Constructive Learning of Recurrent Neural Networks

D. Chen[a], C.L. Giles[a,b], G.Z. Sun[a], H.H. Chen[a], Y.C. Lee[a], M.W. Goudreau[b,c]

[a]Institute for Advanced Computer Studies
University of Maryland
College Park, MD 20742

[b]NEC Research Institute
4 Independence Way
Princeton, NJ 08540

[c]Department of Electrical Engineering
Princeton University
Princeton, NJ 08544

***Abstract*— Recurrent neural networks are a natural model for learning and predicting temporal signals. In addition, simple recurrent networks have been shown to be both theoretically and experimentally capable of learning finite state automata [Cleeremans 89, Giles 92a, Minsky 67, Pollack 91, Siegelmann 92]. However, it is difficult to determine what is the minimal neural network structure for a particular automaton. Using a large recurrent network, which would be versatile in theory, in practice proves to be very difficult to train. Constructive or destructive recurrent methods might offer a solution to this problem. We prove that one current method, Recurrent Cascade Correlation, has fundamental limitations in representation and thus in its learning capabilities. We give a preliminary approach on how to get around these limitations by devising a "simple" constructive training method that adds neurons during training while still preserving the powerful fully recurrent structure. Through simulations we show that such a method can learn many types of regular grammars that the Recurrent Cascade Correlation method is unable to learn.**

I. Introduction

Recurrent Neural Networks have been studied extensively, because of their ability to store and process temporal information and sequential signals; for a summary of these issues see [Hertz 91, Narendra 90]. For example recent studies have shown that various order recurrent networks are able to infer small regular grammars from grammatical examples [Cleeremans 89, Giles 92a, Watrous 92].

For the purpose of good generalization, Occam's Razor would likely conclude that a small a network as possible would give the best generalization; this is also in keeping with the results of systems theory [Ljung 87]. [Alon 91] has given an upper limit on the size of a first order network needed to represent a regular grammar or a state automaton. However, in practice, we often do not have enough information about the nature of the target sequence in order to decide the network size before training. In addition for a particular problem, a much smaller network than that given by the theoretical upper bound solution usually exists.

One way to solve this problem is to train different size networks and find the smallest one that learns the training sequences. In practice this can be very time consuming since each different network is trained independently and there are too many different network architectures to choose from.

Another problem associated with recurrent networks is that the training scales badly with both network and problem size. The convergence can be very slow and training errors are not always guaranteed to reduce to previously defined tolerances. By using constructive training methods, one can hope that the neural network could possibly build itself little by little, and speed up the whole training process. In this paper, we show that an existing constructive method, Recurrent Cascade Correlation, has fundamental limitations. We propose an alternative method which eliminates these limitations and give some encouraging preliminary training results.

II. Simple Driven Recurrent Network

For our purposes a simple driven recurrent neural network consists of three parts (figure 3). We term the recurrent network "driven" to denote that it responds

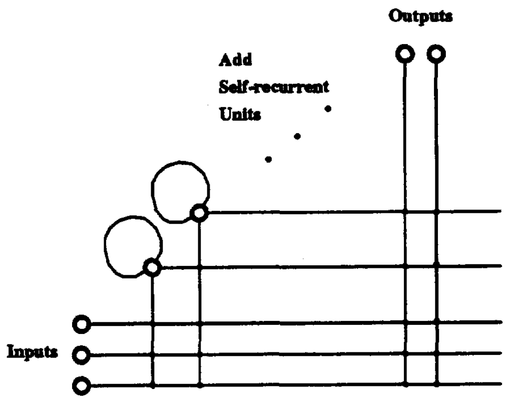

Figure 1: Recurrent Cascade-Correlation Network. The hidden neurons are self-recurrent, and only connect to previously existing neurons.

temporally to inputs. The hidden recurrent layer is activated by both the input neurons and the recurrent layer itself. The output neurons are in general activated by the input and recurrent neurons or, in a special case, by only the recurrent neurons.

Connections between layers can be first, second, or even higher orders; see [Giles 92a, Pollack 91, Sun 90, Watrous 92] for more discussion. In general, the updating rule can be written as,

$$\mathbf{S}^{t+1} = \mathcal{F}(\mathbf{W}, \mathbf{S}^t, \mathbf{I}^t)$$
$$\mathbf{O}^{t+1} = \mathcal{G}(\mathbf{U}, \mathbf{S}^t, \mathbf{I}^t).$$

where $\mathbf{I}^t, \mathbf{S}^t, \mathbf{O}^t$ are the values of input, recurrent and output neurons at time step t and $\mathbf{W}, \mathbf{U}$ are the respective connection weights. The typical learning rule uses gradient decent [Williams 89] to adjust the weights $\mathbf{W}, \mathbf{U}$ so as to minimize the error function:

$$E = \sum_t (\mathbf{T}^t - \mathbf{O}^t)^2.$$

III. "Simple" Constructive Learning

In general a constructive method dynamically changes the network structure during the training - [Gallant 86], [Hanson 90] and for a summary of other methods [Hertz 91]. In addition to the normal updating rule and learning rule of the neural network, a constructive training scheme also requires:

1. a criterion for when the changing takes place,
2. how to connect the newly created neurons to the existing system,
3. how to assign initial values to the newly added connections.

(For simplicity we ignore methods which are both constructive and destructive.) To speed up the training, we hypothesize that in addition to these criteria we must also satisfy the principle that the network preserves previously acquired knowledge *while* the network is changing. Previous work where a priori knowledge such as rules are encoded directly into recurrent networks have shown this to be the case [Frasconi 91, Giles 92b]. Various constructive learning schemes have been proposed for feed-forward networks [Ash 89, Fahlman 90, Gallant 86]. These methods find a minimal structure for the problem and reduce the computational complexity. However, to our knowledge little work besides [Fahlman 91] has focused on recurrent networks.

A. Limitations of the Recurrent Cascade-Correlation Architecture

[Fahlman 91] proposed a constructive training method — a Recurrent Cascade-Correlation (RCC) network (Figure 1). Regardless of the training procedure, this network can be topologically viewed as in Figure 2. It differs from a fully connected recurrent network in the sense that the recurrent connection of the old neurons to the newly added neurons are restricted - *i.e.* nonexistent. Even though this self recurrent restriction simplifies the training (each neuron can be trained sequentially), it *significantly* restricts the representational power of the network. We will show that this type of structure with a *sigmoid updating function* is **not** capable of representing all finite state automata.

To understand the limitations of RCC, we first examine the hard-limit threshold case, where each neuron is only allowed to take on binary values, *e.g.* $\{0, 1\}$ or $\{-1, 1\}$. Suppose we have a constant input sequence, say all 1's. The activation function of the first neuron S_1 then simplifies to

$$S_1^{t+1} = \Theta(W_{11} * S_1^t + \theta_1),$$

where Θ is the threshold function and the constant input term I is implicit. It is easy to verify that under such an update function, S_1 will either remain constant or oscillate at each time step between the two values $0, 1$ or $-1, 1$, since S_1 only depends only on its value at the previous time step. We define this oscillation at each time step as an oscillation of period 2. Oscillations that occur at two or more time steps have a period greater than 2, and a constant value is a period 1 oscillation. An example of all such sequences of period 2 or less is: $\{$*0000000⋯,1111111⋯,0101010⋯*$\}$.

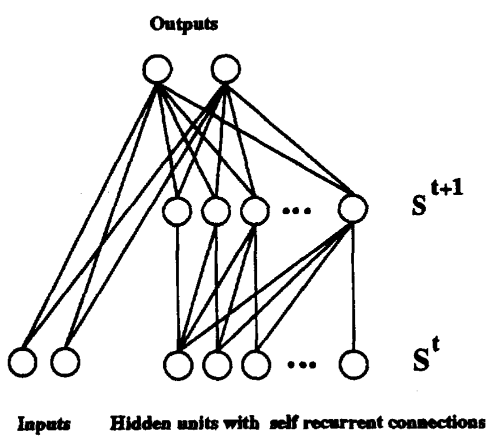

Figure 2: Re-drawing of RCC in figure 1.
The input and output neurons are connected to all hidden recurrent neurons. The hidden neurons are not fully connected to each other.

The activation function for the nth RCC neuron S_n can be written as

$$S_n^{t+1} = \Theta(W_{n1} * S_1^{t+1} + W_{n2} * S_2^{t+1} + \cdots + W_{nn} * S_n^t + \theta_n).$$

Or,

$$S_n^{t+1} = \Theta(\Lambda + W_{nn} * S_n^t), \tag{1}$$

where

$$\Lambda = W_{n1} * S_1^{t+1} + W_{n2} * S_2^{t+1} + \cdots + W_{n(n-1)} * S_{n-1}^{t+1} + \theta_n.$$

We prove this by induction. Assume $S_1, \cdots, S_{n-1}$ at each time step all oscillate with period 2 or remain constant. As Λ is a linear function of $S_1, \cdots, S_{n-1}$, it will at most oscillate at period 2. If Λ remains constant all the time, then S_n will be a constant or oscillate at period 2, the same as for S_1. If Λ oscillates at period 2, then by examining all 14 possible mappings of equation 1 (see Appendix), we find that S_n can only oscillate at period 2 or remains constant all the time. This proves that the binary RCC structure is not be able to represent or to simulate all possible finite state machines, *e.g.* a machine consists of part that has period greater that 2 under a same input. It can also be shown that for an analog RCC network with a constant input signal, the activities of the recurrent neurons will asymptotically become either constant or oscillate at period 2. [There results hold for both first and second order recurrent neural networks. See [Pao 89] for a discussion of order and [Lee 86] for order in recurrent networks.]

It is arguable that the analog RCC network can have more complex dynamics during the transient period. However, a typical finite state automaton can accept infinitely long strings. Thus, the RCC type of network structure (with sigmoid thresholds) will fail at some point (beyond the transient period) after a periodic input sequence is presented. For example, numerical simulations have shown the RCC network is unable to learn the simple double parity grammar. The number of added neurons grow as a function of the longest string length of the training examples!

It should be noted that the sigmoid type updating function is *essential* to the above proof. If the activity function is Gaussian or another non-monotonic shape, or if the activity function is of high order in terms of the hidden neurons, i.e. based on terms such as $S_i * S_j$; much more complicated behavior can occur and the above conclusions may not hold.

B. Simple Expanding Recurrent Neural Network

In order to get around the interconnect restriction imposed by RCC, we propose a very simple scheme to dynamically construct a recurrent network. In this method, the network is the same as the simple driven recurrent structure discussed in Section II. In this method the number of neurons in the recurrent layer is allowed to expand whenever needed. The only criterion for expansion is that the network spend some time learning. As the network is always fully-connected, it does not have the restriction of RCC. Theoretically it has been shown that a fully connected recurrent network is capable of representing any finite state automaton [Minsky 67].

The importance of using a priori knowledge in neural network learning has been described by many, see for example [Towell 90, Giles 92b, Frasconi 91]. We further assume that it is important to effectively maintain some of the networks knowledge acquired during training. To do this we require the network to be expanded smoothly, *i.e.* the newly added weights should remain zero or very small random numbers. Thus, the new network behaves very similar to the old one immediately after the expansion.

First, we present a simple example of the above method. For our example we train a recurrent neural network to be a deterministic finite state automaton. For training we use a second-order fully-recurrent neural network that uses full gradient, real-time recurrent learning [Williams 89]. For detail description see [Giles 92a]. We examine if the network can take advantage of its previous knowledge. We first train the network to learn a 10-state randomly generated deterministic finite state automaton (DFA) (figure 4) as in [Giles 92c]. We use the incremental training method, where the network reads through

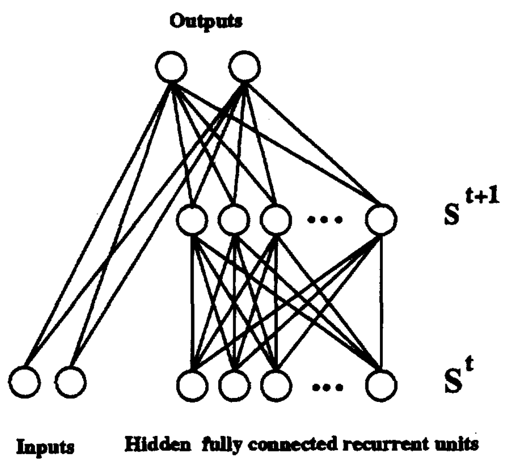

Figure 3: Fully recurrent neural network.
The input and output neurons are connected to all hidden recurrent neurons.

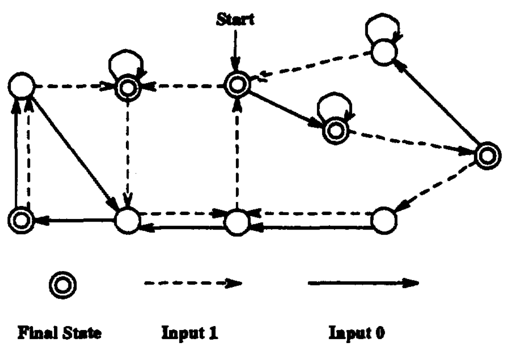

Figure 4: A 10-state finite state automaton, which is randomly generated.

the ordered training samples until the accumulated error exceeds the preset value or all training samples are classified correctly. When training with a fixed size 8 neuron fully-recurrent network, the network converged in 72 epoches after using 10000 positive and negative training examples. However, if we train a similar network with only 7 recurrent neurons for 150 epoches first and then expand the network to 8 recurrent neurons. the network takes only 32 more epoches to converge. Here an epoch is defined as a training cycle in which the network starts again at the beginning of the training samples after finding 5 errors. This indicates that the network does take advantage of previous knowledge and converges faster than a network with no a priori knowledge and random initial weights.

Choosing the criterion for when the network expands is very important. A simple method is to determine if the training error reaches the local minimum. More complex methods based on entropy measure, network capacity, or neuron activity distribution would be more feasible.

In other preliminary experiments, we add one more neuron to the recurrent layer after every 50 epoches, until the network learns all the training samples. This is a very simple constructive criterion; a more sophisticated constructive criterion might generate a smaller network. In training very small grammars (those of [Tomita 82]), we found that the constructive method converges very fast when the network size grows up to the minimal size required. However the convergence speed was not significantly less than that obtained by training the corresponding fixed size networks. This is understandable since for a very simple problem, the fixed size network can learn the training samples very quickly assuming a large enough network is provided. However, by avoiding training all different size networks, the constructive method does appear to save time in finding the smallest size network for each of these grammars.

IV. Conclusions

We presented some preliminary results on a "simple" constructive learning method for recurrent neural networks. This method relies on using some of the knowledge of a partially trained neural network, and more importantly, expands the network in a fully recurrent manner. The constructive learning method permits the network to build itself up from scratch. This type of training method is necessary when only very limited information about the problem to be solved is available. The recurrent cascade-correlation network is certainly a step in the right direction, but it is incapable of representing and thus learning many finite state automata. The simple constructive method proposed avoids the limitations of Recurrent Cascade-Correlation (RCC) networks. We illustrate this method by learning some small grammars of Tomita and a randomly generated 10-state grammar. The reason this simple constructive method outperforms RCC easily explained - the full recurrence of the growing network is preserved. Admittedly, the experiments described are preliminary. Further experiments might show inherent limitations in such a "simple" method, such as uncontrolled neuron growth. Other questions are what are good criteria for expansion? What amount of "captured" knowledge is necessary for effective learning? However, the lesson of maintaining the full recurrence in the "added" neurons seems an important one, especially if the full representational power of the recurrent network is needed.

APPENDIX Binary Recurrent Cascade Correlation Network

Without loss of generality, assume the binary neuron is either 0 or 1. Recall equation 1,

$$S_n^{t+1} = \Theta(\Lambda + W_{nn} * S_n^t).$$

If Λ oscillates with period 2, *i.e.*

$$\Lambda = \begin{cases} \lambda_1 & \text{if } t = \text{odd} \\ \lambda_2 & \text{if } t = \text{even} \end{cases}$$

we can list all possible outcomes of the above equation.

Λ	S^t	S^{t+1}	possible sequences
λ_1	0	0	
λ_1	1	0	*00000000···*
λ_2	0	0	
λ_2	1	0	*10000000···*

Λ	S^t	S^{t+1}	possible sequences
λ_1	0	1	
λ_1	1	0	*01010101···*
λ_2	0	0	
λ_2	1	0	*10010101···*

Λ	S^t	S^{t+1}	possible sequences
λ_1	0	0	
λ_1	1	1	*00000000···*
λ_2	0	0	
λ_2	1	0	*11000000···*

Λ	S^t	S^{t+1}	possible sequences
λ_1	0	1	
λ_1	1	1	*01010101···*
λ_2	0	0	
λ_2	1	0	*11010101···*

Λ	S^t	S^{t+1}	possible sequences
λ_1	0	0	
λ_1	1	0	*00101010···*
λ_2	0	1	
λ_2	1	0	*10101010···*

Λ	S^t	S^{t+1}	possible sequences
λ_1	0	1	
λ_1	1	0	*01010101···*
λ_2	0	1	
λ_2	1	0	*10101010···*

Λ	S^t	S^{t+1}	possible sequences
λ_1	0	1	
λ_1	1	1	*01010101···*
λ_2	0	1	
λ_2	1	0	*10101010···*

Λ	S^t	S^{t+1}	possible sequences
λ_1	0	0	
λ_1	1	0	*00000000···*
λ_2	0	0	
λ_2	1	1	*10000000···*

Λ	S^t	S^{t+1}	possible sequences
λ_1	0	0	
λ_1	1	1	*00000000···*
λ_2	0	0	
λ_2	1	1	*11111111···*

Λ	S^t	S^{t+1}	possible sequences
λ_1	0	1	
λ_1	1	1	*01111111···*
λ_2	0	0	
λ_2	1	1	*11111111···*

Λ	S^t	S^{t+1}	possible sequences
λ_1	0	0	
λ_1	1	0	*00101010···*
λ_2	0	1	
λ_2	1	1	*10101010···*

Λ	S^t	S^{t+1}	possible sequences
λ_1	0	1	
λ_1	1	0	*01101010···*
λ_2	0	1	
λ_2	1	1	*10101010···*

Λ	S^t	S^{t+1}	possible sequences
λ_1	0	0	
λ_1	1	1	*00111111···*
λ_2	0	1	
λ_2	1	1	*11111111···*

Λ	S^t	S^{t+1}	possible sequences
λ_1	0	1	
λ_1	1	1	*11111111···*
λ_2	0	1	
λ_2	1	1	*11111111···*

Two combinations of S^{t+1}

Λ	S^t	S^{t+1}
λ_1	0	0
λ_1	1	1
λ_2	0	1
λ_2	1	0

Λ	S^t	S^{t+1}
λ_1	0	1
λ_1	1	0
λ_2	0	0
λ_2	1	1

are not listed above. This is because the linear threshold function cannot solve the XOR problem. It is easy to see that the sequence of S will either be a constant or oscillate at period 2.

References

[Alon 91] N. Alon, A.K. Dewdney, T.J. Ott, Efficient simulation of finite automata by neural nets, *J. A.C.M.* 38, p. 495 (1991).

[Ash 89] T. Ash, Dynamic Node Creation in Backpropagation Networks, it Connection Science, vol 1, No. 4, p. 365 (1989).

[Cleeremans 89] A. Cleeremans, D. Servan-Schreiber, J. McClelland, Finite State Automata and Simple Recurrent Recurrent Networks, *Neural Computation*, 1(3), p. 372 (1989).

[Fahlman 90] S.E. Fahlman, C. Lebiere, The Cascade-Correlation Learning Architecture, *Advances in Neural Information Systems 2*, D.S. Touretzky (ed), Morgan Kaufmann, San Mateo, Ca, (1990).

[Fahlman 91] S.E. Fahlman, The Recurrent Cascade-Correlation Architecture, in *Advances in Neural Information Processing Systems 3*, R.P. Lippmann, J.E. Moody, D.S. Touretzky (eds), Morgan Kaufmann, San Mateo, Ca., p.190 (1991).

[Frasconi 91] P. Frasconi, M. Gori, M. Maggini, G. Soda, An Unified Approach for Integrating Explicit Knowledge and Learning by Example in Recurrent Networks, *Proceedings of the International Joint Conference on Neural Networks* IJCNN-91-SEATTLE, Vol. I, p. 811, (1991).

[Gallant 86] S.I. Gallant, Three Constructive Algorithms for Network Learning, in *Proceedings, 8th Annual Conference of the Cognitive Science Society.* p. 652 (1986)

[Giles 92a] 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, *Neural Computation*, 4(3), p. 393 (1992).

[Giles 92b] C.L. Giles, C.W. Omlin, Inserting Rules into Recurrent Neural Networks *Neural Networks for Signal Processing II, Proceedings of the 1992 IEEE-SP Workshop*, S.Y. Kung, F. Fallside, J. Aa Sorenson C.A. Kamm (eds), IEEE92TH0430-9, p.13 (1992).

[Giles 92c] C.L. Giles, C.B. Miller, D. Chen, G.Z. Sun, H.H. Chen, Y.C. Lee, Extracting and Learning an Unknown Grammar with Recurrent Neural Networks, *Advances in Neural Information Processing Systems 4*, J.E. Moody, S.J. Hanson and R.P. Lippmann (eds), Morgan Kaufmann, San Mateo, Ca., (1992).

[Hanson 90] S.J. Hanson, Meiosis Networks, in *Advances in Neural Information Processing Systems 2*, D. Touretzky (ed), Morgan Kaufmann, San Mateo, Ca., p.533 (1990)

[Hertz 91] J. Hertz, A. Krogh, R.G. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley, Redwood City, CA., p. 163 (1991).

[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-D, No.1-3, p. 276 (1986).

[Ljung 87] L. Ljung, *System Identification - Theory for the User*, Prentice-Hall, Englewood Cliffs, N.J. (1987).

[Minsky 67] M.L. Minsky, *Computation: Finite and Infinite Machines*, Ch 3.5, Prentice-Hall, Englewood Cliffs, N.J. (1967).

[Narendra 90] K.S. Narendra, K. Parthasarathy, Identification and Control of Dynamical Systems Using Neural Networks, *IEEE Trans. on Neural Networks*, Vol. 1, No. 1, page 4 (1990).

[Pao 89] Y-H. Pao, *Adaptive Pattern Recognition and Neural Networks*, Addison-Wesley Publishing Co., Inc., Reading, MA (1989).

[Pollack 91] J.B. Pollack, The Induction of Dynamical Recognizers, *Machine Learning*, vol 7, p. 227 (1991).

[Siegelmann 92] H.T. Siegelmann, E.D. Sontag, On the computational power of neural nets, in *Proc. Fifth ACM Workshop on Computational Learning Theory*, Pittsburgh PA, ACM Press, p. 440 (1992).

[Sun 90] G.Z. Sun, H.H. Chen, C.L. Giles, Y.C. Lee, D. Chen, Connectionist Pushdown Automata that Learn Context-Free Grammars, *Proceedings of the International Joint Conference on Neural Networks*, IJCNN-90-WASH-DC, Lawrence Erlbaum, Hillsdale, N.J., Vol I, p. 577 (1990).

[Tomita 82] M. Tomita, Dynamic Construction of Finite-state Automata from Examples Using Hill-climbing. *Proceedings of the Fourth Annual Cognitive Science Conference* p. 105 (1982).

[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, *Neural Computation* 4(3), p.406 (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).