5.4 Extensions of Backprop for Temporal Learning

Up to this point we have been concerned with "static" mapping networks which are trained to produce a spatial output pattern in response to a particular spatial input pattern. However, in many engineering, scientific, and economic applications, the need arises to model dynamical processes where a time sequence is required in response to certain temporal input signal(s). One such example is plant modeling in control applications. Here, it is desired to capture the dynamics of an unknown plant (usually nonlinear) by modeling a flexible-structured network that will imitate the plant by adaptively changing its parameters to track the plant's observable output signals when driven by the same input signals. The resulting model is referred to as a temporal association network.

Temporal association networks must have a recurrent (as opposed to static) architecture in order to handle the time dependent nature of associations. Thus, it would be very useful to extend the multilayer feedforward network and its associated training algorithm(s) (e.g., backprop) into the temporal domain. In general, this requires a recurrent architecture (nets with feedback connections) and proper associated learning algorithms.

Two special cases of temporal association networks are sequence reproduction and sequence recognition networks. For sequence reproduction, a network must be able to generate the rest of a sequence from a part of that sequence. This is appropriate, for example, for predicting the price trend of a given stock market from its past history or predicting the future course of a time series from examples. In sequence recognition, a network produces a spatial pattern or a fixed output in response to a specific input sequence. This is appropriate, for example, for speech recognition, where the output encodes the word corresponding to the speech signal. NETtalk and Glove-Talk of Section 5.3 are two other examples of sequence recognition networks.

In the following, neural net architectures having various degrees of recurrency and their associated learning methods are introduced which are capable of processing time sequences.

5.4.1 Time-Delay Neural Networks

Consider the time-delay neural network architecture shown in Figure 5.4.1. This maps a finite time sequence $\{x(t), x(t-\Delta), x(t-2\Delta), \cdots, x(t-m\Delta)\}$ into a single output y (this can also be generalized for the case when x and/or y are vectors). One may view this neural network as a discrete-time nonlinear filter (we may also use the borrowed terms finite-duration impulse response (FIR) filter or nonrecursive filter from the linear filtering literature).

The architecture in Figure 5.4.1 is equivalent to a single hidden layer feedforward neural network receiving the $(m + 1)$-dimensional "spatial" pattern $\mathbf{x}$ generated by a tapped delay line preprocessor from a temporal sequence. Thus, if target values for the output unit are specified for various times t, backprop may be used to train the above network to act as a sequence recognizer.

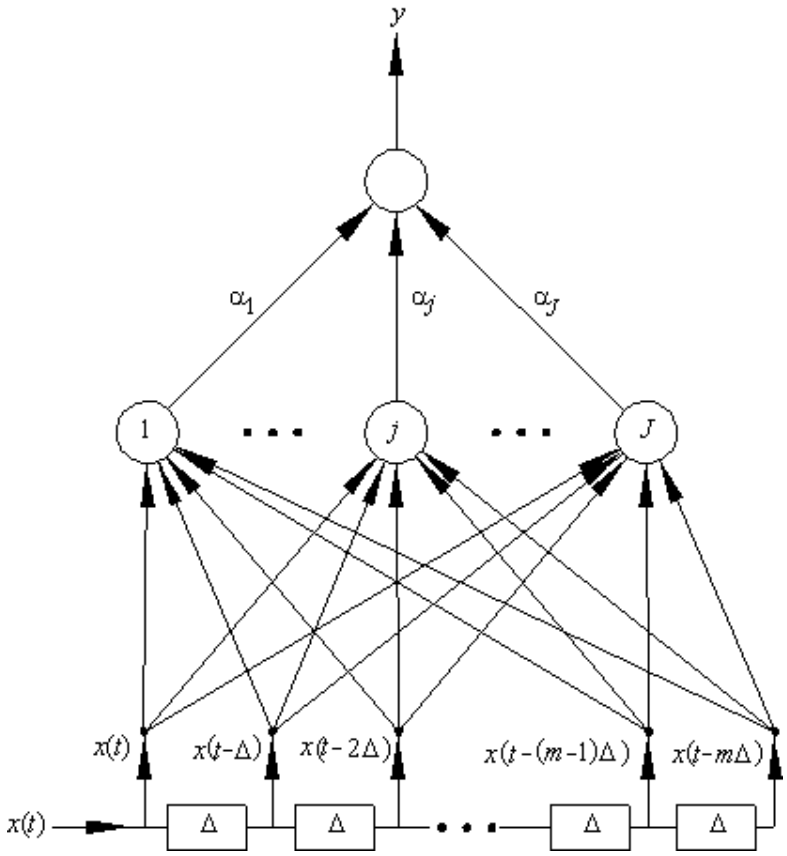


Figure 5.4.1. A time-delay neural network for one-dimensional input/output signals.

The time-delay neural net has been successfully applied to the problem of speech recognition (e.g., Tank and Hopfield, 1987; Elamn and Zipser, 1988; Waibel, 1989; Waibel et al., 1989; Lippmann, 1989) and time series prediction (Lapedes and Farber, 1988; Weigend et al., 1991). Here, we discuss time series prediction since it captures the spirit of the type of processing done by the time-delay neural net. Given observed values of the state x of a (nonlinear) dynamical system at discrete times less than t, the goal is to use these values to accurately predict $x(t+p)$, where p is some prediction time step into the future (for simplicity, we assume a one dimensional state x). Clearly, as p increases the quality of the predicted value will degrade for any predictive method. A method is robust if it can maintain prediction accuracy for a wide range of p values.

As is normally done in linear signal processing applications (e.g., Widrow and Stearns, 1985), one may use the tapped delay line nonlinear filter of Figure 5.4.1 as the basis for predicting $x(t+p)$. Here, a training set is constructed of pairs $\{\mathbf{x}^k, x(t_k+p)\}$, where $\mathbf{x}^k = [x(t_k), x(t_k-\Delta), x(t_k-2\Delta), \cdots, x(t_k-m\Delta)]^T$. Backprop may now be employed to learn such a training set. Reported simulation results of this prediction method show comparable or better performance compared to other non neural network-based techniques (Lapedes and Farber, 1988; Weigend et al., 1991; Weigend and Gershenfeld, 1993).

Theoretical justification for the above approach is available in the form of a very powerful theorem by Takens (1981), which states that there exists a functional relation of the form

$$x(t+p) = g[x(t), x(t-\Delta), \cdots, x(t-m\Delta)] \quad (5.4.1)$$

with $d < m < 2d$, as long as the trajectory $x(t)$ evolves towards compact attracting manifolds of dimension d. This theorem, however, provides no information on the form of g or the value of . The time-delay neural network approach provides a robust approximation for g in Equation (5.4.1) in the form of the continuous, adaptive parameter model

$$y = \sum_{j=1}^{J} \alpha_j f_h\left(\sum_{i=1}^{m+1} w_{ji} x(t-(i-1)\Delta)\right) \quad (5.4.2)$$

where a linear activation is assumed for the output unit, and f_h is the nonlinear activation of hidden units.

A simple modification to the time-delay net makes it suitable for sequence reproduction. The training procedure is identical to the one for the above prediction network. However, during retrieval, the output y [predicting $x(t+1)$] is propagated through a single delay element, with the output of this delay element connected to the input of the time-delay net as is shown in Figure 5.4.2. This sequence reproduction net will only work if the prediction $y = \hat{x}(t+1)$ is very accurate since any error in the predicted signal has a multiplicative effect due to the iterated scheme employed.

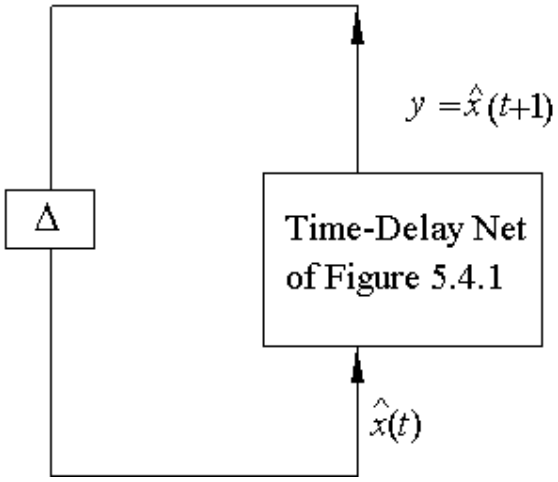


Figure 5.4.2. Sequence reproduction network.

Further generalization of the above ideas can result in a network for temporal association. We present such modifications in the context of nonlinear dynamical plant identification/modeling of control theory. Consider the following general nonlinear single input, single output plant described by the difference equation:

$$x(t+1) = g\left[x(t), x(t-1), \cdots, x(t-n); u(t), u(t-1), \cdots, u(t-m)\right] \quad (5.4.3)$$

where $u(t)$ and $x(t)$ are, respectively, the input and output signals of the plant at time t, g is a nonlinear function, and $m \leq n$. We are interested in training a suitable layered neural network to capture the dynamics of the plant in Equation (5.4.3), thus modeling the plant. Here, we assume that the order of the plant is known (m and n are known). The general form of Equation (5.4.3) suggests the use of a time-delay neural network shown inside the dashed rectangle in Figure 5.4.3. This may also be viewed as a (nonlinear) recursive filter, termed infinite-duration impulse response (IIR) filter in the linear filtering literature.

using

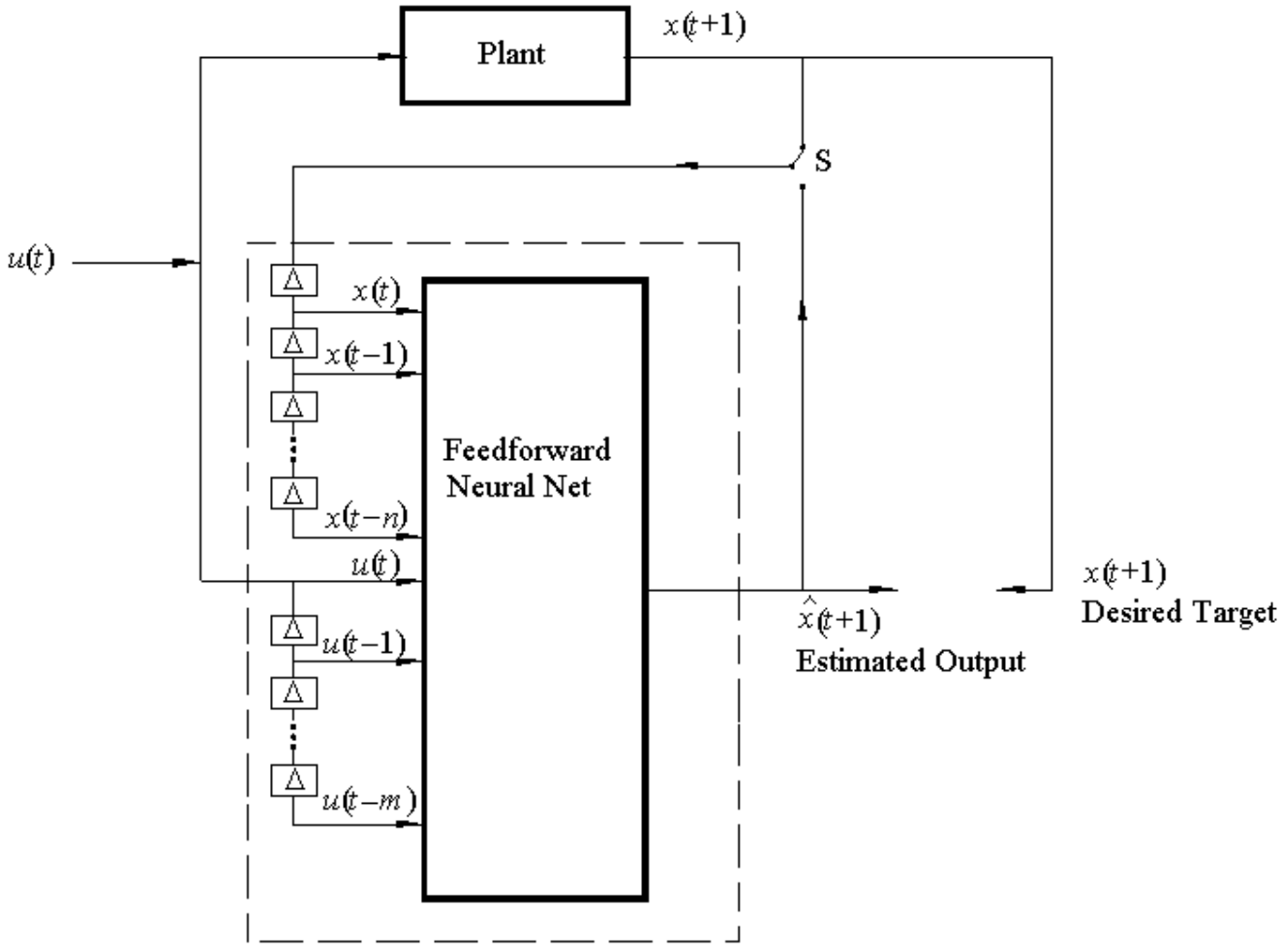


Figure 5.4.3. A time-delay neural network setup for the identification of a nonlinear plant.

During training, the neural network and the plant receive the same input $u(t)$. The neural network also receives the plant's output $x(t+1)$ (switch S in the up position in Figure 5.4.3). Backprop can be used to update the weights of the neural network based on the "static" mapping pairs

$$\left\{\left[x(t)\quad x(t-1)\quad \cdots\quad x(t-n)\quad u(t)\quad u(t-1)\quad \cdots\quad u(t-m)\right]^{\mathrm{T}},\ x(t+1)\right\}$$

for various values of t. This identification scheme is referred to as series-parallel identification model (Narendra and Parthasarathy, 1990). After training, the neural network with the switch S in the down position ($\hat{x}(t+1)$ is fed back as the input to the top delay line in Figure 5.4.3) will generate (recursively) an output time sequence in response to an input time sequence. If the training was successful, one would expect the output $\hat{x}(t+1)$ to approximate the actual output of the plant, $x(t+1)$, for the same input signal $u(t)$ and same initial conditions. Theoretical justifications for the effectiveness of this neural network identification method can be found in Levin and Narendra (1992).

Narendra and Parthasarathy (1990) reported successful identification of nonlinear plants by time-delay neural networks similar to the one in Figure 5.4.3. In one of their simulations, the feedforward part of the neural network consisted of a two hidden layer network with five inputs and a single linear output unit. The two hidden layers consisted of 20 and 10 units with bipolar sigmoid activations, respectively. This network was used to identify the unknown plant

$$x(t+1)=\frac{x(t)x(t-1)x(t-2)u(t-1)\left[x(t-2)-1\right]+u(t)}{1+x^2(t-2)+x^2(t-1)} \qquad (5.4.4)$$

The inputs to the neural network during training were $x(t), x(t-1), x(t-2), u(t),$ and $u(t-1)$. Incremental backprop was used to train the network using a uniformly distributed random input signal whose amplitude was in the interval [-1, +1]. The training phase consisted of 100,000 training iterations, which amounts to one training cycle over the random inputs signal $u(t)$, $0 < t$ 100,000. A learning rate of 0.25

was used. Figure 5.4.4 (a) shows the output of the plant (solid line) and the model (dotted line) for the input signal

$$u(t) = \begin{cases} \sin\left(\frac{2\pi t}{250}\right) & \text{for } 0 \le t \le 500 \\ 0.8\sin\left(\frac{2\pi t}{250}\right) + 0.2\sin\left(\frac{2\pi t}{25}\right) & \text{for } t > 500 \end{cases} \quad (5.4.5)$$

It should be noted that in the above simulation no attempt has been made to optimize the network size or to tune the learning process. For example, Figure 5.4.4 (b) shows simulation results with a single hidden layer net consisting of twenty bipolar sigmoid activation hidden units. Here, incremental backprop with a learning rate of 0.25 was used. The training phase consisted of 5 10_6 iterations. This amounts to 10,000 training cycles over a 500 sample input signal having the same characteristics as described above.

Other learning algorithms may be used for training the time-delay neural network discussed above, some of which are extensions of algorithms used in classical linear adaptive filtering or adaptive control. Nerrand et al. (1993) present examples of such algorithms.

(a) (b)

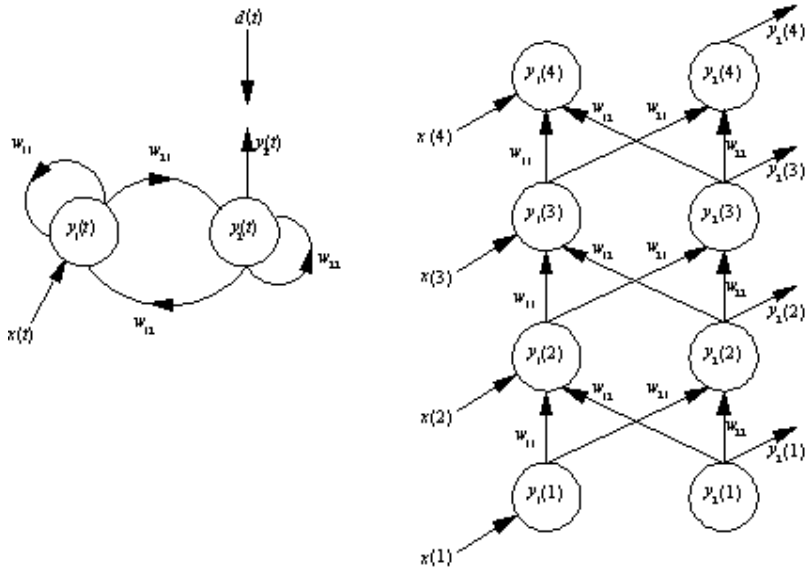
Figure 5.4.4. Identification results for the plant in Equation (5.4.4) a time-delay neural network. Plant output $x(t)$ (solid line) and neural network output $\hat{x}(t)$ (dotted line) in response to the input signal in Equation (5.4.5). (a) The network has two hidden layers and is trained with incremental backprop for one cycle over a 100,000 sample random input signal. (Adapted from K. S. Narendra and K. Parthasarathy, 1990, Identification and Control of Dynamical Systems Containing Neural Networks, *IEEE Trans. on Neural Networks*, **1**(1), 4-27, ©1990 IEEE.) (b) The network has a single hidden layer and is trained for 10,000 cycles over a 500 sample random input signal.

See how it works interactively

5.4.2 Backpropagation Through Time

In the previous section, a partially recurrent neural network is presented which is capable of temporal association. In general, however, a fully recurrent neural net is a more appropriate/economic alternative. Here, individual units may be input units, output units, or both. The desired targets are defined on a set of arbitrary units at certain predetermined times. Also, arbitrary interconnection patterns between units can exist. An example of a simple two-unit fully interconnected network is shown in Figure 5.4.5(a). The network receives an input sequence $x(t)$ at unit 1, and it is desired that the network generates the sequence $d(t)$ as the output $y_2(t)$ of unit 2.

A network which behaves identically to the above simple recurrent net over the time steps t = 1, 2, 3, and 4 is shown in Figure 5.4.5(b). This amounts to unfolding the recurrent network in time (Minsky and Papert, 1969) to arrive at a feedforward layered network. The number of resulting layers is equal to the unfolding time interval T. This idea is effective when T is small and limits the maximum length of sequences that can be generated. Here, all units in the recurrent network are duplicated T times, so that a separate unit in the unfolded network holds the state $y_i(t)$ of the equivalent recurrent network at time t. Note that the connections w_{ij} from unit j to unit i in the unfolded network are identical for all layers.



(a) (b)

Figure 5.4.5. (a) A simple recurrent network. (b) A feedforward network generated by unfolding in time the recurrent net in (a). The two networks are equivalent over the four time steps $t = 1, 2, 3, 4$.

The resulting unfolded network simplifies the training process of encoding the $x(t)$ $d(t)$ sequence association since now backprop learning is applicable. However, we should note a couple of things here. First, targets may be specified for hidden units. Thus, errors at the output of hidden units, and not just the output errors, must be propagated backward from the layer in which they originate. Secondly, it is important to realize the constraint that all copies of each weight w_{ij} must remain identical across duplicated layers (backprop normally produces different increments w_{ij} for each particular weight copy). A simple solution is to add together the individual weight changes for all copies of a partial weight w_{ij} and then change all such copies by the total amount. Once trained, a copy of the weights from any layer of the unfolded net are copied into the recurrent network which, in turn, is used for the temporal association task. Adapting backprop to training unfolded recurrent neural nets results in the so-called backpropagation through time learning method (Rumelhart et al., 1986b). There exist relatively few applications of this technique in the literature (e.g., Rumelhart et al., 1986b; Nowlan, 1988; Nguyen and Widrow, 1989). One reason is its inefficiency in handling long sequences. Another reason is that other learning methods are able to solve the problem without the need for unfolding. These methods are treated next. But first, we describe one interesting application of backpropagation through time: The truck backer-upper problem.

Consider the trailer truck system shown in Figure 5.4.6. The goal is to design a controller which successfully backs-up the truck so that the back of the trailer designated by coordinates (x, y) ends at (0, 0) with the trailer perpendicular to the dock (i.e., the trailer angle t is zero), and where only backward movements of the cab are allowed. The controller receives the observed state $\mathbf{x} = [x, y, t, c]^T$ (c is the cab angle) and produces a steering signal (angle) s. It is assumed that the truck backs-up at a constant speed. The details of the trailer truck kinematics can be found in Miller et al. (1990a). The original application assumes six state variables, including the position of the back of the cab. However, these two additional variables may be eliminated if the length of the cab and that of the trailer are given.

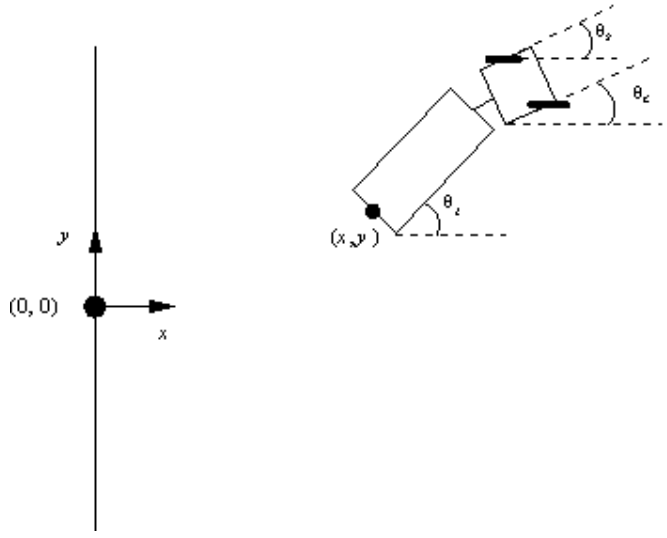


Figure 5.4.6. A pictorial representation of the truck backer-upper problem. The objective here is to design a controller which generates a steering signal s which successfully backs-up the truck so that the back of the trailer ends at the (0, 0) reference point, with the trailer perpendicular to the dock.

Before the controller is designed, a feedforward single hidden layer neural network is trained, using backprop, to emulate the truck and trailer kinematics. This is accomplished by training the network on a large number of backup trajectories (corresponding to random initial trailer truck position configurations), each consisting of a set of association pairs $\{[\mathbf{x}(k\text{-}1)^T \quad s(k\text{-}1)]^T, \mathbf{x}(k)\}$ where $k = 1, 2, ..., T$, and T represents the number of backup steps till the trailer hits the dock or leaves some predesignated borders of the parking lot (T depends on the initial state of the truck and the applied steering signal s). The steering signal was selected randomly during this training process. The general idea for training the emulator is depicted in the block diagram of Figure 5.4.3 for the identification of nonlinear dynamical systems by a neural network. However, the tapped delay lines are not needed here because of the kinematic nature of the trailer truck system. Next, the trained emulator network is used to train the controller. Once trained, the controller is used to control the real system. The reason for training the controller with the emulator and not with the real system is justified below.

Figure 5.4.7 shows the controller/emulator system in a retrieval mode. The whole system is recurrent due to the external feedback loops (actually, the system exhibits partial recurrence since the emulator is a feedforward network and since it will be assumed that the controller has a feedforward single hidden layer architecture). The controller and the emulator are labeled C and E, respectively, in the figure. The controller receives the input vector $\mathbf{x}(k)$ and responds with a single output $s(k)$, representing the control signal.

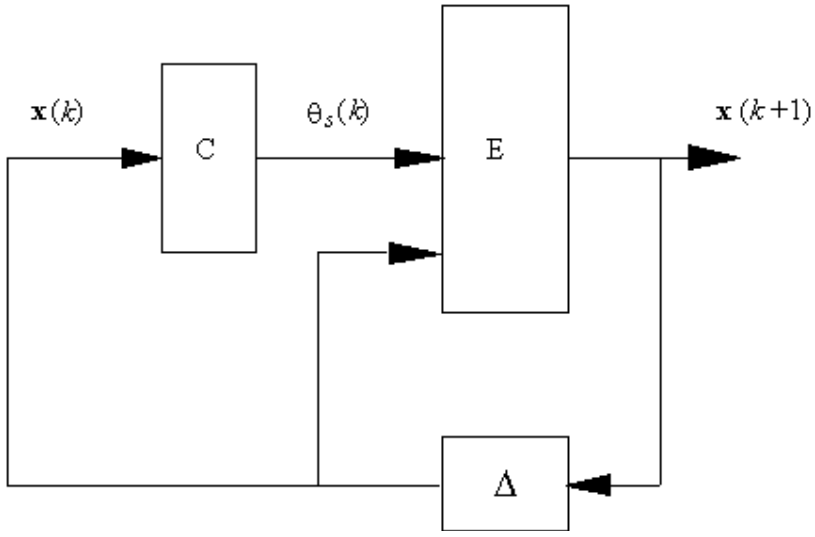


Figure 5.4.7. Controller/emulator retrieval system.

The idea of unfolding in time is applicable here. When initialized with state $\mathbf{x}(0)$, the system with the untrained, randomly initialized controller neural network evolves over T time steps until its state enters a restricted region (i.e., the trailer hits the borders). Unfolding the controller/emulator neural network T time steps results in the T-level feedforward network of Figure 5.4.8. This unfolded network has a total of $4T$ - 1 layers of hidden units. The backpropagation through time technique can now be applied to adapt the controller weights. The only units with specified desired targets are the three units of the output layer at level T representing x, y, and t. The desired target vector is the zero vector.

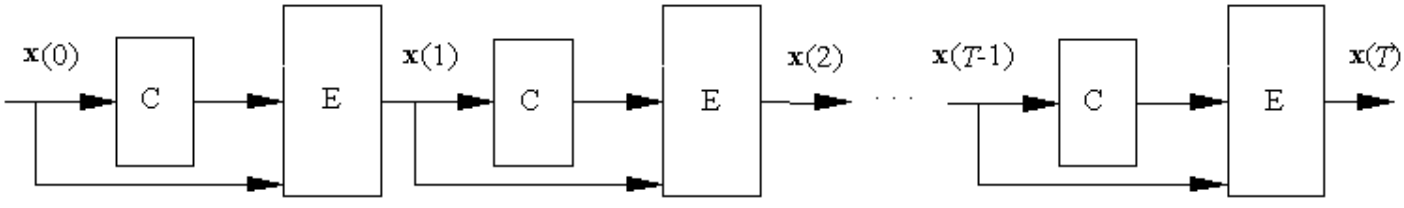


Figure 5.4.8. Unfolded trailer truck controller/emulator network over T time steps.

Once the output layer errors are computed, they are propagated back through the emulator network units and through the controller network units. Here, only the controller weights are adjusted (with equal increments for all copies of the same weight as discussed earlier). The need to propagate the error through the plant block necessitates that a neural network-based plant emulator be used to replace the plant during training. The trained controller is capable of backing the truck from any initial state, as long as it has sufficient clearance from the loading dock. Thousands of backups are required to train the controller. It is helpful (but not necessary) to start the learning with "easy" initial cases and then proceed to train with more difficult cases. Typical backup trajectories are shown in Figure 5.4.9.

(a)

(b)

(c)

Figure 5.4.9. Typical backup trajectories for the trailer truck which resulted by employing a backpropagation through time trained controller (a) Initial state, (b) trajectory, and (c) final state. (Courtesy of Lee Feldkamp and Gint Puskorius of Ford Research Laboratory, Dearborn, Michigan.)

5.4.3 Recurrent Backpropagation

This section presents an extension of backprop to fully recurrent networks where the units are assumed to have continuously evolving states. The new algorithm is used to encode " spatial" input/output associations as stable equilibria of the recurrent network; i.e., after training on a set of $\{\mathbf{x}_k, \mathbf{d}_k\}$ pattern pairs, the presentation of $\mathbf{x}_k$ is supposed to drive the network's output $\mathbf{y}(t)$ towards the fixed attractor state $\mathbf{d}_k$. Thus, the extension here is still restricted to learning " static" mappings as opposed to temporal association; however, it will serve as the basis for other extensions of backprop to sequence association which are discussed later in this section. The present extension, usually called recurrent backpropagation, was proposed independently by Pineda (1987, 1988) and Almeida (1987, 1988).

Consider a recurrent network of N units with outputs y_i, connections w_{ij}, and activations $f(net_i)$. A simple example ($N = 2$) of such a network is shown in Figure 5.4.5a. A unit is an input unit if it receives an element x_i^k of the input pattern $\mathbf{x}_k$. By definition, non-input units will be assigned an input $x_i^k = 0$. Output units are designated as units with prespecified desired outputs d_i^k. In general, a unit may belong to the set of input units and the set of output units, simultaneously, or it may be "hidden" in the sense that it is neither an input nor an output unit. Henceforth, the pattern index k is dropped for convenience.

A biologically as well as electronically motivated choice for the state evolution of unit i is given by (refer to Equations (4.7.8) and (7.1.19), respectively).

$$\frac{dy_i}{dt} = -y_i + f\left(\sum_j w_{ij}\, y_j + x_i\right) = -y_i + f(net_i), \qquad i = 1, 2, \cdots, N \tag{5.4.6}$$

where net_i represents the total input activity of unit i and $-y_i$ simulates natural signal decay. By setting $\frac{dy_i}{dt} = 0$, one arrives at the equilibrium points $\mathbf{y}^*$ of the above system, given by:

$$y_i^* = f\left(net_i^*\right) = f\left(\sum_j w_{ij} y_j^* + x_i\right) \tag{5.4.7}$$

The following is a derivation of a learning rule for the system/network in Equation (5.4.6), which assumes the existence and asymptotic stability of at least one equilibrium point $\mathbf{y}^*$, $\mathbf{y}^* = \left[y_1^*\ y_2^* \cdots y_N^*\right]^T$, in Equation (5.4.7). This equilibrium point(s) represent the steady-state response of the network. Suppose that the network has converged to an equilibrium state $\mathbf{y}^*$ in response to an input $\mathbf{x}$. Then, if neuron i is an output neuron, it will respond with y_i^*. This output is compared to the desired response d_i, resulting in an error signal E_i. The goal is to adjust the weights of the network in such a way that the state $\mathbf{y}^*$ ultimately becomes equal to the desired response $\mathbf{d}$ associated with the input $\mathbf{x}$. In other words, our goal is to minimize the error function

$$E = \frac{1}{2}\sum_{i=1}^{N}\left(d_i - y_i^*\right)^2 = \frac{1}{2}\sum_{i=1}^{N} E_i^{*2} \quad (5.4.8)$$

with $E_i^* = 0$ if unit i is not an output unit. Note that an instantaneous error function is used so that the resulting weight update rule is incremental in nature. Using gradient descent search to update the weight w_{pq} gives

$$\Delta w_{pq} = -\rho\frac{\partial E}{\partial w_{pq}} = \rho\sum_i E_i^*\frac{\partial y_i^*}{\partial w_{pq}} \quad (5.4.9)$$

with $\frac{\partial y_i^*}{\partial w_{pq}}$ given by differentiating Equation (5.4.7) to obtain

$$\frac{\partial y_i^*}{\partial w_{pq}} = f'\left(net_i^*\right)\left[\delta_{ip}\, y_q^* + \sum_j w_{ij}\frac{\partial y_j^*}{\partial w_{pq}}\right] \quad (5.4.10)$$

where $_{ip}$ is the Kronecker delta ($_{ip} = 1$ if $i = p$ and zero otherwise). Another way of writing Equation (5.4.10) is

$$\sum_j \mathbf{L}_{ij}\frac{\partial y_j^*}{\partial w_{pq}} = \delta_{ip}\, f'(net_i^*)\, y_q^* \quad (5.4.11)$$

where

$$\mathbf{L}_{ij} = \delta_{ij} - f'(net_i^*)\, w_{ij} \quad (5.4.12)$$

Now, one may solve for $\frac{\partial y_i^*}{\partial w_{pq}}$ by inverting the set of linear equations represented by Equation (5.4.11) and get

$$\frac{\partial y_i^*}{\partial w_{pq}} = \left(\mathbf{L}^{-1}\right)_{ip} f'(net_p^*)\, y_q^* \quad (5.4.13)$$

where $(\mathbf{L}^{-1})_{ip}$ is the ipth element of the inverse matrix $\mathbf{L}^{-1}$. Hence, substituting Equation (5.4.13) in Equation (5.4.9) gives the desired learning rule:

$$\Delta w_{pq} = \rho\, f'\left(net_p^*\right)\sum_i E_i^*\left(\mathbf{L}^{-1}\right)_{ip} y_q^* \quad (5.4.14)$$

When the recurrent network is fully connected, then the matrix $\mathbf{L}$ is $N\,N$ and its inversion requires $O(N^3)$ operations using standard matrix inversion methods. Pineda and Almeida independently showed that a more economical local implementation, utilizing a modified recurrent neural network of the same size as the original network, is possible. This implementation has $O(N^2)$ computational complexity and is usually called recurrent backpropagation. To see this, consider the summation term in Equation (5.4.14) and define it as z_p^*:

$$z_p^* = \sum_i E_i^*\left(\mathbf{L}^{-1}\right)_{ip} \quad (5.4.15)$$

Then, undoing the matrix inversion in Equation (5.4.15) leads to the set of linear equations for z_p^*, as shown by

$$\sum_p \mathbf{L}_{pi} z_p^* = E_i^* \tag{5.4.16}$$

or, substituting for **L** from Equation (5.4.12), renaming the index *p* as *j*, and rearranging terms,

$$z_i^* = \sum_j f'(net_j^*)\, w_{ji} z_j^* + E_i^* \tag{5.4.17}$$

This equation can be solved using an analog network of units z_i with the dynamics

$$\frac{dz_i}{dt} = -z_i + \sum_j f'(net_j^*)\, w_{ji} z_j + E_i^*, \qquad i = 1, 2, \cdots, N \tag{5.4.18}$$

Note that Equation (5.4.17) is satisfied by the equilibria of Equation (5.4.18). Thus, a solution for $\mathbf{z}^*$, $\mathbf{z}^* = \left[z_1^* \; z_2^* \cdots z_N^*\right]^T$, is possible if it is an attractor of the dynamics in Equation (5.4.18). It can be shown (see Problem 5.4.5) that $\mathbf{z}^*$ is an attractor of Equation (5.4.18), if $\mathbf{y}^*$ is an attractor of Equation (5.4.6).

The similarity between Equations (5.4.18) and (5.4.6) suggests that a recurrent network realization for computing $\mathbf{z}^*$ should be possible. In fact, such a network may be arrived at by starting with the original network and replacing the coupling weight w_{ij} from unit *j* to unit *i* by $f'(net_i^*)\, w_{ij}$ from unit *i* to unit *j*, assuming linear activations for all units, setting all inputs to zero, and feeding the error E_i^* as input to the *i*th output unit (of the original network). The resulting network is called the error-propagation network or the adjoint of the original net. Figure 5.4.10 shows the error-propagation network for the simple recurrent net given in Figure 5.4.5(a).

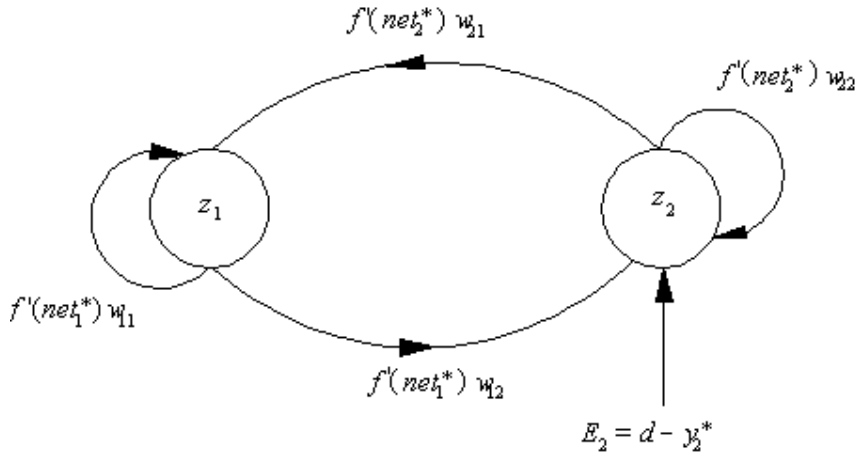


Figure 5.4.10. Error-propagation (adjoint) network for the simple recurrent net in Figure 5.4.5(a).

We may now give a brief outline of the recurrent backpropagation learning procedure. An input pattern $\mathbf{x}_k$ is presented to the recurrent net and a steady-state solution $\mathbf{y}^{*k}$ is computed by iteratively solving Equation (5.4.6). The steady-state outputs of the net are compared with the target $\mathbf{d}_k$ to find the output errors E_i^*. Then, the z_i^{*k}'s are computed by iteratively solving Equation (5.4.18). The weights are finally adjusted using Equation (5.4.14) or its equivalent form

$$\Delta w_{ij} = \rho\, f'\left(net_i^{*k}\right) z_i^{*k}\, y_j^{*k} \tag{5.4.19}$$

where

$$net_i^{*k} = \sum_j w_{ij} y_j^{*k} + x_i^k$$

Next, a new input pattern is presented to the network and the above procedure is repeated, and so on. It should be noted that the recurrent backpropagation reduces to incremental backprop for the special case of a net with no feedback.

The above analysis assumed that finite equilibria $\mathbf{y}^*$ exist and are stable. However, it has been shown (Simard et al., 1988, 1989) that for any recurrent neural network architecture, there always exists divergent trajectories for Equation (5.4.6). In practice, though, if the initial weights are chosen to be small enough, the network almost always converges to a finite stable equilibrium $\mathbf{y}^*$.

One potential application of recurrent backpropagation networks is as associative memories (for definition and details of associative memories, refer to Chapter 7). This is because these networks build attractors $\mathbf{d}_k$ which correspond to input/output association patterns $\{\mathbf{x}^k, \mathbf{d}^k\}$. That is, if a noisy and/or incomplete version of a trained pattern $\mathbf{x}_k$ is presented as input, it potentially causes the network to eventually converge to $\mathbf{d}_k$. These pattern completion/error correction features are superior to those of feedforward networks (Almeida, 1987). Other applications of recurrent backpropagation nets can be found in Qian and Sejnowski (1989) and Barhen et al. (1989).

5.4.4 Time-Dependent Recurrent Backpropagation

The recurrent backpropagation method just discussed can be extended to recurrent networks that produce time-dependent trajectories. One such extension is the time-dependent recurrent backpropagation method of Pearlmutter (1989a, b)[See also Werbos (1988), and Sato (1990)]. In Pearlmutter's method, learning is performed as a gradient descent in the weights of a continuous recurrent network to minimize an error function E of the temporal trajectory of the states. It can be thought of as an extension of recurrent backpropagation to dynamic sequences. The following is a brief outline of this algorithm.

Here, we start with a recurrent net with units y_i having the dynamics

$$\tau_i \frac{dy_i}{dt} = -y_i + f(net_i) + x_i(t), \qquad i = 1, 2, \cdots, N \tag{5.4.20}$$

Note that the inputs $x_i(t)$ are continuous functions of time. Similarly, each output unit y_l has a desired target signal $d_l(t)$ that is also a continuous function of time.

Consider minimizing a criterion $E(\mathbf{y})$, which is some function of the trajectory $\mathbf{y}(t)$ for t between 0 and t_1. Since the objective here is to teach the lth output unit to produce the trajectory $d_l(t)$ upon the presentation of $\mathbf{x}(t)$, an appropriate criterion (error) functional is

$$E = \frac{1}{2} \int_0^{t_1} \sum_l \left[d_l(t) - y_l(t) \right]^2 dt \tag{5.4.21}$$

which measures the deviation of y_l from the function d_l. Now, the partial derivatives of E with respect to the weights may be computed as:

$$\frac{\partial E}{\partial w_{ij}} = \frac{1}{\tau_i} \int_0^{t_1} f'\left[net_i(t) \right] z_i(t)\, y_j(t)\, dt \tag{5.4.22}$$

where $net_i(t) = \sum_j w_{ij}\, y_j(t)$, $y_j(t)$ is the solution to Equation (5.4.20), and $z_i(t)$ is the solution of the dynamical system given by

$$\frac{dz_i}{dt} = +\frac{1}{\tau_i} z_i - \sum_j \frac{1}{\tau_j} w_{ji}\, f'(net_j)\, z_j - E_i(t), \qquad i = 1, 2, \cdots, N \tag{5.4.23}$$

with the boundary condition $z_i(t_1) = 0$. Here, $E_i(t)$ is given by $d_i(t) - y_i(t)$ if unit i is an output unit, and zero otherwise. One may also simultaneously minimize E in the time-constant space by gradient descent, utilizing

$$\frac{\partial E}{\partial \tau_i} = -\frac{1}{\tau_i}\int_0^{t_1} z_i(t)\frac{dy_i(t)}{dt}dt$$

$$= -\frac{1}{\tau_i^2}\int_0^{t_1} z_i(t)\left[-y_i(t) + f\left(net_i(t)\right) + x_i(t)\right]dt \qquad (5.4.24)$$

Equations (5.4.22) and (5.4.24) may be derived by using a finite difference approximation as in Pearlmutter (1988). They may also be obtained using the calculus of variations and Lagrange multipliers as in optimal control theory (Bryson and Denham, 1962).

Using numerical integration (e.g., first order finite difference approximations) one first solves Equation (5.4.20) for t [0, t_1], then set the boundary condition $z_i(t_1) = 0$, and integrate the system in Equation (5.4.23) backward from t_1 to 0. Having determined $y_i(t)$ and $z_i(t)$, we may proceed with computing $\frac{\partial E}{\partial w_{ij}}$ and $\frac{\partial E}{\partial \tau_i}$ from Equations (5.4.22) and (5.4.24), respectively. Next, the w_{ij}'s and τ_i are computed from $\Delta w_{ij} = -\rho\frac{\partial E}{\partial w_{ij}}$ and $\Delta\tau_i = -\eta\frac{\partial E}{\partial \tau_i}$, respectively.

Due to its memory requirements and continuous-time nature, time-dependent recurrent backpropagation is more appropriate as an off-line training method. Some applications of this technique include learning limit cycles in two-dimensional space (Pearlmutter, 1989a) like the one shown in Figure 5.4.11 and 5.4.12. The trajectory in Figure 5.4.11(b) and (c) are produced by a network of four hidden units, two output units, and no input units, after 1,500 and 12,000 learning cycles, respectively. The desired trajectory [$d_1(t)$ versus $d_2(t)$] is the circle in Figure 5.4.11(a). The state space trajectories in Figure 5.4.12 (b) and (c) are generated by a network with 10 hidden units and two output units, after 3,182 and 20,000 cycles, respectively. The desired trajectory is shown in Figure 5.4.12(a). This method has also been shown to work well in time series prediction (Logar et al., 1993). Fang and Sejnowski (1990) reported improved learning speed and convergence of the above algorithm as the result of allowing independent learning rates for individual weights in the network (for example, they report a better formed figure "eight" compared to the one in Figure 5.4.12(c), after only 2000 cycles.)

(a) (b) (c)

Figure 5.4.11. Learning performance of time-dependent recurrent backpropagation: (a) desired trajectory $d_1(t)$ versus $d_2(t)$, (b) generated state space trajectory, $y_1(t)$ versus $y_2(t)$ after 1,500 cycles, and (c) $y_1(t)$ versus $y_2(t)$ after 12,000 cycles. (From B. A. Pearlmutter, 1989a, with permission of the MIT Press.)

Finally, an important property of the continuous-time recurrent net described by Equation (5.4.20) should be noted. It has been shown (Funahashi and Nakamura, 1993) that the output of a sufficiently large continuous-time recurrent net with hidden units can approximate any continuous state space trajectory to any desired degree of accuracy. This means that recurrent neural nets are universal approximators of dynamical systems. Note, however, that this says nothing about the existence of a learning procedure which will guarantee that any continuous trajectory is learned successfully. What it implies, though, is that the failure of learning a given continuous trajectory by a sufficiently large recurrent net would be attributed to the learning algorithm used.

(a) (b) (c)

Figure 5.4.12. Learning the figure "eight" by a time-dependent recurrent backpropagation net. (a) Desired state space trajectory, (b) generated trajectory after 3,182 cycles, (c) generated trajectory after 20,000 cycles. (From B. A. Pearlmutter, 1989a, with permission of the MIT Press.)

5.4.5 Real-Time Recurrent Learning

Another method that allows sequences to be associated is the real-time recurrent learning (RTRL) method proposed by Williams and Zipser

(1989a, b). This method allows recurrent networks to learn tasks that require retention of information over time periods having either fixed or indefinite length. RTRL assumes recurrent nets with discrete-time states that evolve according to

$$y_i(t) = f\left[net_i(t-1)\right] = f\left[\sum_j w_{ij}\ y_j(t-1) + x_i(t-1)\right] \quad (5.4.25)$$

A desired target trajectory **d**(*t*) is associated with each input trajectory **x**(*t*). As before, the quadratic error measure is used

$$E_{total} = \sum_{t=0}^{T} E(t) \quad (5.4.26)$$

where

$$E(t) = \frac{1}{2}\sum_l \left[d_l(t) - y_l(t)\right]^2$$

Thus, gradient descent on E_{total} gives

$$\Delta w_{pq} = \sum_{t=0}^{T} \Delta w_{pq}(t) \quad (5.4.27)$$

with

$$\Delta w_{pq}(t) = -\rho\frac{\partial E(t)}{\partial w_{pq}} = \rho\sum_l \left[d_l(t) - y_l(t)\right]\frac{\partial y_l(t)}{\partial w_{pq}} \quad (5.4.28)$$

The partial derivative $\frac{\partial y_l}{\partial w_{pq}}$ in Equation (5.4.28) can now be computed from Equation (5.4.25) as

$$\frac{\partial y_l(t)}{\partial w_{pq}} = f'\left[net_l(t-1)\right]\left[\delta_{lp}\ y_q(t-1) + \sum_j w_{lj}\frac{\partial y_j(t-1)}{\partial w_{pq}}\right] \quad (5.4.29)$$

Since Equation (5.4.29) relates the derivatives at time *t* to those at time *t*-1, we can iterate it forward (starting from some initial value for $\frac{\partial y_l(0)}{\partial w_{pq}}$; e.g., zero) and compute $\frac{\partial y_l(t)}{\partial w_{pq}}$ at any desired time, while using Equation (5.4.25) to iteratively update states at each iteration. Each cycle of this algorithm requires time proportional to N^4, where *N* is the number of units in a fully interconnected net. Instead of using Equation (5.4.27) to update the weights, it was found (Williams and Zipser, 1989a) that updating the weights after each time step according to Equation (5.4.28) works well as long as the learning rate r is kept sufficiently small; thus the name real-time recurrent learning. This avoids the need for allocating memory proportional to the maximum sequence length and leads to simple on-line implementations. The power of this method was demonstrated through a series of simulations (Williams and Zipser, 1989b). In one particular simulation, a 12 unit recurrent net learned to detect whether a string of arbitrary length comprised of left and right parentheses consists entirely of sets of balanced parentheses, by observing only the action of a Turing machine performing the same task. In some of the simulations, it was found that learning speed (and sometimes convergence) improved by setting the states of units $y_l(t)$ with known targets to their target values, but only after computing $E_l(t)$ and the derivatives in Equation (5.4.29). This heuristic is known as teacher forcing; it helps keep the network closer to the desired trajectory. The reader may refer to Robinson and Fallside (1988), Rohwer (1990), and Sun et al. (1992) for other methods for learning sequences in recurrent networks. (The reader is also referred to the Special Issue of the IEEE Transactions on Neural Networks, volume **5**(2), 1994, for further exploration into recurrent neural networks and their applications.)

Goto [5.0] [5.1] [5.2] [5.3] [5.5]


Back to the Table of Contents


Back to Main Menu