Temporally Continuous vs. Clocked Networks

Barak A. Pearlmutter
Yale University
Department of Psychology
11A Yale Station
New Haven, CT 06520-7447
Pearlmutter-Barak@YALE.EDU

Abstract

We discuss advantages and disadvantages of temporally continuous neural networks in contrast to clocked ones, and continue with some "tricks of the trade" of continuous time and recurrent neural networks.

Introduction

Other papers in this volume discuss systems incorporating as subcomponents black boxes whose job is to learn to behave so as to minimize some error measure. Typically these black boxes are filled with some sort of neural network. Here we advocate consideration of temporally continuous networks, and even temporally continuous recurrent networks, to go inside these boxes.

We will restrict our attention to networks with hidden units, and learning algorithms that operate upon them, because networks without hidden units are well treated in the control theory literature, sometimes under more traditional names.

Consider a neural network governed by the equations

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

where y is the time-varying state vector, w the parameters to be modified by the learning, and I a time-varying external input. Given some error metric $E(y, t)$, our task is to modify w to reduce $\int E(y, t)dt$. Our strategy will be gradient descent, so the main portion of our work will be finding algorithms to calculate dE/dw.

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 always assume $\Delta t = 1$, giving

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

with t an integer.

Certainly the behavior of (1) can be precisely duplicated by (2) with suitable choice of f in the latter. For this reason, in order to determine the practical tradeoffs 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

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

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 (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 $\sigma'(\xi) = (1 + \sigma(\xi))(1 - \sigma(\xi)) = 1 - \sigma^2(\xi)$. Although the latter symmetric squashing function is usually preferable, as it has a number of computational advantages, in particular leading to a better conditioned Hessian, which speeds gradient descent [3], the former was used in all the simulations presented below.) The initial conditions $y_i(t_0)$ and driving functions $I_i(t)$ are the inputs to the system.

For the clocked alternative, we consider the analogous

$$y_i(t + 1) = \sigma(x_i(t)) + I_i(t). \tag{5}$$

Both (3) and (5) define rather general dynamic systems. Even assuming that the external input terms $I_i(t)$ are held constant, it is possible for them to exhibit wide ranges of asymptotic behavior. The simplest is a stable fixpoint, but limit cycles and chaos are also possible.

Special learning algorithms are available for various restrictions. There are fixpoint learning algorithms (surveyed in [29], see [31, 1, 13] for more detail or [2] for recent developments) that take advantage of the special relationships holding at a fixpoint 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 feed-forward 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 (surveyed in [30], or see [12, 18, 38, 7] for more detail.)

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. Although in this paper we are concerning ourselves with the general case, it should be noted that for many problems recurrent networks are overkill. For instance, Crutchfield et al. [6] and Lapedes and Farber [20] have had success at the identification of chaotic systems using models without temporally hidden units.

Continuous vs. Discrete Time

Continuous time networks have a number of potential advantages over clocked networks:

- 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 a simple first order difference equations, which is formally identical to a discrete time network. However, regarding the discrete time network running on the computer as a simulation of a continuous time network has a number of advantages. First, more sophisticated and faster simulation techniques than simple first order difference equations can be used, such as higher order forward-backward techniques. 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.
- Continuous time units tend to maintain their values through time, particularly at the start of training. Another way of putting this is that their bias in the learning theory sense is towards temporally continuous tasks, which is certainly advantageous if the task being performed is also temporally continuous.

  Another way of phrasing this is that, although (1) and (2) can describe the same system in general, when we fix f, and the dimensionality of w, only a tiny fraction of this function space, a small manifold, remains available to us. There are two contradictory influences: we would like to be able to perform the task well after much training, so we sould like the best possible solution to be close to something in our manifold; and we would like the manifold itself to be as small as possible, to ease the estimation of w. This means that we are better off whenever we discard from the manifold things that can not possibly be close to the right solution, as this decreases that accessible space without hurting potential performance. Thus, if we know the task to be learned is temporally continuous, we are better off discarding all discontinuous candidates. This is what using a temporally continuous architecture ammounts to.

- A somewhat more subtle advantage is that even for tasks which themselves have no temporal content, such as constraint satisfaction, the best 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 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 benefit of units tending to maintain their state through time is that it helps make information about the past decay more slowly, speeding up learning about the relationship between temporally distant events.
- Embelishments we discuss that are applicable only to continuous time systems, such as mutable time constants and delays, are equally applicable to recurrent and non-recurrent systems, although we discuss them in the former context here.
- It is easier to do stability analysis with a continuous time system.
- Allow error functionals that involve derivatives of the states y to be used. This is important for applications like control, where one might wish to find minimum jerk trajectories. Similarly, error functionals like the elastic network error measure [9, 4] can be used without the inner loop involving sliding the beads along the trajectory, by considering the limit of a continous string of beads.

Algorithms for Computing dE/dw

Now we get to the heart of the matter–the computation of the necessary derivatives. We will consider two major techniques, and then a few more derived from them. The first is the obvious extension of backpropagation through time (BPTT) to continuous time.

Backpropagation Through Time or BPTT

Adjoint equations to (1) are

$$\frac{dz}{dt} = \frac{df(y,w,I)}{dy} z + \frac{\delta E}{\delta y} \tag{6}$$

$$\frac{dE}{dw} \quad = \quad \int_{t_0}^{t_1} y \frac{df(y,w,I)}{dw} z dt. \tag{7}$$

with boundary condition $z(t_1) = 0$. This is similar to the clocked backwards error equations

$$\begin{aligned} z(t-1) &= \frac{df(y,w,I)}{dy} z + \frac{\partial E}{\partial y(t)} \\ \frac{dE}{dw} &= \sum_t y \frac{df(y,w,I)}{dw} z. \end{aligned}$$

where the error to be minimized is E. If this error is of the usual form of an integral $E = \int g(y(t),t)dt$ then we get the simple form $\delta E/\delta y = dg/dy$. For the particular form of (3), this comes to

$$\frac{dz_i}{dt} = z_i - e_i - \sum_j w_{ij} \sigma'(x_j) z_j. \tag{8}$$

$$\frac{\partial E}{\partial w_{ij}} = \int_{t_0}^{t_1} y_i \sigma'(x_j) z_j dt. \tag{9}$$

There are two ways to go about finding such derivations. One is direct, using the calculus of variations [17]. 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 [29]. 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.

Forward Propagation or RTRL

An online, 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 was known in the physics and controls literature, and discovered and applied to recurrent neural networks by Robinson and Fallside [32], and later rediscovered independently by others ([11, 45], for reviews see also [30, 29].) It is called by various researchers *forward propagation* or *RTRL* for real time recurrent learning. In the general case of (1) it is

$$\frac{dE}{dw} = \int_{t_0}^{t_1} \gamma \frac{\delta E}{\delta y} dt \tag{10}$$

where $\gamma(t_0) = 0$ and

$$\frac{d\gamma}{dt} = \frac{df(y,w,I)}{dw} + \frac{df(y,w,I)}{dy} \gamma. \tag{11}$$

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, this can be made an online algorithm by updating the weights continuously instead of actually integrating (10),

$$\frac{dw}{dt} = -\eta\gamma\frac{\delta E}{\delta y}, \tag{12}$$

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

$$\alpha\frac{d^2 w}{dt^2} + (1-\alpha)\frac{dw}{dt} + \eta\gamma\frac{\delta E}{\delta y} = 0. \tag{13}$$

Regretably, the computation of γ is very expensive, and also non-local. 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 $O(n^3 m)$ operations in the general case, but the particular structure of a neural network causes some of the matrices to be sparse, which reduces the burden to $O(n^2 m)$. Nevertheless, this is too high to make the technique practical.

Faster Online Techniques

One way to reduce the complexity of the 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 [46].

Another is to use BPTT with a history cutoff of k units of time, termed BPTT(k) by Williams and Peng [43], and make a small weight change each timestep. This obviates the need for epochs, resulting in a purely online 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 $O(s^{-1}n^2 m + nm)$ 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 [41, 33].

Finally, one can note that, although the forward equations for y are nonlinear, and therefore require numeric integration, the backwards 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 [35], this allows one, during the forward pass, to compute a matrix relating the external inputs to the dE/dw terms, allowing a fully online algorithm with $O(nm)$ time and space complexity.

A number of researchers have recently pointed out that RTRL bears a close relationship to the Extended Kalmann Filter [26, 42]. One advantage of the Extended Kalmann Filter approach is that it rationalizes teacher forcing, in that 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.

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 Ron 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 dilema 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

$$\Delta y = \frac{dy}{dw} \Delta w. \tag{14}$$

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

Time Constants

A major advantage of temporally continuous networks is that one can add additional parameters that control the temporal bahavior in ways known to relate to natural tasks. An example of this is time constants. If we add a time constant T_i to each unit i, modifying (3) to

$$T_i \frac{dy_i}{dt} = -y_i + \sigma(x_i) + I_i, \tag{15}$$

and carry these terms through the derivation of section , equations (8) and (9) become

$$\frac{dz_i}{dt} = \frac{1}{T_i} z_i - e_i - \sum_j \frac{1}{T_j} w_{ij} \sigma'(x_j) z_j. \tag{16}$$

and

$$\frac{\partial E}{\partial w_{ij}} = \frac{1}{T_j} \int_{t_0}^{t_1} y_i \sigma'(x_j) z_j dt. \tag{17}$$

In order to learn these time constants rather than just set them by hand, we need to compute $\partial E(\mathbf{y})/\partial T_i$. This is easily calculated by

$$\frac{\partial E}{\partial T_i} = -\frac{1}{T_i} \int_{t_0}^{t_1} z_i \frac{dy_i}{dt} dt. \tag{18}$$

Time Delays

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

$$x_i(t) = \sum_j w_{ji} y_j(t - \tau_{ji}), \tag{19}$$

τ_{ji} being the time delay along the connection from unit j to unit i. Let us include the variable time constants of section as well. Such time delays merely add analogous time delays to the adjoint error equations, giving

$$\frac{dz_i}{dt}(t) = \frac{1}{T_i} z_i(t) - e_i(t) - \sum_j w_{ij} \sigma'(x_j(t + \tau_{ij})) \frac{1}{T_j} z_j(t + \tau_{ij}), \tag{20}$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{1}{T_j} \int_{t_0}^{t_1} y_i(t) \sigma'(x_j(t + \tau_{ij})) z_j(t + \tau_{ij}) dt, \tag{21}$$

while (18) remains unchanged.

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

$$\frac{\partial E}{\partial \tau_{ij}} = \int_{t_0}^{t_1} z_j(t) \sigma'(x_j(t)) w_{ij} \frac{dy_i}{dt}(t - \tau_{ij}) dt. \tag{22}$$

Watrous et al applied recurrent networks with immutable time delays in the domain of speech [39]. Feedforward networks with immutable time delays (TDNNs) have been applied with great success in the same domain by Lang et al [19]. A variant of TDNNs which learn the time delays was explored by Bodenhausen et al [5]. 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 [36, 8]. Day and Davenport successfully applied a continuous time feedforward network with learned time delays to a difficult time-series prediction task [7].

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 [29].

Some Simulations

In the following simulations, we used networks without time delays, but with mutable time constants. An extra unit whose value was held at 1 by a constant external input, and which had outgoing connections to all other units, was used to implement biases.

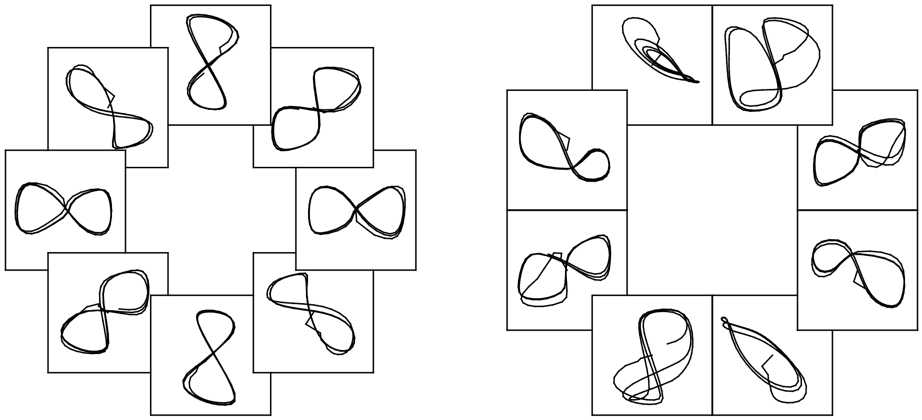

Figure 1: The output of the rotated figure eight network at all the trained angles (left) and some untrained angles (right).

Using first order finite difference approximations, we integrated the system $\mathbf{y}$ forward from t_0 to t_1, set the boundary conditions $z_i(t_1) = 0$, and integrated the system $\mathbf{z}$ backwards 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 backwards as well. We also stored and replayed y_i as it is used in expressions being numerically integrated.

We used the error functional $E = \frac{1}{2}\sum_i \int_{t_0}^{t_1} s_i(\,y_i - d_i)^2 dt$ 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 1 when i was an output unit and 0 otherwise. Throughout, we used $\sigma(\xi) = (1 + e^{-\xi})^{-1}$. Time constants were initialized to 1, 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 equations with $\Delta t = 0.1$.

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. 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 figure 1. But when the network is run with the eight input angles furthest the training angles, as shown on the right in figure 1, generalization is poor.

The task would be simple to solve using second order connections, as they would

Figure 2: 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 eight network (right) in ± 0.05.

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.

Stability and Perturbation Experiments

In an attempt to judge the stability of the limit cycles exhibited by networks of this sort trained in this fashion, we introduced random perturbations and observed the effects of these perturbations upon the evolution of two networks: one trained to trace out of circular trajectory, and the other trained to trace out a figure eight. The results are shown in figure 2.

The limit cycle on the right is symmetric without disturbances, but when perturbations are introduced, symmetry is broken. The portion of the limit cycle moving from the upper left hand corner towards 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 towards 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 towards 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, Simard, Rayzs and Victorri have developed a technique for learning the local maximum eigenvalue of the transfer function [34], optionally projecting out directions whose eigenvalues are not of interest. Their technique explicitly modulates the behavior we only measured above.

Leech Simulations

Lockery et al. used BPTT in continuous time to fit a low level neurophysiological model of the leech local bending reflex to data on sensory and motor neuron activity [22, 23, 24, 25]. They modified the dynamic equations substantially in order to model their system at a low level, using activity levels to represent currents rather than voltages. Their trained model disagreed with human intuition concerning what the synaptic strengths, and in fact signs, would be, but qualitatively matched empirical measurements of interneuron synaptic strengths in the leech *Hirudo medicinalis.*

Teacher Forcing

Teacher forcing [44] consists of jamming the desired output values into 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. Williams and Zipser report that their teacher forcing technique radically reduced training time for their recurrent networks [44], although when this author used teacher forcing on networks with a larger number of hidden units he had difficulties[29].

However, by taking the limit as the step size goes to zero, it is possible to show that the continuous time analogue 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 descrete time case, which can be seen as nearly the same thing.

Regretably 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 Williams and Zipser for some cases where oscillations trained with teacher forcing exhibited radically and systematically lower frequency and amplitude when running free [44].

We also note that “Jordan Nets” [16] can be regarded as networks in which the teacher-forced output units cut all recurrent pathways.

Learning with Scale Parameters

The parameters usually modified 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 other, 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 match some criteria. Frequently another term is added as an expression of some a-priori on 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 [40, 28].

However, we have added some new sorts of parameters, namely time constants and time delays, here represented generically by T. These are *scale parameters,* which are different in a number of ways. For instance, they must not become negative. As they approach zero, the dynamics of the associated network becomes more and more sensitive to small changes. This means that we must add machinery to enforce the constraint, and that our gradient descent may become unstable as they approach zero. Also, the flat zero-knowledge prior no longer seems appropriate.

All these problems can be solved in a single stroke by noting that the correct null hypothesis for scale parameters is not flat in their values, but rather flat in their log values. This corresponds to doing gradient descent in $\mathcal{L}_T = \log T$ rather than in T itself. Doing this also solves our other problems, as it makes the parameters more stiff near zero, compensating for the network's increased sensitivity; and it enforces $T > 0$ since $T = \exp \mathcal{L}_T > 0$ for real $\mathcal{L}_T$, enforcing the constraint for free.

In addition, weight decay of scale parameters becomes more natural, as decaying $\mathcal{L}_T$ towards 0 corresponds to decaying T towards 1, which is a reasonable target. Of course, a constant factor can be put in to make the decay towards some other a-priori most likely scale.

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 apparently proven useful in this particular situation is that of Robert Jacobs [15] which was applied by Fang and Sejnowski to the single figure eight problem with great success [10]. It was also used in the leech simulations of Lockery et al. described in section , again leading to much faster training.

Conclusions

Certainly there is no reason to use a recurrent network when a 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 practicle than they seemed only a few years ago.

References

[1] L. B. Almeida. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In Maureen Caudill and Charles Butler, editors, *IEEE First International Conference on Neural Networks*, volume 2, pages 609–618, San Diego, CA, June 21–24 1987.

[2] Pierre Baldi and Fernando Pineda. Contrastive learning and neural oscillations. *Neural Computation*, 3(4):526–545, 1991.

[3] Sue Becker and Yann le Cun. Improving the convergence of backpropagation learning with second order methods. In David S. Touretzky, Geoffrey E. Hinton, and Terrence J. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School*. Morgan Kaufmann, 1989. Also published as Technical Report CRG-TR-88-5, Department of Computer Science, University of Toronto.

[4] Hugues Bersini, Luis Gonzalez Sotelino, and Eric Decossaux. Hopfield net generation of trajectories in constrained environment. *In this volume*.

[5] U. Bodenhausen. Learning internal representations of pattern sequences in a neural network with adaptive time-delays. In *International Joint Conference on Neural Networks*, San Diego, CA, June 1990. IEEE.

[6] J. P. Crutchfield and B. S. McNamara. Equations of motion from a data series. *Complex Systems*, 1:417–452, 1987.

[7] Shawn P. Day and Michael R. Davenport. Continuous-time temporal backpropagation with adaptable time delays. Available by ftp, archive.cis.ohio-state.edu, pub/neuroprose/day.temporal.ps.Z, August 1991.

[8] Bert de Vries and Jose C. Principe. A theory for neural networks with time delays. In Lippmann et al. [21], pages 162–168.

[9] R. Durbin and D. Willshaw. An analogue approach to the travelling salesman problem using an elastic net method. *Nature*, 326:689–691, 1987.

[10] Yan Fang and Terrence J. Sejnowski. Faster learning for dynamic recurrent backpropagation. *Neural Computation*, 2(3):270–273, 1990.

[11] Michael Gherrity. A learning algorithm for analog, fully recurrent neural networks. In IJCNN89 [14], pages 643–644.

[12] Marco Gori, Yoshua Bengio, and Renato De Mori. Bps: A learning algorithm for capturing the dynamic nature of speech. In IJCNN89 [14], pages 417–423.

[13] Geoffrey E. Hinton. Deterministic Boltzmann learning performs steepest descent in weight-space. *Neural Computation*, 1(1):143–150, 1989.

[14] *International Joint Conference on Neural Networks*, Washington DC, June 18–22 1989. IEEE.

[15] Robert Jacobs. Increased rates of convergence through learning rate adaptation. Technical Report COINS 87-117, University of Massachusetts, Amherst, MA 01003, 1987.

[16] Michael I. Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the 1986 Cognitive Science Conference*, pages 531–546. Lawrence Erlbaum Associates, 1986.

[17] Arthur E. Bryson Jr. A steepest ascent method for solving optimum programming problems. *Journal of Applied Mechanics*, 29(2):247, 1962.

[18] Gary Kuhn. A first look at phonetic discrimination using connectionist models with recurrent links. SCIMP working paper 82018, Institute for Defense Analysis, Princeton, New Jersey, April 1987.

[19] Kevin J. Lang, Geoffrey E. Hinton, and Alex Waibel. A time-delay neural network architecture for isolated word recognition. *Neural Networks*, 3(1):23–43, 1990.

[20] Alan Lapedes and Robert Farber. Nonlinear signal processing using neural networks: Prediction and system modelling. Technical report, Theoretical Division, Los Alamos National Laboratory, 1987.

[21] Richard P. Lippmann, John E. Moody, and David S. Touretzky, editors. *Advances in Neural Information Processing Systems 3*. Morgan Kaufmann, 1991.

[22] Shawn R. Lockery, Yan Fang, and Terrence J. Sejnowski. A dynamic neural network model of sensorimotor transformations in the leech. *Neural Computation*, 2(3):274–282, 1990.

[23] Shawn R. Lockery and W. B. Kristan Jr. Distributed processing of sensory information in the leech i: Input-output relations of the local bending relex. *Journal of Neuroscience*, 1990.

[24] Shawn R. Lockery and W. B. Kristan Jr. Distributed processing of sensory information in the leech ii: Identification of interneurons contributing to the local bending reflex. *Journal of Neuroscience*, 1990.

[25] Shawn R. Lockery, G. Wittenberg, W. B. Kristan Jr., N. Qian, and T. J. Sejnowski. Neural network analysis of distributed representations of sensory information in the leech. In Touretzky [37], pages 28–35.

[26] M. B. Matthews. Neural network nonlinear adaptive filtering using the extended kalman filter algorithm. In *Proceedings of the International Neural Networks Conference*, volume 1, pages 115–119, Paris, France, July 1990.

[27] John E. Moody, Steve J. Hanson, and Richard P. Lippmann, editors. *Advances in Neural Information Processing Systems 4*. Morgan Kaufmann, 1992.

[28] S. J. Nowlan and G. E. Hinton. Adaptive soft weight tying using gaussian mixtures. In Moody et al. [27].

[29] Barak A. Pearlmutter. Dynamic recurrent neural networks. Technical Report CMU-CS-90-196, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, December 1990.

[30] Barak A. Pearlmutter. Two new learning procedures for recurrent networks. *Neural Network Review*, 3(3):99–101, 1990.

[31] Fernando Pineda. Generalization of back-propagation to recurrent neural networks. *Physical Review Letters*, 19(59):2229–2232, 1987.

[32] A. J. Robinson and F. Fallside. Static and dynamic error propagation networks with application to speech coding. In Dana Z. Anderson, editor, *Neural Information Processing Systems*, pages 632–641, New York, New York, 1987. American Institute of Physics.

[33] Juergen Schmidhuber. An $O(n^3)$ learning algorithm for fully recurrent networks. Technical Report FKI-151-91, Institut fuer Informatik, Muenchen, Germany, May 1991. Or ftp flop.informatik.tu-muenchen.de pub/fki/fki151.ps.Z.

[34] Patrice Y. Simard, Jean Pierre Rayzs, and Bernard Victorri. Shaping the state space landscape in recurrent networks. In Lippmann et al. [21], pages 105–112.

[35] G.Z. Sun, H.H. Chen, and Y.C. Lee. Green's function method for fast on-line learning algorithm of recurrent method for fast on-line learning algorithm of recurrent nn. In Moody et al. [27].

[36] D.W. Tank and J.J. Hopfield. Neural computation by time compression. *Proceedings of the National Academy of Sciences*, 84:1896–1900, 1987.

[37] David S. Touretzky, editor. *Advances in Neural Information Processing Systems II*. Morgan Kaufmann, 1990.

[38] Tadasu Uchiyama, Katsunori Shimohara, and Yukio Tokunaga. A modified leaky integrator network for temporal pattern recognition. In IJCNN89 [14], pages 469–475.

[39] R. L. Watrous, B. Laedendorf, and G. Kuhn. Complete gradient optimization of a recurrent network applied to bdg descrimination. *Journal of the Acoustic Society*, 1989, in press.

[40] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman. Generalization by weight-elimination with application to forecasting. In Lippmann et al. [21], pages 875–882.

[41] R. J. Williams. Complexity of exact gradient computation algorithms for recurrent neural networks. Technical Report NU-CCS-89-27, College of Computer Science, Northeastern University, Boston, MA, 1989.

[42] R. J. Williams. Some observations on the use of the extended kalman filter as a recurrent network learning algorithm. Technical Report NU-CCS-92-1, College of Computer Science, Northeastern University, Boston, MA, 1992.

[43] Ronald J. WIlliams and Jing Peng. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural Computation*, 2(4):490–501, 1990.

[44] Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. Technical Report ICS Report 8805, UCSD, La Jolla, CA 92093, November 1988.

[45] Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989.

[46] David Zipser. Subgrouping reduces complexity and speeds up learning in recurrent networks. In Touretzky [37], pages 638–641.