Communicated by Haim Sompolinsky

Learning in the Recurrent Random Neural Network

Erol Gelenbe
Ecole des Hautes Etudes en Informatique,
Université René Descartes (Paris V), 45 rue des Saints-Pères,
75006 Paris, France

The capacity to learn from examples is one of the most desirable features of neural network models. We present a learning algorithm for the recurrent random network model (Gelenbe 1989, 1990) using gradient descent of a quadratic error function. The analytical properties of the model lead to a "backpropagation" type algorithm that requires the solution of a system of n linear and n nonlinear equations each time the n-neuron network "learns" a new input–output pair.

1 Introduction

The capability to learn from examples is one of the most desirable features of neural network models. Therefore this issue has been at the center of much research in neural network theory and applications (Ackley *et al.* 1985; Le Cun 1985; Rumelhart *et al.* 1986).

Learning theory in general is of major interest because of its numerous implications in machine intelligence, as well as its ability to provide a better understanding of the relationship between natural and artificial intelligence.

In the area of artificial neural networks, learning has been extensively studied in the context of feedforward networks, primarily on the basis of the backpropagation algorithm (Rumelhart *et al.* 1986).

Designing effective learning algorithms for general (i.e., recurrent) networks is a current and legitimate scientific concern in neural network theory. There are numerous examples where recurrent networks constitute a natural approach to problems. Such examples include, in particular, image processing and pattern analysis and recognition (see, for instance, Atalay *et al.* 1991), where local interactions between picture elements lead to mutual interactions between neighboring neurons, which are naturally represented by recurrent networks. In such cases, it is clear that effective learning algorithms for recurrent networks can enhance the value of neural network methodology. Another area where recurrent networks are indispensable is in combinatorial optimization, and it would be interesting to explore further the relationship between the application

Neural Computation **5, 154–164** (1993)

of neural networks to control and optimization (Gelenbe and Batty 1992) and network learning.

Several authors have considered learning algorithms for recurrent connectionist networks (Almeida 1987; Pineda 1987, 1989; Pearlmutter 1989; Behrens *et al.* 1991). These are based on neural network dynamics, which exhibit a fixed-point behavior. The work presented in this paper extends this approach to the random network model (Gelenbe 1989, 1990), which has the advantage of possessing well-defined fixed-point equations representing the stationary solution of the stochastic network equations.

Applications of the random network model to image texture generation, associative memory, pattern recognition, and combinatorial optimization have been described elsewhere (Atalay *et al.* 1991; Gelenbe *et al.* 1991; Mokhtari 1991; Gelenbe and Batty 1992).

In this paper we present a "backpropagation" type learning algorithm for the recurrent random network model (Gelenbe 1989, 1990), using gradient descent of a quadratic error function when a set of input–output pairs is presented to the network. Both the excitation and inhibition weights of the random network model must be learned by the algorithm. Thus, it requires the solution of a system of $2n$ linear and n nonlinear equations each time the n-neuron network "learns" a new input–output pair. The system of nonlinear equations describes the networks fixed-point, while the linear equations are obtained from the partial derivatives of these equations with respect to the network weights.

To justify the use of the algorithm, we prove (in the Appendix) a general theorem concerning necessary and sufficient conditions for the existence of the stationary or fixed-point solution to the network. This general result completes the work presented in Gelenbe (1990) where only more restrictive sufficient conditions were given. Note that for our network existence implies uniqueness of the solution, due to the fact that the random network model is characterized by Chapman–Kolmogorov equations. Furthermore existence implies stability, since all moments of the state distribution can be explicitly computed from the model's product-form property.

2 The Random Network Model

In the random network model (RN), n neurons exchange *positive and negative* impulse signals. Each neuron accumulates signals as they arrive, and fires if its total signal count at a given instant of time is positive. Firing then occurs at random according to an exponential distribution of constant rate, and signals are sent out to other neurons or to the outside of the network. Each neuron i of the network is represented at time t by its input signal potential $k_i(t)$, constituted only by positive signals that have accumulated, which have not yet been cancelled by negative signals,

and which have not yet been sent out by the neuron as it fires. Positive signals represent excitation, while negative signals represent inhibition. A negative signal *reduces by 1* the potential of the neuron to which it arrives (i.e., it "cancels" an existing signal) or has no effect on the signal potential if it is already zero, while an arriving positive signal *adds 1* to the neuron potential. This is a simplified representation of biophysical neural behavior (Kandel and Schwartz 1985).

In the RN, signals arrive at a neuron from the outside of the network (exogenous signals) or from other neurons. Each time a neuron fires, a signal leaves it depleting its total input potential. A signal leaving neuron i heads for neuron j with probability $p^+(i,j)$ as a positive (or normal) signal, or as a negative signal with probability $p^-(i,j)$ or it departs from the network with probability $d(i)$. $p(i,j) = p^+(i,j) + p^-(i,j)$ is the transition probability of a Markov chain representing the movement of signals between neurons. We have $\sum_j p(i,j) + d(i) = 1$ for $1 \leq i \leq n$. External (or exogenous) inputs to each neuron i of the network are provided by stationary Poisson processes of rate $\Lambda(i)$, and $\lambda(i)$. A neuron is capable of firing and emitting signals if its potential is strictly positive, and firing times are modeled by iid exponential neuron firing times with rate $r(i)$, at neuron i.

In Gelenbe (1989) it was shown that this network has a product form solution. That is, the network's stationary probability distribution can be written as the product of the marginal probabilities of the state of each neuron. This does not imply that the neurons have a behavior that is independent of each other. Indeed the probabilities that each neuron is excited are obtained from the coupled nonlinear signal flow equations (2) below, which yield the rate of signal arrival and hence the rate of firing of each neuron in steady state.

The RN has a number of interesting features:

1. It represents more closely the manner in which signals are transmitted in a biophysical neural network where they travel as spikes rather than as fixed analog signals.
2. It is computationally efficient.
3. It is easy to simulate, since each neuron is simply represented by a counter; this may lead to a simple hardware implementation.
4. It represents neuron potential and therefore the level of excitation as an integer, rather than as a binary variable, which leads to more detailed information on system state; a neuron is interpreted as being in the "firing state" if its potential is positive.

Let $k(t) = [k_1(t), \ldots, k_n(t)]$ be the vector of signal potentials at time t, and $k = (k_1, \ldots, k_n)$ be a particular value of the vector. $p(k)$ denotes the stationary probability distribution $p(k) = \lim_{t\to\infty} \text{Prob}[k(t) = k]$ if it exists. Since $\{k(t) : t \geq 0\}$ is a continuous time Markov chain it satisfies the

usual Chapman–Kolmogorov equations; thus in steady state $p(k)$ must satisfy the global balance equations:

$$p(k)\sum_i[\Lambda(i)+[\lambda(i)+r(i)]\mathbf{1}[k_i>0]] = \sum_i[p(k_i^+)r(i)d(i)+p(k_i^-)\Lambda(i)\mathbf{1}[k_i>0] + p(k_i^+)\lambda(i)+\sum_j\{p(k_{ij}^{+-})r(i)p^+(i,j)\mathbf{1}[k_j>0] + p(k_{ij}^{++})r(i)p^-(i,j))+p(k_i^+)r(i)p^-(i,j)\mathbf{1}[k_j=0]\}]$$

where the vectors used are

$$\begin{aligned} k_i^+ &= (k_1,\ldots,k_i+1,\ldots,k_n)\\ k_i^- &= (k_1,\ldots,k_i-1,\ldots,k_n)\\ k_{ij}^{+-} &= (k_1,\ldots,k_i+1,\ldots,k_j-1,\ldots,k_n)\\ k_{ij}^{++} &= (k_1,\ldots,k_i+1,\ldots,k_j+1,\ldots,k_n)\end{aligned}$$

and $\mathbf{1}[X]$ is the usual characteristic function which takes the value 1 if X is true and 0 otherwise.

Theorem (Gelenbe 1989). Let

$$q_i \equiv \lambda^+(i)/[r(i)+\lambda^-(i)] \tag{2.1}$$

where the $\lambda^+(i),\lambda^-(i)$ for $i=1,\ldots,n$ satisfy the following system of nonlinear simulataneous equations:

$$\lambda^+(i)=\sum_j q_j r(j)p^+(j,i)+\Lambda(i),\quad \lambda^-(i)=\sum_j q_j r(j)p^-(j.i)+\lambda(i) \tag{2.2}$$

If a unique nonnegative solution $\{\lambda^+(i),\lambda^-(i)\}$ *exists to equations 2.1 and 2.2 such that each* $q_i<1$, then

$$p(k)=\prod_{i=1}^n[1-q_i]q_i^{k_i}$$

As a consequence of this result, whenever the $q_i<1$ can be found, the network is stable in the sense that all moments (marginal or joint) of the neural network state can be found from the above formula, and all moments are finite. For instance, the average potential at a neuron i is simply $q_i/[1-q_i]$. The rate (frequency) of the emission of spikes from neuron i in steady state is then $q_i r(i)$. Furthermore because the underlying model is described by Chapman–Kolmogorov equations, whenever there is a solution, it is necessarily unique and given by the above product form formula.

If for some neuron, we have $\lambda^+(i)>[r(i)+\lambda^-(i)]$, we say that the neuron is unstable or saturated. This implies that it is constantly excited in steady state: $\lim_{t\to\infty}\text{Prob}[k_i(t)>0]=1$. Its rate of spike emission

is then $r(i)$: to another neuron j of the network its output appears as a constant source of positive or negative signals of rates $r(i)p^+(i,j)$ and $r(i)p^-(i,j)$.

For notational convenience let us write

$$\begin{aligned} w^+(j,i) &= r(i)p^+(i,j) \geq 0, \quad w^-(j,i) = r(i)p^-(i,j) \geq 0 \\ N(i) &= \sum_j q_j w^+(j,i) + \Lambda(i), \quad \text{and} \\ D(i) &= r(i) + \sum_j q_j w^-(j,i) + \lambda(i) \end{aligned}$$

Then 1 becomes

$$q_i = N(i)/D(i) \tag{2.3}$$

and $r(i) = \sum_j [w^+(i,j) + w^-(i,j)]$.

2.1 The Role of the Parameters $w^+(j,i)$ and $w^-(j,i)$. The weight parameters $w^+(j,i)$ and $w^-(j,i)$ have a somewhat different effect in the RN model than the weights $w(j,i)$ in the connectionist model. In the RN model, all the $w^+(j,i)$ and $w^-(j,i)$ are *nonnegative* since they represent rates at which positive and negative signals are sent out from any neuron i to neuron j. Furthermore, in the RN model, for a given pair (i,j) it is possible that both $w^+(i,j) > 0$ and $w^-(i,j) > 0$; in general, it is not possible to transform an RN into an equivalent network in which certain connections are only excitatory, while others are only inhibitory, as would be the case in the usual connectionist model. Therefore, in the RN, for each pair (j,i) it will be necessary to learn both $w^+(i,j)$ and $w^-(i,j)$.

3 Learning with the Recurrent Random Network Model

We now present an algorithm for choosing the set of network parameters W in order to learn a given set of K input–output pairs $(\iota, \mathbf{Y})$ where the set of successive inputs is denoted $\iota = \{\iota_1, \ldots, \iota_K\}$, and $\iota_k = (\Lambda_k, \lambda_k)$ are pairs of positive and negative signal flow rates entering each neuron:

$$\Lambda_k = [\Lambda_k(1), \ldots, \Lambda_k(n)], \quad \lambda_k = [\lambda_k(1), \ldots, \lambda_k(n)]$$

The successive desired outputs are the vectors $\mathbf{Y} = \{y_1, \ldots, y_K\}$, where each vector $y_k = (y_{1k}, \ldots, y_{nk})$, whose elements $y_{ik} \in [0,1]$ correspond to the desired values of each neuron. The network approximates the set of desired output vectors in a manner that minimizes a cost function E_k:

$$E_k = (1/2) \sum_{i=1}^{n} a_i (q_i - y_{ik})^2, \quad a_i \geq 0$$

Without loss of generality, we treat each of the n neurons of the network as an output neuron; if we wish to remove some neuron j from network output it suffices to set $a_j = 0$ in the cost function, and to disconsider q_j when constructing the output of the network.

Our algorithm lets the network learn *both* n by n weight matrices $W_k^+ = \{w_k^+(i,j)\}$ and $W_k^- = \{w_k^-(i,j)\}$ by computing for each input $\iota_k = (\Lambda_k, \lambda_k)$, a new value W_k^+ and W_k^- of the weight matrices, using gradient descent. Clearly, we seek only solutions for which all these weights are positive.

Let us denote by the generic term $w(u,v)$ either $w(u,v) \equiv w^-(u,v)$, or $w(u,v) \equiv w^+(u,v)$. The rule for weight update may be written as

$$w_k(u,v) = w_{k-1}(u,v) - \eta \sum_{i=1}^{n} a_i(q_{ik} - y_{ik})[\partial q_i/\partial w(u,v)]_k \tag{3.1}$$

where $\eta > 0$ is some constant, and

1. q_{ik} is calculated using the input ι_k and $w(u,v) = w_{k-1}(u,v)$, in equation 3.
2. $[\partial q_i/\partial w(u,v)]_k$ is evaluated at the values $q_i = q_{ik}$ and $w(u,v) = w_{k-1}(u,v)$.

To compute $[\partial q_i/\partial w(u,v)]_k$ we turn to the expression 3, from which we derive the following equation:

$$\begin{aligned}
\partial q_i/\partial w(u,v) &= \sum_j \partial q_j/\partial w(u,v)[w^+(j,i) - w^-(j,i)q_i]/D(i) \\
&\quad - 1[u=i]q_i/D(i) \\
&\quad + 1[w(u,v) = w^+(u,i)]q_u/D(i) - 1[w(u,v) \\
&\equiv w^-(u,i)]q_u q_i/D(i)
\end{aligned}$$

Let $q = (q_1, \ldots, q_n)$, and define the $n \times n$ matrix

$$W = \{[w^+(i,j) - w^-(i,j)q_j]/D(j)\} \quad i,j = 1, \ldots, n$$

We can now write the vector equations:

$$\begin{aligned}
\partial q/\partial w^+(u,v) &= \partial q/\partial w^+(u,v)W + \gamma^+(u,v)q_u \\
\partial q/\partial w^-(u,v) &= \partial q/\partial w^-(u,v)W + \gamma^-(u,v)q_u
\end{aligned}$$

where the elements of the n-vectors $\gamma^+(u,v) = [\gamma_1^+(u,v), \ldots, \gamma_n^+(u,v)]$, $\gamma(u,v) = [\gamma_1(u,v), \ldots, \gamma_n^-(u,v)]$ are

$$\begin{aligned}
\gamma_i^+(u,v) &= -1/D(i) \quad \text{if } u = i, v \neq i, \\
&= +1/D(i) \quad \text{if } u \neq i, v = i, \\
&= 0 \quad \text{for all other values of } (u,v), \\
\gamma_i^-(u,v) &= -(1+q_i)/D(i) \quad \text{if } u = i, v = i, \\
&= -1/D(i) \quad \text{if } u = i, v \neq i, \\
&= -q_i/D(i) \quad \text{if } u \neq i, v = i, \\
&= 0 \quad \text{for all other values of } (u,v)
\end{aligned}$$

Notice that

$$\begin{aligned} \partial q/\partial w^+(u,v) &= \gamma^+(u,v)q_u[\mathrm{I}-W]^{-1} \\ \partial q/\partial w^-(u,v) &= \gamma^-(u,v)q_u[\mathrm{I}-W]^{-1} \end{aligned} \tag{3.2}$$

where I denotes the n by n identity matrix. Hence the main computational effort in solving 3.2 is simply to obtain $[\mathrm{I}-W]^{-1}$, which can be done in time complexity $O(n^3)$, or $O(mn^2)$ if an m-step relaxation method is used. Since the solution of 3 is necessary for the learning algorithm, in the Appendix we derive necessary and sufficient conditions for the existence of the q_i.

We now have the information to specify the complete learning algorithm for the network:

Initiate the matrices W_0^+ and W_0^- in some appropriate manner. This initiation will be made at random (among nonnegative matrices) if no better information is available; in some cases it may be possible to choose these initial values by using a Hebbian learning rule. Choose a value of η in 4.

1. For each successive value of k, starting with $k = 1$ proceed as follows. Set the input values to $\iota_k = (\Lambda_k, \lambda_k)$.

2. Solve the system of nonlinear equations 3 with these values.

3. Solve the system of linear equations 3.2 with the results of (2).

4. Using equation 3.1 and the results of (2) and (3), update the matrices W_k^+ and W_k^-. Since we seek the "best" matrices (in terms of gradient descent of the quadratic cost function) that satisfy the *nonnegativity* constraint, in any step k of the algorithm, if the iteration yields a negative value of a term, we have two alternatives:

   a. set the term to zero, and stop the iteration for this term in this step k; in the next step $k+1$ we will iterate on this term with the same rule starting from its current null value;

   b. go back to the previous value of the term and iterate with a smaller value of η.

   In our implementation we have used (a).

Note that we may either proceed with a complete gradient descent [iterating on Steps (2), (3), and (4) until the change in the cost function or in the new values of the weights is smaller than some predetermined value], or only one iteration can be carried out for all the weights for each successive value of k (new input).

Clearly, one may either update the weight matrices separately for each successive value of k (i.e., successive input) as suggested, or sum the updates for all inputs at each iteration of the algorithm.

3.1 Complexity. Several authors have examined the complexity of neural network learning algorithms (Pineda 1989; Baum 1991). One viewpoint (Pineda 1989) is to consider the complexity of each network weight update, while another is to consider the complexity of learning a given family of input–output functions (Baum 1991). In the latter approach, it is known that learning even elementary boolean functions using the backpropagation algorithm is NP complete. In fact, the complexity of our learning algorithm is of the same order as that of the algorithms described in Pineda (1989).

Here we merely discuss the complexity of weight update for the algorithm we have presented. Notice that the algorithm requires that for each (u, v) and for each input (successive k) we solve the nonlinear system of equations 3, and the linear system 3.2.

Equations 3.2 have to be solved for each (u, v). $[\mathbf{I}, W]^{-1}$ is obtained in time complexity $O(n^3)$, or in time complexity $O(mn^2)$ if a relaxation method with m iterations is adopted as suggested, for instance in Pineda (1989). The remaining computations for 3.2 are trivial. Similarly for 3, which is a nonlinear system of equations (to be solved once for each step), the complexity will be $O(mn^2)$.

Appendix: Existence and Uniqueness of Network Solutions

As with most neural network models, the signal flow equations 1 and 2, which describe the manner in which each neuron receives inhibitory or excitatory signals from other neurons or from the outside world, are nonlinear. These equations are essential to the construction of the learning algorithm described above. Yet only sufficient conditions for the existence (and uniqueness) of their solution had previously been established for feedforward networks, or for so-called hyperstable networks (Gelenbe 1989, 1990).

Thus in order to implement the learning algorithm it is useful to have necessary and sufficient conditions for their existence. This is precisely what we do in this appendix.

Rewrite 1 and 2 as follows:

$$\begin{aligned} \lambda^+(i) &= \sum_j \lambda^+(j)p^+(j,i)r(i)/[r(i)+\lambda^-(i)] + \Lambda(i), \\ \lambda^-(i) &= \sum_j \lambda^+(j)p^-(j,i)r(i)/[r(i)+\lambda^-(i)] + \lambda(i) \end{aligned} \tag{A.1}$$

where the q_i have disappeared from the equations. The $\lambda^+(i)$ and $\lambda^-(i)$ represent the total arrival rates of positive and negative signals to neuron i.

Define the following vectors:

λ^+ with elements $\lambda^+(i)$

λ^- with elements $\lambda^-(i)$

Λ with elements $\Lambda(i)$

λ with elements $\lambda(i)$

Let F be the diagonal matrix with elements $f_i = r(i)/[r(i) + \lambda^-(i)] \leq 1$. Then A.1 may be written as

$$\lambda^+ = \lambda^+ FP^+ + \Lambda, \quad \lambda^- = \lambda^+ FP^- + \lambda$$

or,

$$\lambda^+(\mathbf{I} - FP^+) = \Lambda \tag{A.2}$$

$$\lambda^- = \lambda^+ FP^- + \lambda \tag{A.3}$$

Proposition 1. Equations A.2 and A.3 have a solution (λ^+, λ^-).

Proof. Since the series $\sum_{n=0}^{\infty}(FP^+)^n$ is geometrically convergent (Kemeny and Snell 1960, p. 43 ff), we can write A.2 as

$$\lambda^+ = \Lambda \sum_{n=0}^{\infty}(FP^+)^n$$

so that A.3 becomes

$$\lambda^- - \lambda = \Lambda \sum_{n=0}^{\infty}(FP^+)^n FP^- \tag{A.4}$$

Now define $\mathbf{y} = \lambda^- - \lambda$, and call the vector function

$$G(y) = \Lambda \sum_{n=0}^{\infty}(FP^+)^n FP^-$$

where the dependence of G on $\mathbf{y}$ comes from F, which depends on λ^-. Notice that G is continuous. Therefore by Brouwer's fixed-point theorem,

$$\mathbf{y} = G(\mathbf{y}) \tag{A.5}$$

has a fixed-point $\mathbf{y}^*$. This fixed point will in turn yield the solution of A.2 and A.3:

$$\lambda^-(\mathbf{y}^*) = \lambda + \mathbf{y}^*, \quad \lambda^+(\mathbf{y}^*) = \Lambda \sum_{n=0}^{\infty} [F(\mathbf{y}^*)P^+]^n$$

completing the proof. □

If this computation yields a fixed-point $\mathbf{y}^*$ such that for any neuron i such that $\lambda^+(i) \geq [r(i) + \lambda^-(i)]$, the stationary solution for neuron i does not exist; this simply means that in steady state neuron i is constantly excited, and we set $q_i(\mathbf{y}^*) = 1$. If on the other hand we obtain $\lambda^+(i) < [r(i) + \lambda^-(i)]$, then we set $q_i(\mathbf{y}^*) = \lambda^+(i)/[r(i) + \lambda^-(i)]$. Since $p(k)$ is a probability distribution it must sum to 1, which is the case if $q_i(\mathbf{y}^*) < 1$, for all i and hence $p(k)$ exists. Let us insist on the fact that $p(k)$ is indeed unique, and that $q_i(\mathbf{y}^*) < 1$ for all i implies stability (in the sense of finiteness of all moments of the state).

Remark. This reduces the problem of determining existence and uniqueness of the steady-state distributions of a random network to that of computing $\mathbf{y}^*$ from A.5, which always exists by Proposition 1, and then of verifying the condition $q_i(\mathbf{y}^*) < 1$, for each $i = 1, \ldots, n$.

Acknowledgments

The author acknowledges the support of Pôle Algorithmique Répartie, C3 CNRS, the French National Program in Distributed Computing, and of a Grant from the Ministère de la Recherche et de la Technologie (Paris, France).

References

Ackley, D. H., Hinton, G. E., and Sejnowski, T. J. 1985. A learning algorithm for Boltzmann machines. *Cog. Sci.* 9, 147–169.

Almeida, L. B. 1987. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. *Proc. IEEE First International Conf. Neural Networks*, San Diego, CA, Vol. II, pp. 609–618.

Atalay, V., Gelenbe, E., and Yalabik, N. 1991. Texture generation with the random neural network model. In *Artificial Neural Networks*, Vol. I, T. Kohonen, ed., pp. 111–117. North-Holland, Amsterdam.

Baum, E. B. 1991. Neural net algorithms that learn in polynomial time from examples and queries. Draft Paper May 11 (private communication).

Behrens, H., Gawronska, D., Hollatz, J., and Schürmann, B. 1991. Recurrent and feedforward backpropagation: Performance studies. In *Artificial Neural Networks*, Vol. II, T. Kohonen et al., eds., pp. 1511–1514. North-Holland, Amsterdam.

Gelenbe, E. 1990. Stability of the random neural network model. *Neural Comp.* 2(2), 239–247.

Gelenbe, E. 1989. Random neural networks with negative and positive signals and product form solution. *Neural Comp.* 1(4), 502–510.

Gelenbe, E., and Batty, F. 1992. Minimum cost graph covering with the random network model. *ORSA TC on Computer Science Conference*, Williamsburg, VA, January. Pergamon Press, Oxford.

Gelenbe, E., Stafilopatis, A., and Likas, A. 1991. In *Artificial Neural Networks*, Vol. I, T. Kohonen, ed., pp. 307–315. North-Holland, Amsterdam.

Kandel, E. C., and Schwartz, J. H. 1985. *Principles of Neural Science*, Elsevier, Amsterdam.

Kemeny, J. G., and Snell, J. L. 1965. *Finite Markov Chains*. Van Nostrand, Princeton, NJ.

Le Cun, Y. 1985. A learning procedure for asymmetric threshold networks. *Proc. Cognitiva* 85, 599–604.

Mokhtari, M. 1992. Recognition of typed images with the random network model. *Int. J. Artificial Intelligence Pattern Recognition*, in press.

Pearlmutter, B. A. 1989. Learning state space trajectories in recurrent neural networks. *Neural Comp.* 1(2), 263–269.

Pineda, F. J. 1987. Generalization of backpropagation to recurrent and higher order neural networks. In *Neural Information Processing Systems*, D. Z. Anderson, ed., p. 602. American Institute of Physics.

Pineda, F. J. 1989. Recurrent backpropagation and the dynamical approach to adaptive neural computation. *Neural Comp.* 1(2), 161–172.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. 1986. Learning internal representations by error propagation. In *Parallel Distributed Processing*, D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, eds., Vols. I and II, Bradford Books and MIT Press, Cambridge, MA.

Received 3 September 1991; accepted 27 May 1992.