it took a few hundred thousand sessions. Since the training procedure and learning rate are the same, it appears that the difference lies in the initial random selection of connection weights. That is, the computational efficiency of learning depends on initial weights of connections in the network.

It could be argued that the above experiment violated the "standard" procedure of starting the initial weights very "close" to zero ($|w| < 0.3$ typically) so that the slope of the output function is large enough. In anticipation of this objection, we repeated the above experiment, this time varying several parameters in the simulation: initial weight range, learning rate, η, and momentum rate, α. We were interested in the number of initial weight states which did not converge to the target function within 50000 epochs.[8] During the simulation, network outputs of less than or equal to 0.49 were regarded as 0, while outputs greater than or equal to 0.51 were considered to be 1. Figure 4 illustrates the results of this experiment. In this graph, the x axis represents the range of initial weights which were randomly generated between 0.0 and x, and the y axis represents the percentage of initial weight configurations which did not converge in 50000 epochs. On can observe from the graph that for small initial weight ranges (less than 0.2), convergence is hampered. As the range gets farther from the origin, convergence occurs almost everywhere. As the upper bound increases from about 2.0, more and more networks no longer converge within $50,000$ epochs. Note that this behavior is similar for each combination of learning and momentum rates.

Insert Figure 4 about here.

Two conclusions can be drawn from these results. First, convergence to the "correct" set of weights is considerably more likely to occur within $50,000$ epochs if the initial weights are within a window of convergence. Second, the learning rate (L) and momentum rate (M) apparently have relatively little effect on the percentage of non-convergent initial weight states. Similar results were obtained when the absolute values of the initial weights were bounded by a parameter, as shown in Figure 5. Two important differences did arise, even though the two graphs were similar. First, the percentage of non-convergent networks initialized near the origin rose dramatically. Second, for all learning and momentum rate pairings, at least ten percent of the initial conditions always converged to XOR.

In sum, *the computational efficiency of the learning scheme is strongly dependent on the initial choice of weights of connections in the network.* Kolen and Pollack[24] report on additional experimental results in support of this conclusion.

Insert Figure 5 about here.

Learning Tic-Tac-Toe

The second set of experiments involved networks learning to play *Tic-tac-toe*[9]. This set of experiments had two goals. The first goal was to identify the information content of what PDP architectures learn by the method of back propagation of corrective feedback. The second goal was to identify the sources of power for learning in PDP networks.

The Original Experiment

Rumelhart, Smolensky, McClelland and Hinton [37] have earlier reported on a PDP network that learns to play *Tic-tac-toe*. Their network contains 67 units, including 9 input units for the network's current board position, 9 input units for the opponent's current position, and 9 output units for the next move by the network. There are also 40 hidden units, which are organized in 8 groups corresponding to the eight ways in which an entire row, column or diagonal can be captured. Each such group contains 5 units corresponding to the five abstractions (domain concepts) of *an empty line*, *friendly singlet*, *friendly doublet*, *opponent singlet*, and *opponent doublet*. These abstractions provide the network with a model of the *Tic-Tac-Toe* domain.

Once again, it is rather difficult to analyze the implications of this experiment for learning since it is not clear what has the network really learned. It could not have learned the domain concepts of *a row*, *a diagonal*, *an empty line*, *a friendly singlet*, *an opponent doublet*, etc. The system designers, in fact, "programmed" these abstractions into the network in a "compiled" form [5]. Did their network, then, learn situation-specific responses to various board positions? Moreover, the source of power for learning even the situation-specific responses is not clear. Does the source of power reside in the abstractions explicitly represented in the network or the method of back propagation (or both)? Can the network learn the abstractions by the back propagation method?

A Variation on the Original Experiment

The task of learning to play *Tic-tac-toe* would be much harder if the network lacked the high-level abstractions to begin with. In fact, the real test of learning by the method of back propagation would be whether the network can learn some of these abstractions. Thus, we designed a PDP network that learns to play *Tic-tac-toe* but without providing it with any of these abstractions. The network, shown in Figure 6, contains 9 input units for the network's current board position, 9 input units for the opponents current position, and 9 output units for the next move, just as in the network of Rumelhart *et al.*[10]The network also contains hidden units configured in a single layer, where the number of the hidden units is kept a parameter of the network.

Insert Figure 6 about here.

In order to reduce the time required for a training session, we used a variation of the generalized delta rule. For a fixed precision, the continuous and monotonic activation function degenerates to the linear threshold function as temperature T approaches 0, as can be seen from Equation 1 in section 2. Since the activation function is now a step function, this implies that the derivative of the function would be 0. However, the derivative only adjusts the magnitude of the change in the connection weights, not its direction. Thus, if the steepest gradient descent method is used for calculating the change in connection weights but the derivative term from the calculations is dropped, then the weights would be changed in the right direction but the actual distance traversed in the error space would typically be somewhat larger. This tends to reduce the time for a training session and also makes the back propagation less sensitive to local perturbations in the error space.

As the opponent to the neural network, we used a *symbolic algorithm*. The symbolic algorithm used the high-level abstractions (e.g., *row*, *opponent doublet*) that were denied to the neural network. The algorithm used a simple heuristic. For any given board position, it selected the move that would maximize the number of rows, columns, and diagonals of which it had sole possession while minimizing the number of rows, columns, and diagonals possessed by its opponent (the network). Thus, given the same board configuration the symbolic algorithm would always generate the same move. Moreover, because of the nature of the heuristic, there existed a sequence of moves which, for certain board positions, could beat the algorithm. This made the symbolic algorithm a strong, yet imperfect, opponent for the network.

In the training procedure we adopted, the network was first made to play the symbolic opponent, and then learn from the moves made by the winning side. Initially, the network played the game by evaluating the current board position and generating a list of the moves it would like to make. This list includes both legal and illegal moves, where an illegal move involves moving into an already occupied square. Illegal moves were then stripped from the list, and the actual move was selected (at random) from the remaining moves.[11] A record was kept of the moves throughout the play of a game. At the end of the game, the network was trained to make the same responses to various board positions as by the victorious opponent. That is, the network was trained to be the best of the two players. This was done by using a variation of the generalized delta rule described above. Thus, the network started out as a random move generator, and then it gradually improved its performance by learning from its opponent each time it lost a game or by reinforcement of what it had previously learned if it won the game.

In this manner, the network played 10 matches with the symbolic algorithm, where each match consisted of a $1,000$ games. The results are shown in Table 1.[12] Each row in the table corresponds to a particular match and contains the number of games out of $1,000$ that the symbolic algorithm won, the number that the network won, and that ended in a stalemate. The matches are sorted by the network's performance. We found that in each of the 10 matches the network learned to draw its opponent in about 100 games. We also found that although it was possible to beat the imperfect symbolic algorithm, in fact, the network won exactly one game out of a total of $10,000$ games.

Insert Table 1 about here.

To draw or win against the symbolic algorithm often and consistently requires that our network learn the abstractions of *a row*, *an opponent doublet*, *etc.*, which were "programmed" into the network of Rumelhart *et al.* However, we could detect no such abstractions despite a careful examination of the activation levels of the hidden units in the network.

The ability to generalize, of course, is sensitive to the number of hidden units in the network. If the number of hidden units is too small, then the learning problem is over-constrained and there is not enough structure for the network to capture all the needed abstractions; and if the number is too large, the problem is under-constrained and generalization is not possible [6]. In anticipation of this objection, we repeated the above experiment, changing the number of hidden units in the network from 3 to 60 in steps of 3. While the number of games played before the network learned to draw its opponent showed some change with change in the number of hidden units, we could detect no evidence of generalization to the needed abstractions.

Two conclusions may be drawn from the results of this experiment taken together with the experimental results of Rumelhart *et al*. First, without the high-level abstractions, our network only learned to mimic the situation-specific responses of its symbolic opponent. It could not generalize to the high-level abstractions needed to win against the opponent. In contrast, given the needed abstractions, the network of Rumelhart *et al*, appears to have learned the strategy for playing and winning *Tic-Tac-Toe*. Thus, *the content of what is learned by the method of back propagation is strongly dependent on the initial abstractions present in the network*. Second, *these high-level abstractions are a major source of power for learning in PDP networks*. We have suggested elsewhere [5] that the success of PDP networks is due to the "programming" of these abstractions in the networks in a 'compiled" form, and the results of the above experiment demonstrate this. These abstractions provide a domain model to the network, which decomposes the learning space into smaller and simpler spaces, and guides the network in navigating these spaces. In fact, it is these abstractions that form the *computational theory* [25] of the given information processing task.

Computational Complexity of Learning

So far we have analyzed only the connectionist *methods* for learning, specifically, the method of back propagation of corrective feedback. We now turn our attention to the learning *task* addressed by these methods including the back propagation method. Specifically, we show that the learning task addressed by connectionist methods is computationally intractable. While the computational intractability of learning by connectionist methods has been demonstrated earlier (e.g., [18, 1]), the following proof is based on the demonstration [23] that any learning mechanism employed by neural networks can also be used to solve the conjunctive-normal form[13] (CNF) satisfiability problem which is known to be *NP-complete* [8].[14]

A Proof for the Intractability of Learning

To illustrate the proof, consider the hypothetical network shown in Figure 7. The network contains fixed and plastic weights such that the network realizes the logical functions associated with evaluation of a given CNF expression and the selection of a set of satisfying variable assignments. The network has a single input and a single output and four layers of units, where each unit performs a linear threshold operation on its net input. (The numbers within circles of Figure 7 represent the bias of the processing unit, while numbers adjacent to the arrows signify the strength of the connection.) The fourth layer in the network is a single unit performing conjunction (logical AND) of the third layer's output. The third layer implements the individual sets of disjunctions (logical OR's) used in the given CNF expression. The second layer provides negated values for the disjunctive layer. These three levels, consisting of fixed weights determined by a given CNF expression, perform the evaluation of the CNF within the network. The first layer, with plastic weights, determines the truth assignments for the variables of the boolean expression according to the weights on that layer.

Insert Figure 7 about here.

Clearly, this hypothetical network directly implements the CNF expression when an active input is applied to it. CNF-satisfiability, therefore, is equivalent to finding a set of weights which generate an active output to this input. Since CNF-satisfiability is known to be NP-complete, it follows that the problem of learning in linear threshold networks is also NP-complete.[15]Since an arbitrary network which uses a sigmoid activation function can be simulated by a network of linear threshold units, the above proof also implies that learning task is computationally intractable for learning schemes that use sigmoid functions as well. Thus, *the learning task that connectionist methods seek to solve is computationally intractable.*

Concluding Discussion

We suggested earlier that an information processing system capable of learning must have several interrelated abilities.

> **Representational Adequacy.** It must be able to represent what it knows and also what it is to learn.
>
> **Credit Assignment** It must be able to identify its structural components which are responsible for incorrect performance for a specific exemplar of a given task.
>
> **Structure Modification.** It must be able to change its structure so that it can correctly perform the given task for the specific exemplar.
>
> **Generalization.** It must be able to generalize (and specialize) what it learns to exemplar-independent abstractions that are needed to perform a given task.
>
> **Computational Complexity.** It must be computationally efficient so that the learning can occur in a reasonable amount of time.

However, the experiment on learning to compute the XOR function suggests that the computational efficiency of learning in a PDP network depends on the choice of initial weights of connections in the network. Similarly, the experiment on learning to play *Tic-Tac-Toe* suggests that the content of what is learned in a PDP network depends on the choice of initial abstractions stored in the network. It also suggests that these abstractions are a major source of power for learning in PDP networks. In addition, we have shown that the learning task that the connectionist methods address is NP-complete. It follows that the method of back propagation of corrective feedback is quite limited relative to the learning task it seeks to solve. In most successful PDP networks built so far, the system designers first identified the abstractions and weights needed for efficient learning, and then explicitly represented the needed abstractions in the network and carefully set the initial weights close to the needed weights.

Limitations of Connectionist Methods

The limitations of connectionist methods for learning described here raise two interrelated issues: (i) what are the causes of these limitations, and (ii) what are the potential solutions for them. There appear to be at least three interrelated causes for the limitations of connectionist methods such as back propagation of corrective feedback. First, PDP networks are poor in their capacity for representing knowledge. For instance, McCarthy [26] has pointed to the "unary fixation" of PDP networks. These networks, he has argued, cannot represent higher-order relations. This representational poverty leads to an incapacity for generalization to higher-order relations since a network can learn only what it can represent. Although McCarthy's critique has been limited to feed-forward, hidden layer networks such as PDP networks, neural networks in general appear to offer relatively few representational primitives.

Second, connectionist methods for learning are "weak methods" in that they (by themselves) make relatively little use of domain knowledge. In the method of back propagation, for instance, the structural components responsible for incorrect system performance are "identified" by back propagation of corrective feedback to individual connections in the network, and the system structure is "modified" by adjusting the connection weights in the direction of steepest gradient descent in the error space. These two ideas are captured in the generalized delta rule. This has led to some claims that the credit assignment problem of learning has been solved (e.g., [40]). However, the generalized delta rule is only a special, recursive form of the method of hill climbing [35, 30]). The problems with the use of weak methods such as hill climbing for navigation of complex search spaces which contain local minima are well known (e.g., [31]). Connectionist methods are most successful, we have shown above, when the domain concepts are explicitly represented in the network.

Third, the connectionist methods for learning do not reflect the structure of the tasks they address. For instance, Rumelhart *et al* use the same method of back propagation of corrective feedback for the very different tasks of computing XOR and playing *Tic-Tac-Toe*. The differences between the learning schemes for these two tasks lie in the network configurations and the domain abstractions that the system designer explicitly represented in the networks. The method for learning (back propagation), however, is task-neutral. Such task-neutral general-purpose methods do not make use of the constraints that characterize the tasks and provide decompositions of the learning space into smaller and simpler spaces that can be navigated more efficiently and effectively.

Potential Solutions

The issue of representational adequacy of neural networks in general and PDP networks in particular has recently attracted much attention in research on connectionist methods for learning. Proposals for distributed representations [19] and recursive representations [29, 34] appear to hold considerable promise for at least partially addressing this problem. These representational schemes, however, do not by themselves provide any clue to how the task constraints and the domain concepts can be used to guide the process of learning.

Goel [14] has proposed the development of *task-specific* connectionist methods as a methodology for exploring the use of task constraints and domain concepts for problem solving and learning. The tasks and the corresponding methods can serve as high-level *building blocks* in constructing neural architectures for solving complex problems. Such task-specific methods have previously been investigated in symbolic, algorithmic approaches to problem solving (e.g., [4, 7, 28]). Chandrasekaran [4] in particular has advocated the use of such "elementary" task-specific methods as building blocks in the construction of knowledge-based systems for solving complex problems. Within the framework of neural networks, the proposal for task-specific connectionist methods is similar to the proposal for "structured" neural networks of Feldman and others [10]. The main difference between the two lies in that the connectionist "structures" in our proposal are functional in nature. This functional character of connectionist methods in our proposal is closer to Touretzky's proposal [42] for developing connectionist methods for tasks such as *pattern substitution*.

A detailed discussion of task-specific connectionist methods is beyond the scope of this paper. Briefly, the development of a neural architecture for solving a given complex task by task-specific connectionist methods involves several steps, including (i) decomposition of the task into a set of smaller and simpler subtasks, (ii) identification of high-level abstractions useful for solving the various subtasks, (iii) representation of these abstractions in a set of neural networks corresponding to the set of subtasks, (iv) identification of task-specific connectionist methods for learning the use of these abstractions to solve the various subtasks, and (v) the composition of individual neural networks into a neural architecture. We [15] are presently exploring this methodology for construction of neural architectures for solving the complex task of *abduction*. Abduction is inference to the best explanation for a given set of data. Recently, several neural architectures for abductive inference have been proposed [32, 41], including our own proposal [17]. Our proposal is based on a decomposition of the abduction task into the subtasks of *data classification* and *hypothesis synthesis*. The task of data classification maps the given set of data into belief values for a set of elementary hypotheses stored in memory. The task of hypothesis synthesis maps the elementary explanatory hypotheses with high belief values into a composite explanation for the data set. The classification task can be solved by a variety of connectionist methods [36]. The synthesis task can be viewed as a constrained-optimization problem [17] and solved on a constrained-optimization network [22]. The tasks of data classification and hypothesis synthesis and the connectionist methods corresponding to them provide high-level building blocks for construction of a neural architecture for solving the more complex task of abduction. Many of the hard issues in using task-specific connectionist methods have yet to be solved. For example, how to recursively decompose a complex task into smaller and simpler subtasks, and how to compose neural networks for solving the subtasks into a neural architecture for solving the given task. Nevertheless, we believe that this approach holds considerable promise for constructing neural architectures for solving complex problems.

Acknowledgments

This research was conducted when AKG, along with JFK, was with the Laboratory for Artificial Intelligence Research at the Ohio State University. There, this work benefited from scholarly contributions by Dean Allemang. It also benefited from discussions with B. Chandrasekaran and Jordan Pollack. In addition, this paper has benefited from comments on previous versions by Tom Bylander, Robert McCurley, and anonymous reviewers. As usual, the authors are responsible for any errors.

References

[1] A. Blum and R. Rivest, Training a 3-node Neural Network is NP-Complete. *Proceedings of IEEE Conference on Neural Information Processing Systems*, Denver, Colorado, 1988.

[2] J. Carbonell, R. Michalski, and T. Mitchell, An Overview of Machine Learning. In *Machine Learning: An Artificial Intelligence Approach*, J. Carbonell, R. Michalski, and T. Mitchell (editors). Palo Alto, California: Tioga Publications, 1983.

[3] G. Carpenter and S. Grossberg, A Massively Parallel Architecture for a Self-Organizing Neural Pattern Recognition Machine. *Computer Vision, Graphics, and Image Processing*, vol. 37, pp. 54-115, 1987.

[4] B. Chandrasekaran, Generic Tasks as Building Blocks for Knowledge-Based Systems: The Diagnosis and Routine Design Examples. *Knowledge Engineering Review*, vol. 3, no. 3, pp.183-219, 1988.

[5] B. Chandrasekaran, A. Goel, and D. Allemang, Connectionism and Information Processing Abstractions: The Message Still Counts More Than the Medium. *AI Magazine*, vol. 9, no. 4, pp. 24-34, Winter 1988.

[6] B. Chandrasekaran and A. Goel, From Numbers to Symbols to Knowledge Structures: Artificial Intelligence Perspectives on the Classification Task. *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 18, no. 3, pp. 415-424, May/June 1988.

[7] W. Clancey, Heuristic Classification. *Artificial Intelligence*, vol. 27, no. 3, pp. 289-350, 1985.

[8] S. Cook, The Complexity of Theorem-Proving Procedures. *Proceedings of the Third ACM Symposium on Theory of Computing*, pp. 151-158, 1971.

[9] J. A. Feldman and D. H. Ballard, Connectionist Models and their properties. *Cognitive Science*, vol. 6, pp. 205-254, 1982.

[10] J. Feldman, M. Fanty, N. Goddard and K. Lynne, Computing with Structured Connectionist Networks. *Communications of ACM*, February 1988, pp. 170-187.

[11] S. Franklin and M. Garzon, Neural Network Implementation of Turing Machines. Poster presentation at the Second IEEE International Conference on Neural Networks, San Diego, California, 1988.

[12] M. Garey and D. Johnson, *Computers and Intractability*. New York, New York: W. H. Freeman and Company, 1979.

[13] A. Goel. A Report on the 1988 AAAI Spring Symposium on Parallel Models of Intelligence. *SIGART*, vol. 106, pp. 28-31, October 1988.

[14] A. Goel, (1990). Abductive Inference on Neural Networks: Back to Basics. To appear in *Neural Network Review*, 1990.

[15] A. Goel and J. Ramanujam. A Neural Architecture for a Class of Abduction Problems. Technical Report, School of Information and Computer Science, Georgia Institute of Technology, 1990.

[16] A. Goel. J. F. Kolen, and D. Allemang, Learning in Connectionist Networks: Has the Credit Assignment Problem Been Solved? *Proceedings of the SIGART Aerospace Applications of Artificial Intelligence Conference*, Dayton, Ohio, Vol. II, pp. 74-80, 1988.

[17] A. Goel, J. Ramanujam, and P. Sadayappan, Towards a Neural Architecture for Abductive Reasoning. In *Proceedings of the Second IEEE International Conference on Neural Networks*, San Diego, California, pp. I:680-688, 1988.

[18] S. Judd, Learning in Networks is Hard. *Journal of Complexity*, vol. 4, pp. 177-192, 1988.

[19] G. Hinton and T. Sejnowski, Learning and Relearning in Boltzman Machines. In *Parallel Distributed Processing: Explorations in The Microstructure of Cognition; Volume 1*, D. Rumelhart, J. McClelland, and the PDP Research Group (editors). Cambridge, MA: MIT Press. A Bradford book, 1986.

[20] J. Hopfield, Neural Networks and Physical Systems with Emergent Collective Computational Abilities. *Proceedings National Academy of Sciences, USA*, vol. 79, pp. 2554-2558, 1982.

[21] K. Hornik, M. Stinchcombe, and H. White, ”Multi-layer Feedforward Networks are Universal Approximators,” *Neural Networks*, vol. 2, pp. 359, 1989.

[22] J. Hopfield and D. Tank, Neural Computation of Decisions in Optimization Problems. *Biological Cybernetics,* vol 52, pp. 141-152, 1985.

[23] J. Kolen, Faster Learning Through a Probabilistic Approximation Algorithm. *Proceedings of the Second IEEE International Conference on Neural Networks*, San Diego, California, pp. I:449-454, 1988.

[24] J. Kolen and J. Pollack, Back Propagation is Sensitive to Initial Conditions. Technical Report, Laboratory for Artificial Intelligence Research, The Ohio State University, 1990. To appear in *The Proceedings of the Twelfth Annual Conference of the Cognitive Science Society*.

[25] D. Marr, *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. San Francisco, CA: Freeman, 1982.

[26] J. McCarthy, Epistemological Challenges for Connectionism. *Behavioral and Brain Sciences*, vol. 11, no. 1, pp. 44, 1988.

[27] W. McCulloch and W. Pitts, A logical Calculus of the Ideas Immanent in Neural Nets, *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115-137, 1943.

[28] J. McDermott, Preliminary Steps Towards a Taxonomy of Problem Solving Methods. *Automating Knowledge Acquisition for Expert Systems*, S. Marcus (editor). Boston, MA: Kluwer, 1988.

[29] R. Miikkulainen and M. Dyer, Forming Global Representations With Extended Backpropagation. *Proceedings of the Second IEEE International Conference on Neural Networks*, San Diego, California, pp. I:285-292 1988.

[30] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*, Expanded Edition; Cambridge, MA: MIT Press, 1988.

[31] N. J. Nillson, *Principles of Artificial Intelligence*. Palo Alto, CA: Tioga, 1980.

[32] Y. Peng and J. Reggia, A Connectionist Model for Diagnostic Problem Solving. *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 19, pp. 285-298, 1989.

[33] J. Pollack, On Connectionist Models of Natural Language Processing. Ph.D. Thesis, Computer Science Department, University of Illinois, Urbana, 1987.

[34] J. Pollack, Recursive Auto-Associative Memories: Devising Compositional Distributed Representations. *Proceedings Tenth Annual Conference of the Cognitive Science Society*, Montreal, Canada, August 1988.

[35] D. Rumelhart, G. Hinton, and R. Williams. Learning Internal Representation by Error Propagation. In *Parallel Distributed Processing: Explorations in The Microstructure of Cognition; Volume 1*, D. Rumelhart, J. McClelland, and the PDP Research Group (editors). Cambridge, MA: MIT Press. A Bradford book. 1986.

[36] D. Rumelhart, J. McClelland, and the PDP Research Group (editors), *Parallel Distributed Processing: Explorations in The Microstructure of Cognition; Volume 1*, Cambridge, MA: MIT Press. A Bradford book, 1986.

[37] D. Rumelhart, P. Smolensky, J. McClelland, and G. Hinton, Schemata and Sequential Thought Processes in PDP Models. In *Parallel Distributed Processing: Explorations in The Microstructure of Cognition; Volume 2*,D. Rumelhart, J. McClelland, and the PDP Research Group (editors); Cambridge, MA: MIT Press, A Bradford book, 1986.

[38] A. Samuel, Some Studies in Machine Learning Using the Game of Chequers II - Recent Progress. *IBM Journal of Research an Development*, vol. 11, no. 6, pp. 601-617, 1967.

[39] P. Smolensky, Information Processing in Dynamical Systems: Foundations of Harmony Theory. In *Parallel Distributed Processing: Explorations in The Microstructure of Cognition; Volume 1*,D. Rumelhart, J. McClelland, and the PDP Research Group (editors); Cambridge, MA: MIT Press, A Bradford book, 1986.

[40] P. Smolensky, On the Proper Treatment of Connectionism. *Behavioral and Brain Sciences*, vol. 11, no. 2, pp. 1-27, March 1988.

[41] P. Thagard, Explanatory Coherence. *Behavioral and Brain Sciences*, vol. 12, no. 3, pp. 435-467, September 1989.

[42] D. Touretzky, Beyond Associative Memory: Connectionists Must Search for Other Cognitive Primitives. *Proc. AAAI Symposium on Parallel Models of Intelligence*, Stanford University, Palo Alto, California, pp. 231-236, March 1988.

Footnotes

1. Manuscript received .... This paper is an expanded and revised version of work presented at the Fourth Annual Aerospace Applications of Artificial Intelligence Conference, Dayton, Ohio, October 1988, and at the Second IEEE International Conference on Neural Networks, San Diego, California, July 1988. This research has been supported by the Office of Naval Research, grant N00014-89-J-1200.

2. J. F. Kolen is with the Laboratory for Artificial Intelligence Research, Department of Computer and Information Science, The Ohio State University, Columbus, Ohio 43210, USA.

3. A. K. Goel is with the Artificial Intelligence and Cognitive Science Group, College of Computing, Georgia Institute of Technology, Atlanta, Georgia 30332-0280, USA.

4. The term "information content" here refers to the types of information a system acquires as a result of learning. For example, upon training, one network may learn a set of "novel" abstractions while another network may acquire a set of situation-specific responses. Obviously, the information content learned by the two networks is quite different. In general, one would prefer a network that learns novel abstractions rather than merely memorizing situation-specific responses.

5. The term "learning scheme"' here refers to the combination of a specific type of neural network, e.g., a PDP network and a specific connectionist method for learning, e.g., the back propagation method.

6. Strictly, the design of Turing-Universal neural networks requires constructs such as *pi* units, cycles, etc. that are not available in feed-forward, hidden layer, PDP networks. Some researchers, however, have shown feed-forward networks can be universal approximators [21].

7. For simplicity, we assume here that each weight configuration specifies a unique function. Some functions, however, may be specified by more than one weight configurations.

8. The term "epoch" refers to the entire exemplar set that is presented to the network. In the XOR case, an epoch consisted of four exemplars.

9. *Tic-tac-toe* is a 3×3 board game in which two players take turns placing distinguishable marks on the board. The player who first succeeds in capturing an entire row, column or diagonal wins the game. Against an experienced opponent, typically the best a player can do is to achieve a stalemate.

10. The network actually contained an extra input unit for "symmetry breaking", the need for which arises when the board is empty and the network has to make the first move [16].

11. If the network suggested no moves or only illegal moves, then the program randomly selected a legal move for the network.

12. We have not provided here any details about the parameters of the network such as the learning rates, the ranges of the connection weights, the unit biases, etc., since the goal of this experiment was not to measure the computational efficiency of learning, but to investigate the content of what is learned.

13. A conjunctive-normal form expression is a boolean expression comprised of the disjunction of conjunctions of boolean variables. In symbolic form, $\bigwedge_{i=1}^{n} C_i$, where $C_i = \bigvee_{j=1}^{m_i} x_j$, where x_j is a boolean variable of the negation of a boolean variable. For example $(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee x_4 \vee \neg x_5) \wedge (x_1 \vee \neg x_4)$ is a CNF expression [12].

14. Nondeterministic polynomial time (NP) is the class of problems where guessing a solution and verifying its correctness can be performed in time bounded by some polynomial function of the input size. Nondeterministic polynomial time complete (NP-complete) is the subset of NP problems such that all problems in NP can be reduced, in polynomial time, to NP-complete problems. Although not proven, it is generally believed that no polynomial time algorithms exist for NP-complete problems since the number of possible solutions grows exponentially with the size of the input [12].

15. This reduction actually demonstrates that learning is NP-hard. The NP-completeness proof is finished by constructing a trivial non-deterministic machine that can "guess" a set of weights and verify that the network computes the desired function in polynomial time.

Biographies

John F. Kolen received his B.A. degree in computer science from the University of California at San Diego in 1987 and his M.S. degree in computer and information sciences from The Ohio State University, Columbus, in 1988. He is currently a research assistant and doctoral candidate at The Ohio State University. His current research interests include neural networks, computational geometry, and discovering ways to exploit the computational capabilities of nonlinear dynamical systems.

Ashok K. Goel received his M.S. in physics in 1980, M.S. in Computer and Information Science in 1982, and Ph.D. in computer and information science in 1989, all from The Ohio State University, Columbus, Ohio. Since 1989, he has been an Assistant Professor with the School of Information and Computer Science, Georgia Institute of Technology, Atlanta, Georgia.

Dr. Goel has published over twenty papers on Computer Science, Cognitive Science, Artificial Intelligence, and Neural Networks, including several refereed journal papers., He is a member of Association for Computing Machinery, Cognitive Science Society, American Association for Artificial Intelligence, American Physical Society, and Institute of Electrical and Electronic Engineers. Recently, he was appointed the chair of the Artificial Intelligence Technical Committee of the Systems, Man, and Cybernetics Society.

Figure and Table Captions

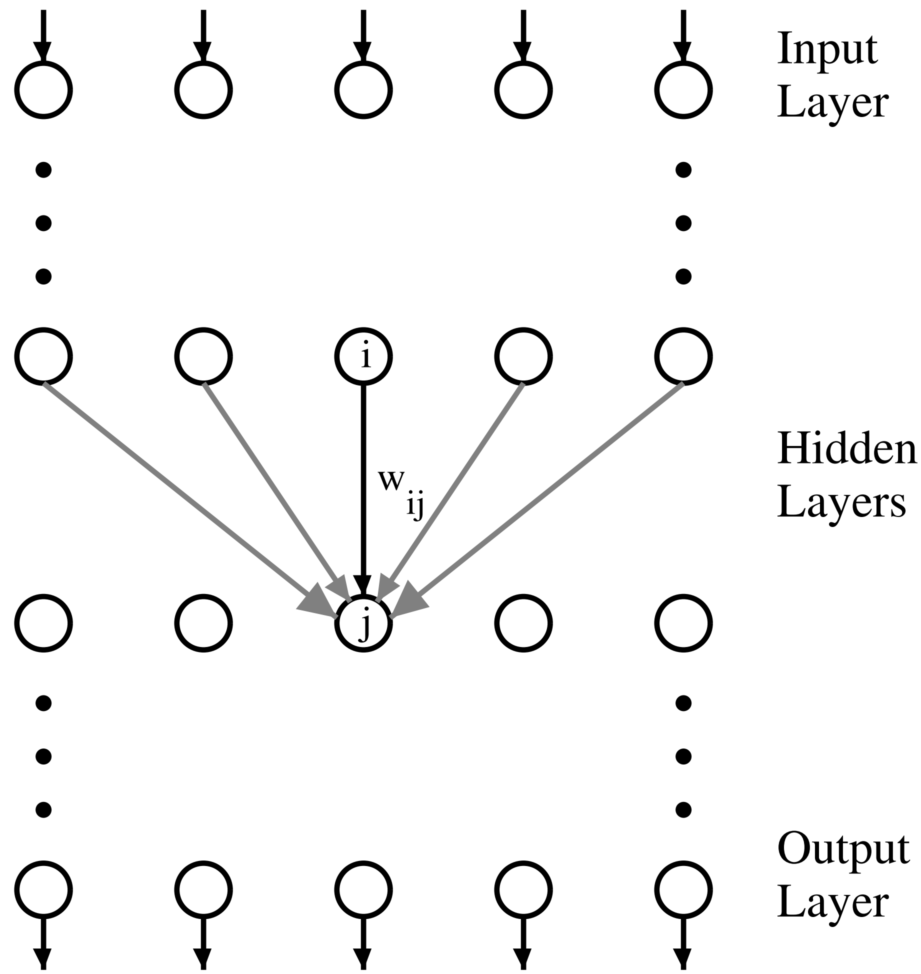

Figure 1: Connection between two processing units

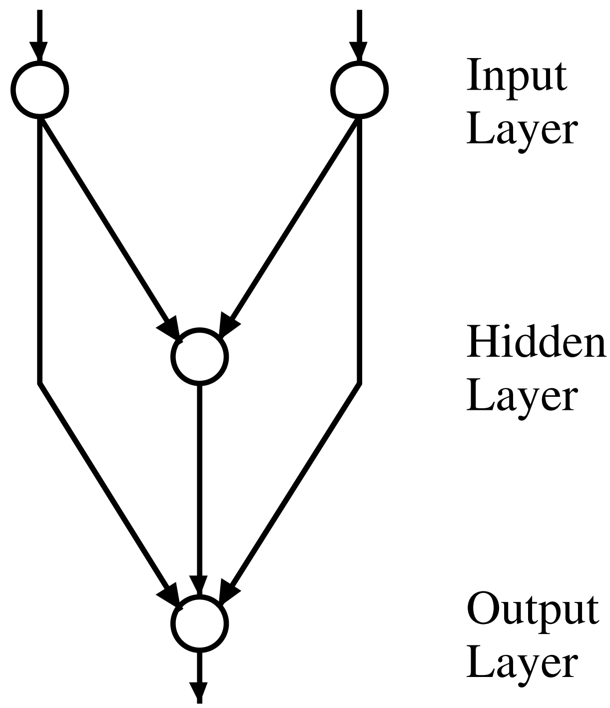

Figure 2: XOR Network

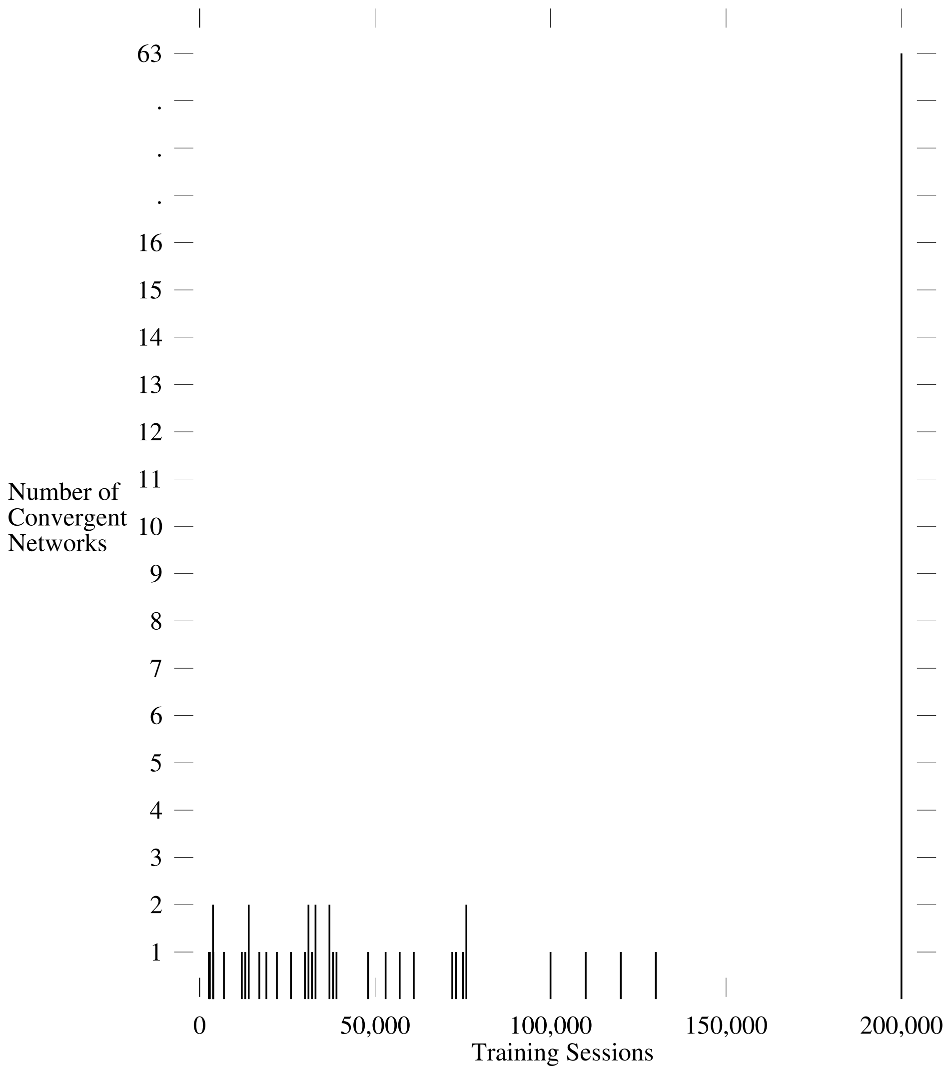

Figure 3: Histogram of XOR Convergence Times

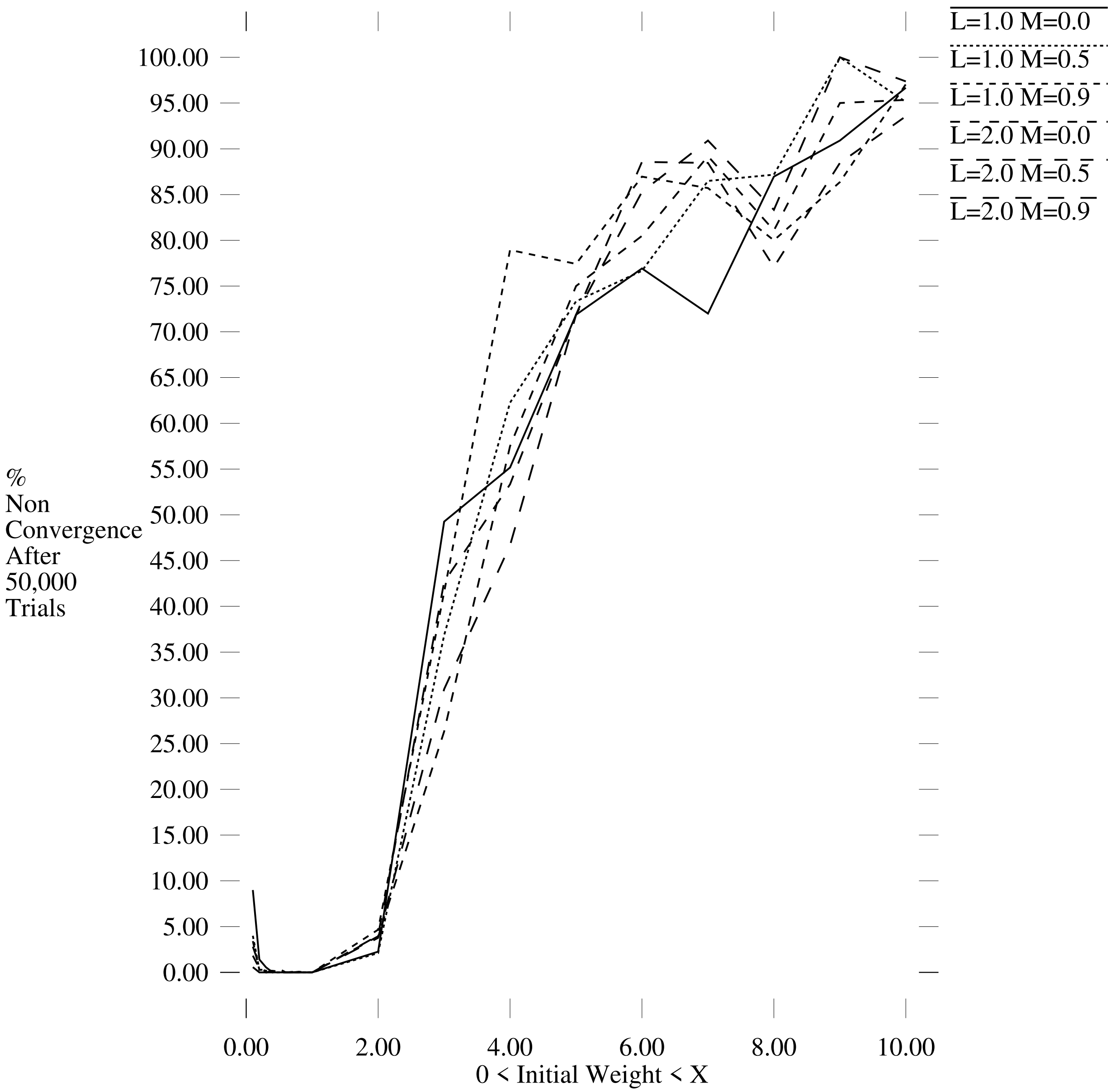

Figure 4: XOR Non-Convergence Percentage versus Initial Weight Range (0.0 to x)

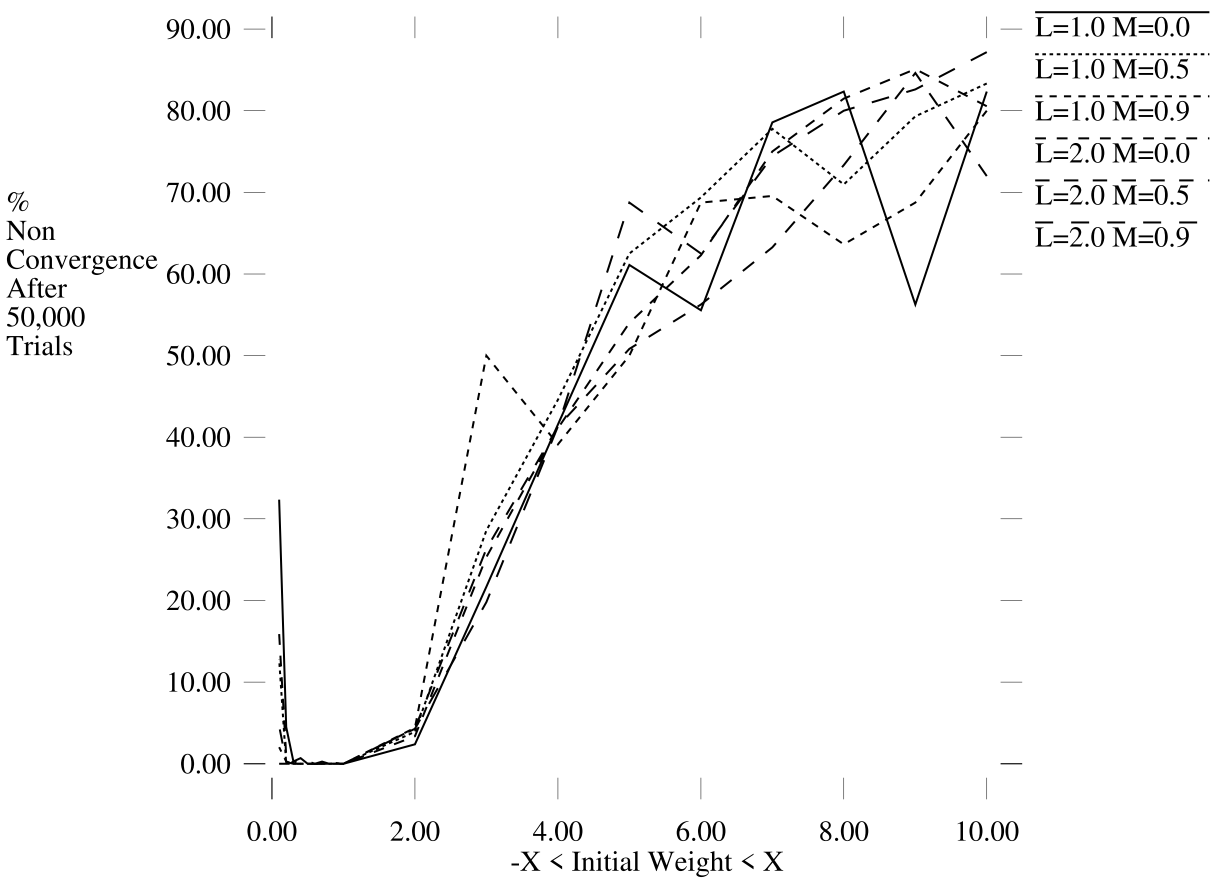

Figure 5: XOR Non-Convergence Percentage versus Initial Weight Range ($-x$ to x)

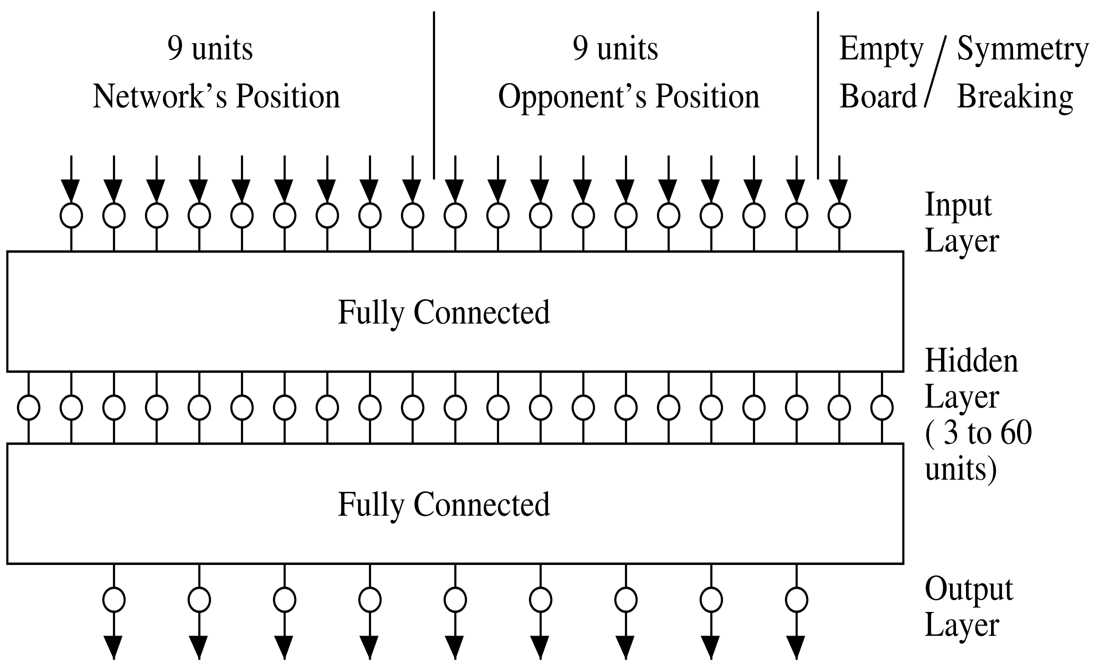

Figure 6: Tic-tac-toe Network

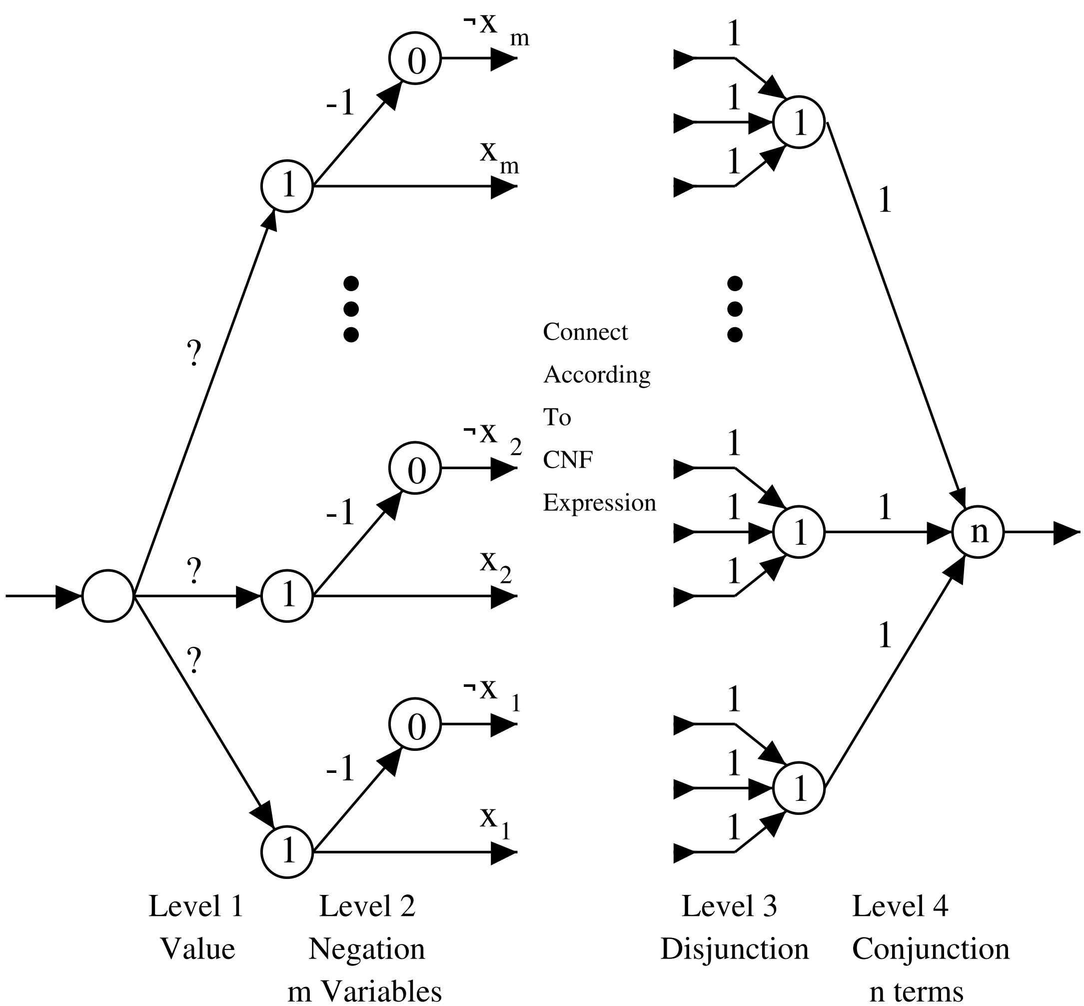

Figure 7: The Generic Conjunctive-Normal Form Expression Network

	Symbolic Algorithm Wins (out of 1000)	Neural Network Wins (out of 1000)	Symbolic Algorithm/ Neural Network Draws (out of 1000)
Match 1	33	0	967
Match 2	54	1	945
Match 3	56	0	944
Match 4	57	0	943
Match 5	63	0	937
Match 6	85	0	915
Match 7	277	0	723
Match 8	298	0	702
Match 9	298	0	702
Match 10	345	0	655

Table 1: Tic-Tac-Toe Results