Gradient Calculations for Dynamic Recurrent Neural Networks: A Survey

Barak A. Pearlmutter

Abstract—**We survey learning algorithms for recurrent neural networks with hidden units and put the various techniques into a common framework. We discuss fixed point learning algorithms, namely recurrent backpropagation and deterministic Boltzmann machines, and nonfixed point algorithms, namely backpropagation through time, Elman's history cutoff, and Jordan's output feedback architecture. Forward propagation, an on-line technique that uses adjoint equations, and variations thereof, are also discussed. In many cases, the unified presentation leads to generalizations of various sorts. We discuss advantages and disadvantages of temporally continuous neural networks in contrast to clocked ones continue with some "tricks of the trade" for training, using, and simulating continuous time and recurrent neural networks. We present some simulations, and at the end, address issues of computational complexity and learning speed.**

I. Introduction

A. Why Recurrent Networks?

THE motivation for exploring recurrent architectures is their potential for dealing with two sorts of temporal behavior. First, recurrent networks are capable of settling to a solution that satisfies many constraints [1], as in a vision system which relaxes to an interpretation of an image which maximally satisfies a complex set of conflicting constraints [2]–[6], a system which relaxes to find a posture for a robot satisfying many criteria [7], and models of language parsing [8]. Although algorithms suitable for building systems of this type are reviewed to some extent below, such as the algorithm used in [9], the bulk of this paper is concerned with the problem of causing networks to exhibit particular desired detailed temporal behavior, which has found application in signal processing [10], [11], speech and language processing [12]–[14], and neuroscience [15]–[17].

It should be noted by engineers that many real-world problems which one might think would require recurrent architectures for their solution turn out to be solvable with feedforward architectures, sometimes augmented with preprocessed inputs such as tapped delay lines and various other architectural embellishments [18]–[35], [10]. For this reason, if one is interested in solving a particular problem, it would be only prudent to try a variety of nonrecurrent architectures before resorting to the more powerful and general recurrent networks.

Manuscript received June 4, 1993; revised February 4, 1994 and July 17, 1994. This research was sponsored in part by NSF EET-8716324, by ONR N00014-86-K-0678, and by a Fannie and John Hertz Foundation Fellowship.

The author is with the Learning Systems Department at Siemens Corporate Research, Princeton, NJ 08540 USA.

IEEE Log Number 9409369.

This paper is concerned with learning algorithms for recurrent networks themselves and not with recurrent networks as elements of larger systems, such as specialized architectures for control [36]–[39]. Also, since we are concerned with learning, we will not discuss the computational power of recurrent networks considered as abstract machines [40]–[42]. Although we consider techniques for trajectory learning, we will not review practical applications thereof. In particular, grammar learning, although intriguing and progressing rapidly [43]–[49], typically involves recurrent neural networks as components of more complex systems and also at present is inferior in practice to discrete algorithmic techniques [50], [51]. Grammar learning is therefore beyond our scope here. Similarly, learning of multiscale phenomena, which again typically consists of larger systems containing recurrent networks as components [52]–[55], will not be discussed.

B. Why Hidden Units?

We will restrict our attention to training procedures for networks which may include hidden units, units which have no particular desired behavior and are not directly involved in the input or output of the network. For the biologically inclined, they can be thought of as interneurons.

With the practical successes of backpropagation, it seems gratuitous to expound the virtues of hidden units and internal representations. Hidden units make it possible for networks to discover and exploit regularities of the task at hand, such as symmetries or replicated structure [56], [57], and training procedures capable of exploiting hidden units, such as the Boltzmann machine learning procedure [58] and backpropagation [59]–[62], are behind much of the current excitement in the neural network field [63]. Also, training algorithms that do not operate with hidden units, such as the Widrow–Hoff LMS procedure [64], can be used to train recurrent networks without hidden units, so recurrent networks without hidden units reduce to nonrecurrent networks without hidden units, and therefore do not need special learning algorithms.

Consider a neural network governed by the equation

$$\frac{dy}{dt} = f(y(t), w, I(t)) \tag{1}$$

where y is the time-varying state vector, w the parameters to be modified by the learning, and I a time-varying vector of external input. Given some error metric $E'(y,t)$, our task is to modify w to reduce $E = \int E'(y,t)\,dt$. Our strategy will be gradient descent, so the main portion of our work will be

finding algorithms to calculate the gradient $\nabla_w E$, the vector whose elements are $\partial E/\partial w_i$.

The above formulation is for a continuous-time system. The alternative to this is a clocked system, which obeys an equation of the form $y(t + \Delta t) = f(y(t), w, I(t))$. Without loss of generality, for clocked systems we will use $\Delta t = 1$, giving

$$y(t + 1) = f(y(t), w, I(t)) \tag{2}$$

with t an integer.

Certainly, barring high-frequency components in I, the behavior of (1) can be precisely duplicated by (2) with suitable choice of f in the latter. For this reason, to determine the practical trade-offs of one against the other, we must consider particular functional forms for f. We will consider the most common neural network formulation

$$\frac{dy_i}{dt} = -y_i + \sigma(x_i) + I_i \tag{3}$$

where y_i is the state or activation level of unit i

$$x_i = \sum_j w_{ji} y_j \tag{4}$$

is the total input to unit i, w_{ij} is the strength of the connection from unit i to unit j, and σ is a differentiable function.[1] The initial conditions $y_i(t_0)$ and driving functions $I_i(t)$ are the inputs to the system.

This defines a rather general dynamic system. Even assuming that the external input terms $I_i(t)$ are held constant, it is possible for the system to exhibit a wide range of asymptotic behaviors. The simplest is that the system reaches a stable fixed point; in the next section, we will discuss two different techniques for modifying the fixed points of networks that exhibit them.

More complicated possible asymptotic behaviors include limit cycles and even chaos. Later, we will describe a number of gradient-based training procedures that can be applied to training networks to exhibit desired limit cycles or particular detailed temporal behavior. We will not discuss specialized nongradient methods for learning limit cycle attractors, such as [66] and [67]. Although it has been theorized that chaotic dynamics play a significant computational role in the brain [68], [69], there are no specialized training procedures for chaotic attractors in networks with hidden units. Some [70], [71], have had success with the identification of chaotic systems using models without hidden state, and there is no reason to believe that learning the dynamics of chaotic systems is more difficult than learning the dynamics of nonchaotic ones.

Special learning algorithms are available for various restricted cases. There are fixed point learning algorithms (for details, see [72]–[75], or for a survey see [76]) that take advantage of the special relationships holding at a fixed point to reduce the storage requirements to $O(m)$, the number of weights, and the time requirements to the time required for the network to settle down. There are continuous-time feedforward learning algorithms that are as efficient in both time and space as algorithms for pure feedforward networks, but are applicable only when w is upper-triangular but not necessarily zero-diagonal in other words, when the network is feedforward except for recurrent self-connections [77]–[80], [25] or for a survey, [81].

Later, we will describe a number of training procedures that, for a price in space or time, do not rely on such restrictions and can be applied to training networks to exhibit desired limit cycles or particular detailed temporal behavior.

C. Continuous vs. Discrete Time

We will be concerned predominantly with continuous-time networks, as in (3). All of the learning procedures we will discuss, however, can be equally well applied to discrete-time systems, which obey equations like (2). Continuous-time has advantages for expository purposes, in that the derivative of the state of a unit with respect to time is well defined, allowing calculus to be used instead of tedious explicit temporal indexing, making for simpler derivations and exposition.

When a continuous-time system is simulated on a digital computer, it is usually converted into a set of simple first order difference equations, which is formally identical to a discrete-time network. Regarding the discrete-time network running on the computer as a simulation of a continuous-time network, however, has a number of advantages. First, more sophisticated and faster simulation techniques than simple first-order difference equations can be used [82]. Second, even if simple first-order equations are used, the size of the time step can be varied to suit changing circumstances; for instance, if the network is being used for a signal processing application and faster sensors and computers become available, the size of the time step could be decreased without retraining the network. Third, because continuous-time units are stiff in time, they tend to retain information better through time. Another way of putting this is that their bias in the learning theory sense is toward temporally continuous tasks, which is certainly advantageous if the task being performed is in fact temporally continuous.

Another advantage of continuous-time networks is somewhat more subtle. Even for tasks which themselves have no temporal content, such as constraint satisfaction, the natural way for a recurrent network to perform the required computation is for each unit to represent nearly the same thing at nearby points in time. Using continuous-time units makes this the default behavior; in the absence of other forces, units will tend to retain their state through time. In contrast, in discrete-time networks, there is no *a priori* reason for a unit's state at one point in time to have any special relationship to its state at the next point in time.

A pleasant added benefit of units tending to maintain their states through time is that it helps make information about the past decay more slowly, speeding up learning about the relationship between temporally distant events.

[1] Typically $\sigma(\xi) = (1+e^{-\xi})^{-1}$, in which case $\sigma'(\xi) = \sigma(\xi)(1-\sigma(\xi))$, or the scaled $\sigma(\xi) = \tanh(\xi)$, in which case $\sigma'(\xi) = (1 + \sigma(\xi))(1 - \sigma(\xi)) = 1 - \sigma^2(\xi)$. The latter symmetric squashing function is usually preferable, as it leads to a better conditioned Hessian, which speeds gradient descent [65]. The former, however, was used in all the simulations presented in this paper.

II. Learning in Networks with Fixed Points

The fixed point learning algorithms we will discuss assume that the networks involved converge to stable fixed points.[2] Networks that converge to fixed points are interesting because of the class of things they can compute, in particular constraint satisfaction and associative memory tasks. In such tasks, the problem is usually given to the network either by the initial conditions or by a constant external input, and the answer is given by the state of the network once it has reached its fixed point. This is precisely analogous to the relaxation algorithms used to solve such things as steady-state heat equations, except that the constraints need not have spatial structure or uniformity.

A. Will a Fixed Point Exist?

One problem with fixed points is that recurrent networks do not always converge to them. There are, however, a number of special cases that guarantee convergence to a fixed point.

- Some simple linear conditions on the weights, such as zero-diagonal symmetry ($w_{ij} = w_{ji}, w_{ii} = 0$) guarantee that the Lyapunov function

$$L = -\sum_{i,j} w_{ij} y_i y_j + \sum_i (y_i \log y_i + (1 - y_i)\log(1 - y_i)) \quad (5)$$

  decreases until a fixed point is reached [83]. This weight symmetry condition arises naturally if weights are considered to be Bayesian constraints, as in Boltzmann machines [84].
- A unique fixed point is reached regardless of initial conditions if $\sum_{ij} w_{ij}^2 < \max(\sigma')$ where $\max(\sigma')$ is the maximal value of $\sigma'(x)$ for any x [85], but in practice much weaker bounds on the weights seem to suffice, as indicated by empirical studies of the dynamics of networks with random weights [86].
- Other empirical studies indicate that applying fixed point learning algorithms stabilizes networks, causing them to exhibit asymptotic fixed point behavior [87], [88]. There is as yet no theoretical explanation for this phenomenon, and it has not been replicated with larger networks.

One algorithm that is capable of learning fixed points, but does not require the network being trained to settle to a fixed point to operate, is backpropagation through time [59]. This has been used to train a constraint satisfaction network for the eight queens problem, where shaping was used to gradually train a discrete-time network without hidden units to exhibit the desired attractors [89]. The other fixed point algorithms we will consider, however, take advantage of the special properties of a fixed point to simplify the learning algorithm.

B. Problems with Fixed Points

Even when it can be guaranteed that a network settles to a fixed point, fixed-point learning algorithms can still run into trouble. The learning procedures discussed here all compute the derivative of some error measure with respect to the internal parameters of the network. This gradient is then used by an optimization procedure, typically some variant of gradient descent, to minimize the error. Such optimization procedures assume that the mapping from the network's internal parameters to the consequent error is continuous and can fail spectacularly when this assumption is violated.

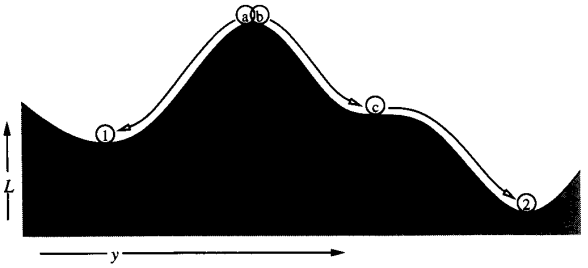


Fig. 1. This energy landscape, represented by the curved surface, and the balls, representing states of the network, illustrate some potential problems with fixed points. The initial conditions **a** and **b** can differ infinitesimally but map to different fixed points, so the mapping of initial conditions to fixed points is not continuous. Likewise, an infinitesimal change to the weights can change which fixed point the system evolves to from a given starting point by moving the boundary between the basins of attraction of two attractors. Similarly, point **c** can be changed from a fixed point to a nonfixed point by an infinitesimal change to the weights.

Consider mapping the initial conditions $\tilde{y}(t_0)$ to the resultant fixed points, $\tilde{y}(t_\infty) = \mathcal{F}(\tilde{y}(t_0))$. Although the dynamics of the network are all continuous, $\mathcal{F}$ need not be. For purposes of visualization, consider a symmetric network, whose dynamics thus cause the state of the network to descend the energy function of (5). As shown schematically in Fig. 1, even an infinitesimal change to the initial conditions, or to the location of a ridge, or to the slope of an intermediate point along the trajectory, can change which fixed point the system ends up in. In other words, $\mathcal{F}$ is not continuous. This means that as a learning algorithm changes the locations of the fixed points by changing the weights, it is possible for it to cross such a discontinuity, making the error jump suddenly; this remains true no matter how gradually the weights are changed.

C. Recurrent Backpropagation

It was shown independently by Pineda [72] and Alemeida [73] that the error backpropagation algorithm [59]–[61] is a special case of a more general error gradient computation procedure. The backpropagation equations are

$$x_i = \sum_j w_{ji} y_j$$

$$y_i = \sigma(x_i) + I_i \quad (6)$$

$$z_i = \sigma'(x_i) \sum_j w_{ij} z_j + e_i \quad (7)$$

$$\frac{\partial E}{\partial w_{ij}} = y_i z_j \quad (8)$$

where z_i is the ordered partial derivative of E with respect to y_i as defined in [60], E is an error over $y(t_\infty)$, and $e_i = \partial E / \partial y_i(t_\infty)$ is the simple derivative of E with respect

[2] Technically, these algorithms only require that a fixed point be reached, not that it be stable. It is unlikely (with probability zero), however, that a network will converge to an unstable fixed point, and in practice the possibility of convergence to unstable fixed points can be safely ignored.

to the final state of a unit. In the original derivations of backpropagation, the weight matrix is assumed to be triangular with zero diagonal elements, which is another way of saying that the connections are acyclic. This ensures that a fixed point is reached, and allows it to be computed very efficiently in a single pass through the units. But the backpropagation equations remain valid even with recurrent connections, assuming a fixed point is found.

If we assume that (3) reaches a fixed point, which we will denote $y(t_\infty)$, then (6) must be satisfied. And if (6) is satisfied, and assuming we can find z_i that satisfy (7), then (8) will give us the derivatives we seek, even in the presence of recurrent connections. (For a simple task, [90] reports that reaching the precise fixed point is not crucial to learning.)

One way to compute a fixed point for (6) is to relax to a solution. By subtracting y_i from each side, we get

$$0 = -y_i + \sigma(x_i) + I_i.$$

At a fixed point, $dy_i/dt = 0$, so the equation

$$k\frac{dy_i}{dt} = -y_i + \sigma(x_i) + I_i$$

has the appropriate fixed points. Now we note that when $-y_i + \sigma(x_i) + I_i$ is greater than zero, we can reduce its value by increasing y_i, so under these circumstances dy_i/dt should be positive, so k should be greater than zero. We can choose $k = 1$, giving (3) as a technique for relaxing to a fixed point of (6).

Equation (7) is linear once y is determined (y appears in the equation through the intermediate variable x and also through the error terms e_i), so (7) has a unique solution. Any technique for solving a set of linear equations could be used. Since we are computing a fixed point of (6) using associated differential equation (3), it is tempting to do the same for (7) using

$$\frac{dz_i}{dt} = -z_i + \sigma'(x_i)\sum_j w_{ij}z_j + e_i. \tag{9}$$

These equations admit to direct analog implementation. In a real analog implementation, different time constants would probably be used for (3) and (9), and under the assumption that the time y and z spend settling is negligible compared to the time they spend at their fixed points and that the rate of weight change η is slow compared to the speed of presentation of new training samples, the weights would likely be updated continuously by an equation like

$$\frac{dw_{ij}}{dt} = -\eta\frac{dE}{dw_{ij}} = -\eta y_i z_j \tag{10}$$

or, if a momentum term $0 < \alpha < 1$ is desired

$$\frac{d^2w_{ij}}{dt^2} + (1-\alpha)\frac{dw_{ij}}{dt} + \eta y_i z_j = 0. \tag{11}$$

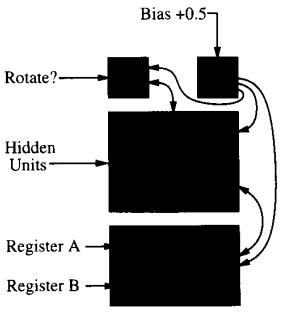


Fig. 2. The architecture of a network to solve an associative version of the four-bit rotation problem.

1) Simulation of an Associative Network: In this section we will simulate a recurrent backpropagation network learning a higher order associative task, that of associating three pieces of information: two four-bit shift registers, A and B, and a direction bit, D. If D is off, then B is equal to A. If D is on, then B is equal to A rotated one bit to the right, with wraparound. The task is to reconstruct one of these three pieces of information, given the other two.

The architecture of the network is shown in Fig. 2. Three groups of visible units hold A, B, and D. An undifferentiated group of 10 hidden units is fully and bidirectionally connected to all the visible units. There are no connections between visible units. An extra unit, called a bias unit, is used to implement thresholds. This unit has no incoming connections and is forced to always have a value of one by a constant external input of 0.5. Connections go from it to each other unit, allowing units to have biases, which are equivalent to the negative of the threshold, without complicating the mathematics. Inputs are represented by an external input of $+0.5$ for an on bit, -0.5 for an off bit, and zero for a bit to be completed by the network.

The network was trained by giving it external inputs that put randomly chosen consistent patterns on two of the three visible groups and training the third group to attain the correct value. The error metric was the squared deviation of each input–output unit from its desired state, except that units were not penalized for being "too correct."[3] All 96 patterns were successfully learned, except for the ones which were ambiguous, as shown in the state diagrams of Fig. 4. The weights after this training, which took about 300 epochs, are shown in Fig. 3. By inspection, many weights are large and decidedly asymmetric; but during training, no instabilities were observed. The network consistently settled to a fixed point within 20 simulated time units. When the network was tested on untrained completion problems, such as reconstructing D as well as half of A and B from partially, but unambiguously, specified A and B, performance was poor. Redoing the training with weight symmetry enforced, however, caused the network to learn not only the training data but also to do well on these untrained completions.

[3] A unit with external input could be pushed outside the [0, 1] bounds of the range of the $\sigma(\cdot)$ used.

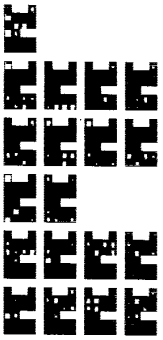


Fig. 3. A Hinton diagram of weights learned by the network of Fig. 2.

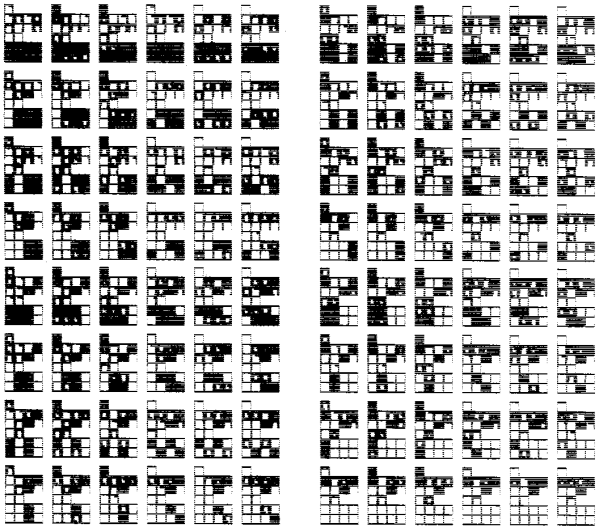


Fig. 4. Network state for all the cases in the four-bit rotation problem. This display shows the states of the units, arranged as in Fig. 2. Each row of six shows one value for register A. There are $2^4 = 16$ such rows. Within each row, the three diagrams on the left show the network's state when competing the direction bit, register B, and register A, unshifted. The right three are the same, except with a shift. Note that all completions are correct except in the two cases where the rotation bit can not be determined from the two shift registers, namely a pattern of 0000 or 1111.

[9] successfully applied the [72], [73] recurrent backpropagation learning procedure to learning weights for a relaxation procedure for dense stereo disparity problems with transparent surfaces. By training on examples, they were able to learn appropriate weights instead of deriving them from a simplified and unrealistic analytical model of the distribution of surfaces to be encountered, as is usual.

D. Deterministic Boltzmann Machines

The mean field form of the stochastic Boltzmann machine learning rule or MFT Boltzmann machines [91] have been shown to descend an error functional [74]. Stochastic Boltzmann machines themselves [58] are beyond our scope here; instead, we give only the probabilistic interpretation of MFT Boltzmann machines, without derivation.

In a deterministic Boltzmann machine, the transfer function of (3) is $\sigma(\xi) = (1 + e^{-\xi/T})^{-1}$, where T is the temperature, which starts at a high value and is gradually lowered to a target temperature each time the network is presented with a new input; without loss of generality, we assume this target temperature to be $T = 1$. The weights are assumed to be symmetric and zero-diagonal. Input is handled in a different way than in the other procedures we discuss: the external inputs I_i are set to zero, and a subset of the units, rather than obeying (3), have their values set externally. Such units are said to be clamped.

In learning, a set of input units (states over which we will index with α) are clamped to some values, the output units are similarly clamped to their correct corresponding values, the network is allowed to settle, and the quantities

$$p_{ij}^{+} = \langle y_i y_j \rangle = \sum_{\alpha,\beta} P(\alpha) y_i^{(\alpha,\beta)} y_j^{(\alpha,\beta)} \tag{12}$$

are accumulated, where $\langle \cdot \rangle$ denotes an average over the environmental distribution, the $+$ superscript denote clamping of both input and output, and α is used to index the input units and β indexes the output units. The same procedure is then repeated, but with the output units (states of which we will index by β) not clamped, yielding

$$p_{ij}^{-} = \langle y_i y_j \rangle = \sum_{\alpha} P(\alpha) y_i^{(\alpha)} y_j^{(\alpha)} \tag{13}$$

where the $-$ superscript denotes clamping of only the inputs and not the outputs. At this point, it is the case that

$$\frac{\partial G}{\partial w_{ij}} = p_{ij}^{+} - p_{ij}^{-} \tag{14}$$

where

$$G = \sum_{\alpha,\beta} P(\alpha) \log \frac{P(\beta \mid \alpha)}{P^{-}(\beta \mid \alpha)} \tag{15}$$

is a measure of the information theoretic difference between the clamped and unclamped distribution of the output units given the clamped input units. $P^{-}(\beta \mid \alpha)$ measures how probable the network says β is given α, and its definition is beyond the scope of this paper, while $P(\beta \mid \alpha)$ is the probability of β being the correct output when α is the input, as given by the target distribution to be learned.

This learning rule (14) is a version of Hebb's rule in which the sign of synaptic modification is alternated, positive during the "waking" phase and negative during the "hallucinating" phase.

Even before the learning rule was rigorously justified, deterministic Boltzmann machines were applied to a number of tasks [91], [92]. Although weight symmetry is assumed in the definition of energy which is used in the definition of probability, and is thus fundamental to these mathematics, it seems that in practice weight asymmetry can be tolerated in large networks [88]. This makes MFT Boltzmann machines the most biologically plausible of the various learning procedures we discuss, but it is difficult to see how it would be possible to extend them to learning more complex phenomena, like limit cycles or paths through state space. And thus, although

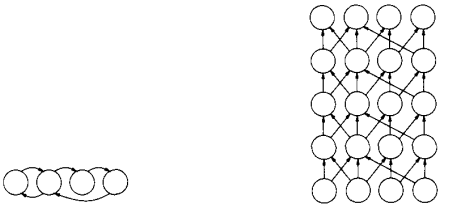


Fig. 5. A recurrent network is shown on the left, and of that network unfolded in time through four timesteps is shown on the right.

they are probably the best current technique in their domain of application, we now turn our attention to procedures suitable for learning more dynamic sorts of behaviors.

III. Computing the Gradient without Assuming a Fixed Point

Now we get to the heart of the matter—the computation of $\nabla_w E$, the gradient of the error E with respect to the vector of free parameters w, where the error is not defined at a fixed point but rather is a function of the network's detailed temporal behavior. The techniques we will discuss here, like those of Section II, are quite general purpose: they can accommodate hidden units as well as various architectural embellishments, such as second-order connections [93], [34], [94], [44], weight sharing [23], [35], and in general any of the architectural modifications made to neural networks to customize them for their problem domain. We will consider two major gradient calculation techniques and then a few more derived from them. The first is the obvious extension of backpropagation through time to continuous time [95], [96], [62].

A. Backpropagation Through Time

The fixed point learning procedures discussed above are unable to learn nonfixed point attractors or to produce desired temporal behavior over a bounded interval, or even to learn to reach their fixed points quickly. Here, we turn to a learning procedure suitable for such nonfixed point situations. This learning procedure essentially converts a network evolving through time into a network whose activation is flowing through a number of layers, translating time into space, as shown in Fig. 5. Backpropagation then becomes applicable. The technique is therefore called backpropagation through time, or BPTT.

Consider minimizing $E(\mathbf{y})$, some functional of the trajectory taken by $\mathbf{y}$ between t_0 and t_1. For instance, $E = \int_{t_0}^{t_1} (y_0(t) - d(t))^2 dt$ measures the deviation of $y_0(t)$ from the function $d(t)$, and minimizing this E would teach the network to have $y_0(t)$ imitate $d(t)$. Below, we derive a technique for computing $\partial E(\mathbf{y})/\partial w_{ij}$ efficiently, thus allowing us to do gradient descent in the weights so as to minimize E. Backpropagation through time has been used to train discrete-time networks to perform a variety of tasks [59], [89]. Here, we will derive the continuous-time version of backpropagation through time, as in [96], and use it in two toy domains.

In this derivation, we take the conceptually simple approach of unfolding the continuous-time network into a discrete-time network with a step of Δt, applying backpropagation to this discrete-time network and taking the limit as Δt approaches zero to get a continuous time learning rule. The derivative in (3) can be approximated with

$$\frac{dy_i}{dt}(t) \approx \frac{y_i(t + \Delta t) - y_i(t)}{\Delta t} \tag{16}$$

which yields a first-order difference approximation to (3)

$$\tilde{y}_i(t + \Delta t) = (1 - \Delta t)\tilde{y}_i(t) + \Delta t \sigma(\tilde{x}_i(t)) + \Delta t I_i(t). \tag{17}$$

Tildes are used throughout for temporally discretized versions of continuous functions.

Let us define e_i to be the first variation of E with respect to the function $y_i(t)$

$$e_i(t) = \frac{\delta E}{\delta y_i(t)}. \tag{18}$$

In the usual case E is of the form

$$E = \int_{t_0}^{t_1} f(\mathbf{y}(t), t)\, dt \tag{19}$$

so $e_i(t) = \partial f(\mathbf{y}(t), t)/\partial y_i(t)$. Intuitively, $e_i(t)$ measures how much a small change to y_i at time t affects E if everything else is left unchanged.

As usual in backpropagation, let us define

$$\tilde{z}_i(t) = \frac{\partial^+ E}{\partial \tilde{y}_i(t)} \tag{20}$$

where the ∂^+ denotes the ordered derivative of [97], with variables ordered here by time and not unit index. Intuitively, $\tilde{z}_i(t)$ measures how much a small change to $\tilde{y}_i$ at time t affects E when this change is propagated forward through time and influences the remainder of the trajectory, as in Fig. 7. Of course, z_i is the limit of $\tilde{z}_i$ as $\Delta t \to 0$. This z is the δ of the standard backpropagation "generalized δ rule."

We can use the chain rule for ordered derivatives to calculate $\tilde{z}_i(t)$ in terms of the $\tilde{z}_j(t + \Delta t)$. According to the chain rule, we add all the separate influences that varying $\tilde{y}_i(t)$ has on E. It has a direct contribution of $\Delta t e_i(t)$, which comprises the first term of our equation for $\tilde{z}_i(t)$. Varying $\tilde{y}_i(t)$ by $d\tilde{y}_i(t)$ has an effect on $\tilde{y}_i(t + \Delta t)$ of $d\tilde{y}_i(t)\ (1 - \Delta t)$, giving us a second term, namely $(1 - \Delta t)\tilde{z}(t + \Delta t)$.

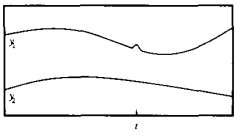


Fig. 6. The infinitesimal changes to $\mathbf{y}$ considered in $e_1(t)$.

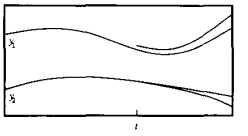


Fig. 7. The infinitesimal changes to $\mathbf{y}$ considered in $z_1(t)$.

Each weight w_{ij} makes $\tilde{y}_i(t)$ influence $\tilde{y}_j(t + \Delta t)$, $i \neq j$. Let us compute this influence in stages. Varying $\tilde{y}_i(t)$ by $d\tilde{y}_i(t)$ varies $\tilde{x}_j(t)$ by $d\tilde{y}_i(t)w_{ij}$, which varies $\sigma(\tilde{x}_j(t))$ by $d\tilde{y}_i(t)w_{ij}\sigma'(\tilde{x}_j(t))$, which varies $\tilde{y}_j(t + \Delta t)$ by $d\tilde{y}_i(t)w_{ij}\sigma'(\tilde{x}_j(t))\Delta t$. This gives us our third and final term, $\sum_j w_{ij}\ \sigma'(\tilde{x}_j(t))\ \Delta t\tilde{z}_j(t + \Delta t)$. Combining these

$$\tilde{z}_i(t) = \Delta t\, e_i(t) + (1 - \Delta t)\tilde{z}_i(t + \Delta t) + \sum_j w_{ij}\sigma'(\tilde{x}_j(t))\Delta t\tilde{z}_j(t + \Delta t). \tag{21}$$

If we put this in the form of (16) and take the limit as $\Delta t \to 0$ we obtain the differential equation

$$\frac{dz}{dt} = \frac{df(y, w, I)}{dy}z + \frac{\delta E}{\delta y} \tag{22}$$

$$\frac{dE}{dw} = \int_{t_0}^{t_1} y\frac{df(y, w, I)}{dw}z\, dt \tag{23}$$

with boundary condition $z(t_1) = 0$. Thus we have derived appropriate adjoint equations to (1). They are similar to the analogous discrete-time backward error equations

$$z(t - 1) = \frac{df(y, w, I)}{dy}z + \frac{\partial E}{\partial y(t)} \tag{24}$$

$$\frac{dE}{dw} = \sum_t y\frac{df(y, w, I)}{dw}z \tag{25}$$

where the error to be minimized is E. If this error is of the usual form of an integral $E = \int E'(y(t), t)\, dt$ then we get the simple form $\delta E/\delta y = dE'/dy$.

For the particular form of (3), this comes to

$$\frac{dz_i}{dt} = z_i - e_i - \sum_j w_{ij}\sigma'(x_j)z_j. \tag{26}$$

For boundary conditions note that by (18) and (20) $\tilde{z}_i(t_1) = \Delta t e_i(t_1)$, so in the limit as $\Delta t \to 0$ we have $z_i(t_1) = 0$.

Consider making an infinitesimal change dw_{ij} to w_{ij} for a period Δt starting at t. This will cause a corresponding infinitesimal change in E of $y_i(t)\sigma'(x_j(t))\Delta t z_j(t)\, dw_{ij}$. Since we wish to know the effect of making this infinitesimal change to w_{ij} throughout time, we integrate over the entire interval, yielding

$$\frac{\partial E}{\partial w_{ij}} = \int_{t_0}^{t_1} y_i\sigma'(x_j)z_j\, dt. \tag{27}$$

One can also derive (26), (27) and (37) using the calculus of variations and Lagrange multipliers, as in optimal control theory [98], [99]. In fact, the idea of using gradient descent to optimize complex systems was explored by control theorists in the late 1950's. Although their mathematical techniques and algorithms are identical to those reviewed here, and thus handled hidden units, they refrained from exploring systems with so many degrees of freedom, perhaps in fear of local minima.

It is also interesting to note that the recurrent backpropagation learning rule (Section II-C) can be derived from these. Let I_i be held constant, assume that the network settles to a fixed point, and let E be integrated for one time unit before t_1. As $t_1 \to \infty$, (26) and (27) reduce to the recurrent backpropagation (9) and (8), so in this sense BPTT is a generalization of recurrent backpropagation.

There are two ways to go about finding such derivations. One is direct, using the calculus of variations [98]. The other is to take the continuous-time equations, approximate them by difference equations, precisely calculate the adjoint equations for this discrete-time system, and then approximate back to get the continuous-time adjoint equations, as in [76]. An advantage of the latter approach is that, when simulating on a digital computer, one actually simulates the difference equations. The derivation ensures that the simulated adjoint difference equations are the precise adjoints to the simulated forward difference equations, so the computed derivatives contain no approximation errors.

B. Real Time Recurrent Learning

An on-line, exact, and stable, but computationally expensive, procedure for determining the derivatives of functions of the states of a dynamic system with respect to that system's internal parameters has been discovered and applied to recurrent neural networks a number of times [100]–[103]; for reviews see also [81], [104]. It is called by various researchers forward propagation, forward perturbation, or real time recurrent learning, (RTRL). Like BPTT, the technique was known and applied to other sorts of systems since the 1950's; for a hook into this literature see [105], [106] or the closely related extended Kalman filter [107]. In the general case of (1), RTRL is

$$\frac{dE}{dw} = \int_{t_0}^{t_1} \gamma\frac{\delta E}{\delta y}\, dt \tag{28}$$

where $\gamma(t_0) = 0$ and

$$\frac{d\gamma}{dt} = \frac{df(y, w, I)}{dw} + \frac{df(y, w, I)}{dy}\gamma. \tag{29}$$

The γ matrix is the sensitivity of the states $y(t)$ to a change of the weights w.

Under the assumption that the weights are changing slowly, RTRL can be made an on-line algorithm by updating the weights continuously instead of actually integrating (28)

$$\frac{dw}{dt} = -\eta\gamma\frac{\delta E}{\delta y} \tag{30}$$

where η is the learning rate, or, if a momentum term $0 < \alpha < 1$ is also desired

$$\alpha\frac{d^2w}{dt^2} + (1 - \alpha)\frac{dw}{dt} + \eta\gamma\frac{\delta E}{\delta y} = 0. \tag{31}$$

For the special case of a fully connected recurrent neural network, as described by (3), applying the general RTRL formulas above yields

$$\frac{d\gamma_{ijk}}{dt} = \frac{\partial f_k}{\partial y_k}\gamma_{ijk} + \left([j = k]y_j + \sum_l w_{lk}\gamma_{ijl}\right)\frac{\partial f_k}{\partial \operatorname{net}_k} \tag{32}$$

$$\frac{dw_{ij}}{dt}(t) = -\eta\sum_k \frac{\partial g}{\partial y_k}(t)\gamma_{ijk}(t). \tag{33}$$

Regrettably, the computation of γ is very expensive and also nonlocal. The γ array has nm elements, where n is the number of states and m the number of weights, which is typically on the order of n^2. Updating γ requires $O(n^3m)$ operations in the general case, but the particular structure of a neural network causes some of the matrixes to be sparse, which reduces the burden to $O(n^2m)$. This remains too high to make the technique practical for large networks. Nevertheless, because of its ease of implementation, RTRL is used by many researchers working with small networks.

C. Less Computationally Burdensome on-line Techniques

One way to reduce the complexity of the RTRL algorithm is to simply leave out elements of γ that one has reason to believe will remain approximately zero. This approach, in particular ignoring the coupling terms which relate the states of units in one module to weights in another, has been explored by Zipser [108].

Another is to use BPTT with a history cutoff of k units of time, termed BPTT(k) by Williams and Peng [109], and make a small weight change each timestep. This obviates the need for epochs, resulting in a purely on-line technique and is probably the best technique for most practical problems.

A third is to take blocks of s timesteps using BPTT, but use RTRL to encapsulate the history before the start of each block. This requires $O(s^{-1}n^2m+nm)$ time per step, on average, and $O(nm+sm)$ space. Choosing $s=n$ makes this $O(nm)$ time and $O(nm)$ space, which dominates RTRL. This technique has been discovered independently a number of times [110], [111].

Finally, one can note that, although the forward equations for y are nonlinear, and therefore require numeric integration, the backward equations for z in BPTT are linear. Since the dE/dw terms are linear integrations of the z, this means that they are linear functions of the external inputs, namely the e_i terms. As shown by Sun *et al.* [112], this allows one, during the forward pass, to compute a matrix relating the external error signal to the elements of ∇_w, allowing a fully on-line algorithm with $O(nm)$ time and space complexity.

D. Time Constants

A major advantage of temporally continuous networks is that one can add additional parameters that control the temporal behavior in ways known to relate to natural tasks. An example of this is time constants, which were learned in the context of neural networks in [79], [52], [53]. If we add a time constant T_i to each unit i, modifying (3) to

$$T_i\frac{dy_i}{dt} = -y_i + \sigma(x_i) + I_i \tag{34}$$

and carry these terms through the derivation of Section III-A, (26) and (27) become

$$\frac{dz_i}{dt} = \frac{1}{T_i}z_i - e_i - \sum_j \frac{1}{T_j}w_{ij}\sigma'(x_j)z_j \tag{35}$$

and

$$\frac{\partial E}{\partial w_{ij}} = \frac{1}{T_j}\int_{t_0}^{t_1} y_i\sigma'(x_j)z_j\,dt. \tag{36}$$

To learn these time constants rather than just set them by hand, we need to compute $\partial E(\mathbf{y})/\partial T_i$. If we substitute $\rho_i = T_i^{-1}$ into (34), find $\partial E/\partial\rho_i$ with a derivation similar to that of (27), and substitute T_i back in we get

$$\frac{\partial E}{\partial T_i} = -\frac{1}{T_i}\int_{t_0}^{t_1} z_i\frac{dy_i}{dt}\,dt. \tag{37}$$

E. Time Delays

Consider a network in which signals take finite time to travel over each link, so that (4) is modified to

$$x_i(t) = \sum_j w_{ji}y_j(t-\tau_{ji}) \tag{38}$$

τ_{ji} being the time delay along the connection from unit j to unit i. Let us include the variable time constants of Section III-D as well. Such time delays merely add analogous time delays to (35) and (36)

$$\frac{dz_i}{dt}(t) = \frac{1}{T_i}z_i(t) - e_i(t) - \sum_j w_{ij}\sigma'(x_j(t+\tau_{ij})) \times \frac{1}{T_j}z_j(t+\tau_{ij}). \tag{39}$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{1}{T_j}\int_{t_0}^{t_1} y_i(t)\sigma'(x_j(t+\tau_{ij}))z_j(t+\tau_{ij})\,dt \tag{40}$$

while (37) remains unchanged. If we set $\tau_{ij} = \Delta t$, these modified equations alleviate concern over time skew when simulating networks of this sort, obviating any need for accurate numerical simulations of the differential equations and allowing simple difference equations to be used without fear of inaccurate error derivatives.

Instead of regarding the time delays as a fixed part of the architecture, we can imagine modifiable time delays. Given modifiable time delays, we would like to be able to learn appropriate values for them, which can be accomplished using gradient descent by

$$\frac{\partial E}{\partial \tau_{ij}} = \int_{t_0}^{t_1} z_j(t)\sigma'(x_j(t))w_{ij}\frac{dy_i}{dt}(t-\tau_{ij})\,dt. \tag{41}$$

Reference [12] applied recurrent networks with immutable time delays in the domain of speech. Feedforward networks with immutable time delays (TDNN's) have been applied with great success in the same domain by Lang *et al.* [22]. A variant of TDNN's which learn the time delays was explored by [113]. The synapses in their networks, rather than having point taps, have Gaussian envelopes whose widths and centers were both learned. Similar synaptic architectures using alpha function envelopes (which obviate the need for a history buffer) whose parameters were learned were proposed and used in systems without hidden units [114], [29]. A continuous time feedforward network with learned time delays was successfully applied to a difficult time-series prediction task by [25].

In the sections on time constants and delays, we have carried out the derivative derivations for BPTT. All the other techniques also remain applicable to this case, with straightforward derivations. The analogous derivations for RTRL are carried

out in [76]. We will not, however, simulate here networks with modifiable time delays.

An interesting class of architectures would have the state of one unit modulate the time delay along some arbitrary link in the network or the time constant of some other unit. Such a "higher-order time delay" architecture seems appropriate for tasks in which time warping is an issue, such as speech recognition. The gradients with respect to higher-order time delay can be readily calculated by appropriate augmentation of either BPTT or RTRL.

In the presence of time delays, it is reasonable to have more than one connection between a single pair of units, with different time delays along the different connections. Such "time delay neural networks" have proven useful in the domain of speech recognition [20]–[22], [115]. Having more than one connection from one unit to another requires us to modify our notation somewhat; weights and time delays are modified to take a single index, and we introduce some external apparatus to specify the source and destination of each connection. Thus w_i is the weight on a connection between unit $\mathcal{L}(i)$ and unit $\mathcal{R}(i)$, and τ_i is the time delay along that connection. Using this notation we write (38) as

$$x_i(t) = \sum_{j|\mathcal{L}(j)=i} w_j y_{\mathcal{R}(j)}(t - \tau_j). \tag{42}$$

Our equations would be more general if written in this notation, but readability would suffer, and the translation is quite mechanical.

F. Extending RTRL to Time Constants and Time Delays

We have seen that BPTT can be easily applied to these new sorts of free parameters we have been adding to our networks, namely time constants and time delays. Other gradient calculation procedures also can be naturally applied to these new sorts of free parameters. In this section, we apply RTRL, first to incorporate time constants and then time delays.

If we begin with (34), first we must generalize (32) and (33) to correctly modify the weights in the presence of time constants. If we substitute k for i in (34), take the partial with respect to w_{ij}, and substitute in γ where possible, we have a differential equation for γ

$$T_k \frac{\gamma kij}{dt} = -\gamma kij + \sigma'(x_k) \sum_l w_{lk} \gamma lij \tag{43}$$

nearly the same as (32) except for a time constant.

We can derive analogous equations for the time constants themselves; define

$$q_j^i(t) = \frac{\partial y_i(t)}{\partial T_j} \tag{44}$$

take the partial of (3) with respect to T_j, and substitute in q. This yields

$$T_i \frac{dq_j^i}{dt} = -q_j^i - \frac{dy_i}{dt} + \sigma'(x_i) \sum_k w_{ki} q_j^k \tag{45}$$

which can be used to update the time constants using the continuous update rule

$$\frac{dT_i}{dt} = -\eta \sum_j e_j q_i^j. \tag{46}$$

Similarly, let us derive equations for modifying the time delays of Section III-E. Define

$$r_{ij}^k(t) = \frac{\partial y_k(t)}{\partial \tau_{ij}} \tag{47}$$

and take the partial of (3) with respect to τ_{ij}, arriving at a differential equations for r

$$T_k \frac{dr_{ij}^k}{dt} = -r_{ij}^k + \sigma'(x_k) \left(\underbrace{w_{ij} \frac{dy_i}{dt}(t - \tau_{ij})}_{\text{included if} j=k} - \sum_l w_{lk} r_{ij}^l (t - \tau_{lk}) \right). \tag{48}$$

The time delays can be updated on-line using the continuous update equation

$$\frac{d\tau_{ij}}{dt} = -\eta \sum_k e_k r_{ij}^k. \tag{49}$$

IV. Some Simulations

In the following simulations, we used networks without time delays, but with mutable time constants. As in the associative network of Section II-C1), an extra input unit whose value was always held at one by a constant external input of 0.5, and which had outgoing connections to all other units, was used to implement biases.

Using firs-t order finite difference approximations, we integrated the system **y** forward from t_0 to t_1, set the boundary conditions $z_i(t_1) = 0$, and integrated the system **z** backward from t_1 to t_0 while numerically integrating $z_j \sigma'(x_j) y_i$ and $z_i \, dy_i/dt$, thus computing $\partial E/\partial w_{ij}$ and $\partial E/\partial T_i$. Since computing dz_i/dt requires $\sigma'(x_i)$, we stored it and replayed it backward as well. We also stored and replayed y_i as it is used in expressions being numerically integrated.

We used the error functional

$$E = \frac{1}{2} \sum_i \int_{t_0}^{t_1} s_i (y_i - d_i)^2 \, dt \tag{50}$$

where $d_i(t)$ is the desired state of unit i at time t and $s_i(t)$ is the importance of unit i achieving that state at that time, in this case zero except when i was an output unit and after some time (five units) had elapsed for the network to settle down. Throughout, we used $\sigma(\xi) = (1 + e^{-\xi})^{-1}$. Time constants were initialized to one, weights were initialized to uniformly distributed random values between 1 and -1, and the initial values $y_i(t_0)$ were set to $I_i(t_0) + \sigma(0)$. The simulator used first-order difference (17) and (21) with $\Delta t = 0.1$.

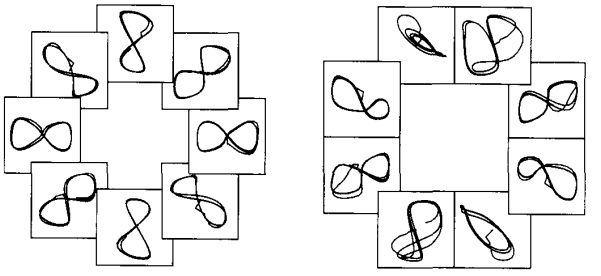


Fig. 8. The output of the rotated figure eight network at all the trained angles (left) and some untrained angles (right).

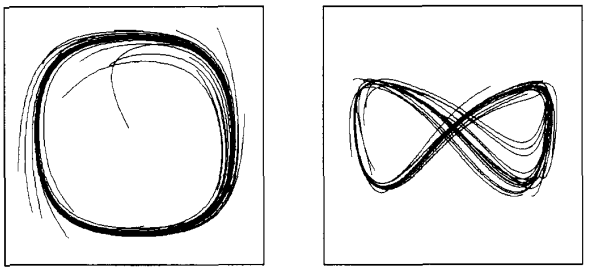


Fig. 9. The output states y_1 and y_2 plotted against each other for a 1000 time unit run, with all the units in the network perturbed by a random amount about every 40 units of time. The perturbations in the circle network (left) were uniform in ± 0.1, and in the figure eight network (right) in ± 0.05.

A. A Rotated Figure Eight

In this simulation a network was trained to generate a figure eight shaped trajectory in two of its units, designated output units. The figure eight was to be rotated about its center by an angle θ which was input to the network through two input units which held the coordinates of a unit vector in the appropriate direction. The target vector for the two output units was generated by

$$\text{target} = 0.4\begin{pmatrix}\cos\theta & -\sin\theta \\ \sin\theta & \cos\theta\end{pmatrix}\begin{pmatrix}\sin \pi t/16 \\ \cos \pi t/16\end{pmatrix} + \begin{pmatrix}0.5 \\ 0.5\end{pmatrix} \quad (51)$$

while the input to the network was simply the angle θ, represented to avoid blemishes as the direction vector

$$\begin{pmatrix}\sin\theta \\ \cos\theta\end{pmatrix}.$$

Eight different values of θ, equally spaced about the circle, were used to generate the training data. In experiments with 20 hidden units, the network was unable to learn the task. Increasing the number of hidden units to 30 allowed the network to learn the task, as shown on the left in Fig. 8. But as shown on the right in Fig. 8, generalization is poor when the network is run with the eight input angles furthest from the training angles, i.e., 22.5 degrees off.

The task would be simple to solve using second —order connections, as they would allow the problem to be decoupled. A few units could be devoted to each of the orthogonal oscillations, and the connections could form a rotation matrix. The poor generalization of the network shows that it is not solving the problem in such a straightforward fashion and suggests that for tasks of this sort it might be better to use slightly higher-order units.

V. Stability and Perturbation Experiments

We can analytically determine the stability of the network by measuring the eigenvalues of Df where f is the function that maps the state of the network at one point in time to its state at a later time. For instance, for a network exhibiting a limit cycle, one would typically use the function that maps the network's state at some time in the cycle to its state at the corresponding time in the next cycle. Unfortunately, this gives only a local stability measure and also does not factor out the effect of hidden units.

In our attempt to judge the stability of the limit cycles exhibited above, rather than calculating Df, where $f(y(t)) = y(t + 16)$, we simply modified the simulator to introduce random perturbations and observed the effects of these perturbations upon the evolution of the system.[4] The two output units in the unrotated figure eight task appear to be phase locked, as their phase relationship remains invariant even in the face of major perturbations. This phase locking is unlike the solution that a human would create by analytically determining weights through decoupling the two output units and using linearized subnets to generate the desired oscillatory behavior, as suggested by Merrick Furst.

The networks to which we introduced these perturbations had been trained to produce simple limit cycles, one in a circular shape and the other in a figure eight shape. Neither of the networks had any input units; they produced only a single limit cycle.

The unperturbed limit cycle of the figure eight network is symmetric, but when perturbations are introduced, as in the right of Fig. 9, symmetry is broken. The portion of the limit cycle moving from the upper left-hand corner toward the lower right-hand corner has diverging lines, but we do not believe that they indicate high eigenvalues and instability. The lines converge rapidly in the upward stroke on the right-hand side of the figure, and analogous unstable behavior is not present in the symmetric downward stroke from the upper right-hand corner toward the lower left. Analysis shows that the instability is caused by the initialization circuitry being inappropriately activated. Since the initialization circuitry is adapted for controlling just the initial behavior of the network, when the net must delay at (0.5, 0.5) for a time before beginning the cycle by moving toward the lower left corner, this circuitry is explicitly not symmetric. The diverging lines seem to be caused by this circuitry being activated and exerting a strong influence on the output units while the circuitry itself deactivates.

In fact, [116] developed a technique for learning the local maximum eigenvalue of the transfer function, optionally projecting out directions whose eigenvalues are not of interest.

[4] Actually, we would not care about the eigenvalues of Df per se, because we wouldn't care about perturbations in the direction of travel, as these effect only the phase, or perturbations that effect only the hidden units. For this reason, we would want to project these out of the matrix Df before computing the eigenvalues. This effect is achieved automatically in our display in Fig. 9.

This technique, which explicitly modulates the behavior we only measured above, has not yet been applied in a control domain.

VI. Other Nonfixed Point Techniques

A. "Elman Nets"

Reference [117] considers a version of backpropagation through time in discrete time in which the temporal history is cut off. Typically, only one or two timesteps are preserved, at the discretion of the architect. This cutoff makes backpropagation through time an on-line algorithm, as the backpropagation to be done to account for the error at each point in time is done immediately. It makes, however, the computational expense per time step scale linearly with the number of timesteps of history being maintained. This accuracy of the computed derivative is smoothly traded off against storage and computation.

The real question with Elman networks is whether the contribution to the error from the history that has been cut off is significant. This question can only be answered relative to a particular task. For instance, [118] finds some problems amenable to the history cutoff, but resorts to full fledged backpropagation through time for other tasks. Reference [43] describes a regular language token prediction task which is difficult for Elman nets when the transition probabilities are equal, but find that breaking this symmetry allows these nets to learn the task.

B. The Moving Targets Method

[119]–[121] propose a moving targets learning algorithm. Such an algorithm maintains a target value for each hidden unit at each point in time. These target values are typically initialized either randomly or to the units' initial untrained behavior. In learning, two phases alternate. In one phase, the hidden units' targets are improved, such that if the targets are attained, better performance would be achieved. In the other phase, the weights are modified such that each unit comes closer to attaining its target values. The error can be regarded as having two terms, one term which penalizes the units being too far from their targets, and another which penalizes the targets for being too far from the values actually attained. This technique has the appeal of decoupling temporally distant actions during the learning of weights, and the disadvantage of requiring the targets to be stored and updated. In the limit, as learning rates are decreased, the moving targets method becomes equivalent to backpropagation though time.

In continuous time, the moving targets method would entail decoupling the units during learning and storing a target trajectory for each unit, including the hidden units. The weights would then be modified to make the trajectories consistent with each other, while the trajectories of the hidden units would be similarly modified. Unfortunately, as with teacher forcing, even if the error is driven to very low levels by such a procedure, there would be no guarantee that the resulting network, if allowed to run free, would have dynamics close to that of the forced dynamics.

The primary disadvantage of the technique is that each pattern to be learned must have associated with it the targets for the hidden units, and these targets must be learned just as the weights are. This makes the technique inapplicable for on-line learning, in which each pattern is seen only once.

C. Feedforward Networks with State

It is noteworthy that the same basic mathematical technique of forward propagation can be applied to networks with a restricted architecture, feedforward networks whose units have state [77], [78], [80]. This is the same as requiring the w_{ij} matrix to be triangular, but allowing nonzero diagonal terms. If we let the γ quantities be ordered derivatives, as in standard backpropagation, then this simplified architecture reduces the computational burden substantially. The elimination of almost all temporal interaction makes $\gamma_{ijk} = 0$ unless $i = k$, leaving only $O(n^2)$ auxiliary equations, each of which can be updated with $O(1)$ computation, for a total update burden of $O(n^2)$, which is the same as conventional backpropagation. This favorable computational complexity makes it of practical significance even for large feedforward recurrent networks. But these feedforward networks are outside the scope of this paper.

D. Teacher Forcing in Continuous Time

Williams and Zisper [122] coin the term teacher forcing, which consists of jamming the desired output values into output units as the network runs. Thus, the teacher forces the output units to have the correct states, even as the network runs, and hence the name. This technique is applied to discrete-time clocked networks, as only then does the concept of changing the state of an output unit each time step make sense.

The error is as usual, with the caveat that errors are to be measured before output units are forced, not after. [122] reports that their teacher forcing technique radically reduced training time for their recurrent networks, although [76] reports difficulties when teacher forcing was used on networks with a larger number of hidden units.

[122]'s application of teacher forcing to their networks is dependent on discrete-time steps, so applying teacher forcing to temporally continuous networks requires a different approach. The approach we shall take is to add some controls that one imagines being used to control the states of the output units and use them to keep the output units locked at their desired states. The error function to be minimized will measure the amount of control that it was necessary to exert, with zero error coming only when the no external forces at all need to be exerted.

Let

$$F_i = \frac{1}{T_i}(-y_i + \sigma(x_i) + I_i) \tag{52}$$

so that (3) is just $dy_i/dt = F_i$, and let us add a new forcing term $f_i(t)$ to (3)

$$\frac{dy_i}{dt} = F_i + f_i. \tag{53}$$

Using Φ to denote the set of units to be forced, we will let d_i be the trajectory that we will force y_i to follow, for each $i \in \Phi$. So we set

$$f_i = \frac{dd_i}{dt} - F_i \tag{54}$$

and $y_i(t_0) = d_i(t_0)$ for $i \in \Phi$ and $f_i = 0$ for $i \notin \Phi$, with the consequence that $y_i = d_i$ for $i \in \Phi$. Now let the error functional be of the form

$$E = \int_{t_0}^{t_1} L(f(t).t)\,dt \tag{55}$$

where typically $L = \sum_{i \in \Phi} f_i^2$.

We can modify the derivation in Section III-A for this teacher forced system. For $i \in \Phi$ a change to $\tilde{y}_i$ will be canceled immediately, so taking the limit as $\Delta t \to 0$ yields $z_i = 0$. Because of this, it does not matter what e_i is for $i \in \Phi$.

We can apply (18) to calculate e_i for $i \notin \Phi$. The chain rule is used to calculate how a change in y_i effects E through the f_i, yielding

$$e_i = \sum_{j \in \Phi} \frac{\delta E}{\delta f_j} \frac{\partial f_j}{\partial y_i}$$

or

$$e_i = \sum_{j \in \Phi} \frac{\partial L}{\partial f_j} - \frac{1}{T_j} \sigma'(x_j) w_{ij}. \tag{56}$$

For $i \notin \Phi$ (26) and (37) are unchanged, and for $j \notin \Phi$ and any i (27) also remains unchanged. The only equations still required are $\partial E / \partial w_{ij}$ for $j \in \Phi$ and $\partial E / \partial T_i$ for $i \in \Phi$. To derive the first, consider the instantaneous effect of a small change to w_{ij}, giving

$$\frac{\partial E}{\partial w_{ij}} = \frac{1}{T_j} \int_{t_0}^{t_1} y_i \sigma'(x_j) \frac{\partial L}{\partial f_i}\,dt. \tag{57}$$

Analogously, for $i \in \Phi$

$$\frac{\partial E}{\partial T_i} = -\frac{1}{T_i} \int_{t_0}^{t_1} \frac{\partial L}{\partial f_i} \frac{dy_i}{dt}\,dt. \tag{58}$$

We are left with a system with a number of special cases depending on whether units are in Φ or not. Interestingly, an equivalent system results if we leave (26), (27), and (37) unchanged except for setting $z_i = \partial L / \partial f_i$ for $i \in \Phi$ and setting all the $e_i = 0$. It is an open question as to whether there is some other way of defining z_i and e_i that results in this simplification.

By taking the limit as the step size goes to zero, however, it is possible to show that the continuous-time analog of teacher forcing is to force the output states to follow desired trajectories, with the error being the difference between the derivative that the network attempts to apply to these units and the derivative of the desired trajectory. This casts light on teacher forcing in the discrete-time case, which can be seen as nearly the same thing.

Regrettably, it also shows that teacher forcing can result in a network with a systematic bias, or a network which, although when being forced has little error, when running free rapidly drifts far from the desired trajectory, in a qualitative sense, as reported by [122] for some cases where oscillations trained with teacher forcing exhibited radically and systematically lower frequency and amplitude when running free.

E. Jordan Nets

Jordan [123] used a backpropagation network with the outputs clocked back to the inputs to generate temporal sequences. Although these networks were used long before teacher forcing, from our perspective Jordan nets are simply a restricted class of teacher forced recurrent networks, in particular, discrete-time networks in which the only recurrent connections emanate from output units. By teacher forcing these output units, no real recurrent paths remain, so simple backpropagation through a single time step suffices for training.

The main disadvantage of such an architecture is that the state to be retained by the network across time must be manifested in the desired outputs of the network, so new persistent internal representations of temporal structures cannot be created. For instance, it would be impossible to train such networks to perform the figure eight task of just a single one of the patterns shown in Fig. 8. In the usual control theory way, this difficulty can be partially alleviated by cycling back to the inputs not just the previous timestep's outputs, but also those from a small number of previous timesteps. The tradeoffs between using hidden units to encapsulate temporally hidden structure and using a temporal window of values which must contain the desired information is problem dependent and depends in essence on how long a hidden variable can remain hidden without being manifested in the observable state variables.

It is easy to construct a continuous-time Jordan network, in which the units' values are continuous in time, the output units constantly have corrected values jammed into them from external sources, and the only recurrent connections are from the outputs back to the inputs. Above we explored teacher forcing in the general setting of fully recurrent networks, but when applied to a Jordan network, the result is a system that is no longer truly recurrent, at least as far as learning is concerned. This is because the network maps the current visible state to the next visible state, with no other information retained in the network. For this reason, a continuous-time Jordan network is precisely equivalent to training a layered network whose input is the current measured value of the signal we wish the Jordan network to learn, and whose target output is the first derivative of this signal to be learned.

F. Teacher Forcing, RTRL, and the Kalman Filter

Matthews [124] and Williams [125] have pointed out that RTRL is related to a version of the Kalman [126] filter, in the extension that allows it to apply to nonlinear systems, namely the extended Kalman filter (ELF) [127], [128], [107]. The EKF has time and space complexity of the same order as those of RTRL. One advantage of using the EKF (instead of RTRL) for learning the weights of a recurrent neural network is that the EKF rationalizes teacher forcing: it modifies both the weights

and the states on an equal basis. This solves the dilema of teacher forcing that if the "true output" units are extra added units whose values are directly copied from those of the old output units, teacher forcing fails to maintain synchronization between the network and its teacher. The EKF does not have this problem, in that it would adjust the new extra and the old output units on an equal basis.

Another way of attempting to rationalize teacher forcing is to note that gradient descent itself generates dE/dy in addition to dE/dw terms. One might think this would make it natural to use $\Delta y = -\eta dE/dy$, thus treating the states on an equal basis with the weights. The problem with this, as pointed out by Williams (personal communication) is that it is difficult to determine exactly what this means. Should the derivative be taken just with respect to the current states or to their histories too? One way alleviate this dilemma is to note that, when we change the weights, we wish we had changed them earlier. To this end, it would be natural to change the states to what they would have been had we changed the weights earlier. This gives

$$\Delta y = \frac{dy}{dw}\Delta w. \tag{59}$$

The involved matrix, dy/dw, is already available as γ in RTRL.

VII. Learning with Scale Parameters

The parameters usually modified by neural network learning algorithms are the weights. There are no *a priori* restrictions on these values; they can be positive, negative, or zero, and the behavior of a network is continuous with respect to changes in its weights. These factors, along with the tractable shape of the error surface, make simple gradient descent algorithms, $\Delta w = -\eta dE/dw$, surprisingly effective.

The error term E being used generally contains one term which has to do with how well the network's outputs meet some criterion. Frequently another term is added as an expression of some *a priori* known probability distribution of the weights. For instance, adding $\sum_i w_i^2$ is equivalent to assuming that the weights are Gaussian distributed. Not adding such a term is equivalent to assuming that the *a priori* distribution on what the weights will turn out to be is flat—not a totally unreasonable prior [28], [129].

We have, however, added some new sorts of parameters, namely time constants and time delays, here represented generically by the vector T. These are scale parameters, which differ from positional parameters in a number of ways. The most telling property of a scale parameter is that the dynamics of the system are affected about as much by multiplying a scale parameter by some constant, irrespective of the scale parameter's value. For instance, changing a time constant from 2 s to 2.2 s can be expected to have about the same qualitative effect as changing it from 200 to 220. Other properties of scale parameters is that they must not become negative, and that as they approach zero, the dynamics of the associated system becomes more and more sensitive to small changes. This means that in practice one must add machinery to enforce the constraint of positiveness and that gradient descent will become increasingly unstable as a scale parameter approaches zero, due to the system's growing sensitivity to its value. Also, the flat prior is no longer the appropriate zero-knowledge prior.

All these problems can be solved in a single stroke by noting that the correct zero-knowledge hypothesis for scale parameters is not flat in their values, but rather flat in their log values [130]. In practice, this corresponds to doing gradient descent in $\mathcal{L}_T = \log T$ rather than in T itself; in other words, to not manipulating T directly but rather using $\Delta\mathcal{L}_T = -\eta dE/d\mathcal{L}_T$. Such a policy also solves the practical problems with scale parameters noted above, as it makes the gradient descent process stiffer as T approaches zero, compensating for the system's increased sensitivity in that region, and it naturally enforces $T > 0$ since $T = \exp \mathcal{L}_T > 0$, which enforces this constraint without any additional mechanism. This last property led to the independent invention and use of this technique by [131].

In addition, weight decay of scale parameters becomes simpler, as decaying $\mathcal{L}_T$ toward zero corresponds to decaying T toward one, which is a reasonable target. Of course, a constant factor can be inserted to make the decay toward some other *a priori* most likely value. Note, however, that the force exerted by the decay term will scale with the log parameter, which is more appropriate, since the additional force exerted should correspond to the change's effect on the dynamics of the system, to pass dimensional analysis.

VIII. Summary and Conclusion

A. Complexity Comparison

Consider a network with n units and m weights which is run for s time steps (variable grid methods [132] would reduce s by dynamically varying Δt) where $s = (t_1 - t_0)/\Delta t$. Additionally, assume that the computation of each $e_i(t)$ is $O(1)$ and that the network is not partitioned.

Under these conditions, simulating the $\mathbf{y}$ system takes $O(m+n) = O(m)$ time for each time step, as does simulating the $\mathbf{z}$ system. This means that using the technique described in Section IV, the entire simulation takes $O(m)$ time per time step, the best that could be hoped for. Storing the activations and weights takes $O(n+m) = O(m)$ space, and storing $\mathbf{y}$ during the forward simulation to replay while simulating $\mathbf{z}$ backward takes $O(sn)$ space, so if we use this technique the entire computation takes $O(sn+m)$ space. If we simulate $\mathbf{y}$ backward during the backward simulation of $\mathbf{z}$, the simulation requires $O(n+m)$ space, again the best that could be hoped for. This later technique, however, is susceptible to numeric stability problems.

The on-line technique of RTRL described in Section III-B requires $O(n^2m)$ time each time step, and $O(nm)$ space. The other techniques discussed in that section require less time and space and retain all of the advantages of being on-line (with the possible exception of simplicity of implementation), so it would appear that these new on-line methods dominate RTRL. These time complexity results are for sequential machines, and are summarized in Table I.

TABLE I
A SUMMARY OF THE COMPLEXITY OF SOME LEARNING PROCEDURES FOR RECURRENT NETWORKS. IN THE "STORING Y'" TECHNIQUE WE STORE Y' AS TIME IS RUN FORWARD AND REPLAY IT AS WE RUN TIME BACKWARD COMPUTING Z. IN "Y' BACKWARD" WE DO NOT STORE Y', INSTEAD RECOMPUTING IT AS TIME IS RUN BACKWARD. "FORWARD PROPAGATION" ONE AND TWO ARE THE ON-LINE TECHNIQUES DESCRIBED IN SECTION III-B. THE TIMES GIVEN ARE FOR COMPUTING THE GRADIENT WITH RESPECT TO ONE PATTERN

technique	*time/Δt*	*space*	*online*	*stable*	*local*	*exact*
BPTT, storing y	$O(m)$	$O(sn+m)$	no	yes	yes	yes
RTRL	$O(n^2m)$	$O(nm)$	yes	yes	no	yes
BPTT, only h steps	$O(hm)$	$O(hn+m)$	yes	yes	yes	no
Williams-Peng, h steps	$O(m)$	$O(hn+m)$	yes	yes	yes	no
hybrid BPTT/RTRL	$O(nm)$	$O(nm)$	yes	yes	no	yes
Sun-Chen-Lee	$O(nm)$	$O(n^2+m)$	yes	yes	no	yes
BPTT, recalc. y	$O(m)$	$O(m)$	no	no	yes	yes

Note that in this section we are concerning ourselves with how much computation it takes to obtain the gradient information. This is generally just the inner loop of a more complex algorithm to adjust the weights, which uses the gradient information, such as a stochastic gradient descent algorithm.

B. Speeding the Optimization

Experience has shown that learning in these networks tends to be "stiff" in the sense that the Hessian of the error with respect to the weights (the matrix of second derivatives) tends to have a wide eigenvalue spread. One technique that has proven useful in this particular situation is that of [133] which was applied to the single figure eight problem perturbed in Fig. 9 with great success [134]. For a modern variant of this technique which is suitable to on-line pattern presentation, see [135]–[137].

Since the acceleration of convergence in these gradient systems is such an important issue, it can be helpful to know some of the techniques used to analyze the limitations of convergence under various conditions in systems of this sort, and of some other techniques for accelerating their convergence; see [138, p. 304] and [139]–[147].

C. Prospects and Future Work

Control domains are the most natural application for continuous-time recurrent networks, but signal processing and speech generation (and recognition using generative techniques) are also domains to which this type of network might be naturally applied. Such domains may lead us to complex architectures like those discussed in Section III-E. For control domains, it seems important to have ways to force the learning toward solutions that are stable in the control sense of the term.

On the other hand, we can turn the logic of Section V around. Consider a difficult constraint satisfaction task of the sort that neural networks are sometimes applied to, such as the traveling salesman problem [148]. Two competing techniques for such problems are simulated annealing [149], [58] and mean field theory [92]. By providing a network with a noise source which can be modulated (by second-order connections, say) we could see if the learning algorithm constructs a network that makes use of the noise to generate networks that do simulated annealing or if pure gradient descent techniques are evolved. If a hybrid network evolves, its structure may give us insight into the relative advantages of these two different optimization techniques, and into the best ways to structure annealing schedules.

D. Conclusions

Recurrent networks are often avoided because of a fear of inordinate learning times and incomprehensible algorithms and mathematics. It should be clear from the above that such fears are unjustified. Certainly there is no reason to use a recurrent network when a feedforward layered architecture suffices; but on the other hand, if recurrence is needed, there are a plethora of learning algorithms available across the spectrum of quiescence vs. dynamics and across the spectrum of accuracy vs. complexity and across the spectrum of space vs. time. These new learning algorithms, and experience with recurrent and temporally continuous networks, has made them much more tractable and practical than they seemed only a few years ago.

ACKNOWLEDGMENT

For provocative and enlightening discussions, the author would like to thank L. Giles, G. Hinton, G. Kuhn, F. Pineda, R. Szeliski, D. Touretzky, R. Williams, and members of the old Boltzmann group at CMU. The author would also like to thank a number of anonymous reviewers.

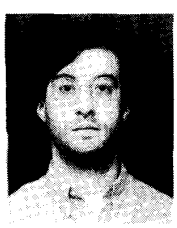
REFERENCES

[1] J. L. McClelland, D. E. Rumelhart, and G. E. Hinton, "The appeal of parallel distributed processing," in D. E. Rumelhart, J. L. McClelland, and the PDP research group, Eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*. Cambridge, MA: MIT Press, 1986.
[2] D. Marr and T. Poggio, "Cooperative computation of stereo disparity," *Sci.*, vol. 194, pp. 283–287, 1976.
[3] G. Marr, D. Palm, and T. Poggio, "Analysis of a cooperative stereo algorithm," *Biol. Cybern.*, vol. 28, pp. 223–229, 1978.
[4] G. E. Hinton, "Relaxation and its role in vision," Ph.D. dissertation, Univ. Edinburgh, 1977.
[5] L. S. Davis and A. Rosenfeld, "Cooperating processes for low-level vision: A survey," *Artificial Intell.*, vol. 3, pp. 245–264, 1981.
[6] R. Szeliski, "Cooperative algorithms for solving random-dot stereograms," Carnegie Mellon Univ. Tech. Rep. CMU-CS-86-133, 1986.
[7] G. E. Hinton, "Using relaxation to find a puppet," in *Proc A.I.S.B. Summer Conf.*, Univ. Edinburgh, July 1976.
[8] D. L. Waltz and J. B. Pollack, "Massively parallel parsing: A strongly interactive model of natural language interpretation," *Cognitive Sci.*, vol. 9, pp. 51–74, 1985.
[9] N. Qian and T. J. Sejnowski, "Learning to solve random-dot stereograms of dense and transparent surfaces with recurrent backpropagation," in *Proc. 1988 Connectionist Models Summer School*, D. S. Touretzky, G. E. Hinton, and T. J. Sejnowski, Eds., 1989, pp. 435–443.
[10] O. Nerrand, P. Roussel-Ragot, G. D. L. Personnaz, and S. Marcos, "Neural networks and nonlinear adaptive filtering: Unifying concepts and new algorithms," *Neural Computa.*, vol. 5, no. 2, pp. 165–197, 1993.
[11] T. W. Karjala, D. M. Himmelblau. and R. Miikkulainen, "Data rectification using recurrent (Elman) neural networks," in *Proc. IJCNN '92*, Baltimore, pp. 901–905.
[12] R. L. Watrous, B. Laedendorf, and G. M. Kuhn, "Complete gradient optimization of a recurrent network applied to BDG discrimination," *J Acoustical Soc. Amer.*, vol. 87, no. 3, pp. 1301–1309, Mar. 1990.
[13] P. Poddar and K. P. Unnikrishnan, "nonlinear prediction of speech signals using memory neuron networks," in *Proc. 1991 IEEE Workshop Neural Networks Signal Process.*, IEEE Press, 1991, pp. 395–404.
[14] D. Albesano, R. Gemello, and F, Mana, "Word recognition with recurrent network automata," in *Proc. IJCNN '92*, Baltimore, pp. 308–313.

[15] S. R. Lockery, Y. Fang, and T. J. Sejnowski, "A dynamic neural network model of sensorimotor transformations in the leech," *Neural Computa.*, vol. 2, no. 3, pp. 274–282, 1990.
[16] K. Doya, M. E. T. Boyle, and A. I. Selverston, "Mapping between neural and physical activities of the lobster gastric mill system," in *Advances in Neural Information Processing Systems 5*, S. J. Hanson, Jack D. Cowan, and C. Lee Giles, Eds. San Mateo, CA: Morgan Kaufmann, 1993, pp. 913–920.
[17] K. Doya, A. I. Selverston, and P. F. Rowat, "A Hodgkin–Huxley type neuron model that learns slow nonspike oscillation," in *Advances in Neural Information Processing Systems 6*, J. D. Cowan, G. Tesauro and J. Alspector, Eds. San Mateo, CA: Morgan Kaufmann, 1994.
[18] Y. Chauvin and D. E. Rumelhart, Eds., *Bcckpropagation: Theory, Architectures and Applications*. Hillsdale, NJ: Lawrence Erlbaum Associates, 1994, in Press.
[19] P. R. Gorman and T. J. Sejnowski, "Analysis of hidden units in a layered network trained to classify sonar targets," *Neural Networks*, vol. 1, no. 1, pp. 75–89, 1988.
[20] K. Lang and G. Hinton, "The development of the time-delay neural network architecture for speech recognition," Dep. Comput. Sci., Carnegie Mellon Univ., Tech. Rep. CMU-CS-88-152, Nov. 1988.
[21] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang, "Phoneme recognition using time-delay networks," *IEEE Trans. Acoustics, Speech, Signal Process.*, vol. 37, no. 3, pp. 328–339, 1989.
[22] K. J. Lang, G. E. Hinton, and A. Waibel, "A time-delay neural network architecture for isolated word recognition," *Neural Networks*, vol. 3, no. 1, pp. 23–43, 1990.
[23] K. J. Lang and G. E. Hinton, "Dimensionality reduction and prior knowledge in *e*-set recognition," in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1990, pp. 178–185.
[24] D. W. Tank and J. J. Hopfield, "Concentrating information in time: Analog neural networks with applications to speech recognition problems," in *Proc. IEEE 1st Int. Conf. Neural Networks*, San Diego, CA, June 21–24 1987, pp. 455–468.
[25] S. P. Day and M. R. Davenport, "Continuous-time temporal backpropagation with adaptable time delays," *IEEE Trans. Neural Networks*, vol. 4, no. 2, pp. 348–354, 1993.
[26] M. I. Jordan, "Generic constraints on underspecified target trajectories," in *Proc. IJCNN '89 (Int. Joint Conf. Neural Networks)*, Washington DC, June 18–22 1989.
[27] *Applications of Artificial Neural Networks*, number 1294 in APIE Proceedings Series, Orlando, FL, Apr. 18–20, 1990.
[28] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman, "Generalization by weight-elimination with application to forecasting," in *Advances in Neural Information Processing Systems 3*, R. P. Lippmann, J. E. Moody, and D. S. Touretzky, Eds. San Mateo, CA: Morgan Kaufmann, 1991, pp. 875–882.
[29] B. de Vries and J. C. Principe, "A theory for neural networks with time delays," in *Advances in Neural Information Processing Systems 3*, R. P. Lippmann, J. E. Moody, and D. S. Touretzky, Eds. San Mateo, CA: Morgan Kaufmann, 1991, pp. 162–168.
[30] A. D. Back and A. C. Tsoi, "FIR and IIR synapses, a new neural network architecture for time series modeling," *Neural Computa.*, vol. 3, no. 3, pp. 337–350, 1991.
[31] B. H. Juang, S. Y. Kung, and C. A. Camm, Eds., in *Proc. 1991 IEEE Workshop Neural Networks Signal Process.*, IEEE Press, 1991.
[32] D. Hush and B. Horne, "Progress in supervised neural networks," *IEEE Signal Process. Mag.*, vol. 10, no. 1, pp. 8–39, 1993.
[33] B. de Vries and J. Principe, "The gamma model—A new neural network for temporal processing," *Neural Networks*, vol. 5, no. 4, pp. 565–576, 1992.
[34] T. Maxwell, C. L. Giles, Y. C. Lee, and H. H. Chen, "nonlinear dynamics of artificial neural systems," in *Proc. Snowbird Conf. Neural Networks Computing*, Amer. Institute Physics, no. 151, 1986.
[35] Y. Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.
[36] M. Kawato, T. Setoyama, and R. Suzuki, "Feedback error learning of movement by multi-layer neural networks," in *Proc. Int. Neural Networks Soc. 1st Annu. Meeting*, 1988, p. 342.
[37] M. I. Jordan and R. A. Jacobs, "Learning to control an unstable system with forward modeling," in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1990.
[38] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Networks*, vol. 1, pp. 4–27, Mar. 1990.
[39] W. Thomas Miller, III, R. S. Sutton, and P. J. Werbos, Eds., *Neural Networks for Control*. Cambridge, MA: MIT Press, 1990.
[40] H. T. Siegelmann and E. D. Sontag, "Turing computability with neural networks," *Appl. Math. Lett.*, vol. 4, no. 6, pp. 77–80, 1991.
[41] J. Kilian and H. T. Siegelmann, "Computability with the classical sigmoid," in *Proc. 6th Annu. ACM Workshop Computational Learning Theory*, Santa Cruz, CA, July 1993, pp. 137–143.
[42] H. T. Siegelmann and E. D. Sontag, "Analog computation via neural networks," in *Proc. 2nd Israel Symp. Theory Comput. Syst.*, Natanya, Israel, June 1993.
[43] A. Cleeremans, D. Servan-Schreiber, and J. McClelland, "Finite state automata and simple recurrent networks," *Neural Computa.*, vol. 1, no. 3, pp. 372–381, 1989.
[44] R. L. Watrous and G. M. Kuhn, "Induction of finite-state automata using second-order recurrent networks," in *Advances in Neural Information Processing Systems 4*, J. E. Moody, S. J. Hanson, and R. P. Lippmann, Eds. San Mateo, CA: Morgan Kaufmann, 1992, pp. 309–316.
[45] C. L. Giles, C. B. Miller, D. Chen, G. Z. Sun, H. H. Chen, and Y. C. Lee, "Extracting and learning an unknown grammar with recurrent neural networks," in *Advances in Neural Information Processing Systems 4*J. E. Moody, S. J. Hanson, and R. P. Lippmann, Eds. San Mateo, CA: Morgan Kaufmann, 1992, pp. 317–324.
[46] M. C. Mozer and S. Das, "A connectionist symbol manipulator that discovers the structure of context-free languages," in *Advances in Neural Information Processing Systems 5*S. J. Hanson, Jack D. Cowan, and C. Lee Giles, Eds. San Mateo, CA: Morgan Kaufmann, 1993, pp. 863–870.
[47] S. Das, C. L. Giles, and G.-Z. Sun, "Using prior knowledge in a NNPDA to learn context-free languages," in *Advances in Neural Information Processing Systems 5*S. J. Hanson, Jack D. Cowan, and C. Lee Giles, Eds. San Mateo, CA: Morgan Kaufmann, 1993, pp. 65–72.
[48] J. F. Kolen, "Fool's gold: Extracting finite state machines from recurrent network dynamics," in *Advances in Neural Information Processing Systems 6*J. D. Cowan, G. Tesauro and J. Alspector, Eds.. San Mateo, CA: Morgan Kaufmann, 1994, In Press.
[49] S. Das and M. C. Mozer, "A unified gradient-descent/clustering architecture for finite state machine induction," in *Advances in Neural Information Processing Systems 6*J. D. Cowan, G. Tesauro and J. Alspector, Eds.. San Mateo, CA: Morgan Kaufmann, 1994, In Press.
[50] D. Angluin, "Learning regular sets from queries and counter examples," *Inform. Computa.*, vol. 75, pp. 87–106, 1987.
[51] K. J. Lang, "Random DFA's can be approximately learned from sparse uniform examples," in *Proc. 5th Annu. ACM Workshop Computational Learning Theory*, Pittsburgh, PA, July 1992, pp. 45–52.
[52] J. Hochreiter, "Untersuchungen zu dynamischen neuronalen netzen," 1991, Diplomarbeit, Institut für Informatik, Technische Universität München.
[53] M. C. Mozer, "Induction of multiscale temporal structure," in *Advances in Neural Information Processing Systems 4*J. E. Moody, S. J. Hanson, and R. P. Lippmann, Eds. San Mateo, CA: Morgan Kaufmann, 1992, pp. 275–282.
[54] J. H. Schmidhuber, "Learning complex, extended sequences using the principle of history compression," *Neural Computa.*, vol. 4, no. 2, pp. 234–242, 1992.
[55] ——, "Learning unambiguous reduced sequence descriptions," *Advances in Neural Information Processing Systems 4*in J. E. Moody, S. J. Hanson, and R. P. Lippmann, Eds. San Mateo, CA: Morgan Kaufmann, 1992, pp. 291–298.
[56] G. E. Hinton, "Learning distributed representations of concepts," in *Proc. 8th Annu. Cognitive Sci. Conf.*, 1986.
[57] T. J. Sejnowski, P. K. Kienker, and G. Hinton, "Learning symmetry groups with hidden units: Beyond the perceptron," *Physica D*, vol. 22, pp. 260–275, 1986.
[58] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for Boltzmann Machines," *Cognitive Sci.*, vol. 9, pp. 147–169, 1985.
[59] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations* D. E. Rumelhart, J. L. McClelland, and the PDP research group, Eds. Cambridge, MA: MIT Press, 1986.
[60] P. J. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," Ph.D. dissertation, Harvard Univ., 1974.
[61] D. B. Parker, "Learning-logic," MIT Center Res. Computational Economics Manage. Sci., Tech. Rep. TR-47, 1985.
[62] R. Howard, *Dynamic Programming and Karkhov Processes*. Cambridge, MA: MIT Press, 1960.
[63] D. S. Touretzky and D. A. Pomerleau, "What's hidden in the hidden

layers?," *BYTE*, pp. 227–233, Aug. 1989.
[64] B. Widrow and M. Hoff, "Adaptive switching circuits," in *Proc. Western Electron. Show Conv.*, 1960, vol. 4, pp. 96–104, Institute of Radio Engineers (now IEEE).
[65] Y. L. Cun, I. Kanter, and S. A. Solla, "Second-order properties of error surfaces: Learning time and generalization," in R. P. Lippmann, J. E. Moody, and D. S. Touretzky, Eds., *Advances in Neural Information Processing Systems 3*. San Mateo, CA: Morgan Kaufmann, 1991, pp. 918–924.
[66] B. Baird, "A learning rule for CAM storage of continuous periodic sequences," in *Proc. IJCNN '90 II (Int. Joint Conf. Neural Networks)*, San Diego, CA, June 1990, pp. 493–498.
[67] B. Baird and F. Eeckman, "CAM storage of analog patterns and continuous sequences with $3n^2$ weights," in *Advances in Neural Information Processing Systems 3*R. P. Lippmann, J. E. Moody, and D. S. Touretzky, Eds. San Mateo, CA: Morgan Kaufmann, 1991, pp. 91–97.
[68] C. A. Skarda and W. J. Freeman, "How brains make chaos to make sense of the world," *Brain Behavioral Sci.*, vol. 10, Nov. 1987.
[69] W. J. Freeman, "Simulation of chaotic EEG patterns with a dynamic model of the olfactory system," *Biol. Cybern.*, vol. 56, p. 139, 1987.
[70] J. P. Crutchfield and B. S. McNamara, "Equations of motion from a data series," *Complex Syst.*, vol. 1, pp. 417–452, 1987.
[71] A. Lapedes and R. Farber, "nonlinear signal processing using neural networks: Prediction and system modeling," Theoretical Division, Los Alamos Nat. Lab., Los Alamos, NM, Tech. Rep. LA-UR-87-2662, 1987.
[72] F. Pineda, "Generalization of backpropagation to recurrent neural networks," *Physical Rev. Lett.*, vol. 19, no. 59, pp. 2229–2232, 1987.
[73] L. B. Almeida, "A learning rule for asynchronous perceptrons with feedback in a combinatorial environment," in *Proc. IEEE 1st Int. Conf. Neural Networks*, San Diego, CA, June 21–24, 1987, pp. 609–618.
[74] G. E. Hinton, "Deterministic Boltzmann learning performs steepest descent in weight-space," *Neural Computa.*, vol. 1, no. 1, pp. 143–150, 1989.
[75] P. Baldi and F. Pineda, "Contrastive learning and neural oscillations," *Neural Computa.*, vol. 3, no. 4, pp. 526–545, 1991.
[76] B. A. Pearlmutter, "Dynamic recurrent neural networks," Carnegie Mellon Univ. School Comput. Sci., Pittsburgh, PA, Tech. Rep. CMU-CS-90-196, Dec. 1990.
[77] M. Gori, Y. Bengio, and R. de Mori, "BPS: A learning algorithm for capturing the dynamic nature of speech," in *Proc. IJCNN '89 Int. Joint Conf. Neural Networks*, Washington DC, June 18–22 1989, pp. 417–423.
[78] G. Kuhn, "A first look at phonetic discrimination using connectionist models with recurrent links," Institute Defense Anal., Princeton, NJ, SCIMP Working Paper 82018, Apr. 1987.
[79] M. C. Mozer, "A focused backpropagation algorithm for temporal pattern recognition," *Complex Syst.*, vol. 3, no. 4, pp. 349–381, Aug. 1989.
[80] T. Uchiyama, K. Shimohara, and Y. Tokunaga, "A modified leaky integrator network for temporal pattern recognition," in *Proc. IJCNN '89 (Int. Joint Conf. Neural Networks)*, Washington DC, June 18–22 1989, pp. 469–475.
[81] B. A. Pearlmutter, "Two new learning procedures for recurrent networks," *Neural Network Rev.*, vol. 3, no. 3, pp. 99–101, 1990.
[82] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vererling, *Numerical Recipes in C*. Cambridge, UK: Cambridge Univ. Press, 1988.
[83] M. A. Cohen and S. Grossberg, "Stability of global pattern formation and parallel memory storage by competitive neural networks," *IEEE Trans. Syst., Man Cybern.*, vol. 13, pp. 815–826, 1983.
[84] G. E. Hinton and T. J. Sejnowski, "Optimal perceptual inference," in *Proc. IEEE Conf. Comput. Vision Pattern Recognition*, Washington DC, June 1983, pp. 448–453.
[85] A. F. Atiya, "Learning on a general network," in *Neural Information Processing Systems*D. Z. Anderson, Ed. New York: American Institute of Physics, 1988 pp. 22–30.
[86] S. Renals and R. Rohwer, "A study of network dynamics," *J. Statistical Physics*, vol. 58, pp. 825–848, June 1990.
[87] R. B. Allen and J. Alspector, "Learning of stable states in stochastic asymmetric networks," Bell Commun. Res., Morristown, NJ, Tech. Rep. TM-ARH-015240, Nov. 1989.
[88] C. C. Galland and G. E. Hinton, "Deterministic Boltzmann learning in networks with asymmetric connectivity," Univ. Toronto Dep. Comput. Sci., Tech. Rep. CRG-TR-89-6, 1989.
[89] S. J. Nowlan, "Gain variation in recurrent error propagation networks," *Complex Syst.*, vol. 2, no. 3, pp. 305–320, June 1988.
[90] M. B. Ottaway, P. Y. Simard, and D. H. Ballard, "Fixed point analysis for recurrent neural networks," in *Advances in Neural Information Processing Systems I*, David S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1989.
[91] C. Peterson and J. R. Anderson, "A mean field theory learning algorithm for neural nets," *Complex Syst.*, vol. 1, 1987.
[92] C. Peterson and J. R. Anderson, "A mean field theory learning algorithm for neural networks," MCC (Microelectronics and Computer Technology Corporation), Austin, TX, tech. rep. EI-259-87, Aug. 1987.
[93] G. E. Hinton and K. J. Lang, "Shape recognition and illusory conjunctions," in *Proc. 9th Int. Joint Conf. Artificial Intell.*, Los Angeles, Aug. 1985, vol. 1, pp. 252–259.
[94] T. J. Sejnowski, "Higher-order Boltzmann machines," in *Proc. Snowbird Conf. Neural Networks Computing*, Amer. Institute Physics, no. 151, 1986, pp. 398–403.
[95] P. J. Werbos, "Backpropagation through time: What it does and how to do it," in *Proc. IEEE*, vol. 78, pp. 1550–1560, 1990.
[96] B. Pearlmutter, "Learning state space trajectories in recurrent neural networks," *Neural Computa.*, vol. 1, no. 2, pp. 263–269, 1989.
[97] P. J. Werbos, "Generalization of backpropagation with application to a recurrent gas market model," *Neural Networks*, vol. 1, pp. 339–356, 1988.
[98] A. E. Bryson, Jr., "A steepest ascent method for solving optimum programming problems," *J. Appl. Mechanics*, vol. 29, no. 2, p. 247, 1962.
[99] S. E. Dreyfus, *Dynamic Programming and the Calculus of Variations*, vol. 21 of *Mathematics in Science and Engineering*. New York: Academic, 1965.
[100] P. J. Werbos, "Applications of advances in nonlinear sensitivity analysis," in *Proc. 10th IFIP Conf. Syst. Modeling Optimization*, New York, Aug. 31–Sep. 4, 1981.
[101] A. J. Robinson and F. Fallside, "Static and dynamic error propagation networks with application to speech coding," in *Proc. Int. Joint Conf. Neural Networks*, Washington DC, June 18–22 1989, pp. 632–641.
[102] M. Gherrity, "A learning algorithm for analog, fully recurrent neural networks," in *Proc. IJCNN '89 Int. Joint Conf. Neural Networks*, Washington DC, June 18–22 1989, pp. 643–644.
[103] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Computa.*, vol. 1, no. 2, pp. 270–280, 1989.
[104] K. S. Narendra and K. Parthasarathy, "Gradient methods for the optimization of dynamical systems containing neural networks," *IEEE Trans. Neural Networks*, vol. 2, no. 2, pp. 252–262, Mar. 1991.
[105] D. H. Jacobson, "New second-order and first-order algorithm for determining optimal control: A differential dynamic programming approach," *J. Optimization Theory Applicat.*, vol. 2, 1968.
[106] R. E. Bellman, *Methods of nonlinear Analysis: Volume II*. New York: Academic, 1973.
[107] A. Gelb *et al.*, Eds., *Applied Optimal Estimation*. Cambridge, MA: MIT Press, 1974.
[108] D. Zipser, "Subgrouping reduces complexity and speeds up learning in recurrent networks," in *Advances in Neural Information Processing Systems 2*D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1990, pp. 638–641.
[109] R. J. Williams and J. Peng, "An efficient gradient-based algorithm for on-line training of recurrent network trajectories," *Neural Computa.*, vol. 2, no. 4, pp. 490–501, 1990.
[110] R. J. Williams and D. Zipser, "Gradient-based learning algorithms for recurrent networks and their computational complexity," in *Backpropagation: Theory, Architectures and Applications*, Y. Chauvin and D. E. Rumelhart, Eds. Hillsdale, NJ: Lawrence Erlbaum Associates, 1994, in Press.
[111] Jürgen H. Schmidhuber, "A fixed size storage $O(n^3)$ time complexity learning algorithm for fully recurrent continually running networks," *Neural Computa.*, vol. 4, no. 2, pp. 243–248, 1992.
[112] G.-Z. Sun, H.-H. Chen, and Y.-C. Lee, "Green's function method for fast on-line learning algorithm of recurrent neural networks," in *Advances in Neural Information Processing Systems 4*J. E. Moody, S. J. Hanson, and R. P. Lippmann, Eds. San Mateo, CA: Morgan Kaufmann, 1992, pp. 333–340.
[113] U. Bodenhausen, "Learning internal representations of pattern sequences in a neural network with adaptive time-delays," in *Proc. IJCNN '90 II (Int. Joint Conf. Neural Networks)*, San Diego, CA, June 1990.
[114] D. W. Tank and J. J. Hopfield, "Neural computation by time compression," *Proc. National Academy Sci.*, vol. 84, pp. 1896–1900, 1987.
[115] R. L. Watrous, "Speech recognition using connectionist networks," Ph.D. dissertation, Univ. Pennsylvania, Oct. 1988.
[116] P. Y. Simard, J. P. Rayzs, and B. Victorri, "Shaping the state space landscape in recurrent networks," in *Advances in Neural Information Processing Systems 3*, R. P. Lippmann, J. E. Moody, and D. S. Touretzky, Eds. San Mateo, CA: Morgan Kaufmann, 1991, pp. 105–112.
[117] J. L. Elman, "Finding structure in time," *Cognitive Sci.*, vol. 14, pp.

179–211, 1990.
[118] ——, "Finding structure in time," Center Res. Language, Univ. Calif. San Diego Tech. Rep. CRL-8801, Apr. 1988.
[119] Y. Le Cun, "Une procédure d'apprentissage pour réseau à seuil asymétrique," in *Cognitiva 85: A la Frontière de l'Intelligence Artificielle des Sciences de la Connaissance des Neurosciences*, Paris 1985, 1985, CESTA, Paris, pp. 599–604.
[120] T. Grossman, R. Meir, and E. Domany, "Learning by choice of internal representations," *Complex Syst.*, vol. 2, pp. 555–575, 1989.
[121] R. Rohwer, "The 'moving targets' training algorithm," in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1990, pp. 558–565.
[122] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," Univ. Calif San Diego, La Jolla, CA , Tech. Rep. ICS 8805, Nov. 1988.
[123] M. I. Jordan, "Attractor dynamics and parallelism in a connectionist sequential machine," in *Proc. 9th Annu. Conf. Cognitive Sci. Soc.*, 1986, pp. 531–546.
[124] M. B. Matthews, "Neural network nonlinear adaptive filtering using the extended Kalman filter algorithm," in *Proc. Int. Neural Networks Conf.*, Paris, July 1990, vol. 1, pp. 115–119.
[125] R. J. Williams, "Training recurrent networks using the extended Kalman filter," in *Proc. IJCNN '92, (Int. Joint Conf. Neural Networks)*, Baltimore, MD, Apr. 1992, pp. 241–250.
[126] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Trans. ASME J. Basic Eng.*, vol. 82, no. 1, pp. 35–45, Mar. 1960.
[127] R. K. Mahra, "On the identification of variances and adaptive Kalman filtering," *IEEE Trans. Automat. Contr.*, vol. AC-15, no. 2, pp. 175–184, Apr. 1970.
[128] B. D. O. Anderson and J. B. Moore, *Optimal Filtering*. Englewood Cliffs, NJ: Prentice-Hall, 1979.
[129] S. J. Nowlan and G. E. Hinton, "Adaptive soft weight tying using Gaussian mixtures," in *Advances in Neural Information Processing Systems 4*, J. E. Moody, S. J. Hanson, and R. P. Lippmann, Eds. San Mateo, CA: Morgan Kaufmann, 1992, pp. 993–1000.
[130] J. Skilling, Ed., *Maximum Entropy and Bayesian Methods*. Norwell, MA: Kluwer, 1989.
[131] P. F. Rowat and A. I. Selverston, "Learning algorithms for oscillatory networks with gap junctions and membrane currents," *Network: Computation Neural Syst.*, vol. 2, no. 1, pp. 17–42, Feb. 1991.
[132] J. G. Blom, J. M. Sanz-Serna, and J. G. Verwer, *On Simple Moving Grid Methods for One-Dimensional Evolutionary Partial Differential Equations*. Amsterdam: Stichting Mathematisch Centrum, 1986.
[133] R. A. Jacobs, "Increased rates of convergence through learning rate adaptation," *Neural Networks*, vol. 1, no. 4, pp. 295–307, 1988.
[134] Y. Fang and T. J. Sejnowski, "Faster learning for dynamic recurrent backpropagation," *Neural Computa.*, vol. 2, no. 3, pp. 270–273, 1990.
[135] R. S. Sutton, "Gain adaptation beats least squares?," in *Proc. 7th Yale Workshop Adaptive Learning Syst.*, 1992, pp. 161–166.
[136] ——, "Adapting bias by gradient descent: An incremental version of delta-bar-delta," in *Proc. Nat. Conf. Artificial Intell. AAAI-92*, 1992.
[137] M. A. Gluck, P. T. Glauthier, and R. S. Sutton, "Adaptation of cue-specific learning rates in network models of human category learning," in *Proc. 14th Annu. Meeting Cognitive Sci. Soc.*, Bloomington, IN, 1992.
[138] D. J. Wilde and C. S. Beightler, *Foundations of Optimization*. Englewood Cliffs, NJ: Prentice-Hall, 1967.
[139] B. Widrow, J. M. McCool, M. G. Larimore, and C. R. Johnson, Jr., "Stationary and nonstationary learning characteristics of the LMS adaptive filter," *Proc. IEEE*, vol. 64, pp. 1151–1162, 1976.
[140] B. Widrow and S. D. Stearns, *Adaptive Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1985.
[141] S. T. Alexander, *Adaptive Signal Processing*. New York: Springer-Verlag, 1986.
[142] D. B. Parker, "Optimal algorithms for adaptive networks: Second-order backpropagation, second-order direct propagation and second-order Hebbian learning," in *Proc. IEEE 1st Int. Conf. Neural Networks*, San Diego, CA, June 21–24, 1987, pp. 593–600.
[143] R. Watrous, "Learning algorithms for connectionist networks: Applied gradient methods of nonlinear optimization," in *Proc. IEEE 1st Int. Conf. Neural Networks*, San Diego, CA, June 21–24, 1987, pp. 619–627.
[144] J. J. Shynk and S. Roy, "The LMS algorithm with momentum updating," in *Proc. IEEE Int. Symp. Circuits Syst.*, June 6–9 1988, pp. 2651–2654.
[145] G. Tesauro, Y. He, and S. Ahmad, "Asymptotic convergence of backpropagation," *Neural Computa.*, vol. 1, no. 3, pp. 382–391, 1989.
[146] M. A. Tuğay and Y. Tanik, "Properties of the momentum LMS algorithm," *Signal Process.*, vol. 18, no. 2, pp. 117–127, Oct. 1989.
[147] B. A. Pearlmutter, "Gradient descent: Second-order momentum and saturating error," in *Advances in Neural Information Processing Systems 4*, J. E. Moody, S. J. Hanson, and R. P. Lippmann, Eds. San Mateo, CA: Morgan Kaufmann, 1992, pp. 887–894.
[148] J. J. Hopfield and D. W. Tank, "'Neural' computation of decisions in optimization problems," *Biol. Cybern.*, vol. 52, pp. 141–152, 1985.
[149] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Sci.*, vol. 220, pp. 671–680, 1983.



Barak A. Pearlmutter received the bachelor's degree in mathematics from Case Western Reserve University, Cleveland, OH. He received the Ph.D. degree in computer science from Carnegie Mellon University, Pittsburgh, PA.

He is currently with the learning group at Siemens Corporate Research. He has been working with neural networks for over a decade and is interested in understanding adaptive behavior and the brain. Secondary interests include programming language implementation and computer architecture.