Training Second-Order Recurrent Neural Networks using Hints *

Christian W. Omlin
Computer Science Department
Rensselaer Polytechnic Institute
Troy, N.Y. 12180 USA

C. Lee Giles
NEC Research Institute
4 Independence Way
Princeton, N.J. 08540 USA

Abstract

We investigate a method for inserting rules into discrete-time second-order recurrent neural networks which are trained to recognize regular languages. The rules defining regular languages can be expressed in the form of transitions in the corresponding deterministic finite-state automaton. Inserting these rules as hints into networks with second-order connections is straightforward. Our simulation results show that even weak hints seem to improve the convergence time by an order of magnitude.

1 MOTIVATION

Often, we have a priori knowledge about a learning task and we wish to make effective use of this knowledge. We will discuss a method for inserting prior knowledge into recurrent neural networks. For an initial testbed, we will train networks to recognize regular languages, thus behaving like deterministic finite-state automata ([Giles 91], [Giles 92]). We show that, as might be expected, the convergence time is significantly decreased by placing partial knowledge about a deterministic finite-state automaton (DFA), particularly about the transitions emerging from individual states and about which states are accepting states, into a network. We interpret this type of a priori knowledge as *hints* about when to make a change (change state) as a function of the input symbol.

[Abu-Mostafa 90] has pointed out that using partial information about the implementation of a function f using input-output examples may be valuable to the learning process in two ways: It may reduce the number of functions that are candidates for f and it may reduce the number of steps needed to find the implementation. In related work, [Al-Mashouq 91] trains feed-forward networks using hints thus improving the learning time and the generalization performance. [Towell 90] shows how approximate rules about some domain are translated into a feed-forward network which is used as the basis for constructive induction. [Pratt 92] uses feed-forward networks that were previously trained for a learning task to improve the learning time for a new task. [Berenji 91] uses reinforcement learning to refine reasoning-based controllers. [Georgiou 92] extracts information from a data set prior to training and inserts this knowledge into feed-forward networks to improve convergence time for a classification problem. [Suddarth 91] introduces a measure for guiding learning processes through constraints for faster learning and better generalization. [Giles 87] and [Perantonis 92] encode geometric invariances of two-dimensional objects into higher-order networks thus improving training and generalization performance.

We show how hints can be inserted into second-order completely recurrent neural networks by setting some of the weights to larger values rather than starting with small random initial values for all the weights. The training algorithm is expected to use these preset weights to find a solution in weight space thus improving the convergence time. We will also investigate whether hints can improve the generalization performance. We test our method using the simple DFA that accepts the regular language 2-parity.

2 RECURRENT NETWORK

2.1 Architecture

To learn grammars, we use a second-order recurrent neural network ([Lee 86], [Giles 91], [Pollack 91], [Giles 92], [Watrous 92]). The network architecture is illustrated in figure 1. This net has N recurrent hidden neurons labeled S_j; L special, nonrecurrent input neurons labeled I_k; and $N^2 \times L$ real-valued weights labeled W_{ijk}. We refer to the values of the hidden

*To appear in D. Sleeman and P. Edwards (Eds), *Machine Learning: Proceedings of the Ninth International Conference (ML92)*, Morgan Kaufmann, San Mateo, CA

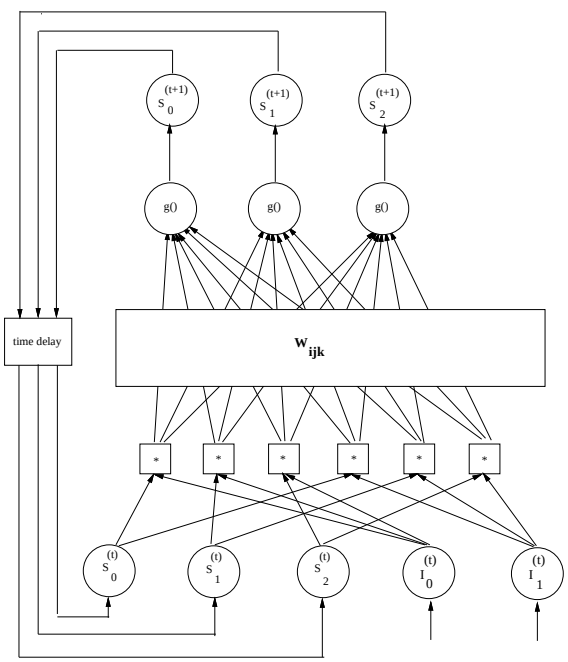


Figure 1: A second-order, single layer recurrent neural network. $S_i^{(t)}$ and $I_k^{(t)}$ represent the values of the i^{th} state and k^{th} input neuron, respectively, at time t. The block marked '*' represent the operation $W_{ijk} \times S_j^{(t)} \times I_k^{(t)}$. $g()$ is the (sigmoidal) transfer function.

neurons collectively as a *state vector* **S** in the finite N-dimensional space $[0,1]^N$. This recurrent network accepts a time-ordered sequence of inputs and evolves with dynamics defined by the following equations:

$$S_i^{(t+1)} = g(\Xi_i), \qquad \Xi_i \equiv \sum_{j,k} W_{ijk} S_j^{(t)} I_k^{(t)},$$

where g is a sigmoid discriminant function. Each input string is encoded into the input neurons one character per discrete time step t. The above equation is then evaluated for each hidden neuron S_i to compute the state vector **S** of the hidden neurons at the next time step $t+1$. With unary encoding the neural network is constructed with one input neuron for each character in the alphabet of the relevant language.

2.2 Training Algorithm

For any training procedure, one must consider the error criteria, the method by which errors change the learning process, and the presentation of the training samples. The error function E_0 is defined by selecting a special "response" neuron S_0 which is either on ($S_0 > 1-\epsilon$) if an input string is accepted, or off ($S_0 < \epsilon$) if rejected, where ϵ is the response tolerance of the response neuron. We define two error cases: (1) the network fails to reject a negative string $\mathcal{I}_-$ (*i.e.* $S_0 > \epsilon$); (2) the network fails to accept a positive string $\mathcal{I}_+$ (*i.e.* $S_0 < 1-\epsilon$). For these studies, the acceptance or rejection of an input string is determined only at the end of the presentation of *each* string. The error function is defined as:

$$E_0 = \frac{1}{2}(\tau_0 - S_0^{(f)})^2,$$

where τ_0 is the desired or *target* response value for the response neuron S_0. The target response is defined as $\tau_0 = 0.8$ for positive examples and $\tau_0 = 0.2$ for negative. The notation $S_0^{(f)}$ indicates the *final* value of S_0, *i.e.*, after the final input symbol.

The training is an on-line (real-time) algorithm that updates the weights at the end of each sample string presentation (assuming there is an error $E_0 > .5\epsilon^2$) with a gradient-descent weight update rule:

$$\Delta W_{lmn} = -\alpha \frac{\partial E_0}{\partial W_{lmn}} = \alpha(\tau_0 - S_0^{(f)}) \cdot \frac{\partial S_0^{(f)}}{\partial W_{lmn}},$$

where α is the learning rate. We also add a momentum term, an additive update to ΔW_{lmn}, which is η, the momentum, times the previous ΔW_{lmn}. To determine ΔW_{lmn}, the $\partial S_i^{(f)}/\partial W_{lmn}$ must be evaluated. From the recursive network state equation, we see that

$$\frac{\partial S_i^{(f)}}{\partial W_{lmn}} = g'(\Xi_i) \cdot \left[\delta_{il} S_m^{(f-1)} I_n^{(f-1)} + \sum_{j,k} W_{ijk} I_k^{(f-1)} \frac{\partial S_j^{(f-1)}}{\partial W_{lmn}} \right],$$

where g' is the derivative of the discriminant function. In general, f and $f-1$ can be replaced by any t and $t-1$, respectively. These partial derivative terms are calculated iteratively as the equation suggests, with one iteration per input symbol. This on-line learning rule is a second order form of the RTRL net of [Williams 89]. The initial terms $\partial S_i^{(0)}/\partial W_{lmn}$ are set to zero. After the choice of the initial weight values, the $\partial S_i^{(t)}/\partial W_{lmn}$ can be evaluated *in real time* as each input symbol $I_k^{(t)}$ enters the network. In this way, the error term is forward-propagated and accumulated at each time step t. However, each update of $\partial S_i^{(t)}/\partial W_{lmn}$ requires $O(N^4 \times L^2)$ terms. For $N >> L$, this update is $O(N^4)$ which is the same as a linear network. This could seriously prohibit the size of the recurrent net if it remains fully interconnected.

3 INSERTING RULES

Given a set of positive and negative example strings generated by a DFA $(\Sigma, Q, R, F, \delta)$ with alphabet $\Sigma = \{a_1, \ldots, a_k\}$, states $Q = \{s_1, \ldots, s_M\}$, a start state R, a set $F \epsilon Q$ of accepting states and state transitions $\delta : Q \times \Sigma \rightarrow Q$, we insert rules for *known* transitions (defined as hints) by programming some of the initial

weights of a second-order recurrent network state neurons. Although the number of states in a DFA is not known a priori, we assume that $N > M$ and that the network is large enough to learn the unknown regular grammar.

Before we insert rules into a network, we initialize all weights to small random values in the interval $[a, b]$. Our method of inserting rules into a network to be trained follows directly from the similarity of state transitions in a DFA and the dynamics of a recurrent neural network.

Consider a known transition $\delta(s_j, a_k) = s_i$. We identify DFA states s_j and s_i with state neurons S_j and S_i, respectively, and we further postulate that state neuron S_i has a high output close to 1 and that state neuron S_j has a small output close to 0 after the symbol a_k has entered the network via input neuron I_k. This can be accomplished as follows: Setting W_{ijk} to a large positive value will ensure that S_i^{t+1} will be high and setting W_{jjk} to a large negative value will guarantee that the output S_j^{t+1} will be low. The assumption is that the total contribution of the weighted output of all other state neurons can be neglected and that each state neuron be assigned to only one known state in the DFA.

If it is known whether or not DFA state s_i is an accepting or non-accepting state, then we can bias the output S_0^{t+1}. If state s_i is an accepting state, then we program the weight W_{0jk} to a large positive value; otherwise, we initialize the weight W_{0jk} to a large negative value. If it is unknown whether or not state s_i is an accepting state, then we do not modify the weight W_{0jk}.

The problem remains to determine values for the weights to be programmed. For reasons of simplicity and in order to make our approach accessible to analytic methods, we choose large values to be $+H$ and small values to be $-H$ depending on the weight to be programmed where H is an arbitrary rational number. We will refer to H as the *strength* of a hint.

We assume that the DFA generated the example strings starting in its initial state. Therefore, we can arbitrarily select the output of one of the state neurons to be 1 and set the output of all other state neurons initially to zero. After all known transitions have been inserted into the network by programming the weights according to the above scheme, we train the network on some given training set. All weights including the ones that have been programmed are adaptable.

We can program the change of states and thus (partially) define the network state vector **S** in second-order networks because the input a state neuron S_i receives depends on the current state of all other neurons S_j and the current input symbol. Programming

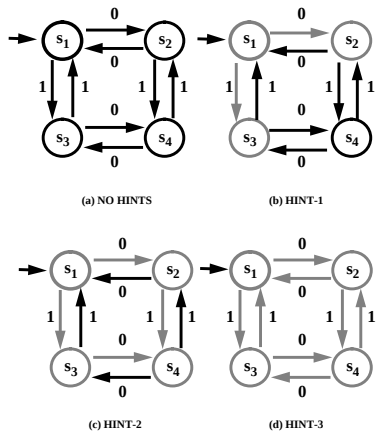


Figure 2: The deterministic finite-state automaton which recognizes strings which have an even number of 0's and 1's. HINTS are encoded as the gray production rules and nodes. For HINT-3 all productions are encoded. The heavy-circled node is a final state and the arrow indicates the start state (in this case the same state).

the weights W_{ijk} *jointly* influences the contributions of state neurons S_j and input neurons I_k. In first-order networks, state and input neurons are independent of each other and have different weights associated with them. Hints can therefore not be inserted into a network by programming some of the weights in a straightforward manner. It remains to be seen whether hints can be inserted into first-order recurrent neural networks in a way similar to our method for second-order networks. Obviously, our algorithm defines a sparse distribution of DFA states among a network's state neurons through the *orthogonal* state encoding. It is possible to extend the algorithm such that fewer state neurons are necessary. However, applying the encoding scheme to smaller networks can lead to conflicts in terms of the values for the programmed weights. The resolution of these conflicts is not obvious and is beyond the scope of this paper.

The hint insertion method discussed above is not unique. There are potentially many other approaches ([Maclin 92]).

4 LEARNING WITH HINTS

4.1 Hints

Consider strings over the alphabet {0, 1}. A string is a member of the language 2-parity if the number of both 0's and 1's is even. The ideal, minimal DFA which accepts strings in the language is shown in figure 2a.

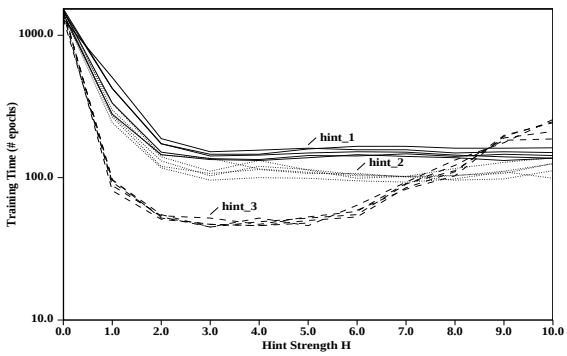
We inserted hints according to figures 2b, 2c and 2d. Hint 1 corresponds to the knowledge that the initial state is an accepting state and that the transitions

Table 1: Programming Weights for Hint 3

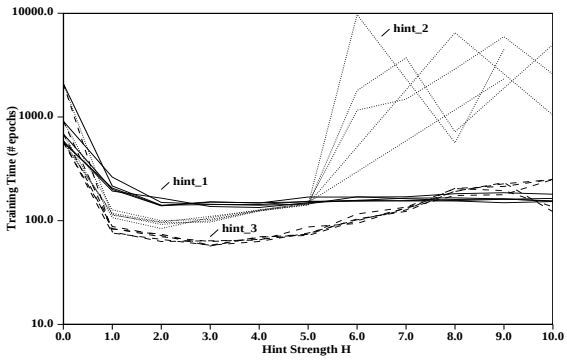
W_{0jk}	W_{1jk}	W_{2jk}	W_{3jk}	W_{4jk}
0.013	0.004	-0.038	-0.006	-0.024
-0.057	-0.045	0.007	-0.094	0.032
0.077	0.019	0.016	0.000	-0.059
$-H$	$-H$	H	-0.087	-0.008
$-H$	$-H$	0.011	H	0.070
-0.099	0.099	0.051	-0.095	0.052
H	H	$-H$	0.012	-0.056
$-H$	0.094	$-H$	0.027	H
-0.093	0.083	0.043	0.098	-0.043
$-H$	-0.08	0.046	$-H$	H
H	H	0.094	$-H$	0.056
-0.030	-0.02	0.045	0.070	-0.049
$-H$	0.032	0.015	H	$-H$
$-H$	0.078	H	-0.029	H
0.077	0.074	0.046	-0.005	0.010

from this initial state on input symbols '0' and '1' lead to two distinct, non-accepting states of the DFA (figure 2b). A stronger hint (hint 2) is shown in figure 2c. Compared to hint 1, our a priori knowledge has increased in that we know that the transitions from the start state on input strings '01' and '10' lead to the same non-accepting state. Hint 3 represents our knowledge of the complete DFA (figure 2d).

We used a training set consisting of 1,000 alternating positive and negative example strings in alphabetical order to train networks with 8 and 5 state neurons. Since we assumed that state neuron S_1 corresponds to the start state s_1 in the DFA, we initialized the output of state neuron S_1^0 to 1 and the output of all other state neurons to 0. In order to assess the influence of varying the hint strength, we first initialized the weights of several networks to small random values in the interval [-0.1, +0.1], then we determined which weights were to be programmed, we inserted the rules into the networks and we trained each of the networks starting with different values for H. The networks were trained with a learning rate $\alpha = 0.5$, a momentum $\eta = 0.5$ and a response neuron tolerance $\epsilon = 0.2$. The initial weights for a network with 5 state neurons where all the transitions (figure 2d) have been programmed into the network with hint strength H are shown in table 1. The hint strength is the same for all programmed weights ($+H$ or $-H$). Each column shows the weight values W_{ijk} feeding from state neuron S_j and input neuron I_k to state neuron S_i. The indexes j and k run in alphabetical order from 0 to 4 and 0 to 2, respectively. Besides the input neurons for symbols '0' and '1', we also provide an input neuron for an end symbol, indicating the end of a string. Although *all* the transitions of the DFA are programmed into the network, our method for inserting rules chooses *only* 24 of the available 75 weights.



(a) Training Performance of 8-Neuron Networks



(b) Training Performance of 5-Neuron Networks

Figure 3: Convergence time for networks with (a) 8 and (b) 5 state neurons trained to recognize 2-parity for different hints as a function of the hint strength H. Hint 1 represents the smallest amount of a priori knowledge, whereas hint 3 represents the knowledge of the complete DFA.

4.2 TRAINING PERFORMANCE

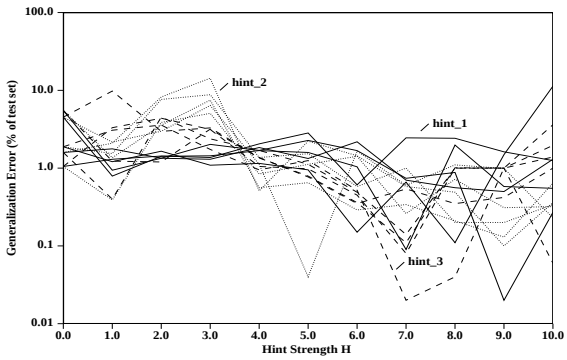
Some representative training times for networks with 8 state neurons as a function of the hint strength are shown in figure 3a on a logarithmic scale. The training times without hints for the 5 experiments shown varied between 1302 and 1532 epochs. Although the initial weights were randomly distributed in the interval [-0.1, +0.1], we show these training times at $H = 0$. We investigated how our algorithm for encoding (partial) knowledge about a DFA affects the amount of training necessary for a network to learn to correctly classify a training data set. The graphs show the training times for the three different hints. We observe that for all hints, the training time is quite insensitive to the initial configuration of the small, random weights. The training times for the strongest hint (hint 3) are smaller than the training times for the other two hints for an appropriate choice of the hint strength. When the hint becomes too strong (H above 7), the training times necessary to train a network with all the information about the DFA increase compared to training

with less a priori knowledge. Our interpretation of this phenomenon is as follows: At each weight update, the gradient descent algorithm chooses the direction of the nearest local minimum, but because the weight values are large, the algorithm overshoots the local minimum during the initial phase of training. As the training proceeds, the momentum term becomes smaller, thus preventing the algorithm from constantly missing the local minimum. This observation illustrates that it is important to find the proper hint strength in order to achieve a significant improvement in convergence time. For weak hints, the training time is not significantly influenced by the hint strength for values of H above 2. The learning speed-up achieved with hint 1 demonstrates that even little prior knowledge can significantly improve convergence time assuming a good hint strength is chosen.

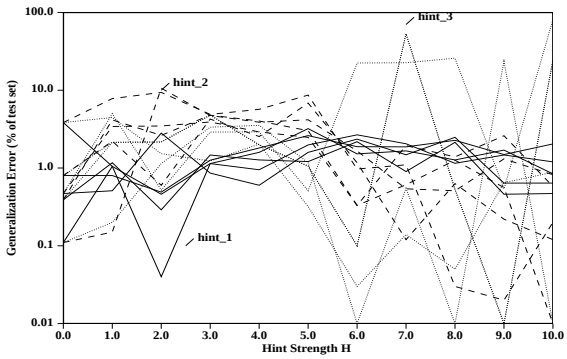
In order to assess the influence of the network size on training time improvements, we trained networks with 5 state neurons. The training times as a function of the hint strength are shown in figure 3b. The convergence time for training without hints varied between 567 and 2099 epochs. If we compare the training times of the smaller networks with the training times for the larger networks, then we observe that for the weakest (hint 1) and the strongest hint (hint 3) the training times as a function of the hint strength show the same general behavior. However, for hint 2, the training times increase significantly for hint strengths greater than 5. In some cases, the training even failed to converge within 10,000 epochs; no training times are shown in the graph for these cases. From these experiments, we conjecture that the training time improvements depend strongly on the particular hint and its strength and that these improvements are fairly independent of the network size and the initial conditions of a network, i.e. the distribution of the small, initial values of the weights.

4.3 GENERALIZATION PERFORMANCE

Besides the effect of hints on the convergence time, we also compared the generalization performance of networks trained with and without hints. We measured the generalization performance of the networks by testing them on all strings of length up to 15 (32,768 strings). The results are shown in figure 4. The graphs in figure 4a show the percentage of errors made by 5 networks with 8 state neurons trained using the hints above as a function of the hint strength H on a logarithmic scale. The performance of networks trained without hints is shown at $H=0$. Clearly, programming some of the initial weights to large values does not necessarily hurt the generalization performance. For some values of the hint strength H the generalization performance even improved. Some of the networks with 5 state neurons failed to learn the training set within 10,000 epochs. The high generalization errors for some



(a) Generalization Performance of 8-Neuron Networks



(b) Generalization Performance of 5-Neuron Networks

Figure 4: Generalization performance of networks with (a) 8 and (b) 5 state neurons on all strings of length up to 15 (32,768 strings) in percentage of misclassified strings. Networks which failed to converge show a very poor generalization performance.

hint values shown in figure 4b reflects this.

We extracted finite-state automata from networks using a clustering heuristic in the n-dimensional output space of the state neurons ([Giles 92]). Some of the minimized automata were identical with the original automaton that generated the strings for an appropriate choice of the hint strength H.

5 CONCLUSIONS

We have demonstrated how partial knowledge about the transitions in the deterministic finite-state automaton (DFA) of some unknown regular grammar can be used to improve the time needed to train networks. Our method uses *second-order weights* and assumes that the size of the network be larger than the number of states in the DFA. Although theoretically possible, it is not always easy to insert rules into first-order networks. We insert rules by programming a small subset of all the weights to some arbitrary *hint strength* ($+H$ or $-H$) rather than setting all weights

to small initial random values. We trained networks of different sizes to recognize the regular language 2-parity. The time necessary to train networks for these simple problems can be improved by an order of magnitude using hints. In many cases the improvement was independent of the value H. The generalization performance did not suffer significantly by using hints. In most cases, the generalization performance even improved. We hypothesize that considerable improvements in convergence time can be achieved by defining an intended orthogonal internal state representation independent of the particular language to be learned. It would be useful to have a heuristic for finding a value of H prior to training which allows fast learning. The optimal hint strength depends on the provided hints and the training set and is less sensitive to the network size and the distribution of the random initial weights. Further work should investigate the insertion of rules into networks without the restriction that the network be larger than the number of states in the unknown DFA while avoiding to insert rules into the network that are inconsistent with the partial knowledge about the DFA. The relationship between the learning time improvement and the generalization performance for networks that are trained using hints remains an open question.

Acknowledgements

We would like to acknowledge useful discussions with D. Chen, H.H. Chen, S. Das, M.W. Goudreau, Y.C. Lee, C.B. Miller, H.T. Siegelman and G.Z. Sun.

References

[Abu-Mostafa 90] Y.S. Abu-Mostafa, Learning from Hints in Neural Networks, Journal of Complexity, Vol. 6, p. 192 (1990).

[Al-Mashouq 91] K.A. Al-Mashouq, I.S. Reed, Including Hints in Training Neural Nets, *Neural Computation* , Vol. 3, No. 4, p. 418, (1991).

[Berenji 91] H.R. Berenji, Refinement of Approximate Reasoning-Based Controllers By Reinforcement Learning, *Proceedings of the Eighth International Machine Learning Workshop*, Evanston, IL, p. 475, (1991).

[Georgiou 92] G.M. Georgiou, C. Koutsougeras, Embedding Discriminant Directions in Backpropagation, to appear in *Proceedings of the IEEE Southeastcon*, Birmingham, (1992).

[Giles 87] Learning, Invariance, and Generalization in High-Order Neural Networks, *Applied Optics*, Vol. 26, No. 23, p. 4972, (1987).

[Giles 91] C.L. Giles, D. Chen, C.B. Miller, H.H. Chen, G.Z. Sun, Y.C. Lee, Second-Order Recurrent Neural Networks for Grammatical Inference, *Proceedings of the International Joint Conference on Neural Networks,* IJCNN-91-SEATTLE, Vol. II, p. 273, (1991).

[Giles 92] C.L. Giles, C.B. Miller, D. Chen, H.H. Chen, G.Z. Sun, Y.C. Lee, Learning and Extracting Finite State Automata with Second-Order Recurrent Neural Networks, to appear in *Neural Computation*, (1992).

[Lee 86] Y.C. Lee, G. Doolen, H.H. Chen, G.Z. Sun, T. Maxwell, H.Y. Lee, C.L. Giles, Machine Learning using a Higher Order Correlational Network, Physica D, Vol. 22, No. 1-3, p. 276, (1986).

[Maclin 92] R. Maclin, J.W. Shavlik, Refining Algorithms with Knowledge-Based Neural Networks: Improving the Chou-Fasman Algorithm for Protein Folding, S. Hanson, G. Drastal, R. Rivest (Eds), *Computational Learning Theory and Natural Learning Systems*, MIT Press, to appear, (1992).

[Perantonis 92] S.J. Perantonis, P.J.G. Lisboa, Translation, Rotation, and Scale Invariant Pattern Recognition by Higher-Order Neural Networks and Moment Classifiers, *IEEE Transactions on Neural Networks*, Vol. 3, No. 2, p.241, (1992).

[Pollack 91] J.B. Pollack, The Induction of Dynamical Recognizers, *Machine Learning*, Kluwer Academic Publishers, Boston, MA, Vol. 7, p. 227, (1991).

[Pratt 92] L.Y. Pratt, Non-Literal Transfer of Information among Inductive Learners, R.J. Mammone & Y.Y. Zeevi (Eds), *Neural Networks: Theory and Applications II*, Academic Press, to appear, (1992). Preprint, (1992).

[Suddarth 91] S. Suddarth, A. Holden, Symbolic Neural Systems and the Use of Hints for Developing Complex Systems, *International Journal of Man-Machine Studies*, Vol. 35, p. 291, (1991).

[Towell 90] G.G. Towell, J.W. Shavlik, M.O. Noordewier, Refinement of Approximately Correct Domain Theories by Knowledge-Based Neural Networks, *Proceedings of the Eighth National Conference on Artificial Intelligence* , Boston, MA, p. 861, (1990).

[Watrous 92] R.L. Watrous, G.M. Kuhn, Induction of Finite-State Languages Using Second-Order Recurrent Networks, to appear in *Neural Computation*, (1992).

[Williams 89] R.J. Williams, D. Zipser, A Learning Algorithm for Continually Running Fully Recurrent Neural Networks, *Neural Computation*, Vol. 1, No. 2, p. 270, (1989).