Recurrent Backpropagation and the Dynamical Approach to Adaptive Neural Computation

Fernando J. Pineda
Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, CA 91109, USA

Error backpropagation in feedforward neural network models is a popular learning algorithm that has its roots in nonlinear estimation and optimization. It is being used routinely to calculate error gradients in nonlinear systems with hundreds of thousands of parameters. However, the classical architecture for backpropagation has severe restrictions. The extension of backpropagation to networks with recurrent connections will be reviewed. It is now possible to efficiently compute the error gradients for networks that have temporal dynamics, which opens applications to a host of problems in systems identification and control.

1 Introduction

The problem of loading a neural network model with a nonlinear mapping is like the problem of finding the parameters of a multidimensional nonlinear curve fit. The traditional way of estimating the parameters is to minimize a measure of the error between the actual output and the "target" output. Many useful optimization techniques exist, but the most common methods make use of gradient information. In general, if there are N free parameters in the objective function, the number of operations required to calculate the gradient numerically is at best proportional to N^2. Neural networks are special because their *mathematical* form permits two tricks that reduce the complexity of the gradient calculation, as discussed below. When these two tricks are implemented, the gradient calculation scales linearly with the number of parameters (weights), rather than quadratically. The resulting algorithm is known as a backpropagation algorithm.

Classical backpropagation was introduced to the neural network community by Rumelhart, Hinton and Williams (1986). Essentially the same algorithm was developed independently by Werbos (1974) and Parker (1982) in different contexts. Le Cun (1988) has provided a brief overview of backpropagation pre-history and stresses that the independent discovery of the technique and its interpretation in the context of connectionist

Neural Computation **1, 161–172** (1989)

systems is a recent and important development. He points out that within the framework of optimal control the essential features of the algorithm were known even earlier (Bryson and Ho 1969).

In this paper, the term "backpropagation" will be used generically to refer to any technique that calculates the gradient by exploiting the two tricks. Furthermore, since one can write a backpropagation routine for evaluating the gradient and then use this routine in any prepackaged numerical optimization package, it is reasonable to take the position that the term "backpropagation" should be attached to the way the gradient is calculated rather than to the particular algorithm for using the gradient (conjugate gradient, line search, etc.).

Recurrent backpropagation is a non-algorithmic continuous-time formalism for adaptive recurrent and nonrecurrent networks in which the dynamical aspects of the computation are stressed (Pineda 1987a; 1987b; 1988). The formalism is expressed in the language of differential equations so that the connection to collective physical systems is more natural. Recurrent backpropagation can be put into an algorithmic form to optimize the performance of the network on digital machines, nevertheless, the intent of the formalism is to stay as close to collective dynamics as possible.

Recurrent backpropagation has proven to be a rich and useful computational tool. Qian and Sejnowski (1988) have demonstrated that a recurrent backpropagation network can be trained to calculate stereo disparity in random-dot stereograms. For dense disparity maps the network converges to the algorithm introduced by Marr and Poggio (1976) and for sparse disparity maps it converges to a new algorithm for transparent surfaces. Barhen et al. (1989) have developed a new methodology for constrained supervised learning and have extended RBP to handle constraints and to include terminal attractors (Zak 1988). They have applied their algorithms to inverse kinematics in robotic applications. The formalism has also been fertile soil for theoretical developments. Pearlmutter (1989) has extended the technique to time-dependent trajectories while Simard et al. (1988) have investigated its convergence properties.

2 Overview of a Dynamical Model

The class of neural network models which can be trained by recurrent backpropagation is very general, but it is useful to pick a definite system as an example, therefore consider a neural network model based on differential equations of the form

$$\tau_x dx_i/dt = -x_i + \sum_j w_{ij} f(x_j) + I_i \ . \tag{2.1}$$

The vector **x** represents the state vector of the network, **I** represents an external input vector and **w** represents a matrix of coupling constants

(weights) which represent the strengths of the interactions between the various neurons. The relaxation time scale is τ_x. By hypothesis, the vector valued function $f(x_i)$ is differentiable and chosen so as to give the system appropriate dynamical properties. For example, biologically motivated choices are the logistic or hyperbolic tangent functions (Cowan 1968). When the matrix $\mathbf{w}$ is symmetric with zero diagonals, this system corresponds to the Hopfield model with graded neurons (1984).

In general, the solutions of equation (2.1) exhibit oscillations, convergence onto isolated fixed points or chaos. For our purposes, convergence onto isolated fixed points is the desired behavior, because we use the value of the fixed point as the output of the system. When the network is loaded, the weights are adjusted so that the output of the network is the desired output.

There are several ways to guarantee convergence. One way is to impose structure on the connectivity of the network, such as requiring the weight matrix to be lower triangular or symmetric. Symmetry, although mathematically elegant, is quite stringent because it constrains microscopic connectivity by requiring pairs of neurons to be symmetrically connected. A less stringent constraint is to require that the Jacobian matrix be diagonally dominant. For equation (2.1), the Jacobian matrix has the form

$$L_{ij} = \delta_{ij} - w_{ij} f'(x_j) \tag{2.2}$$

where δ_{ij} are the elements of the identity matrix and $f'(x_j)$ is the derivative of $f(x_j)$. This condition has been used by Guez et al. (1988).

If the feedforward, symmetry or diagonal dominance stability conditions are imposed as initial conditions on a network, gradient descent dynamics will typically evolve a network which violates the conditions. Nevertheless, this author has never observed an initially stable network becoming unstable while undergoing simple gradient descent dynamics. This fact points out that the above stability conditions are merely sufficient conditions — they are not necessary. This fact also motivates the stability assumption upon which recurrent backpropagation on equation (2.1) is based: that if the initial network is stable, then the gradient descent dynamics will not change the stability of the network. The need for this assumption can be eliminated by choosing a dynamical system which admits only stable behavior, even under learning, as was done by Barhen et al. (1989).

In gradient descent learning, the computational problem is to optimize an objective function whose free parameters are the weights. Let the number of weights be denoted by "N" and let the number of processing units be denoted by "n". Then, N is proportional to n^2 provided the fan-in/fan-out of the units is proportional to n. For the neural network given by equation (2.1) it requires $O(mN)$ or $O(mn^2)$ operations to relax the network and to calculate a separable objective function based on the steady state $\mathbf{x}^0$. (In this discussion, the precision of the calculation is fixed

and m is the number of time steps required to relax equation (2.1) to a given tolerance.) Accordingly, to calculate the gradient of the objective function by numerical differentiation requires $O(mN^2)$ or $O(mn^4)$ calculations. For problems with lots of connections this becomes intractable very rapidly. The scaling referred to here should not be confused with the number of gradient evaluations required for convergence to a solution. Indeed, for some problems, such as parity, the required number of gradient evaluations may diverge at critical training set sizes (Tesauro 1987).

Now, as already mentioned, backpropagation adaptive dynamics is based on gradient descent and exploits two tricks to reduce the amount of computation. The first trick uses the fact that, for equations of the form (2.1), the gradient of an objective function $E(\mathbf{x}^0)$ can be written as an outer-product, that is,

$$\nabla_w E = \mathbf{y}^0 \mathbf{f}(\mathbf{x}^0)^T \tag{2.3}$$

where $\mathbf{x}^0$ is the fixed point of equation (2.1) and where the "error vector" $\mathbf{y}^0$ is given by

$$\mathbf{y}^0 = (\mathbf{L}^T)^{-1}\mathbf{J} \tag{2.4}$$

where $\mathbf{L}^T$ is the transpose of the $n \times n$ matrix defined in equation (2.2) and $\mathbf{J}$ is an external error signal which depends on the objective function and on $\mathbf{x}^0$. This trick reduces the computational complexity of the gradient calculation by a factor of n because $\mathbf{L}^{-1}$ can be calculated from $\mathbf{L}$ by direct matrix inversion in $O(n^3)$ operations and because $\mathbf{x}^0$ can be calculated in only $O(mn^2)$ calculations. Thus the entire calculation scales like $O(mn^3)$ or $O(mN^{3/2})$.

The second trick exploits the fact that $\mathbf{y}^0$ can be calculated by relaxation or equivalently it is the (stable) fixed point of the linear differential equation

$$\tau_y dy_i/dt = -y_i + f'(x_i)\sum_j w_{ji}y_j + J_i \ . \tag{2.5}$$

A form of this equation was derived by Pineda (1987b). A discrete-time version was derived independently by Almeida (1987). To relax $\mathbf{y}$ (that is, to integrate equation (2.5) until $\mathbf{y}$ reaches steady state) requires $O(n^2)$ operations per time step. Therefore, if the system does not wander chaotically, the required amount of computation scales like $O(mn^2)$ or $O(mN)$. The method is computationally efficient provided the network is sufficiently large and sparse and provided that the fixed points are not marginally stable. These results are summarized in Table 1. Note that the two backpropagation algorithms have reduced the amount of computation by a factor of N. The classical feedforward algorithm is more efficient because it does not have to relax to a steady state.

For all its faults, backpropagation has permitted optimization techniques to be applied to many problems which were previously considered

Numerical algorithm complexity:	
Worst case (e.g. numerical differentiation)	$O(mN^2)$
Matrix inversion (e.g. gaussian elimination)	$O(mN^{3/2})$
Matrix inversion by relaxation (e.g. recurrent backpropagation)	$O(mN)$
Recursion (e.g. classical feedforward back-propagation)	$O(N)$

Table 1: Scaling of various algorithms with the number of connections. m is the number of time steps required to relax equations (2.1) or (2.5). It is assumed that the number of time steps required to relax them is the same.

numerically intractable. The N-fold reduction of the amount of calculation is perhaps the single most important reason that backpropagation algorithms have made such an impact in neural computation. The idea of using gradient descent is certainly not new, but whereas it was previously only tractable on problems with few parameters, it is now also tractable on problems with many parameters. It is interesting to observe that a similar situation arose after the development of the FFT algorithm. The idea of numerical fourier transforms had been around for a long time before the FFT, but the FFT caused a computational revolution by reducing the complexity of an n-point fourier transform, from $O(n^2)$ to $O(n \log(n))$.

3 Dynamics vs. Algorithms

Backpropagation algorithms are usually viewed from an algorithmic viewpoint. For example, the gradient descent version of the algorithm is expressed in the following pseudo-code:

```
while(E > ε)
{
  initialize weight change Δw = 0
  repeat for each pattern
  {
    relax eqn. (2.1) to obtain x^0
    relax eqn. (2.4) to obtain y^0
    calculate gradient ∇E = y^0 f(x^0)^T
    accumulate gradients Δw = Δw + ∇E
  }
```

update weights $\mathbf{w} + \mathbf{w} + \Delta\mathbf{w}$

}

Note that all the patterns are presented before a weight update. On the other hand, a "dynamical algorithm" can be obtained by replacing the weight update step with a differential equation, that is,

$$\tau_w dw_{ij}/dt = y_i f(x_j) \tag{3.1}$$

and integrating it simultaneously with the forward-propagation and backward-propagation equations. A constant pattern is presented through the input pattern vector, **I**, and the error signal is presented through the error vector, **J**. The dynamics of this system is capable of learning a single pattern so long as the relaxation time of the forward and backward propagations (τ_x and τ_y) is much slower than the relaxation time of the weights, τ_w. Since the forward and backward equations settle rapidly after a presentation, the outer product $\mathbf{yf(x)}^T$ is a very good approximation for the gradient during most of the integration. To learn multiple patterns, the patterns must be switched slowly compared to the relaxation time of the forward and backward equations, but rapidly compared to τ_w, the time scale over which the weights change.

The conceptual advantage of this approach is that one now has a dynamical system which can be studied and perhaps used as a basis for models of actual physical or biological systems. This is not to say that merely converting an algorithm into a dynamical form makes it biologically or physically plausible. It simply provides a starting point for further development and investigation.

Intuition and formal results concerning algorithmic models do not necessarily apply to the corresponding dynamical models. For example, consider the well-known "fact" that gradient descent is a poor algorithm compared to conjugate gradient. In fact this conventional wisdom is incorrect when it comes to physical dynamical systems. The reason is that the disease which makes gradient descent inefficient is a consequence of discretization. For example a difficulty occurs when descending down a long narrow valley. Gradient descent can wind up taking many tiny steps crossing and re-crossing the actual gradient direction. This is inefficient because the gradient must be recomputed for each step and because the amount of computation required to recalculate the gradient from one step to the next is approximately constant. Conjugate gradient is a technique which assures that the new direction is conjugate to the previous direction and therefore avoids the problem. Accordingly larger steps may be taken and less gradient evaluations are required.

On the other hand gradient descent is quite satisfactory in physical dynamical systems simply because time is continuous. The "steps" are by definition infinitely small and the gradient is evaluated continuously. No repeated crossing of the gradient direction occurs. For the same reason, the ultimate performance of physical neural networks cannot be determined from how quickly or how slowly a "neural" simulation runs on a

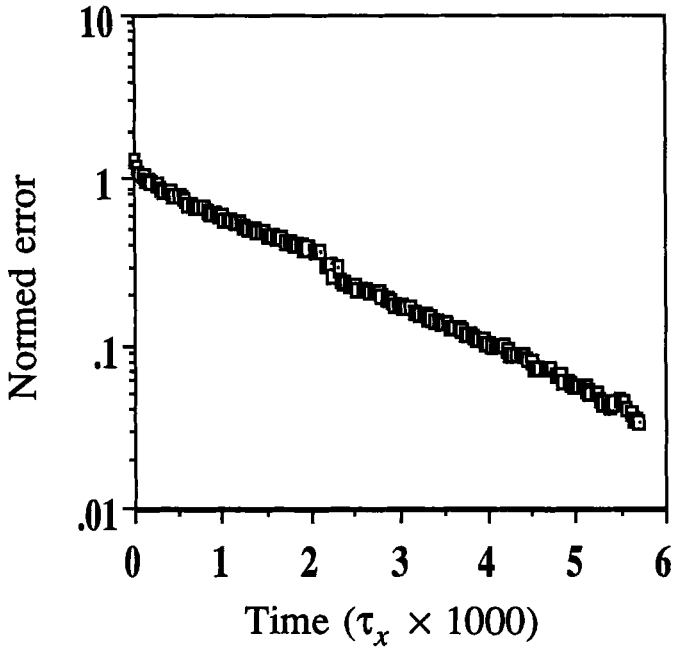

Figure 1: Mean squared error as a function of time.

digital machine. Instead one must integrate the simultaneous equations and measure how long it takes to learn, in multiples of the fundamental time scales of the equations. As an example, consider the following illustrative problem. Choose input and output vectors to be randomly selected 5 digit binary vectors scaled between 0.1 and 0.9. Use a network with two layers of five units each with connections going in both directions (50 weights). For dimensionless time scales choose $\tau_x = \tau_y = 1.0$, $\tau_w = 32\tau_x$ and select a new pattern at random every $4\tau_x$. The equations may be integrated crudely, for example, use the Euler method with ($\Delta t = 0.02\tau_x$). One finds that the error reaches $E = 0.1$ in approximately $4 \times 10^3 \tau_x$ or after 10^3 presentations. Figure 1 shows the error as a function of time.

To estimate the performance of an electronic physical system we can replace these time scales with electronic time scales. Therefore, suppose patterns are presented every 10^{-5} sec (100 kHz). This is the performance bottleneck of the system, since the relaxation time of the circuit, τ_x, is then approximately 2.5×10^{-6} sec, which is slow compared with what

then approximately 2.5×10^{-6} sec, which is slow compared with what can be achieved in analog VLSI. Hence in this case the patterns would be learned in approximately 10 milliseconds.

Unlike simple feedforward networks, recurrent networks exhibit dynamical phenomena. For example, a peculiar phenomenon can occur if a recurrent network is trained as an associative memory to store multiple memories: it is found that the objective function can be reduced to some very small value, yet when the network is tested for recall, the supposedly stored memory is missing! This is due to a fundamental limitation of gradient descent. Gradient descent is capable of moving existing fixed points only. It cannot create new fixed points. To create new fixed points requires a technique whereby some degrees of freedom in the network are clamped during the loading phase and released during the recall phase. The analogous technique in feedforward networks is called "teacher forcing." It can be shown that this technique causes the creation of new fixed points. Unfortunately, after the suppressed degrees of freedom are released, there is no guarantee that the system is stable with respect to the suppressed degrees of freedom. Therefore the fixed points sometimes turn out to be repellers instead of attractors. In feedforward nets teacher forcing causes no such difficulties because there is no dynamics in feedforward networks and hence no attractors or repellers.

4 Recent Developments

Zak (1988) has suggested the use of fixed points with infinite stability in recurrent networks. These fixed points, denoted "terminal attractors," have two properties which follow from their infinite stability. First, their stability is always guaranteed, hence the repeller problem never occurs, and second, trajectories converge onto them in a finite amount of time, rather than an infinite amount of time. In particular, if a terminal attractor is used in the weight update equation, a remarkable speedup in learning time occurs (see for example Barhen et al. 1989). These interesting properties are a consequence of the fact that the attractors violate the Lipschitz condition.

Pearlmutter (1989) has extended the recurrent formalism to include time-dependent trajectories (time-dependent recurrent backpropagation). In this approach the objective function of the fixed point is replaced with an objective functional of the trajectory. The technique is the continous time generalization of the sequence generating network discussed by Rumelhart et al. (1986). Like all backpropagation algorithms the amount of calculation is reduced by $O(N)$ for each gradient evaluation. However, like the Rumelhart network, it requires that the network be unfolded in time during training. Hence the storage and computation during training scales like $O(mN)$ where m is the number of unfolded time steps. Furthermore, the technique is acausal in that the backpropagation equation

one is solving a two-point boundary problem of the kind familiar from control theory. For problems where the target trajectories are known a priori and on-line learning is not required, this is the technique of choice.

On the other hand a causal algorithm has been suggested by Williams and Zipser (1989). This algorithm does not take advantage of the back-propagation tricks and therefore the complexity scales like $O(mN^2)$ for each gradient evaluation while the storage scales like $O(n^3)$. Nevertheless, for small problems where on-line learning is required it is the technique of choice. Both techniques seek to minimize a measure of the error between a target trajectory and an actual trajectory by performing gradient descent. Only the method used for the gradient evaluation differs. Therefore one expects that, to the extent that on-line training is not an issue and to the extent that complexity is not an issue, one could use the two techniques interchangeably to create networks. Both techniques can suffer from the repeller problem if an attempt is made to introduce multiple attractors. As before, this problem could be solved by introducing a time dependent terminal attractor.

5 Constraints

Biologically and physically plausible adaptive systems which are massively parallel should satisfy certain constraints. 1) They should scale well with connectivity, 2) they should require little or no global synchronization, 3) they should use low precision components, and 4) they should not impose unreasonable structural contraints, such as symmetric weights or bi-directional signal propagation. Backpropagation algorithms in general and recurrent backpropagation and time-dependent recurrent backpropagation in particular can be viewed in light of each of these constraints.

Linear scaling of the gradient calculation in backpropagation algorithms is a consequence of the local nature of the computation; that is, each unit only requires information from the units to which it is connected. This notion of locality, which arises from the analysis of the numerical algorithm is distinct from the notion of spatial locality, which is a constraint imposed by physical space on physical networks. Spatial locality is how one avoids the $O(n^2)$ growth of wires in networks. Both locality constraints could be satisfied by physical backpropagation networks.

Global synchronization requires global connections, therefore it is undesirable if the network is to scale up. In one sense, the problem of synchronization has been eliminated in recurrent backpropagation because there is no longer any need for separate forward, backward and update steps, indeed equations (2.1), (2.5), and (3.1) are "integrated" simultaneously by the dynamical system as it evolves. There is another sense in which synchronization causes difficulties. In physical systems and in

massively parallel digital simulations, time delays and asynchronous updates can give rise to chaotic or exponential stochastic behavior (Barhen and Golati 1989). Barhen et al. have shown that this "emergent chaos" can be suppressed easily by the appropriate choice of dynamical parameters.

It is still an open question as to whether backpropagation algorithms require low precision or high precision components. Formal results suggest that some problems, like parity in single layer nets (Minsky and Papert 1988), may lead to exponential growth of weights. In practice it appears that 16 bits of precision for the weights and 8 bits of precision for the activations and error signals are sufficient for many useful problems (Durbin 1987).

Structurally, recurrent backpropagation and time-dependent recurrent backpropagation impose no constraints on the weight matrix. This would help the biological plausiblity of the model were it not for the requirement that the connections be bi-directional. Bi-directionality is perhaps the biggest plausibility problem with the algorithms based on backpropagation. Biologically, this requires bi-directional synapses or separate, but equal and opposite, paths for error and activation signals. There is no evidence for either structure in biological systems. The same difficulties arise in electronic implementations where engineering solutions to this problem have been developed (Furman and Abidi 1988), but one would hope that a better adaptive dynamics would eliminate the problem altogether.

6 Discussion

If neural networks were merely clever numerical algorithms it would be difficult to completely account for the recent excitement in the field. To my mind, much of the excitement started with the work of Hopfield (1982) who made explicit the profound relationship between information storage and dynamically stable configurations of collective physical systems. Hopfield nets are based on the physics of interacting spins which together form a system known as a spin glass. The relevant physical property of spin glasses which make them useful for computation is that the collective interactions between all the spins can result in stable patterns which can be identified with stored memories. Hopfield nets serve as an explicit example of the principle of collective computation even though they may not be the best networks for practical computing.

Digital computers, on the other hand, can compute because they are physical realizations of finite state machines. In digital computers collective dynamics does not play a role at the algorithm level, although it certainly plays a role at the implementation level since the physics of transistors is collective physics. Collective dynamics can play a role at both the algorithmic and the implementation levels if the physical

dynamics of the machine is reflected in the computation directly. Rather than search for machine-independent algorithms, one should search for just the opposite — dynamical algorithms that can fully exploit the collective behavior of physical hardware.

Acknowledgments

The author wishes to acknowledge very helpful discussions with Pierre Baldi, Richard Durbin, and Terrence Sejnowski.

The work described in this paper was performed at the Applied Physics Laboratory, The Johns Hopkins University, sponsored by the Air Force Office of Scientific Research (AFOSR-87-354). The writing and publication of this paper was supported by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply any endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

References

Almeida, L.B. 1987. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. *In:* Proceedings of the IEEE First International Conference on Neural Networks, San Diego, CA, eds. M. Caudil and C. Butler, **2**, 609–618.

Barhen, J., S. Gulati, and M. Zak. 1989. Neural learning of inverse kinematics for redundant manipulators in unstructured environments. To appear in: *IEEE Computer*, June 1989, Special issue on Autonomous Intelligent Machines.

Barhen, J. and S. Gulati. 1989. 'Chaotic relaxation' in concurrently asynchronous neurodynamics. Submitted to: 1989 International Joint Conference on Neural Networks, June 18–19, Washington, D.C.

Bryson, A.E. Jr. and Y-C. Ho. 1969. *Applied Optimal Control.* Blaisdell Publishing Co.

Cowan, J.D. 1968. Statistical mechanics of nervous nets. *In:* Neural Networks, ed. E.R. Caianiello. Berlin: Springer-Verlag, 181–188.

Durbin, R. 1987. Backpropagation with integers. Abstracts of the meeting, Neural Networks for Computing, Snowbird, UT.

Furman, B. and A. Abidi. 1988. A CMOS backward error propagation LSI. *Proceedings of the Twenty-second Asilomar Conference on Signals, Systems, and Computers.* Pacific Grove, CA.

Guez, A., V. Protopopsecu, and J. Barhen. 1988. On the stability, storage capacity, and design of nonlinear continuous neural networks. *IEEE Transactions on Systems, Man, and Cybernetics*, **18**, 80–87.

Hopfield, J.J. 1982. Neural networks as physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Science USA*, **79**, 2554–2558.

———. 1984. Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Science USA*, **81**, 3088–3092.

le Cun, Y. 1988. A theoretical framework for backpropagation. *Proceedings of the 1988 Connectionist Models Summer School*, Carnegie-Mellon University, eds. D. Touretzky, G. Hinton, T. Sejnowski, 21–29. San Mateo, CA: Morgan-Kaufmann Publishers.

Marr, D. and T. Poggio. 1976. Cooperative computation of stereo disparity. *Science*, **194**, 283–287.

Minsky, M. and S. Papert. 1988. *Perceptrons*, 2nd edition. Cambridge, MA: MIT Press.

Parker, David B. 1982. *Learning-Logic*. Invention Report, S81-64, File 1, Office of Technology Licensing, Stanford University.

Pearlmutter, Barak A. 1989. Learning state space trajectories in recurrent neural networks. *Neural Computation*, **1** 263–269.

Pineda, F.J. 1987a. Generalization of backpropagation to recurrent neural networks. *Physical Review Letters*, **18**, 2229–2232.

———. 1987b. Generalization of backpropagation to recurrent and higher order networks. *In:* Proceedings of IEEE Conference on Neural Information Processing Systems, Denver, Colorado, Nov. 8–12, ed. D.Z. Anderson, 602–611.

———. 1988. Dynamics and architecture for neural computation. *Journal of Complexity*, **4**, 216–245.

Qian, N. and T.J. Sejnowski. 1988. Learning to solve random-dot stereograms of dense transparent surfaces with recurrent backpropagation. *In:* Proceedings of the 1988 Connectionist Models Summer School, eds. D. Touretzky, G. Hinton and T. Sejnowski, 435–443. San Mateo, CA: Morgan-Kaufmann Publishers.

Rumelhart, D.E., G.E. Hinton, and R.J. Williams. 1986. Learning internal representations by error propagation. *In:* Parallel Distributed Processing, **1**, eds. D.E. Rumelhart and J.L. McClelland, 318–362.

Simard, P.Y., M.B. Ottaway, and D.H. Ballard. 1988. Analysis of recurrent backpropagation. *Proceedings of the 1988 Connectionist Models Summer School*, June 17–26, 1988, Carnegie Mellon Univ., Morgan-Kaufmann Publishers.

Tesauro, G. 1987. Scaling relationships in backpropagation learning: dependence on training set size. *Complex Systems*, **1**, 367–372.

Werbos, P. 1974. *Beyond regression: New tools for prediction and analysis in the behavioral sciences*, Ph.D. thesis, Harvard University.

Williams, R.J. and D. Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, **1**, 270–280.

Zak, M. 1988. Terminal attractors for addressable memory in neural networks. *Physics Letters A*, **133**, 18–22.

Received 14 March 1989; accepted 17 April 1989.