14 Dynamic Networks

Objectives

Neural networks can be classified into static and dynamic categories. The multilayer network that we have discussed in the last three chapters is a static network. This means that the output can be calculated directly from the input through feedforward connections. In dynamic networks, the output depends not only on the current input to the network, but also on the current or previous inputs, outputs or states of the network. For example, the adaptive filter networks we discussed in Chapter 10 are dynamic networks, since the output is computed from a tapped delay line of previous inputs. The Hopfield network we discussed in Chapter 3 is also a dynamic network. It has recurrent (feedback) connections, which means that the current output is a function of outputs at previous times.

We will begin this chapter with a brief introduction to the operation of dynamic networks, and then we will describe how these types of networks can be trained. The training will be based on optimization algorithms that use gradients (as in steepest descent and conjugate gradient algorithms) or Jacobians (as in Gauss-Newton and Levenberg-Marquardt algorithms) These algorithms were described in Chapters 10, 11 and 12 for static networks. The difference between the training of static and dynamic networks is in the manner in which the gradient or Jacobian is computed. In this chapter, we will present methods for computing gradients for dynamic networks.

Theory and Examples

Dynamic Networks

Dynamic networks are networks that contain delays (or integrators, for continuous-time networks) and that operate on a sequence of inputs. (In other words, the ordering of the inputs is important to the operation of the network.) These dynamic networks can have purely feedforward connections, like the adaptive filters of Chapter 10, or they can also have some feedback (*recurrent*) connections, like the Hopfield network of Chapter 3. Dynamic networks have memory. Their response at any given time will depend not only on the current input, but on the history of the input sequence.

Recurrent

Because dynamic networks have memory, they can be trained to learn sequential or time-varying patterns. Instead of approximating functions, like the static multilayer perceptron network of Chapter 11, a dynmic network can approximate a dynamic system. This has applications in such diverse areas as control of dynamic systems, prediction in financial markets, channel equalization in communication systems, phase detection in power systems, sorting, fault detection, speech recognition, learning of grammars in natural languages, and even the prediction of protein structure in genetics.

Dynamic networks can be trained using the standard optimization methods that we have discussed in Chapters 9 through 12. However, the gradients and Jacobians that are required for these methods cannot be computed using the standard backpropagation algorithm. In this chapter we will present the dynamic backpropagation algorithms that are required for computing the gradients for dynamic networks.

There are two general approaches (with many variations) to gradient and Jacobian calculations in dynamic networks: backpropagation-through-time (BPTT) [Werb90] and real-time recurrent learning (RTRL) [WiZi89]. In the BPTT algorithm, the network response is computed for all time points, and then the gradient is computed by starting at the last time point and working backward in time. This algorithm is efficient for the gradient calculation, but it is difficult to implement on-line, because the algorithm works backward in time from the last time step.

In the RTRL algorithm, the gradient can be computed at the same time as the network response, since it is computed by starting at the first time point, and then working forward through time. RTRL requires more calculations than BPTT for calculating the gradient, but RTRL allows a convenient framework for on-line implementation. For Jacobian calculations, the RTRL algorithm is generally more efficient than the BPTT algorithm.

In order to more easily present general BPTT and RTRL algorithms, it will be helpful to introduce modified notation for networks that can have recurrent connections. In the next section we will introduce this notation, and then the remainder of the chapter will present general BPTT and RTRL algorithms for dynamic networks.

Layered Digital Dynamic Networks

LDDN

In this section we want to introduce the neural network framework that we will use to represent general dynamic networks. We call this framework Layered Digital Dynamic Networks (LDDN). It is an extension of the notation that we have used to represent static multilayer networks. With this new notation, we can conveniently represent networks with multiple recurrent (feedback) connections and tapped delay lines.

To help us introduce the LDDN notation, consider the example dynamic network given in Figure 14.1.

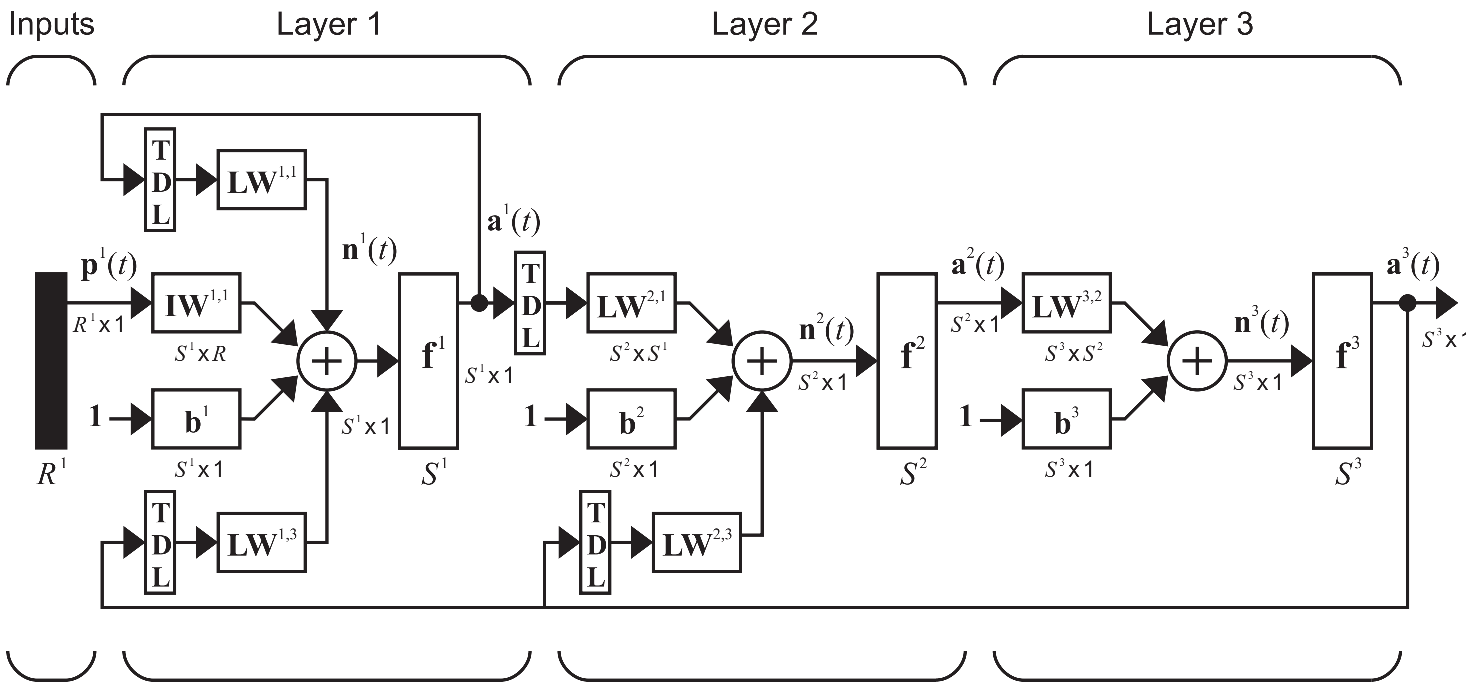


Figure 14.1 Example Dynamic Network

The general equations for the computation of the net input $\mathbf{n}^m(t)$ for layer m of an LDDN are

$$\mathbf{n}^m(t) = \sum_{l \in L_m^f} \sum_{d \in DL_{m,l}} \mathbf{LW}^{m,l}(d)\mathbf{a}^l(t-d) + \sum_{l \in I_m} \sum_{d \in DI_{m,l}} \mathbf{IW}^{m,l}(d)\mathbf{p}^l(t-d) + \mathbf{b}^m \tag{14.1}$$

Input Weight

Layer Weight

where $\mathbf{p}^l(t)$ is the l th input vector to the network at time t, $\mathbf{IW}^{m,l}$ is the *input weight* between input l and layer m, $\mathbf{LW}^{m,l}$ is the *layer weight* between layer l and layer m, $\mathbf{b}^m$ is the bias vector for layer m, $DL_{m,l}$ is the set of all delays in the tapped delay line between Layer l and Layer m, $DI_{m,l}$ is the set of all delays in the tapped delay line between Input l and Layer m,

I_m is the set of indices of input vectors that connect to layer m, and L_m^f is the set of indices of layers that directly connect *forward* to layer m. The output of layer m is then computed as

$$\mathbf{a}^m(t) = \mathbf{f}^m(\mathbf{n}^m(t)) . \tag{14.2}$$

Compare this with the static multilayer network of Eq. (11.6). LDDN networks can have several layers connecting to layer m. Some of the connections can be recurrent through tapped delay lines. An LDDN can also have multiple input vectors, and the input vectors can be connected to any layer in the network; for static multilayer networks, we assumed that the single input vector connected only to Layer 1.

With static multilayer networks, the layers were connected to each other in numerical order. In other words, Layer 1 was connected to Layer 2, which was connected to Layer 3, etc. Within the LDDN framework, any layer can connect to any other layer, even to itself. However, in order to use Eq. (14.1), we need to compute the layer outputs in a specific order. The order in which the layer outputs must be computed to obtain the correct network output is called the *simulation order*. (This order need not be unique; there may be several valid simulation orders.) In order to backpropagate the derivatives for the gradient calculations, we must proceed in the opposite order, which is called the *backpropagation order*. In Figure 14.1, the standard numerical order, 1-2-3, is the simulation order, and the backpropagation order is 3-2-1.

Simulation Order

Backpropagation Order

As with the multilayer network, the fundamental unit of the LDDN is the layer. Each layer in the LDDN is made up of five components:

1. a set of weight matrices that come into that layer (which may connect from other layers or from external inputs),
2. any tapped delay lines (represented by $DL_{m,l}$ or $DI_{m,l}$) that appear at the input of a set of weight matrices (Any set of weight matrices can be preceded by a TDL. For example, Layer 1 of Figure 14.1 contains the weights $\mathbf{LW}^{1,3}(d)$ and the corresponding TDL.),
3. a bias vector,
4. a summing junction, and
5. a transfer function.

The output of the LDDN is a function not only of the weights, biases, and current network inputs, but also of some layer outputs at previous points in time. For this reason, it is not a simple matter to calculate the gradient of the network output with respect to the weights and biases. The weights and biases have two different effects on the network output. The first is the direct effect, which can be calculated using the standard backpropagation algorithm from Chapter 11. The second is an indirect effect, since some of the inputs to the network are previous outputs, which are also functions of the weights and biases. The main development of the next two sections is a general gradient calculation for arbitrary LDDNs.

Example Dynamic Networks

Before we introduce dynamic training, let's get a feeling for the types of responses we can expect to see from dynamic networks. Consider first the feedforward dynamic network shown in Figure 14.2.

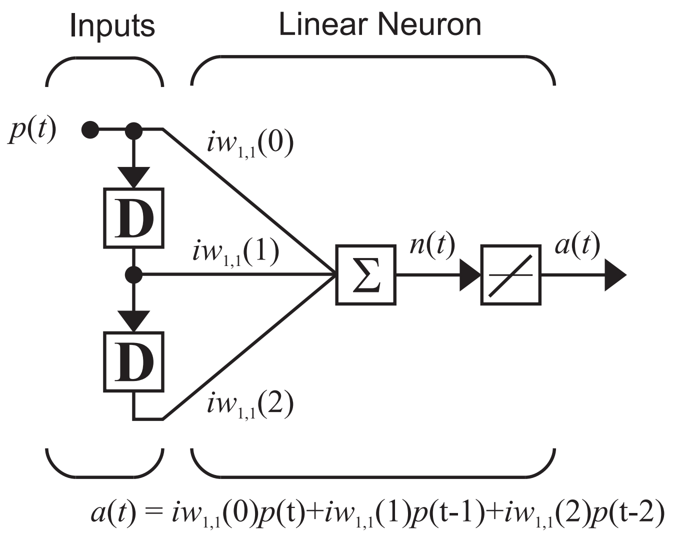


Figure 14.2 Example Feedforward Dynamic Network

This is an ADALINE filter, which we discussed in Chapter 10 (see Figure 10.5). Here we are representing it in the LDDN framework. The network has a TDL on the input, with $DI_{1,1} = \{0, 1, 2\}$. To demonstrate the operation of this network, we will apply a square wave as input, and we will set all of the weight values equal to 1/3:

$$iw_{1,1}(0) = \frac{1}{3},\ iw_{1,1}(1) = \frac{1}{3},\ iw_{1,1}(2) = \frac{1}{3}. \tag{14.3}$$

The network response is calculated from:

$$\begin{aligned}\mathbf{a}(t) = \mathbf{n}(t) &= \sum_{d=0}^{2} \mathbf{IW}(d)\mathbf{p}(t-d)\\ &= n_1(t) = iw_{1,1}(0)p(t) + iw_{1,1}(1)p(t-1) + iw_{1,1}(2)p(t-2)\end{aligned} \tag{14.4}$$

where we have left off the superscripts on the weight and the input, since there is only one input and only one layer.

The response of the network is shown in Figure 14.3. The open circles represent the square-wave input signal $p(t)$. The dots represent the network response $a(t)$. For this dynamic network, the response at any time point depends on the previous three input values. If the input is constant, the output will become constant after three time steps. This type of linear network is called a Finite Impulse Response (FIR) filter.

FIR

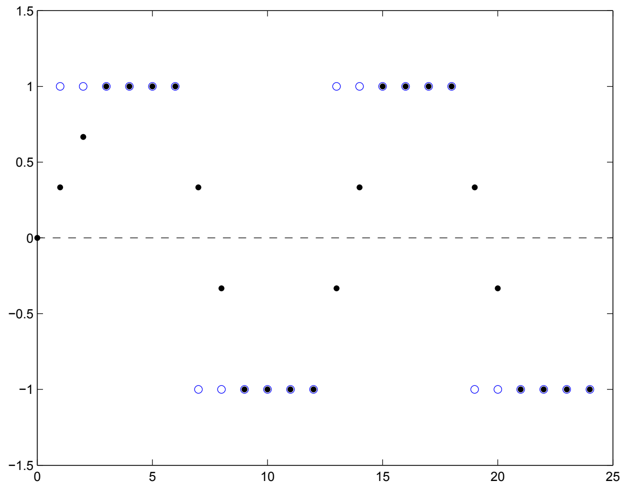


Figure 14.3 Response of ADALINE Filter Network

This dynamic network has memory. Its response at any given time will depend not only on the current input, but on the history of the input sequence. If the network does not have any feedback connections, then only a finite amount of history will affect the response. In the next example, we will consider a network that has an infinite memory.

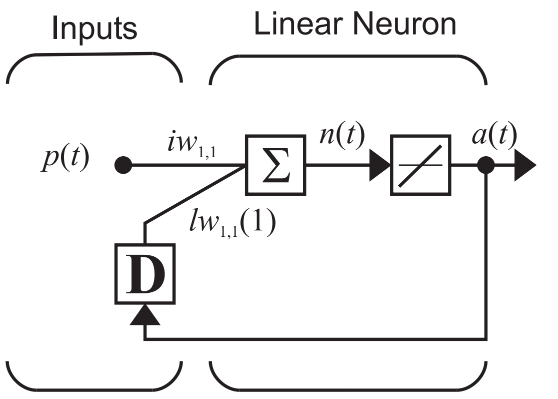
To experiment with the finite impulse response example, use the Neural Network Design Demonstration Finite Impulse Response Network (`nnd14fir`).

Now consider another simple linear dynamic network, but one that has a recurrent connection. The network in Figure 14.4 is a recurrent dynamic network. The equation of operation of the network is

$$\begin{aligned}\mathbf{a}^1(t) &= \mathbf{n}^1(t) = \mathbf{LW}^{1,1}(1)\mathbf{a}^1(t-1) + \mathbf{IW}^{1,1}(0)\mathbf{p}^1(t) \\ &= lw_{1,1}(1)a(t-1) + iw_{1,1}p(t)\end{aligned} \quad (14.5)$$

where, in the last line, we have left off the superscripts, since there is only one neuron and one layer in the network. To demonstrate the operation of this network, we will set the weight values to

$$lw_{1,1}(1) = \frac{1}{2} \text{ and } iw_{1,1} = \frac{1}{2}. \quad (14.6)$$



$$a(t) = iw_{1,1}p(t) + lw_{1,1}(1)a(t\text{-}1)$$

Figure 14.4 Recurrent Linear Neuron

The response of this network to the square wave input is shown in Figure 14.5. The network responds exponentially to a change in the input sequence. Unlike the FIR filter network of Figure 14.2, the exact response of the network at any given time is a function of the infinite history of inputs to the network.

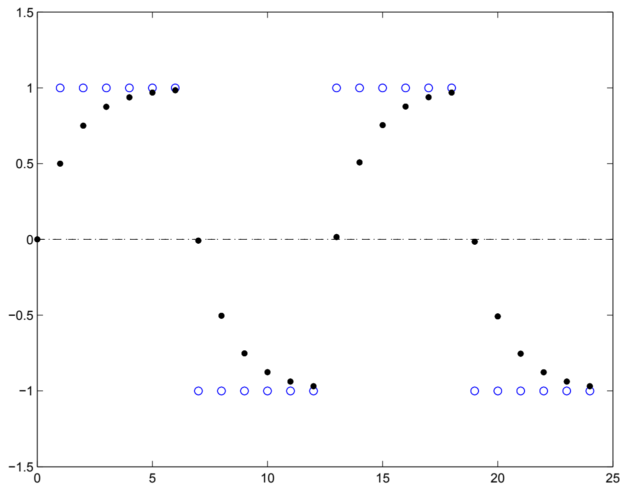


Figure 14.5 Recurrent Neuron Response

To experiment with this infinite impulse response example, use the Neural Network Design Demonstration Infinite Impulse Response Network (`nnd14iir`).

Compare the dynamic networks of the previous two examples with the static, two-layer perceptron of Figure 11.4. Static networks can be trained to approximate static functions, like $\sin(p)$, where the output can be computed directly from the current input. Dynamic networks, on the other hand, can be trained to approximate dynamic systems, such as robot arms, aircraft, biological processes and economic systems, where the current system output depends on a history of previous inputs and outputs. Because dynamic systems are more complex than static functions, we expect that the training process for dynamic networks will be more challenging than static network training.

In the following section, we will discuss the computation of gradients for the training of dynamic networks. For static networks, these gradients were computed using the standard backpropagation algorithm. For dynamic networks, the backpropagation algorithm must be modified.

Principles of Dynamic Learning

Before we get into the details of training dynamic networks, let's first investigate a simple example. Consider again the recurrent network of Figure 14.4. Suppose that we want to train the network using steepest descent. The first step is to compute the gradient of the performance function. For this example we will use sum squared error:

$$F(\mathbf{x}) = \sum_{t=1}^{Q} e^2(t) = \sum_{t=1}^{Q} (t(t) - a(t))^2 . \tag{14.7}$$

The two elements of the gradient will be

$$\frac{\partial F(\mathbf{x})}{\partial lw_{1,1}(1)} = \sum_{t=1}^{Q} \frac{\partial e^2(t)}{\partial lw_{1,1}(1)} = -2\sum_{t=1}^{Q} e(t)\frac{\partial a(t)}{\partial lw_{1,1}(1)} , \tag{14.8}$$

$$\frac{\partial F(\mathbf{x})}{\partial iw_{1,1}} = \sum_{t=1}^{Q} \frac{\partial e^2(t)}{\partial iw_{1,1}} = -2\sum_{t=1}^{Q} e(t)\frac{\partial a(t)}{\partial iw_{1,1}} \tag{14.9}$$

The key terms in these equations are the derivatives of the network output with respect to the weights:

$$\frac{\partial a(t)}{\partial lw_{1,1}(1)} \text{ and } \frac{\partial a(t)}{\partial iw_{1,1}} . \tag{14.10}$$

If we had a static network, then these terms would be very easy to compute. They would correspond to $a(t-1)$ and $p(t)$, respectively. However, for recurrent networks, the weights have two effects on the network output. The first is the direct effect, which is also seen in the corresponding static network. The second is an indirect effect, caused by the fact that one of the network inputs is a previous network output. Let's compute the derivatives of the network output, in order to demonstrate these two effects.

The equation of operation of the network is

$$a(t) = lw_{1,1}(1)a(t-1) + iw_{1,1}p(t). \tag{14.11}$$

We can compute the terms in Eq. (14.10) by taking the derivatives of Eq. (14.11):

$$\frac{\partial a(t)}{\partial lw_{1,1}(1)} = a(t-1) + lw_{1,1}(1)\frac{\partial a(t-1)}{\partial lw_{1,1}(1)}, \tag{14.12}$$

$$\frac{\partial a(t)}{\partial iw_{1,1}} = p(t) + lw_{1,1}(1)\frac{\partial a(t-1)}{\partial iw_{1,1}}. \tag{14.13}$$

The first term in each of these equations represents the direct effect that each weight has on the network output. The second term represents the indirect effect. Note that unlike the gradient computation for static networks, the derivative at each time point depends on the derivative at previous time points (or at future time points, as we will see later).

The following figures illustrate the dynamic derivatives. In Figure 14.6 a) we see the total derivatives $\partial a(t)/\partial iw_{1,1}$ and also the static portions of the derivatives. Note that if we consider only the static portion, we will underestimate the effect of a change in the weight. In Figure 14.6 b) we see the original response of the network (which was also shown in Figure 14.5) and a new response, in which $iw_{1,1}$ is increased from 0.5 to 0.6. By comparing the two parts of Figure 14.6, we can see how the derivative indicates the effect on the network response of a change in the weight $iw_{1,1}$.

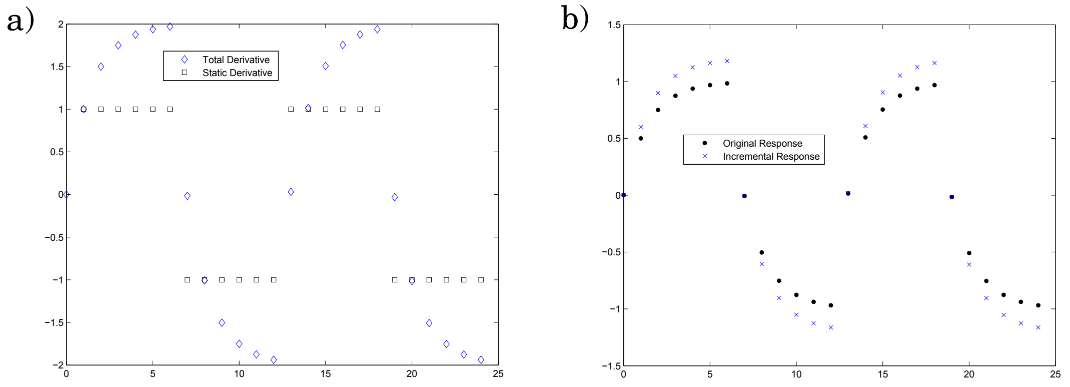


Figure 14.6 Derivatives for $iw_{1,1}$ and Response of Network in Figure 14.4

In Figure 14.7 we see similar results for the weight $lw_{1,1}(1)$. The key ideas to get from this example are: 1) the derivatives have static and dynamic components, and 2) the dynamic component depends on other time points.

To experiment with dynamic derivatives, use the Neural Network Design Demonstration Dynamic Derivatives (`nnd14dynd`).

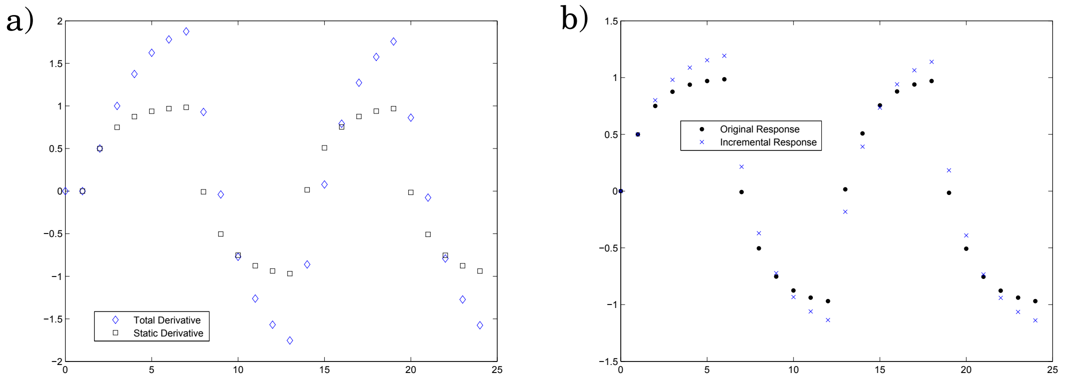


Figure 14.7 Derivatives for $lw_{1,1}(1)$ and Response of Network in Figure 14.4

Having made this initial investigation of a single-neuron network, let's consider the slightly more complex dynamic network that is shown in Figure 14.8. It consists of a static multilayer network with a single feedback loop added from the output of the network to the input of the network through a single delay. In this figure, the vector $\mathbf{x}$ represents all of the network parameters (weights and biases), and the vector $\mathbf{a}(t)$ represents the output of the multilayer network at time step t. This network will help us demonstrate the key steps of dynamic training.

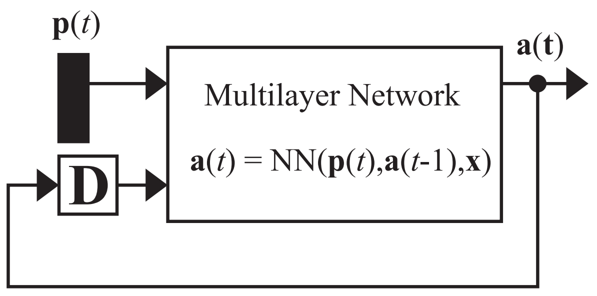


Figure 14.8 Simple Dynamic Network

As with a standard multilayer network, we want to adjust the weights and biases of the network to minimize the performance index, $F(\mathbf{x})$, which is normally chosen to be the mean squared error. In Chapter 11, we derived the backpropagation algorithm for computing the gradient of $F(\mathbf{x})$, which we could then use with any of the optimization methods from Chapter 12 to minimize $F(\mathbf{x})$. With dynamic networks, we need to modify the standard backpropagation algorithm. There are two different approaches to this problem. They both use the chain rule, but are implemented in different ways:

$$\frac{\partial F}{\partial \mathbf{x}} = \sum_{t=1}^{Q} \left[\frac{\partial \mathbf{a}(t)}{\partial \mathbf{x}^T}\right]^T \times \frac{\partial^e F}{\partial \mathbf{a}(t)}, \tag{14.14}$$

or

$$\frac{\partial F}{\partial \mathbf{x}} = \sum_{t=1}^{Q} \left[\frac{\partial^e \mathbf{a}(t)}{\partial \mathbf{x}^T}\right]^T \times \frac{\partial F}{\partial \mathbf{a}(t)}. \tag{14.15}$$

where the superscript e indicates an explicit derivative, not accounting for indirect effects through time. The explicit derivatives can be obtained with the standard backpropagation algorithm of Chapter 11. To find the complete derivatives that are required in Eq. (14.14) and Eq. (14.15), we need the additional equations:

$$\frac{\partial \mathbf{a}(t)}{\partial \mathbf{x}^T} = \frac{\partial^e \mathbf{a}(t)}{\partial \mathbf{x}^T} + \frac{\partial^e \mathbf{a}(t)}{\partial \mathbf{a}^T(t-1)} \times \frac{\partial \mathbf{a}(t-1)}{\partial \mathbf{x}^T} \tag{14.16}$$

and

$$\frac{\partial F}{\partial \mathbf{a}(t)} = \frac{\partial^e F}{\partial \mathbf{a}(t)} + \frac{\partial^e \mathbf{a}(t+1)}{\partial \mathbf{a}^T(t)} \times \frac{\partial F}{\partial \mathbf{a}(t+1)}. \tag{14.17}$$

RTRL

Eq. (14.14) and Eq. (14.16) make up the real-time recurrent learning (RTRL) algorithm. Note that the key term is

$$\frac{\partial \mathbf{a}(t)}{\partial \mathbf{x}^T}, \tag{14.18}$$

BPTT

which must be propagated forward through time. Eq. (14.15) and Eq. (14.17) make up the backpropagation-through-time (BPTT) algorithm. Here the key term is

$$\frac{\partial F}{\partial \mathbf{a}(t)} \tag{14.19}$$

which must be propagated backward through time.

In general, the RTRL algorithm requires somewhat more computation than the BPTT algorithm to compute the gradient. However, the BPTT algorithm cannot be conveniently implemented in real time, since the outputs must be computed for all time steps, and then the derivatives must be backpropagated back to the initial time point. The RTRL algorithm is well suited for real time implementation, since the derivatives can be calculated at each time step. (For Jacobian calculations, which are needed for Levenberg-Marquardt algorithms, the RTRL algorithm is often more efficient than the BPTT algorithm. See [DeHa07].)

Dynamic Backpropagation

In this section, we will develop general RTRL and BPTT algorithms for dynamic networks represented in the LDDN framework. This development will involve generalizing Eq. (14.14) through Eq. (14.17).

Preliminary Definitions

In order to simplify the description of the training algorithm, some layers of the LDDN will be assigned as network outputs, and some will be assigned as network inputs. A layer is an *input layer* if it has an input weight, or if it contains any delays with any of its weight matrices. A layer is an *output layer* if its output will be compared to a target during training, or if it is connected to an input layer through a matrix that has any delays associated with it.

Input Layer

Output Layer

For example, the LDDN shown in Figure 14.1 has two output layers (1 and 3) and two input layers (1 and 2). For this network the simulation order is 1-2-3, and the backpropagation order is 3-2-1. As an aid in later derivations, we will define U as the set of all output layer numbers and X as the set of all input layer numbers. For the LDDN in Figure 14.1, U={1,3} and X={1,2}.

The general equations for simulating an arbitrary LDDN network are given in Eq. (14.1) and Eq. (14.2). At each time point, these equations are iterated forward through the layers, as m is incremented through the simulation order. Time is then incremented from $t = 1$ to $t = Q$.

Real Time Recurrent Learning

In this subsection we will generalize the RTRL algorithm, given in Eq. (14.14) and Eq. (14.16), for LDDN networks. This development will follow in many respects the development of the backpropagation algorithm for static multilayer networks in Chapter 11. You may want to quickly review that material before proceeding.

Eq. (14.14)

The first step in developing the RTRL algorithm is to generalize Eq. (14.14). For the general LDDN network, we can calculate the terms of the gradient by using the chain rule, as in

$$\frac{\partial F}{\partial \mathbf{x}} = \sum_{t=1}^{Q} \sum_{u \in U} \left[\left[\frac{\partial \mathbf{a}^u(t)}{\partial \mathbf{x}^T} \right]^T \times \frac{\partial^e F}{\partial \mathbf{a}^u(t)} \right]. \tag{14.20}$$

If we compare this equation with Eq. (14.14), we notice that in addition to each time step, we also have a term in the sum for each output layer. However, if the performance index $F(\mathbf{x})$ is not explicitly a function of a specific output $\mathbf{a}^u(t)$, then that explicit derivative will be zero.

Eq. (14.16)

The next step of the development of the RTRL algorithm is the generalization of Eq. (14.16). Again, we use the chain rule:

$$\frac{\partial \mathbf{a}^u(t)}{\partial \mathbf{x}^T} = \frac{\partial^e \mathbf{a}^u(t)}{\partial \mathbf{x}^T} + \sum_{u' \in U} \sum_{x \in X} \sum_{d \in DL_{x,u'}} \frac{\partial^e \mathbf{a}^u(t)}{\partial \mathbf{n}^x(t)^T} \times \frac{\partial^e \mathbf{n}^x(t)}{\partial \mathbf{a}^{u'}(t-d)^T} \times \frac{\partial \mathbf{a}^{u'}(t-d)}{\partial \mathbf{x}^T} . \quad (14.21)$$

In Eq. (14.16) we only had one delay in the system. Now we need to account for each output and also for the number of times each output is delayed before it is input to another layer. That is the reason for the first two summations in Eq. (14.21). These equations must be updated forward in time, as t is varied from 1 to Q. The terms

$$\frac{\partial \mathbf{a}^u(t)}{\partial \mathbf{x}^T} \quad (14.22)$$

are generally set to zero for $t \le 0$.

To implement Eq. (14.21), we need to compute the terms

$$\frac{\partial^e \mathbf{a}^u(t)}{\partial \mathbf{n}^x(t)^T} \times \frac{\partial^e \mathbf{n}^x(t)}{\partial \mathbf{a}^{u'}(t-d)^T} . \quad (14.23)$$

To find the second term on the right, we can use

$$n_k^x(t) = \sum_{l \in L_x^f} \sum_{d' \in DL_{x,l}} \left[\sum_{i=1}^{S^l} lw_{k,i}^{x,l}(d') a_i^l(t-d') \right] + \sum_{l \in I_x} \sum_{d' \in DI_{x,l}} \left[\sum_{i=1}^{R^l} iw_{k,i}^{x,l}(d') p_i^l(t-d') \right] + b_k^x \quad (14.24)$$

(Compare with Eq. (11.20).) We can now write

$$\frac{\partial^e n_k^x(t)}{\partial a_j^{u'}(t-d)} = lw_{k,j}^{x,u'}(d) . \quad (14.25)$$

If we define the following sensitivity term

$$s_{k,i}^{u,m}(t) \equiv \frac{\partial^e a_k^u(t)}{\partial n_i^m(t)} , \quad (14.26)$$

which can be used to make up the following matrix

$$\mathbf{S}^{u,m}(t) = \frac{\partial^e \mathbf{a}^u(t)}{\partial \mathbf{n}^m(t)^T} = \begin{bmatrix} s^{u,m}_{1,1}(t) & s^{u,m}_{1,2}(t) & \dots & s^{u,m}_{1,S_m}(t) \\ s^{u,m}_{2,1}(t) & s^{u,m}_{2,2}(t) & \dots & s^{u,m}_{1,S_m}(t) \\ \vdots & \vdots & & \vdots \\ s^{u,m}_{S_u,1}(t) & s^{u,m}_{S_u,2}(t) & \dots & s^{u,m}_{S_u,S_m}(t) \end{bmatrix}, \tag{14.27}$$

then we can write Eq. (14.23) as

$$\left[\frac{\partial^e \mathbf{a}^u(t)}{\partial \mathbf{n}^x(t)^T} \times \frac{\partial^e \mathbf{n}^x(t)}{\partial \mathbf{a}^{u'}(t-d)^T}\right]_{i,j} = \sum_{k=1}^{S^x} s^{u,x}_{i,k}(t+d) \times lw^{x,u'}_{k,j}(d), \tag{14.28}$$

or in matrix form

$$\frac{\partial^e \mathbf{a}^u(t)}{\partial \mathbf{n}^x(t)^T} \times \frac{\partial^e \mathbf{n}^x(t)}{\partial \mathbf{a}^{u'}(t-d)^T} = \mathbf{S}^{u,x}(t) \times \mathbf{LW}^{x,u'}(d). \tag{14.29}$$

Therefore Eq. (14.21) can be written

$$\frac{\partial \mathbf{a}^u(t)}{\partial \mathbf{x}^T} = \frac{\partial^e \mathbf{a}^u(t)}{\partial \mathbf{x}^T} + \sum_{u' \in U} \sum_{x \in X} \sum_{d \in DL_{x,u'}} \mathbf{S}^{u,x}(t) \times \mathbf{LW}^{x,u'}(d) \times \frac{\partial \mathbf{a}^{u'}(t-d)}{\partial \mathbf{x}^T} \tag{14.30}$$

Many of the terms in the summation on the right hand side of Eq. (14.30) will be zero and will not have to be computed. To take advantage of these efficiencies, we introduce some indicator sets. They are sets that tell us for which layers the weights and the sensitivities are nonzero.

The first type of indicator set contains all of the output layers that connect to a specified layer x (which will always be an input layer) with at least some nonzero delay:

$$E^U_{LW}(x) = \{u \in U \ni \exists(\mathbf{LW}^{x,u}(d) \neq 0, d \neq 0)\}, \tag{14.31}$$

where $\ni$ means "such that," and $\exists$ means "there exists."

The second type of indicator set contains the input layers that have a nonzero sensitivity with a specified layer u:

$$E^X_S(u) = \{x \in X \ni \exists(\mathbf{S}^{u,x} \neq 0)\}. \tag{14.32}$$

(When $\mathbf{S}^{u,x}$ is nonzero, there is a static connection from layer x to ouput layer u.) The third type of indicator set contains the layers that have a nonzero sensitivity with a specified layer u:

$$E_S(u) = \{x \ni \exists(\mathbf{S}^{u,x} \neq 0)\}. \tag{14.33}$$

The difference between $E_S^X(u)$ and $E_S(u)$ is that $E_S^X(u)$ contains only input layers. $E_S(u)$ will not be needed in the simplification of Eq. (14.30), but it will be used for the calculation of sensitivities in Eq. (14.38).

Using Eq. (14.31) and Eq. (14.32), we can rearrange the order of the summations in Eq. (14.30) and sum only over nonzero terms:

$$\frac{\partial \mathbf{a}^u(t)}{\partial \mathbf{x}^T} = \frac{\partial^e \mathbf{a}^u(t)}{\partial \mathbf{x}^T} + \sum_{x \in E_S^X(u)} \mathbf{S}^{u,x}(t) \sum_{u' \in E_{LW}^U(x)} \sum_{d \in DL_{x,u'}} \mathbf{LW}^{x,u'}(d) \times \frac{\partial \mathbf{a}^{u'}(t-d)}{\partial \mathbf{x}^T} . \qquad (14.34)$$

Eq. (14.34) makes up the generalization of Eq. (14.16) for the LDDN network. It remains to compute the sensitivity matrices $\mathbf{S}^{u,m}(t)$ and the explicit derivatives $\partial^e \mathbf{a}^u(t)/\partial w$, which are described in the next two subsections.

Sensitivities

In order to compute the elements of the sensitivity matrix, we use a form of standard static backpropagation. The sensitivities at the outputs of the network can be computed as

$$s_{k,i}^{u,u}(t) = \frac{\partial^e a_k^u(t)}{\partial n_i^u(t)} = \begin{bmatrix} \dot{f}^u(n_i^u(t)) & \text{for } i = k \\ 0 & \text{for } i \neq k \end{bmatrix} , \; u \in U , \qquad (14.35)$$

or, in matrix form,

$$\mathbf{S}^{u,u}(t) = \dot{\mathbf{F}}^u(\mathbf{n}^u(t)) , \qquad (14.36)$$

where $\dot{\mathbf{F}}^u(\mathbf{n}^u(t))$ is defined as

$$\dot{\mathbf{F}}^u(\mathbf{n}^u(t)) = \begin{bmatrix} \dot{f}^u(n_1^u(t)) & 0 & \dots & 0 \\ 0 & \dot{f}^u(n_2^u(t)) & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & \dot{f}^u(n_{S^u}^u(t)) \end{bmatrix} \qquad (14.37)$$

(see also Eq. (11.34)). The matrices $\mathbf{S}^{u,m}(t)$ can be computed by backpropagating through the network, from each network output, using

$$\mathbf{S}^{u,m}(t) = \left[\sum_{l \in E_S(u) \cap L_m^b} \mathbf{S}^{u,l}(t) \mathbf{LW}^{l,m}(0) \right] \dot{\mathbf{F}}^m(\mathbf{n}^m(t)) , \; u \in U , \qquad (14.38)$$

where m is decremented from u through the backpropagation order, and L_m^b is the set of indices of layers that are directly connected backwards to layer m (or to which layer m connects forward) and that contain no delays in the connection. The backpropagation step given in Eq. (14.38) is essentially the same as that given in Eq. (11.45), but it is generalized to allow for arbitrary connections between layers.

Explicit Derivatives

We also need to compute the explicit derivatives

$$\frac{\partial^e \mathbf{a}^u(t)}{\partial \mathbf{x}^T} . \tag{14.39}$$

Using the chain rule of calculus, we can derive the following expansion of Eq. (14.39) for input weights:

$$\frac{\partial^e a_k^u(t)}{\partial iw_{i,j}^{m,l}(d)} = \frac{\partial^e a_k^u(t)}{\partial n_i^m(t)} \times \frac{\partial^e n_i^m(t)}{\partial iw_{i,j}^{m,l}(d)} = s_{k,i}^{u,m}(t) \times p_j^l(t-d) . \tag{14.40}$$

In vector form we can write

$$\frac{\partial^e \mathbf{a}^u(t)}{\partial iw_{i,j}^{m,l}(d)} = \mathbf{s}_i^{u,m}(t) \times p_j^l(t-d) . \tag{14.41}$$

In matrix form we have

$$\frac{\partial^e \mathbf{a}^u(t)}{\partial vec(\mathbf{IW}^{m,l}(d))^T} = [\mathbf{p}^l(t-d)]^T \otimes \mathbf{S}^{u,m}(t) , \tag{14.42}$$

and in a similar way we can derive the derivatives for layer weights and biases:

$$\frac{\partial^e \mathbf{a}^u(t)}{\partial vec(\mathbf{LW}^{m,l}(d))^T} = [\mathbf{a}^l(t-d)]^T \otimes \mathbf{S}^{u,m}(t) , \tag{14.43}$$

$$\frac{\partial^e \mathbf{a}^u(t)}{\partial (\mathbf{b}^m)^T} = \mathbf{S}^{u,m}(t) , \tag{14.44}$$

where the *vec* operator transforms a matrix into a vector by stacking the columns of the matrix one underneath the other, and $\mathbf{A} \otimes \mathbf{B}$ is the Kronecker product of $\mathbf{A}$ and $\mathbf{B}$ [MaNe99].

The total RTRL algorithm for the LDDN network is summarized in the following pseudo code.

Real-Time Recurrent Learning Gradient

```
Initialize:
    $\frac{\partial \mathbf{a}^u(t)}{\partial \mathbf{x}^T} = \mathbf{0},\ t \le 0$, for all $u \in U$,
For $t = 1$ to $Q$,
  $U' = \varnothing$, $E_S(u) = \varnothing$ and $E_S^X(u) = \varnothing$ for all $u \in U$.
  For $m$ decremented through the BP order
    For all $u \in U'$, if $E_S(u) \cap L_m^b \neq \varnothing$
      $\mathbf{S}^{u,m}(t) = \left[ \sum_{l \in E_S(u) \cap L_m^b} \mathbf{S}^{u,l}(t)\mathbf{LW}^{l,m}(0) \right] \dot{\mathbf{F}}^m(\mathbf{n}^m(t))$
      add $m$ to the set $E_S(u)$
      if $m \in X$, add $m$ to the set $E_S^X(u)$
    EndFor $u$
    If $m \in U$
      $\mathbf{S}^{m,m}(t) = \dot{\mathbf{F}}^m(\mathbf{n}^m(t))$
      add $m$ to the sets $U'$ and $E_S(m)$
      if $m \in X$, add $m$ to the set $E_S^X(m)$
    EndIf $m$
  EndFor $m$
  For $u \in U$ incremented through the simulation order
    For all weights and biases ($\mathbf{x}$ is a vector containing all weights and biases)
      $\frac{\partial^e \mathbf{a}^u(t)}{\partial vec(\mathbf{IW}^{m,l}(d))^T} = [\mathbf{p}^l(t-d)]^T \otimes \mathbf{S}^{u,m}(t)$
      $\frac{\partial^e \mathbf{a}^u(t)}{\partial vec(\mathbf{LW}^{m,l}(d))^T} = [\mathbf{a}^l(t-d)]^T \otimes \mathbf{S}^{u,m}(t)$
      $\frac{\partial^e \mathbf{a}^u(t)}{\partial (\mathbf{b}^m)^T} = \mathbf{S}^{u,m}(t)$
    EndFor weights and biases
    $\frac{\partial \mathbf{a}^u(t)}{\partial \mathbf{x}^T} = \frac{\partial^e \mathbf{a}^u(t)}{\partial \mathbf{x}^T} + \sum_{x \in E_S^X(u)} \mathbf{S}^{u,x}(t) \sum_{u' \in E_{LW}^U(x)} \sum_{d \in DL_{x,u'}} \mathbf{LW}^{x,u'}(d) \times \frac{\partial \mathbf{a}^{u'}(t-d)}{\partial \mathbf{x}^T}$
  EndFor $u$
EndFor $t$
Compute Gradients
  $\frac{\partial F}{\partial \mathbf{x}} = \sum_{t=1}^{Q} \sum_{u \in U} \left[ \left[ \frac{\partial \mathbf{a}^u(t)}{\partial \mathbf{x}^T} \right]^T \times \frac{\partial^e F}{\partial \mathbf{a}^u(t)} \right]$
```

Example RTRL Implementations (FIR and IIR)

To demonstrate the RTRL algorithm, consider again the feedforward dynamic network of Figure 14.2. The equation of operation of this network is

$$a(t) = n(t) = iw_{1,1}(0)p(t) + iw_{1,1}(1)p(t-1) + iw_{1,1}(2)p(t-2).$$

The architecture of the network is defined by

$$U = \{1\},\ X = \{1\},\ I_1 = \{1\},\ DI_{1,1} = \{0, 1, 2\},\ L_1^f = \varnothing,\ E_{LW}^U(1) = \varnothing.$$

We will choose the following standard performance function with three time points:

$$F = \sum_{t=1}^{Q} (t(t) - a(t))^2 = \sum_{t=1}^{3} e^2(t) = e^2(1) + e^2(2) + e^2(3),$$

with the following inputs and targets:

$$\{p(1), t(1)\}, \{p(2), t(2)\}, \{p(3), t(3)\}.$$

The RTRL algorithm begins with some initialization:

$$U' = \varnothing,\ E_S(1) = \varnothing,\ E_S^X(1) = \varnothing.$$

In addition, the initial conditions for the delays, $p(0), p(-1)$, must be provided.

The network response is then computed for the first time step:

$$a(1) = n(1) = iw_{1,1}(0)p(1) + iw_{1,1}(1)p(0) + iw_{1,1}(2)p(-1)$$

Because the RTRL algorithm proceeds forward through time, we can immediately compute the derivatives for the first time step. We will see in the next section that the BPTT algorithm, which proceeds backward through time, will require that we proceed through all of the time points before we can compute the derivatives.

From the preceding pseudo-code, the first step in the derivative calculation will be

$$\mathbf{S}^{1,1}(1) = \dot{\mathbf{F}}^1(\mathbf{n}^1(1)) = 1,$$

since the transfer function is linear. We also update the following sets:

$$E_S^X(1) = \{1\},\ E_S(1) = \{1\}.$$

The next step is the computation of the explicit derivatives from Eq. (14.42):

$$\frac{\partial^e \mathbf{a}^1(1)}{\partial vec(\mathbf{IW}^{1,1}(0))^T} = \frac{\partial^e a(1)}{\partial iw_{1,1}(0)} = [\mathbf{p}^1(1)]^T \otimes \mathbf{S}^{1,1}(t) = p(1),$$

$$\frac{\partial^e \mathbf{a}^1(1)}{\partial vec(\mathbf{IW}^{1,1}(1))^T} = \frac{\partial^e a(1)}{\partial iw_{1,1}(1)} = [\mathbf{p}^1(0)]^T \otimes \mathbf{S}^{1,1}(t) = p(0),$$

$$\frac{\partial^e \mathbf{a}^1(1)}{\partial vec(\mathbf{IW}^{1,1}(2))^T} = \frac{\partial^e a(1)}{\partial iw_{1,1}(2)} = [\mathbf{p}^1(-1)]^T \otimes \mathbf{S}^{1,1}(t) = p(-1).$$

The next step would be to compute the total derivative, using Eq. (14.34). However, since $E_{LW}^U(1) = \varnothing$, the total derivatives are equal to the explicit derivatives.

All of the above steps would be repeated for each time point, and then the final step is to compute the derivatives of the performance index with respect to the weights, using Eq. (14.20):

$$\frac{\partial F}{\partial \mathbf{x}} = \sum_{t=1}^{Q} \sum_{u \in U} \left[\left[\frac{\partial \mathbf{a}^u(t)}{\partial \mathbf{x}^T}\right]^T \times \frac{\partial^e F}{\partial \mathbf{a}^u(t)}\right] = \sum_{t=1}^{3} \left[\left[\frac{\partial \mathbf{a}^1(t)}{\partial \mathbf{x}^T}\right]^T \times \frac{\partial^e F}{\partial \mathbf{a}^1(t)}\right].$$

If we break this down for each weight, we have

$$\frac{\partial F}{\partial iw_{1,1}(0)} = p(1)(-2e(1)) + p(2)(-2e(2)) + p(3)(-2e(3)),$$

$$\frac{\partial F}{\partial iw_{1,1}(1)} = p(0)(-2e(1)) + p(1)(-2e(2)) + p(2)(-2e(3)),$$

$$\frac{\partial F}{\partial iw_{1,1}(2)} = p(-1)(-2e(1)) + p(0)(-2e(2)) + p(1)(-2e(3)).$$

We can then use this gradient in any of our standard optimization algorithms from Chapters 9 and 12. Note that if we use steepest descent, this result is a batch form of the LMS algorithm (see Eq. (10.33)).

Let's now do an example using a recurrent neural network. Consider again the simple recurrent network in Figure 14.4. From Eq. (14.5), the equation of operation of this network is

$$a(t) = lw_{1,1}(1)a(t-1) + iw_{1,1}p(t)$$

The architecture of the network is defined by

$$U = \{1\},\ X = \{1\},\ I_1 = \{1\},\ DI_{1,1} = \{0\},$$

$$DL_{1,1} = \{1\},\ L_1^f = \{1\},\ E_{LW}^U(1) = \{1\}.$$

We will choose the same performance function as the previous example:

$$F = \sum_{t=1}^{Q} (t(t) - a(t))^2 = \sum_{t=1}^{3} e^2(t) = e^2(1) + e^2(2) + e^2(3),$$

with the following inputs and targets:

$$\{p(1), t(1)\}, \{p(2), t(2)\}, \{p(3), t(3)\}.$$

We initialize with

$$U' = \varnothing,\ E_S(1) = \varnothing,\ E_S^X(1) = \varnothing.$$

In addition, the initial condition for the delay, $a(0)$, and the initial derivatives

$$\frac{\partial a(0)}{\partial iw_{1,1}} \text{ and } \frac{\partial a(0)}{\partial lw_{1,1}(1)}$$

must be provided. (The initial derivatives are usually set to zero.)

The network response is then computed for the first time step:

$$a(1) = lw_{1,1}(1)a(0) + iw_{1,1}p(1)$$

The derivative calculation begins with

$$\mathbf{S}^{1,1}(1) = \dot{\mathbf{F}}^1(\mathbf{n}^1(1)) = 1,$$

since the transfer function is linear. We also update the following sets:

$$E_S^X(1) = \{1\},\ E_S(1) = \{1\}.$$

The next step is the computation of the explicit derivatives:

$$\frac{\partial^e \mathbf{a}^1(1)}{\partial vec(\mathbf{IW}^{1,1}(0))^T} = \frac{\partial^e a(1)}{\partial iw_{1,1}} = [\mathbf{p}^1(1)]^T \otimes \mathbf{S}^{1,1}(1) = p(1),$$

$$\frac{\partial^e \mathbf{a}^1(1)}{\partial vec(\mathbf{LW}^{1,1}(1))^T} = \frac{\partial^e a(1)}{\partial lw_{1,1}(1)} = [\mathbf{a}^1(0)]^T \otimes \mathbf{S}^{1,1}(1) = a(0).$$

The next step is to compute the total derivative, using Eq. (14.34):

$$\frac{\partial \mathbf{a}^1(t)}{\partial \mathbf{x}^T} = \frac{\partial^e \mathbf{a}^1(t)}{\partial \mathbf{x}^T} + \mathbf{S}^{1,1}(t)\mathbf{LW}^{1,1}(1)\frac{\partial \mathbf{a}^1(t-1)}{\partial \mathbf{x}^T}. \qquad (14.45)$$

Replicating this formula for each of our weights for this network, for $t = 1$, we have

$$\frac{\partial a(1)}{\partial iw_{1,1}} = p(1) + lw_{1,1}(1)\frac{\partial a(0)}{\partial iw_{1,1}} = p(1),$$

$$\frac{\partial a(1)}{\partial lw_{1,1}(1)} = a(0) + lw_{1,1}(1)\frac{\partial a(0)}{\partial lw_{1,1}(1)} = a(0).$$

Note that unlike the corresponding equation in the previous example, these equations are recursive. The derivative at the current time depends on the derivative at the previous time. (Note that the two initial derivatives on the right side of this equation would normally be set to zero, but at the next time step they would be nonzero.) As we mentioned earlier, the weights in a recurrent network have two different effects on the network output. The first is the direct effect, which is represented by the explicit derivative in Eq. (14.45). The second is an indirect effect, since one of the inputs to the network is a previous output, which is also a function of the weights. This effect causes the second term in Eq. (14.45).

All of the above steps would be repeated for each time point:

$$\frac{\partial^e a(2)}{\partial iw_{1,1}} = p(2), \quad \frac{\partial^e a(2)}{\partial lw_{1,1}(1)} = a(1),$$

$$\frac{\partial a(2)}{\partial iw_{1,1}} = p(2) + lw_{1,1}(1)\frac{\partial a(1)}{\partial iw_{1,1}} = p(2) + lw_{1,1}(1)p(1),$$

$$\frac{\partial a(2)}{\partial lw_{1,1}(1)} = a(1) + lw_{1,1}(1)\frac{\partial a(1)}{\partial lw_{1,1}(1)} = a(1) + lw_{1,1}(1)a(0),$$

$$\frac{\partial^e a(3)}{\partial iw_{1,1}} = p(3), \quad \frac{\partial^e a(3)}{\partial lw_{1,1}(1)} = a(2),$$

$$\frac{\partial a(3)}{\partial iw_{1,1}} = p(3) + lw_{1,1}(1)\frac{\partial a(2)}{\partial iw_{1,1}} = p(3) + lw_{1,1}(1)p(2) + (lw_{1,1}(1))^2 p(1),$$

$$\frac{\partial a(3)}{\partial lw_{1,1}(1)} = a(2) + lw_{1,1}(1)\frac{\partial a(2)}{\partial lw_{1,1}(1)} = a(2) + lw_{1,1}(1)a(1) + (lw_{1,1}(1))^2 a(0).$$

The final step is to compute the derivatives of the performance index with respect to the weights, using Eq. (14.20):

$$\frac{\partial F}{\partial \mathbf{x}} = \sum_{t=1}^{Q} \sum_{u \in U} \left[\left[\frac{\partial \mathbf{a}^u(t)}{\partial \mathbf{x}^T} \right]^T \times \frac{\partial^e F}{\partial \mathbf{a}^u(t)} \right] = \sum_{t=1}^{3} \left[\left[\frac{\partial \mathbf{a}^1(t)}{\partial \mathbf{x}^T} \right]^T \times \frac{\partial^e F}{\partial \mathbf{a}^1(t)} \right].$$

If we break this down for each weight, we have

$$\begin{aligned}\frac{\partial F}{\partial iw_{1,1}} &= \frac{\partial a(1)}{\partial iw_{1,1}}(-2e(1)) + \frac{\partial a(2)}{\partial iw_{1,1}}(-2e(2)) + \frac{\partial a(3)}{\partial iw_{1,1}}(-2e(3)) \\ &= -2e(1)[p(1)] - 2e(2)[p(2) + lw_{1,1}(1)p(1)] \\ &\quad -2e(3)[p(3) + lw_{1,1}(1)p(2) + (lw_{1,1}(1))^2 p(1)]\end{aligned}$$

$$\begin{aligned}\frac{\partial F}{\partial lw_{1,1}(1)} &= \frac{\partial a(1)}{\partial lw_{1,1}(1)}(-2e(1)) + \frac{\partial a(2)}{\partial lw_{1,1}(1)}(-2e(2)) + \frac{\partial a(3)}{\partial lw_{1,1}(1)}(-2e(3)) \\ &= -2e(1)[a(0)] - 2e(2)[a(1) + lw_{1,1}(1)a(0)] \\ &\quad -2e(3)[a(2) + lw_{1,1}(1)a(1) + (lw_{1,1}(1))^2 a(0)]\end{aligned}$$

The expansions that we show in the final two lines of the above equations (and also in some of the previous equations) would not be necessary in practice, since the results would be numerical. We have included them here so that we can compare this result with the BPTT algorithm, which we present next.

Backpropagation-Through-Time

In this section we will generalize the Backpropagation-Through-Time (BPTT) algorithm, given in Eq. (14.15) and Eq. (14.17), for LDDN networks.

Eq. (14.15)

The first step is to generalize Eq. (14.15). For the general LDDN network, we can calculate the terms of the gradient by using the chain rule, as in

$$\frac{\partial F}{\partial lw_{i,j}^{m,l}(d)} = \sum_{t=1}^{Q} \left[\sum_{u \in U} \sum_{k=1}^{S^u} \frac{\partial F}{\partial a_k^u(t)} \times \frac{\partial^e a_k^u(t)}{\partial n_i^m(t)} \right] \frac{\partial^e n_i^m(t)}{\partial lw_{i,j}^{m,l}(d)} \qquad (14.46)$$

(for the layer weights), where u is an output layer, U is the set of all output layers, and S^u is the number of neurons in layer u.

From Eq. (14.24) we can write

$$\frac{\partial^e n_i^m(t)}{\partial lw_{i,j}^{m,l}(d)} = a_j^l(t-d) . \tag{14.47}$$

We will also define

$$d_i^m(t) = \sum_{u \in U} \sum_{k=1}^{S^u} \frac{\partial F}{\partial a_k^u(t)} \times \frac{\partial^e a_k^u(t)}{\partial n_i^m(t)} . \tag{14.48}$$

The terms of the gradient for the layer weights can then be written

$$\frac{\partial F}{\partial lw_{i,j}^{m,l}(d)} = \sum_{t=1}^{Q} d_i^m(t) a_j^l(t-d) , \tag{14.49}$$

If we use the sensitivity term defined in Eq. (14.26),

$$s_{k,i}^{u,m}(t) \equiv \frac{\partial^e a_k^u(t)}{\partial n_i^m(t)} , \tag{14.50}$$

then the elements $d_i^m(t)$ can be written

$$d_i^m(t) = \sum_{u \in U} \sum_{k=1}^{S^u} \frac{\partial F}{\partial a_k^u(t)} \times s_{k,i}^{u,m}(t) . \tag{14.51}$$

In matrix form this becomes

$$\mathbf{d}^m(t) = \sum_{u \in U} [\mathbf{S}^{u,m}(t)]^T \times \frac{\partial F}{\partial \mathbf{a}^u(t)} \tag{14.52}$$

where

$$\frac{\partial F}{\partial \mathbf{a}^u(t)} = \left[\frac{\partial F}{\partial a_1^u(t)} \; \frac{\partial F}{\partial a_2^u(t)} \; \cdots \; \frac{\partial F}{\partial a_{S_u}^u(t)} \right]^T \tag{14.53}$$

Now the gradient can be written in matrix form.

$$\frac{\partial F}{\partial \mathbf{LW}^{m,l}(d)} = \sum_{t=1}^{Q} \mathbf{d}^m(t) \times [\mathbf{a}^l(t-d)]^T , \tag{14.54}$$

and by similar steps we can find the derivatives for the biases and input weights:

$$\frac{\partial F}{\partial \mathbf{IW}^{m,l}(d)} = \sum_{t=1}^{Q} \mathbf{d}^m(t) \times [\mathbf{p}^l(t-d)]^T, \tag{14.55}$$

$$\frac{\partial F}{\partial \mathbf{b}^m} = \sum_{t=1}^{Q} \mathbf{d}^m(t). \tag{14.56}$$

Eq. (14.54) through Eq. (14.56) make up the generalization of Eq. (14.15) for the LDDN network.

Eq. (14.17)

The next step in the development of the BPTT algorithm is the generalization of Eq. (14.17). Again, we use the chain rule:

$$\frac{\partial F}{\partial \mathbf{a}^u(t)} = \frac{\partial^e F}{\partial \mathbf{a}^u(t)} + \sum_{u' \in U} \sum_{x \in X} \sum_{d \in DL_{x,u}} \left[\frac{\partial^e \mathbf{a}^{u'}(t+d)}{\partial \mathbf{n}^x(t+d)^T} \times \frac{\partial^e \mathbf{n}^x(t+d)}{\partial \mathbf{a}^u(t)^T} \right]^T \times \frac{\partial F}{\partial \mathbf{a}^{u'}(t+d)} \tag{14.57}$$

(Many of the terms in these summations will be zero. We will provide a more efficient representation later in this section.) In Eq. (14.17) we only had one delay in the system. Now we need to account for each network output, how that network output is connected back through a network input, and also for the number of times each network output is delayed before it is applied to a network input. That is the reason for the three summations in Eq. (14.57). This equation must be updated backward in time, as t is varied from Q to 1. The terms

$$\frac{\partial F}{\partial \mathbf{a}^{u'}(t)} \tag{14.58}$$

are generally set to zero for $t > Q$.

If we consider the matrix in the brackets on the right side of Eq. (14.57), from Eq. (14.29) we can write

$$\frac{\partial^e \mathbf{a}^{u'}(t+d)}{\partial \mathbf{n}^x(t+d)^T} \times \frac{\partial^e \mathbf{n}^x(t+d)}{\partial \mathbf{a}^u(t)^T} = \mathbf{S}^{u',x}(t+d) \times \mathbf{LW}^{x,u}(d). \tag{14.59}$$

This allows us to write Eq. (14.57) as

$$\frac{\partial F}{\partial \mathbf{a}^u(t)} = \frac{\partial^e F}{\partial \mathbf{a}^u(t)} + \sum_{u' \in U} \sum_{x \in X} \sum_{d \in DL_{x,u}} [\mathbf{S}^{u',x}(t+d) \times \mathbf{LW}^{x,u}(d)]^T \times \frac{\partial F}{\partial \mathbf{a}^{u'}(t+d)} \tag{14.60}$$

Many of the terms in the summation on the right hand side of Eq. (14.60) will be zero and will not have to be computed. In order to provide a more efficient implementation of Eq. (14.60), we define the following indicator sets:

$$E_{LW}^X(u) = \{x \in X \ni \exists(\mathbf{LW}^{x,u}(d) \neq 0, d \neq 0)\}, \tag{14.61}$$

$$E_S^U(x) = \{u \in U \ni \exists(\mathbf{S}^{u,x} \neq 0)\}. \tag{14.62}$$

The first set contains all of the input layers that have a connection from output layer u with at least some nonzero delay. The second set contains output layers that have a nonzero sensitivity with input layer x. When the sensitivity $S^{u,x}$ is nonzero, there is a static connection from input layer x to output layer u.

We can now rearrange the order of the summation in Eq. (14.60) and sum only over the existing terms:

$$\frac{\partial F}{\partial \mathbf{a}^u(t)} = \frac{\partial^e F}{\partial \mathbf{a}^u(t)} + \sum_{x \in E_{LW}^X(u)} \sum_{d \in DL_{x,u}} \mathbf{LW}^{x,u}(d)^T \sum_{u' \in E_S^U(x)} \mathbf{S}^{u',x}(t+d)^T \times \frac{\partial F}{\partial \mathbf{a}^{u'}(t+d)} \tag{14.63}$$

Summary

The total BPTT algorithm is summarized in the following pseudo code.

Backpropagation-Through-Time Gradient

```
Initialize:
    $\frac{\partial F}{\partial \mathbf{a}^u(t)} = \mathbf{0},\ t > Q$, for all $u \in U$,
For $t = Q$ to 1,
    $U' = \varnothing$, $E_S(u) = \varnothing$, and $E_S^U(u) = \varnothing$ for all $u \in U$.
    For $m$ decremented through the BP order
        For all $u \in U'$, if $E_S(u) \cap L_m^b \neq \varnothing$
            $\mathbf{S}^{u,m}(t) = \left[ \sum_{l \in E_S(u) \cap L_m^b} \mathbf{S}^{u,l}(t)\mathbf{LW}^{l,m}(0) \right] \dot{\mathbf{F}}^m(\mathbf{n}^m(t))$
            add $m$ to the set $E_S(u)$
            add $u$ to the set $E_S^U(m)$
        EndFor $u$
        If $m \in U$
            $\mathbf{S}^{m,m}(t) = \dot{\mathbf{F}}^m(\mathbf{n}^m(t))$
            add $m$ to the sets $U'$, $E_S(m)$ and $E_S^U(m)$
        EndIf $m$
    EndFor $m$
    For $u \in U$ decremented through the BP order
        $\frac{\partial F}{\partial \mathbf{a}^u(t)} = \frac{\partial^e F}{\partial \mathbf{a}^u(t)} + \sum_{x \in E_{LW}^X(u)} \sum_{d \in DL_{x,u}} \mathbf{LW}^{x,u}(d)^T \sum_{u' \in E_S^U(x)} \mathbf{S}^{u',x}(t+d)^T \times \frac{\partial F}{\partial \mathbf{a}^{u'}(t+d)}$
    EndFor $u$
    For all layers $m$
        $\mathbf{d}^m(t) = \sum_{u \in E_S^U(m)} [\mathbf{S}^{u,m}(t)]^T \times \frac{\partial F}{\partial \mathbf{a}^u(t)}$
    EndFor $m$
EndFor $t$
Compute Gradients
    $\frac{\partial F}{\partial \mathbf{LW}^{m,l}(d)} = \sum_{t=1}^{Q} \mathbf{d}^m(t) \times [\mathbf{a}^l(t-d)]^T$
    $\frac{\partial F}{\partial \mathbf{IW}^{m,l}(d)} = \sum_{t=1}^{Q} \mathbf{d}^m(t) \times [\mathbf{p}^l(t-d)]^T$
    $\frac{\partial F}{\partial \mathbf{b}^m} = \sum_{t=1}^{Q} \mathbf{d}^m(t)$
```

Example BPTT Implementations (FIR and IIR)

To demonstrate the BPTT algorithm, we will use the same example networks that we used for the RTRL algorithm. First, we use the feedforward dynamic network of Figure 14.2. We defined the network architecture on page 14-18.

Before the gradient can be computed using BPTT, the network response must be computed for all time steps:

$$a(1) = n(1) = iw_{1,1}(0)p(1) + iw_{1,1}(1)p(0) + iw_{1,1}(2)p(-1),$$

$$a(2) = n(2) = iw_{1,1}(0)p(2) + iw_{1,1}(1)p(1) + iw_{1,1}(2)p(0),$$

$$a(3) = n(3) = iw_{1,1}(0)p(3) + iw_{1,1}(2)p(0) + iw_{1,1}(2)p(1).$$

The BPTT algorithm begins with some initialization:

$$U' = \varnothing,\ E_S(1) = \varnothing,\ E_S^U(1) = \varnothing.$$

The first step in the derivative calculation will be the sensitivity calculation. For BPTT, we start at the last time point ($t = 3$):

$$\mathbf{S}^{1,1}(3) = \dot{\mathbf{F}}^1(\mathbf{n}^1(3)) = 1,$$

since the transfer function is linear. We also update the following sets:

$$E_S^U(1) = \{1\},\ E_S(1) = \{1\}.$$

The next step is the calculation of the following derivative using Eq. (14.63):

$$\frac{\partial F}{\partial \mathbf{a}^1(3)} = \frac{\partial^e F}{\partial \mathbf{a}^1(3)} = -2e(3).$$

The final step for $t = 3$ is Eq. (14.52):

$$\mathbf{d}^1(3) = [\mathbf{S}^{1,1}(3)]^T \times \frac{\partial F}{\partial \mathbf{a}^1(3)} = -2e(3).$$

We repeat the previous steps for $t = 2$ and $t = 1$, to obtain

$$\mathbf{d}^1(2) = [\mathbf{S}^{1,1}(2)]^T \times \frac{\partial F}{\partial \mathbf{a}^1(2)} = -2e(2),$$

$$\mathbf{d}^1(1) = [\mathbf{S}^{1,1}(1)]^T \times \frac{\partial F}{\partial \mathbf{a}^1(1)} = -2e(1).$$

Now, all time steps are combined in Eq. (14.55):

$$\frac{\partial F}{\partial \mathbf{IW}^{1,1}(0)} = \frac{\partial F}{\partial iw_{1,1}(0)} = \sum_{t=1}^{3} \mathbf{d}^1(t) \times [\mathbf{p}^1(t)]^T = \sum_{t=1}^{3} -2e(t) \times p(t),$$

$$\frac{\partial F}{\partial \mathbf{IW}^{1,1}(1)} = \frac{\partial F}{\partial iw_{1,1}(1)} = \sum_{t=1}^{3} \mathbf{d}^1(t) \times [\mathbf{p}^1(t-1)]^T = \sum_{t=1}^{3} -2e(t) \times p(t-1),$$

$$\frac{\partial F}{\partial \mathbf{IW}^{1,1}(2)} = \frac{\partial F}{\partial iw_{1,1}(2)} = \sum_{t=1}^{3} \mathbf{d}^1(t) \times [\mathbf{p}^1(t-2)]^T = \sum_{t=1}^{3} -2e(t) \times p(t-2).$$

Note that this is the same result we obtained for the RTRL algorithm example on page 14-19. RTRL and BPTT should always produce the same gradient. The only difference is in the implementation.

Let's now use our previous recurrent neural network example of Figure 14.4. We defined the architecture of this network on page 14-19.

Unlike the RTRL algorithm, where initial conditions for the derivatives must be provided, the BPTT algorithm requires final conditions for the derivatives:

$$\frac{\partial a(4)}{\partial iw_{1,1}} \text{ and } \frac{\partial a(4)}{\partial lw_{1,1}(1)},$$

which are normally set to zero.

The network response is then computed for all time steps:

$$a(1) = lw_{1,1}(1)a(0) + iw_{1,1}p(1)$$

$$a(2) = lw_{1,1}(1)a(1) + iw_{1,1}p(2)$$

$$a(3) = lw_{1,1}(1)a(2) + iw_{1,1}p(3)$$

The derivative calculation begins with

$$\mathbf{S}^{1,1}(3) = \dot{\mathbf{F}}^1(\mathbf{n}^1(3)) = 1,$$

since the transfer function is linear. We also update the following sets:

$$E_S^X(1) = \{1\},\ E_S(1) = \{1\}.$$

Next we compute the following derivative using Eq. (14.63):

$$\frac{\partial F}{\partial \mathbf{a}^1(t)} = \frac{\partial^e F}{\partial \mathbf{a}^1(t)} + \mathbf{LW}^{1,1}(1)^T \mathbf{S}^{1,1}(t+1)^T \times \frac{\partial F}{\partial \mathbf{a}^1(t+1)}$$

For $t = 3$, we find

$$\frac{\partial F}{\partial \mathbf{a}^1(3)} = \frac{\partial^e F}{\partial \mathbf{a}^1(3)} + lw_{1,1}(1)\mathbf{S}^{1,1}(4)^T \times \overset{0}{\cancel{\frac{\partial F}{\partial \mathbf{a}^1(4)}}} = \frac{\partial^e F}{\partial \mathbf{a}^1(3)} = -2e(3)$$

and

$$\mathbf{d}^1(3) = [\mathbf{S}^{1,1}(3)]^T \times \frac{\partial F}{\partial \mathbf{a}^1(3)} = -2e(3)$$

Continuing to $t = 2$,

$$\mathbf{S}^{1,1}(2) = \dot{\mathbf{F}}^1(\mathbf{n}^1(2)) = 1,$$

$$\frac{\partial F}{\partial \mathbf{a}^1(2)} = \frac{\partial^e F}{\partial \mathbf{a}^1(2)} + lw_{1,1}(1)\mathbf{S}^{1,1}(3)^T \times \frac{\partial F}{\partial \mathbf{a}^1(3)}$$
$$= -2e(2) + lw_{1,1}(1)(-2e(3))$$

and

$$\mathbf{d}^1(2) = [\mathbf{S}^{1,1}(2)]^T \times \frac{\partial F}{\partial \mathbf{a}^1(2)} = -2e(2) + lw_{1,1}(1)(-2e(3))$$

Finally, for $t = 1$,

$$\mathbf{S}^{1,1}(1) = \dot{\mathbf{F}}^1(\mathbf{n}^1(1)) = 1,$$

$$\frac{\partial F}{\partial \mathbf{a}^1(1)} = \frac{\partial^e F}{\partial \mathbf{a}^1(1)} + lw_{1,1}(1)\mathbf{S}^{1,1}(2)^T \times \frac{\partial F}{\partial \mathbf{a}^1(2)}$$
$$= -2e(1) + lw_{1,1}(1)(-2e(2)) + (lw_{1,1}(1))^2(-2e(3))$$

and

$$\mathbf{d}^1(1) = [\mathbf{S}^{1,1}(1)]^T \times \frac{\partial F}{\partial \mathbf{a}^1(1)} = -2e(1) + lw_{1,1}(1)(-2e(2)) + (lw_{1,1}(1))^2(-2e(3))$$

Now we can compute the total gradient, using Eq. (14.54) and Eq. (14.55):

$$\frac{\partial F}{\partial \mathbf{LW}^{1,1}(1)} = \frac{\partial F}{\partial lw_{1,1}(1)} = \sum_{t=1}^{3} \mathbf{d}^{1}(t) \times [\mathbf{a}^{1}(t-1)]^{T}$$

$$= a(0)[-2e(1) + lw_{1,1}(1)(-2e(2)) + (lw_{1,1}(1))^{2}(-2e(3))]$$
$$+ a(1)[-2e(2) + lw_{1,1}(1)(-2e(3))] + a(0)[-2e(3)]$$

$$\frac{\partial F}{\partial \mathbf{IW}^{1,1}(0)} = \frac{\partial F}{\partial iw_{1,1}} = \sum_{t=1}^{3} \mathbf{d}^{1}(t) \times [\mathbf{p}^{1}(t)]^{T}$$

$$= p(1)[-2e(1) + lw_{1,1}(1)(-2e(2)) + (lw_{1,1}(1))^{2}(-2e(3))]$$
$$+ p(2)[-2e(2) + lw_{1,1}(1)(-2e(3))] + p(3)[-2e(3)]$$

This is the same result that we obtained with the RTRL algorithm on page 14-22.

Summary and Comments on Dynamic Training

The RTRL and BPTT algorithms represent two methods for computing the gradients for dynamic networks. Both algorithms compute the exact gradient, and therefore they produce the same final results. The RTRL algorithm performs the calculations from the first time point forward, which is suitable for on-line (real-time) implementation. The BPTT algorithm starts from the last time point and works backward in time. The BPTT algorithm generally requires fewer computations for the gradient calculation than RTRL, but BPTT usually requires more memory storage.

In addition to the gradient, versions of BPTT and RTRL can be used to compute Jacobian matrices, as are needed in the Levenberg-Marquardt described in Chapter 12. For Jacobian calculations, the RTRL algorithm is generally more efficient that the BPTT algorithm. See [DeHa07] for details.

Once the gradients or Jacobians are computed, many standard optimization algorithms can be used to train the networks. However, training dynamic networks is generally more difficult than training feedforward networks—for a number of reasons. First, a recurrent net can be thought of as a feedforward network, in which the recurrent network is unfolded in time. For example, consider the simple single-layer recurrent network of Figure 14.4. If this network were to be trained over five time steps, we could unfold the network to create 5 layers - one for each time step. If a sigmoid transfer function is used, then if the output of the network is near the saturation point for any time point, the resulting gradient could be quite small.

Another problem in training dynamic networks is the shape of the error surface. It has been shown (see [PhHa13]) that the error surfaces of recurrent networks can have spurious valleys that are not related to the dynamic system that is being approximated. The underlying cause of these valleys is the fact that recurrent networks have the potential for instabilities. For example, the network of Figure 14.4 will be unstable if $lw_{1,1}(1)$ is greater than one in magnitude. However, it is possible, for a particular input sequence, that the network output can be small for a particular value of $lw_{1,1}(1)$ greater than one in magnitude, or for certain combinations of values for $lw_{1,1}(1)$ and $iw_{1,1}$.

Finally, it is sometimes difficult to get adequate training data for dynamic networks. This is because the inputs to some layers will come from tapped delay lines. This means that the elements of the input vector cannot be selected independently, since the time sequence from which they are sampled is generally correlated in time. Unlike static networks, in which the network response depends only on the input to the network at the current time, dynamic network responses depend on the history of the input sequence. The data used to train the network must be representative of all situations for which the network will be used, both in terms of the ranges for each input, but also in terms of the variation of the inputs over time.

To illustrate the training of dynamic networks, consider again the simple recurrent network of Figure 14.4, but let's use a nonlinear sigmoid transfer function, as shown in Figure 14.9.

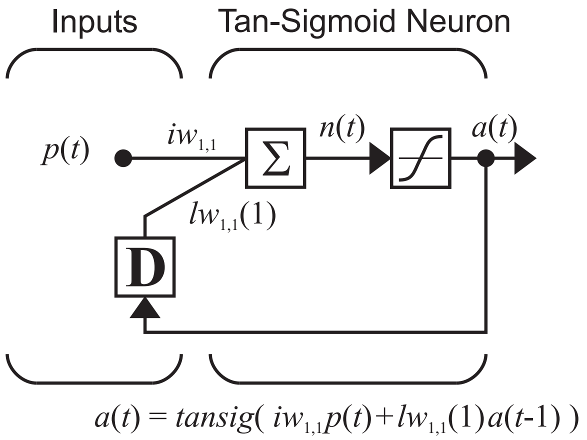


Figure 14.9 Nonlinear Recurrent Network

Recall from Chapter 11 that static multilayer networks can be used to approximate functions. Dynamic networks can be used to approximate dynamic systems. A function maps from one vector space (the domain) to another vector space (the range). A dynamic system maps from one set of time sequences (the input sequences $p(t)$) to another set of time sequences (the output sequences $a(t)$). For example, the network of Figure 14.9 is a dynamic system. It maps from input sequences to output sequences.

In order to simplify our analysis, we will give the network a problem for which we know the optimal solution. The dynamic system we will approximate is the same network, with the following values for the weights:

$$lw_{1,1}(1) = 0.5\,,\; iw_{1,1} = 0.5\,, \tag{14.64}$$

The input sequence that we use to train a dynamic network must be representative of all possible input sequences. Because this network is so simple, it is not difficult to find an appropriate input sequence, but for many practical networks it can be difficult. We will use a standard form of input sequence (called the skyline function), which consists of a series of pulses of varying height and width. The input and target sequences are shown in Figure 14.10. The circles represent the input sequence and the dots represent the target sequence. The targets were created by applying the given input sequence to the network of Figure 14.9, with the weights given by Eq. (14.64).

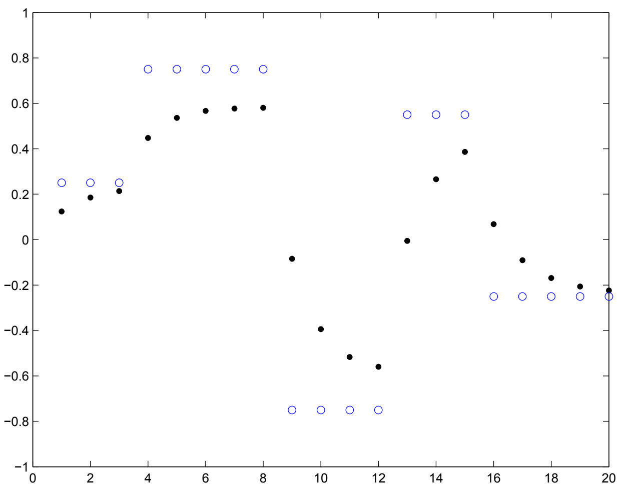
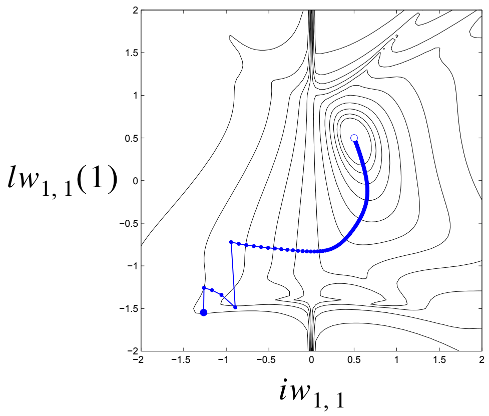


Figure 14.10 Input and Target Sequences

Figure 14.11 shows the squared error performance surface for this problem. Note that as the weight $lw_{1,1}(1)$ becomes greater than one in magnitude, the squared error grows steeply. This effect would be even more prominent, if the length of the training sequence were longer. However, we can also see some narrow valleys in the surface in the regions where $lw_{1,1}(1)$ is greater than one. (This is a very common result, as discussed in [PhHa13]. See Exercise E14.18 to investigate the cause of these valleys.)

The narrow valleys can have an effect on training, since the trajectory can be trapped or misdirected by the spurious valleys. On the left of Figure 14.11 we see a steepest descent path. The path is misdirected at the beginning of the trajectory, because of the narrow valley seen near the bottom of the contour plot.



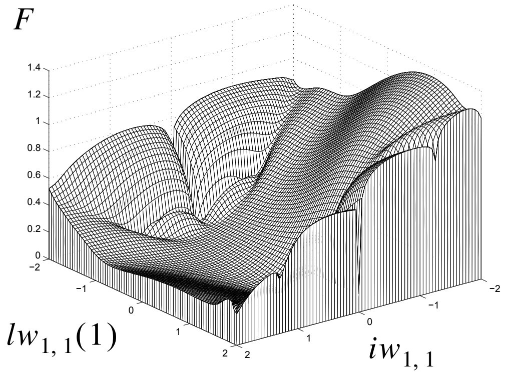


Figure 14.11 Performance Surface and Steepest Descent Trajectory

To experiment with the training of this recurrent network, use the Neural Network Design Demonstration Recurrent Network Training (`nnd14rnt`).

Summary of Results

Real-Time Recurrent Learning Gradient

```
Initialize:
  $\frac{\partial \mathbf{a}^u(t)}{\partial \mathbf{x}^T} = \mathbf{0},\ t \le 0$, for all $u \in U$,
For $t = 1$ to $Q$,
  $U' = \varnothing$, $E_S(u) = \varnothing$ and $E_S^X(u) = \varnothing$ for all $u \in U$.
  For $m$ decremented through the BP order
    For all $u \in U'$, if $E_S(u) \cap L_m^b \neq \varnothing$
      $\mathbf{S}^{u,m}(t) = \left[ \sum_{l \in E_S(u) \cap L_m^b} \mathbf{S}^{u,l}(t)\mathbf{LW}^{l,m}(0) \right] \dot{\mathbf{F}}^m(\mathbf{n}^m(t))$
      add $m$ to the set $E_S(u)$
      if $m \in X$, add $m$ to the set $E_S^X(u)$
    EndFor $u$
    If $m \in U$
      $\mathbf{S}^{m,m}(t) = \dot{\mathbf{F}}^m(\mathbf{n}^m(t))$
      add $m$ to the sets $U'$ and $E_S(m)$
      if $m \in X$, add $m$ to the set $E_S^X(m)$
    EndIf $m$
  EndFor $m$
  For $u \in U$ incremented through the simulation order
    For all weights and biases ($\mathbf{x}$ is a vector containing all weights and biases)
      $\frac{\partial^e \mathbf{a}^u(t)}{\partial vec(\mathbf{IW}^{m,l}(d))^T} = [\mathbf{p}^l(t-d)]^T \otimes \mathbf{S}^{u,m}(t)$
      $\frac{\partial^e \mathbf{a}^u(t)}{\partial vec(\mathbf{LW}^{m,l}(d))^T} = [\mathbf{a}^l(t-d)]^T \otimes \mathbf{S}^{u,m}(t)$
      $\frac{\partial^e \mathbf{a}^u(t)}{\partial (\mathbf{b}^m)^T} = \mathbf{S}^{u,m}(t)$
    EndFor weights and biases
    $\frac{\partial \mathbf{a}^u(t)}{\partial \mathbf{x}^T} = \frac{\partial^e \mathbf{a}^u(t)}{\partial \mathbf{x}^T} + \sum_{x \in E_S^X(u)} \mathbf{S}^{u,x}(t) \sum_{u' \in E_{LW}^U(x)} \sum_{d \in DL_{x,u'}} \mathbf{LW}^{x,u'}(d) \times \frac{\partial \mathbf{a}^{u'}(t-d)}{\partial \mathbf{x}^T}$
  EndFor $u$
EndFor $t$
Compute Gradients
  $\frac{\partial F}{\partial \mathbf{x}} = \sum_{t=1}^{Q} \sum_{u \in U} \left[ \left[ \frac{\partial \mathbf{a}^u(t)}{\partial \mathbf{x}^T} \right]^T \times \frac{\partial^e F}{\partial \mathbf{a}^u(t)} \right]$
```

Backpropagation-Through-Time Gradient

```
Initialize:
    $\frac{\partial F}{\partial \mathbf{a}^u(t)} = \mathbf{0}, t > Q$, for all $u \in U$,
For $t = Q$ to 1,
  $U' = \varnothing$, $E_S(u) = \varnothing$, and $E_S^U(u) = \varnothing$ for all $u \in U$.
  For $m$ decremented through the BP order
    For all $u \in U'$, if $E_S(u) \cap L_m^b \neq \varnothing$
      $\mathbf{S}^{u,m}(t) = \left[ \sum_{l \in E_S(u) \cap L_m^b} \mathbf{S}^{u,l}(t)\mathbf{LW}^{l,m}(0) \right] \dot{\mathbf{F}}^m(\mathbf{n}^m(t))$
      add $m$ to the set $E_S(u)$
      add $u$ to the set $E_S^U(m)$
    EndFor $u$
    If $m \in U$
      $\mathbf{S}^{m,m}(t) = \dot{\mathbf{F}}^m(\mathbf{n}^m(t))$
      add $m$ to the sets $U'$, $E_S(m)$ and $E_S^U(m)$
    EndIf $m$
  EndFor $m$
  For $u \in U$ decremented through the BP order
    $\frac{\partial F}{\partial \mathbf{a}^u(t)} = \frac{\partial^e F}{\partial \mathbf{a}^u(t)} + \sum_{x \in E_{LW}^X(u)} \sum_{d \in DL_{x,u}} \mathbf{LW}^{x,u}(d)^T \sum_{u' \in E_S^U(x)} \mathbf{S}^{u',x}(t+d)^T \times \frac{\partial F}{\partial \mathbf{a}^{u'}(t+d)}$
  EndFor $u$
  For all layers $m$
    $\mathbf{d}^m(t) = \sum_{u \in E_S^U(m)} [\mathbf{S}^{u,m}(t)]^T \times \frac{\partial F}{\partial \mathbf{a}^u(t)}$
  EndFor $m$
EndFor $t$
Compute Gradients
    $\frac{\partial F}{\partial \mathbf{LW}^{m,l}(d)} = \sum_{t=1}^{Q} \mathbf{d}^m(t) \times [\mathbf{a}^l(t-d)]^T$
    $\frac{\partial F}{\partial \mathbf{IW}^{m,l}(d)} = \sum_{t=1}^{Q} \mathbf{d}^m(t) \times [\mathbf{p}^l(t-d)]^T$
    $\frac{\partial F}{\partial \mathbf{b}^m} = \sum_{t=1}^{Q} \mathbf{d}^m(t)$
```

Definitions/Notation

$\mathbf{p}^l(t)$ is the lth *input vector* to the network at time t.

$\mathbf{n}^m(t)$ is the *net input* for layer m.

$\mathbf{f}^m(\ )$ is the *transfer function* for layer m.

$\mathbf{a}^m(t)$ is the *output* for layer m.

$\mathbf{IW}^{m,l}$ is the *input weight* between input l and layer m.

$\mathbf{LW}^{m,l}$ is the *layer weight* between layer l and layer m.

$\mathbf{b}^m$ is the *bias vector* for layer m.

$DL_{m,l}$ is the set of all delays in the tapped delay line between Layer l and Layer m.

$DI_{m,l}$ is the set of all delays in the tapped delay line between Input l and Layer m.

I_m is the set of indices of input vectors that connect to layer m.

L_m^f is the set of indices of layers that directly connect *forward* to layer m.

L_m^b is the set of indices of layers that are directly connected backwards to layer m (or to which layer m connects forward) and that contain no delays in the connection.

A layer is an *input layer* if it has an input weight, or if it contains any delays with any of its weight matrices. The set of input layers is X.

A layer is an *output layer* if its output will be compared to a target during training, or if it is connected to an input layer through a matrix that has any delays associated with it. The set of output layers is U.

The *sensitivity* is defined $s_{k,i}^{u,m}(t) \equiv \dfrac{\partial^e a_k^u(t)}{\partial n_i^m(t)}$.

$$E_{LW}^U(x) = \{u \in U \ni \exists(\mathbf{LW}^{x,u}(d) \neq 0, d \neq 0)\}$$

$$E_S^X(u) = \{x \in X \ni \exists(\mathbf{S}^{u,x} \neq 0)\}$$

$$E_S(u) = \{x \ni \exists(\mathbf{S}^{u,x} \neq 0)\}$$

$$E_{LW}^X(u) = \{x \in X \ni \exists(\mathbf{LW}^{x,u}(d) \neq 0, d \neq 0)\}$$

$$E_S^U(x) = \{u \in U \ni \exists(\mathbf{S}^{u,x} \neq 0)\}$$

Solved Problems

P14.1 Before stating the problem, let's first introduce some notation that will allow us to efficiently represent dynamic networks:

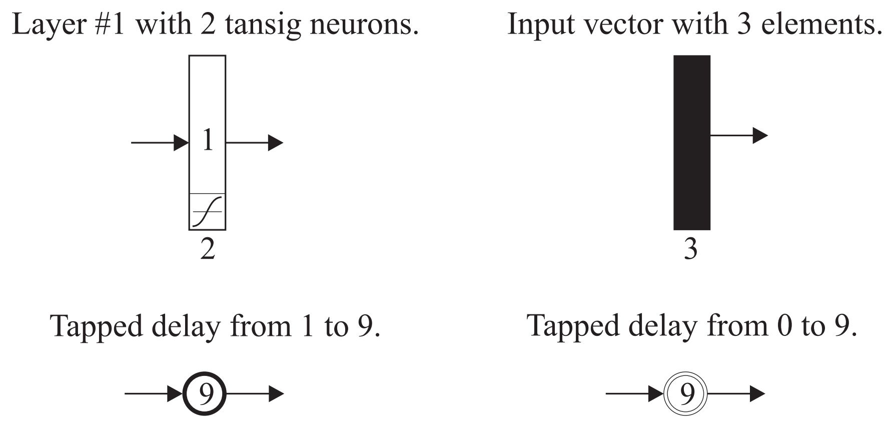

Layer #1 with 2 tansig neurons.
1
2
Input vector with 3 elements.
3
Tapped delay from 1 to 9.
9
Tapped delay from 0 to 9.
9

Figure P14.1 Blocks for Dynamic Network Schematics

Using this notation, consider the following network

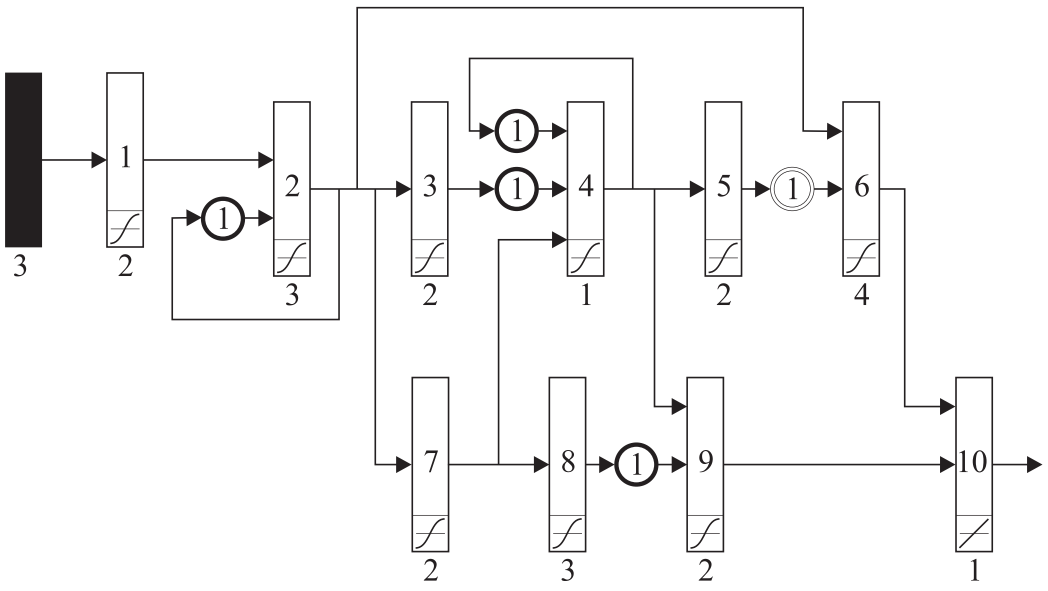


Figure P14.2 Example Dynamic Network for Problem P14.1

Define the network architecture, by showing U, X, I_m, $DI_{m,1}$, $DL_{m,l}$, L_m^f, L_m^b, $E_{LW}^U(x)$, $E_{LW}^X(u)$. Also, select a simulation order and indicate the dimension of each weight matrix.

The input layers have input weights, or have delays with their weight matrices. Therefore $X = \{1, 2, 4, 6, 9\}$. The output layers are compared to targets, or connect to an input layer through a matrix that has any delays. If we assume that only Layer 10 is compared to a target, then $U = \{2, 3, 4, 5, 8, 10\}$. Since the single input vector connects only to Layer 1, the only nonempty set of inputs will be $I_1 = \{1\}$. For the same reason, there will only be one nonempty set of input delays: $DI_{1,1} = \{0\}$. The connections between layers are defined by

$$L_1^f = \varnothing,\ L_2^f = \{1, 2\},\ L_3^f = \{2\},\ L_4^f = \{3, 4, 7\},\ L_5^f = \{4\},$$

$$L_6^f = \{2, 5\},\ L_7^f = \{2\},\ L_8^f = \{7\},\ L_9^f = \{4, 8\},\ L_{10}^f = \{6, 9\}.$$

$$L_1^b = \{2\},\ L_2^b = \{3, 6, 7\},\ L_3^b = \varnothing,\ L_4^b = \{5, 9\},\ L_5^b = \varnothing,$$

$$L_6^b = \{10\},\ L_7^b = \{4, 8\},\ L_8^b = \varnothing,\ L_9^b = \{10\},\ L_{10}^b = \varnothing.$$

Associated with these connections are the following layer delays

$$DL_{2,1} = \{0\},\ DL_{2,2} = \{1\},\ DL_{3,2} = \{0\},\ DL_{4,3} = \{1\},\ DL_{4,4} = \{1\},$$

$$DL_{4,7} = \{0\},\ DL_{5,4} = \{0\},\ DL_{6,2} = \{0\},\ DL_{6,5} = \{0, 1\},\ DL_{7,2} = \{0\},$$

$$DL_{8,7} = \{0\},\ DL_{9,4} = \{0\},\ DL_{9,8} = \{1\},\ DL_{10,6} = \{0\},\ DL_{10,9} = \{0\}.$$

The layers that have connections from output layers are

$$E_{LW}^U(2) = \{2\},\ E_{LW}^U(4) = \{3, 4\},$$

$$E_{LW}^U(6) = \{5\},\ E_{LW}^U(9) = \{8\}.$$

The layers that connect to input layers are

$$E_{LW}^X(2) = \{2\},\ E_{LW}^X(3) = \{4\},\ E_{LW}^X(4) = \{4\},$$

$$E_{LW}^X(5) = \{6\},\ E_{LW}^X(8) = \{9\}.$$

The simulation order can be chosen to be $\{1, 2, 3, 7, 4, 5, 6, 8, 9, 10\}$. The dimensions of the weight matrices are

$$\mathbf{IW}^{1,1}(0) \Rightarrow 2 \times 3,\ \mathbf{LW}^{2,1}(0) \Rightarrow 3 \times 2,\ \mathbf{LW}^{2,2}(1) \Rightarrow 3 \times 3,\ \mathbf{LW}^{3,2}(0) \Rightarrow 2 \times 3,$$

$$\mathbf{LW}^{4,3}(1) \Rightarrow 1 \times 2,\ \mathbf{LW}^{4,4}(1) \Rightarrow 1 \times 1,\ \mathbf{LW}^{4,7}(0) \Rightarrow 1 \times 2,\ \mathbf{LW}^{5,4}(0) \Rightarrow 2 \times 1,$$

$$\mathbf{LW}^{6,2}(0) \Rightarrow 4 \times 3,\ \mathbf{LW}^{6,5}(1) \Rightarrow 4 \times 2,\ \mathbf{LW}^{7,2}(0) \Rightarrow 2 \times 3,\ \mathbf{LW}^{8,7}(0) \Rightarrow 3 \times 2,$$

$$\mathbf{LW}^{9,4}(0) \Rightarrow 2 \times 1,\ \mathbf{LW}^{9,8}(1) \Rightarrow 2 \times 3,\ \mathbf{LW}^{10,6}(0) \Rightarrow 1 \times 4,\ \mathbf{LW}^{10,9}(d) \Rightarrow 1 \times 2.$$

P14.2 Write out the BPTT equations for the network presented in Problem P14.1.

We will assume that the network response has been computed for all time points, and we will demonstrate the process for one time step, since they all will be similar. We proceed through the layers according to the backpropagation order, which is the reverse of the simulation order: $\{10, 9, 8, 6, 5, 4, 7, 3, 2, 1\}$.

$$\mathbf{S}^{10,10}(t) = \dot{\mathbf{F}}^{10}(\mathbf{n}^{10}(t))$$

$$\frac{\partial F}{\partial \mathbf{a}^{10}(t)} = \frac{\partial^e F}{\partial \mathbf{a}^{10}(t)}$$

$$\mathbf{d}^{10}(t) = [\mathbf{S}^{10,10}(t)]^T \times \frac{\partial F}{\partial \mathbf{a}^{10}(t)}$$

$$\mathbf{S}^{10,9}(t) = \mathbf{S}^{10,10}(t)\mathbf{LW}^{10,9}(0)\dot{\mathbf{F}}^{9}(\mathbf{n}^{9}(t))$$

$$\mathbf{d}^{9}(t) = [\mathbf{S}^{10,9}(t)]^T \times \frac{\partial F}{\partial \mathbf{a}^{10}(t)}$$

$$\mathbf{S}^{8,8}(t) = \dot{\mathbf{F}}^{8}(\mathbf{n}^{8}(t))$$

$$\frac{\partial F}{\partial \mathbf{a}^{8}(t)} = \frac{\partial^e F}{\partial \mathbf{a}^{8}(t)} + \mathbf{LW}^{9,8}(1)^T\mathbf{S}^{10,9}(t+1)^T \times \frac{\partial F}{\partial \mathbf{a}^{10}(t+1)}$$

$$\mathbf{d}^{8}(t) = [\mathbf{S}^{8,8}(t)]^T \times \frac{\partial F}{\partial \mathbf{a}^{8}(t)}$$

$$\mathbf{S}^{10,6}(t) = \mathbf{S}^{10,10}(t)\mathbf{LW}^{10,6}(0)\dot{\mathbf{F}}^{6}(\mathbf{n}^{6}(t))$$

$$\mathbf{d}^{6}(t) = [\mathbf{S}^{10,6}(t)]^T \times \frac{\partial F}{\partial \mathbf{a}^{10}(t)}$$

$$\mathbf{S}^{5,5}(t) = \dot{\mathbf{F}}^{5}(\mathbf{n}^{5}(t))$$

$$\frac{\partial F}{\partial \mathbf{a}^{5}(t)} = \frac{\partial^e F}{\partial \mathbf{a}^{5}(t)} + \mathbf{LW}^{6,5}(1)^T\mathbf{S}^{10,6}(t+1)^T \times \frac{\partial F}{\partial \mathbf{a}^{10}(t+1)} + \mathbf{LW}^{6,5}(0)^T\mathbf{S}^{10,6}(t)^T \times \frac{\partial F}{\partial \mathbf{a}^{10}(t)}$$

$$\mathbf{d}^{5}(t) = [\mathbf{S}^{5,5}(t)]^T \times \frac{\partial F}{\partial \mathbf{a}^{5}(t)}$$

$$\mathbf{S}^{10,4}(t) = \mathbf{S}^{10,9}(t)\mathbf{LW}^{9,4}(0)\dot{\mathbf{F}}^4(\mathbf{n}^4(t))$$

$$\mathbf{S}^{5,4}(t) = \mathbf{S}^{5,5}(t)\mathbf{LW}^{5,4}(0)\dot{\mathbf{F}}^4(\mathbf{n}^4(t))$$

$$\mathbf{S}^{4,4}(t) = \dot{\mathbf{F}}^4(\mathbf{n}^4(t))$$

$$\frac{\partial F}{\partial \mathbf{a}^4(t)} = \frac{\partial^e F}{\partial \mathbf{a}^4(t)} + \mathbf{LW}^{4,4}(1)^T\left[\mathbf{S}^{4,4}(t+1)^T \times \frac{\partial F}{\partial \mathbf{a}^4(t+1)} + \mathbf{S}^{5,4}(t+1)^T \times \frac{\partial F}{\partial \mathbf{a}^5(t+1)} + \mathbf{S}^{10,4}(t+1)^T \times \frac{\partial F}{\partial \mathbf{a}^{10}(t+1)}\right]$$

$$\mathbf{d}^4(t) = [\mathbf{S}^{4,4}(t)]^T \times \frac{\partial F}{\partial \mathbf{a}^4(t)} + [\mathbf{S}^{5,4}(t)]^T \times \frac{\partial F}{\partial \mathbf{a}^5(t)} + [\mathbf{S}^{10,4}(t)]^T \times \frac{\partial F}{\partial \mathbf{a}^{10}(t)}$$

$$\mathbf{S}^{10,7}(t) = \mathbf{S}^{10,4}(t)\mathbf{LW}^{4,7}(0)\dot{\mathbf{F}}^7(\mathbf{n}^7(t))$$

$$\mathbf{S}^{8,7}(t) = \mathbf{S}^{8,8}(t)\mathbf{LW}^{8,7}(0)\dot{\mathbf{F}}^7(\mathbf{n}^7(t))$$

$$\mathbf{S}^{5,7}(t) = \mathbf{S}^{5,4}(t)\mathbf{LW}^{4,7}(0)\dot{\mathbf{F}}^7(\mathbf{n}^7(t))$$

$$\mathbf{S}^{4,7}(t) = \mathbf{S}^{4,4}(t)\mathbf{LW}^{4,7}(0)\dot{\mathbf{F}}^7(\mathbf{n}^7(t))$$

$$\mathbf{d}^7(t) = [\mathbf{S}^{10,7}(t)]^T \times \frac{\partial F}{\partial \mathbf{a}^{10}(t)} + [\mathbf{S}^{8,7}(t)]^T \times \frac{\partial F}{\partial \mathbf{a}^8(t)} + [\mathbf{S}^{5,7}(t)]^T \times \frac{\partial F}{\partial \mathbf{a}^5(t)} + [\mathbf{S}^{4,7}(t)]^T \times \frac{\partial F}{\partial \mathbf{a}^4(t)}$$

$$\mathbf{S}^{3,3}(t) = \dot{\mathbf{F}}^3(\mathbf{n}^3(t))$$

$$\frac{\partial F}{\partial \mathbf{a}^3(t)} = \frac{\partial^e F}{\partial \mathbf{a}^3(t)} + \mathbf{LW}^{4,3}(1)^T\left[\mathbf{S}^{4,4}(t+1)^T \times \frac{\partial F}{\partial \mathbf{a}^4(t+1)} + \mathbf{S}^{5,4}(t+1)^T \times \frac{\partial F}{\partial \mathbf{a}^5(t+1)} + \mathbf{S}^{10,4}(t+1)^T \times \frac{\partial F}{\partial \mathbf{a}^{10}(t+1)}\right]$$

$$\mathbf{d}^3(t) = [\mathbf{S}^{3,3}(t)]^T \times \frac{\partial F}{\partial \mathbf{a}^3(t)}$$

$$\mathbf{S}^{10,2}(t) = \mathbf{S}^{10,6}(t)\mathbf{LW}^{6,2}(0)\dot{\mathbf{F}}^2(\mathbf{n}^2(t)) + \mathbf{S}^{10,7}(t)\mathbf{LW}^{7,2}(0)\dot{\mathbf{F}}^2(\mathbf{n}^2(t))$$

$$\mathbf{S}^{8,2}(t) = \mathbf{S}^{8,7}(t)\mathbf{LW}^{7,2}(0)\dot{\mathbf{F}}^2(\mathbf{n}^2(t))$$

$$\mathbf{S}^{5,2}(t) = \mathbf{S}^{5,7}(t)\mathbf{LW}^{7,2}(0)\dot{\mathbf{F}}^2(\mathbf{n}^2(t))$$

$$\mathbf{S}^{4,2}(t) = \mathbf{S}^{4,7}(t)\mathbf{LW}^{7,2}(0)\dot{\mathbf{F}}^2(\mathbf{n}^2(t))$$

$$\mathbf{S}^{3,2}(t) = \mathbf{S}^{3,3}(t)\mathbf{LW}^{3,2}(0)\dot{\mathbf{F}}^2(\mathbf{n}^2(t))$$

$$\mathbf{S}^{2,2}(t) = \dot{\mathbf{F}}^{2}(\mathbf{n}^{2}(t))$$

$$\frac{\partial F}{\partial \mathbf{a}^{2}(t)} = \frac{\partial^{e} F}{\partial \mathbf{a}^{2}(t)} + \mathbf{LW}^{2,2}(1)^{T}\Bigg[\mathbf{S}^{2,2}(t+1)^{T} \times \frac{\partial F}{\partial \mathbf{a}^{2}(t+1)} + \mathbf{S}^{3,2}(t+1)^{T} \times \frac{\partial F}{\partial \mathbf{a}^{3}(t+1)}$$
$$+ \mathbf{S}^{4,2}(t+1)^{T} \times \frac{\partial F}{\partial \mathbf{a}^{4}(t+1)} + \mathbf{S}^{5,2}(t+1)^{T} \times \frac{\partial F}{\partial \mathbf{a}^{5}(t+1)}$$
$$+ \mathbf{S}^{8,2}(t+1)^{T} \times \frac{\partial F}{\partial \mathbf{a}^{8}(t+1)} + \mathbf{S}^{10,2}(t+1)^{T} \times \frac{\partial F}{\partial \mathbf{a}^{10}(t+1)}\Bigg]$$

$$\mathbf{d}^{2}(t) = [\mathbf{S}^{10,2}(t)]^{T} \times \frac{\partial F}{\partial \mathbf{a}^{10}(t)} + [\mathbf{S}^{8,2}(t)]^{T} \times \frac{\partial F}{\partial \mathbf{a}^{8}(t)} + [\mathbf{S}^{5,2}(t)]^{T} \times \frac{\partial F}{\partial \mathbf{a}^{5}(t)}$$
$$+ [\mathbf{S}^{4,2}(t)]^{T} \times \frac{\partial F}{\partial \mathbf{a}^{4}(t)} + [\mathbf{S}^{3,2}(t)]^{T} \times \frac{\partial F}{\partial \mathbf{a}^{3}(t)} + [\mathbf{S}^{2,2}(t)]^{T} \times \frac{\partial F}{\partial \mathbf{a}^{2}(t)}$$

$$\mathbf{S}^{10,1}(t) = \mathbf{S}^{10,2}(t)\mathbf{LW}^{2,1}(0)\dot{\mathbf{F}}^{1}(\mathbf{n}^{1}(t))$$

$$\mathbf{S}^{8,1}(t) = \mathbf{S}^{8,2}(t)\mathbf{LW}^{2,1}(0)\dot{\mathbf{F}}^{1}(\mathbf{n}^{1}(t))$$

$$\mathbf{S}^{5,1}(t) = \mathbf{S}^{5,2}(t)\mathbf{LW}^{2,1}(0)\dot{\mathbf{F}}^{1}(\mathbf{n}^{1}(t))$$

$$\mathbf{S}^{4,1}(t) = \mathbf{S}^{4,2}(t)\mathbf{LW}^{2,1}(0)\dot{\mathbf{F}}^{1}(\mathbf{n}^{1}(t))$$

$$\mathbf{S}^{3,1}(t) = \mathbf{S}^{3,2}(t)\mathbf{LW}^{2,1}(0)\dot{\mathbf{F}}^{1}(\mathbf{n}^{1}(t))$$

$$\mathbf{S}^{2,1}(t) = \mathbf{S}^{2,2}(t)\mathbf{LW}^{2,1}(0)\dot{\mathbf{F}}^{1}(\mathbf{n}^{1}(t))$$

$$\mathbf{d}^{1}(t) = [\mathbf{S}^{10,1}(t)]^{T} \times \frac{\partial F}{\partial \mathbf{a}^{10}(t)} + [\mathbf{S}^{8,1}(t)]^{T} \times \frac{\partial F}{\partial \mathbf{a}^{8}(t)} + [\mathbf{S}^{5,1}(t)]^{T} \times \frac{\partial F}{\partial \mathbf{a}^{5}(t)}$$
$$+ [\mathbf{S}^{4,1}(t)]^{T} \times \frac{\partial F}{\partial \mathbf{a}^{4}(t)} + [\mathbf{S}^{3,1}(t)]^{T} \times \frac{\partial F}{\partial \mathbf{a}^{3}(t)} + [\mathbf{S}^{2,1}(t)]^{T} \times \frac{\partial F}{\partial \mathbf{a}^{2}(t)}$$

After the preceding steps have been repeated for all time points, from the last time point back to the first, then the gradient can be computed as follows:

$$\frac{\partial F}{\partial \mathbf{LW}^{m,l}(d)} = \sum_{t=1}^{Q} \mathbf{d}^{m}(t) \times [\mathbf{a}^{l}(t-d)]^{T}$$

$$\frac{\partial F}{\partial \mathbf{IW}^{m,l}(d)} = \sum_{t=1}^{Q} \mathbf{d}^m(t) \times [\mathbf{p}^l(t-d)]^T$$

$$\frac{\partial F}{\partial \mathbf{b}^m} = \sum_{t=1}^{Q} \mathbf{d}^m(t)$$

P14.3 Write out the RTRL equations for the network presented in Problem P14.1.

As in the previous problem, we will demonstrate the process for one time step, since each step is similar. We will proceed through the layers according to the backpropagation order. The sensitivity matrices $\mathbf{S}^{u,m}(t)$ are computed in the same way for the RTRL algorithm as for the BPTT algorithm, so we won't repeat those steps from Problem P14.2.

The explicit derivative calculations for the input weight will be

$$\frac{\partial^e \mathbf{a}^u(t)}{\partial vec(\mathbf{IW}^{1,1}(0))^T} = [\mathbf{p}^1(t)]^T \otimes \mathbf{S}^{u,1}(t)$$

For the layer weights and the biases, the explicit derivatives are calculated by

$$\frac{\partial^e \mathbf{a}^u(t)}{\partial vec(\mathbf{LW}^{m,l}(d))^T} = [\mathbf{a}^l(t-d)]^T \otimes \mathbf{S}^{u,m}(t)$$

$$\frac{\partial^e \mathbf{a}^u(t)}{\partial (\mathbf{b}^m)^T} = \mathbf{S}^{u,m}(t)$$

For the total derivatives, we have

$$\frac{\partial \mathbf{a}^2(t)}{\partial \mathbf{x}^T} = \frac{\partial^e \mathbf{a}^2(t)}{\partial \mathbf{x}^T} + \mathbf{S}^{2,2}(t)\left[\mathbf{LW}^{2,2}(1) \times \frac{\partial \mathbf{a}^2(t-1)}{\partial \mathbf{x}^T}\right]$$

$$\frac{\partial \mathbf{a}^3(t)}{\partial \mathbf{x}^T} = \frac{\partial^e \mathbf{a}^3(t)}{\partial \mathbf{x}^T} + \mathbf{S}^{3,2}(t)\left[\mathbf{LW}^{2,2}(1) \times \frac{\partial \mathbf{a}^2(t-1)}{\partial \mathbf{x}^T}\right]$$

$$\frac{\partial \mathbf{a}^4(t)}{\partial \mathbf{x}^T} = \frac{\partial^e \mathbf{a}^4(t)}{\partial \mathbf{x}^T} + \mathbf{S}^{4,4}(t)\left[\mathbf{LW}^{4,4}(1) \times \frac{\partial \mathbf{a}^4(t-1)}{\partial \mathbf{x}^T} + \mathbf{LW}^{4,3}(1) \times \frac{\partial \mathbf{a}^3(t-1)}{\partial \mathbf{x}^T}\right]$$
$$+ \mathbf{S}^{4,2}(t)\left[\mathbf{LW}^{2,2}(1) \times \frac{\partial \mathbf{a}^2(t-1)}{\partial \mathbf{x}^T}\right]$$

$$\frac{\partial \mathbf{a}^5(t)}{\partial \mathbf{x}^T} = \frac{\partial^e \mathbf{a}^5(t)}{\partial \mathbf{x}^T} + \mathbf{S}^{5,4}(t)\left[\mathbf{LW}^{4,4}(1) \times \frac{\partial \mathbf{a}^4(t-1)}{\partial \mathbf{x}^T} + \mathbf{LW}^{4,3}(1) \times \frac{\partial \mathbf{a}^3(t-1)}{\partial \mathbf{x}^T}\right]$$
$$+ \mathbf{S}^{5,2}(t)\left[\mathbf{LW}^{2,2}(1) \times \frac{\partial \mathbf{a}^2(t-1)}{\partial \mathbf{x}^T}\right]$$

$$\frac{\partial \mathbf{a}^8(t)}{\partial \mathbf{x}^T} = \frac{\partial^e \mathbf{a}^8(t)}{\partial \mathbf{x}^T} + \mathbf{S}^{8,2}(t)\left[\mathbf{LW}^{2,2}(1) \times \frac{\partial \mathbf{a}^2(t-1)}{\partial \mathbf{x}^T}\right]$$

$$\frac{\partial \mathbf{a}^{10}(t)}{\partial \mathbf{x}^T} = \frac{\partial^e \mathbf{a}^{10}(t)}{\partial \mathbf{x}^T} + \mathbf{S}^{10,9}(t)\left[\mathbf{LW}^{9,8}(1) \times \frac{\partial \mathbf{a}^8(t-1)}{\partial \mathbf{x}^T}\right]$$
$$+ \mathbf{S}^{10,6}(t)\left[\mathbf{LW}^{6,5}(0) \times \frac{\partial \mathbf{a}^5(t)}{\partial \mathbf{x}^T} + \mathbf{LW}^{6,5}(1) \times \frac{\partial \mathbf{a}^5(t-1)}{\partial \mathbf{x}^T}\right]$$
$$+ \mathbf{S}^{10,4}(t)\left[\mathbf{LW}^{4,4}(1) \times \frac{\partial \mathbf{a}^4(t-1)}{\partial \mathbf{x}^T} + \mathbf{LW}^{4,3}(1) \times \frac{\partial \mathbf{a}^3(t-1)}{\partial \mathbf{x}^T}\right] + \mathbf{S}^{10,2}(t)\left[\mathbf{LW}^{2,2}(1) \times \frac{\partial \mathbf{a}^2(t-1)}{\partial \mathbf{x}^T}\right]$$

After the above steps are iterated through all time points, we can compute the gradient with

$$\frac{\partial F}{\partial \mathbf{x}^T} = \sum_{t=1}^{Q}\left[\left[\frac{\partial^e F}{\partial \mathbf{a}^2(t)}\right]^T \times \frac{\partial \mathbf{a}^2(t)}{\partial \mathbf{x}^T} + \left[\frac{\partial^e F}{\partial \mathbf{a}^3(t)}\right]^T \times \frac{\partial \mathbf{a}^3(t)}{\partial \mathbf{x}^T} + \left[\frac{\partial^e F}{\partial \mathbf{a}^4(t)}\right]^T \times \frac{\partial \mathbf{a}^4(t)}{\partial \mathbf{x}^T}\right.$$
$$\left. + \left[\frac{\partial^e F}{\partial \mathbf{a}^5(t)}\right]^T \times \frac{\partial \mathbf{a}^5(t)}{\partial \mathbf{x}^T} + \left[\frac{\partial^e F}{\partial \mathbf{a}^8(t)}\right]^T \times \frac{\partial \mathbf{a}^8(t)}{\partial \mathbf{x}^T} + \left[\frac{\partial^e F}{\partial \mathbf{a}^{10}(t)}\right]^T \times \frac{\partial \mathbf{a}^{10}(t)}{\partial \mathbf{x}^T}\right]$$

P14.4 From the previous problem, show the detail of the calculations involved in the explicit derivative term

$$\frac{\partial^e \mathbf{a}^2(t)}{\partial vec(\mathbf{IW}^{1,1}(0))^T} = [\mathbf{p}^1(t)]^T \otimes \mathbf{S}^{2,1}(t)$$

First, let's display the details of the individual vectors and matrices in this expression:

$$\mathbf{IW}^{1,1}(0) = \begin{bmatrix} iw_{1,1}^{1,1} & iw_{1,2}^{1,1} & iw_{1,3}^{1,1} \\ iw_{2,1}^{1,1} & iw_{2,2}^{1,1} & iw_{2,3}^{1,1} \end{bmatrix}$$

$$vec(\mathbf{IW}^{1,1}(0))^T = \begin{bmatrix} iw_{1,1}^{1,1} & iw_{2,1}^{1,1} & iw_{1,2}^{1,1} & iw_{2,2}^{1,1} & iw_{1,3}^{1,1} & iw_{2,3}^{1,1} \end{bmatrix}$$

$$\mathbf{p}^1(t) = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} \quad \mathbf{S}^{2,1}(t) = \frac{\partial^e \mathbf{a}^2(t)}{\partial \mathbf{n}^1(t)^T} = \begin{bmatrix} s_{1,1}^{2,1} & s_{1,2}^{2,1} \\ s_{2,1}^{2,1} & s_{2,2}^{2,1} \\ s_{3,1}^{2,1} & s_{3,2}^{2,1} \end{bmatrix}$$

$$\frac{\partial^e \mathbf{a}^2(t)}{\partial vec(\mathbf{IW}^{1,1}(0))^T} = \begin{bmatrix} \frac{\partial a_1^2}{\partial iw_{1,1}^{1,1}} & \frac{\partial a_1^2}{\partial iw_{2,1}^{1,1}} & \frac{\partial a_1^2}{\partial iw_{1,2}^{1,1}} & \frac{\partial a_1^2}{\partial iw_{2,2}^{1,1}} & \frac{\partial a_1^2}{\partial iw_{1,3}^{1,1}} & \frac{\partial a_1^2}{\partial iw_{2,3}^{1,1}} \\ \frac{\partial a_2^2}{\partial iw_{1,1}^{1,1}} & \frac{\partial a_2^2}{\partial iw_{2,1}^{1,1}} & \frac{\partial a_2^2}{\partial iw_{1,2}^{1,1}} & \frac{\partial a_2^2}{\partial iw_{2,2}^{1,1}} & \frac{\partial a_2^2}{\partial iw_{1,3}^{1,1}} & \frac{\partial a_2^2}{\partial iw_{2,3}^{1,1}} \\ \frac{\partial a_3^2}{\partial iw_{1,1}^{1,1}} & \frac{\partial a_3^2}{\partial iw_{2,1}^{1,1}} & \frac{\partial a_3^2}{\partial iw_{1,2}^{1,1}} & \frac{\partial a_3^2}{\partial iw_{2,2}^{1,1}} & \frac{\partial a_3^2}{\partial iw_{1,3}^{1,1}} & \frac{\partial a_3^2}{\partial iw_{2,3}^{1,1}} \end{bmatrix}$$

The Kronecker product is defined as

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{1,1}\mathbf{B} & \dots & a_{1,m}\mathbf{B} \\ \vdots & & \vdots \\ a_{n,1}\mathbf{B} & \dots & a_{n,m}\mathbf{B} \end{bmatrix},$$

therefore

$$[\mathbf{p}^1(t)]^T \otimes \mathbf{S}^{2,1}(t) = \begin{bmatrix} p_1 s_{1,1}^{2,1} & p_1 s_{1,2}^{2,1} & p_2 s_{1,1}^{2,1} & p_2 s_{1,2}^{2,1} & p_3 s_{1,1}^{2,1} & p_3 s_{1,2}^{2,1} \\ p_1 s_{2,1}^{2,1} & p_1 s_{2,2}^{2,1} & p_2 s_{2,1}^{2,1} & p_2 s_{2,2}^{2,1} & p_3 s_{2,1}^{2,1} & p_3 s_{2,2}^{2,1} \\ p_1 s_{3,1}^{2,1} & p_1 s_{3,2}^{2,1} & p_2 s_{3,1}^{2,1} & p_2 s_{3,2}^{2,1} & p_3 s_{3,1}^{2,1} & p_3 s_{3,2}^{2,1} \end{bmatrix}.$$

P14.5 Find the computational complexity for the BPTT and RTRL algorithms applied to the sample network in Figure P14.3 as a function of the number of neurons in Layer 1 (S^1), the number of delays in the tapped delay line (D) and the length of the training sequence (Q).

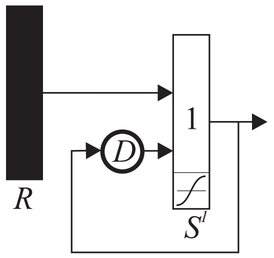


Figure P14.3 Sample Network for Exercise E14.1

The complexity of the BPTT gradient calculation is generally determined by Eq. (14.54). For this network, the most important weights will be $\mathbf{LW}^{1,1}(d)$:

$$\frac{\partial F}{\partial \mathbf{LW}^{1,1}(d)} = \sum_{t=1}^{Q} \mathbf{d}^1(t) \times [\mathbf{a}^1(t-d)]^T,$$

The outer product calculation involves $(S^1)^2$ operations, which must be done for Q time steps and for D delays, so the BPTT gradient calculation is $O[(S^1)^2 DQ]$.

The complexity of the RTRL gradient is based generally on Eq. (14.34). For this sample network, we can consider the equation for $u = 2$:

$$\frac{\partial \mathbf{a}^2(t)}{\partial \mathbf{x}^T} = \frac{\partial^e \mathbf{a}^2(t)}{\partial \mathbf{x}^T} + \mathbf{S}^{2,1}(t)\left[\sum_{d=1}^{D} \mathbf{LW}^{1,1}(d) \times \frac{\partial \mathbf{a}^1(t-d)}{\partial \mathbf{x}^T}\right]$$

Inside the summation we have a matrix multiplication involving an $S^1 \times S^1$ matrix times an $S^1 \times \{(DS^1 + 3)S^1 + 1\}$ matrix. This multiplication will be $O[(S^1)^4 D]$. It must be done for every d and every t, therefore the RTRL gradient calculations are $O[(S^1)^4 D^2 Q]$. The multiplication by the sensitivity matrix does not change the order of the complexity.

Epilogue

Dynamic networks can be trained using the same optimization procedures that we described in Chapter 12 for static multilayer networks. However, the calculation of the gradient for dynamic networks is more complex than for static networks. There are two basic approaches to the gradient calculation in dynamic networks. One approach, backpropagation through time (BPTT), starts at the last time point, and works backward in time to compute the gradient. The second approach, real-time recurrent learning (RTRL), starts at the first time point, and then works forward through time.

RTRL requires more calculations than BPTT for calculating the gradient, but RTRL allows a convenient framework for on-line implementation. Also, RTRL generally requires less storage than BPTT. For Jacobian calculations, RTRL is often more efficient than the BPTT algorithm.

Chapter 27 presents a real-world case study for using dynamic networks to solve a prediction problem.

Further Reading

[DeHa07] O. De Jesús and M. Hagan, “Backpropagation Algorithms for a Broad Class of Dynamic Networks,” *IEEE Transactions on Neural Networks*, vol. 18, no. 1, pp., 2007.

This paper presents a general development of BPTT and RTRL algorithms for gradient and Jacobian calculations. Experimental results are presented that compare the computational complexities of the two algorithms for a variety of network architectures.

[MaNe99] J.R. Magnus and H. Neudecker, *Matrix Differential Calculus*, John Wiley & Sons, Ltd., Chichester, 1999.

This textbook provides a very clear and complete description of matrix theory and matrix differential calculus.

[PhHa13] M. Phan and M. Hagan, “Error Surface of Recurrent Networks,” *IEEE Transactions on Neural Networks and Learning Systems*, Vol. 24, No. 11, pp. 1709 - 1721, October, 2013.

This paper describes spurious valleys that appear in the error surfaces of recurrent networks. It also describes some procedures that can be used to improve training of recurrent networks.

[Werb90] P. J. Werbos, “Backpropagation through time: What it is and how to do it,” *Proceedings of the IEEE*, vol. 78, pp. 1550–1560, 1990.

The backpropagation through time algorithm is one of the two principal approaches to computing the gradients in recurrent neural network. This paper describes the general framework for backpropagation through time.

[WiZi89] R. J. Williams and D. Zipser, “A learning algorithm for continually running fully recurrent neural networks,” *Neural Computation*, vol. 1, pp. 270–280, 1989.

This paper introduced the real-time, recurrent learning algorithm for computing the gradients of dynamic networks. Using this method, the gradients are computed by starting at the first time point, and then working forward through time. The algorithm is suitable for on-line, or real-time, implementation.

Exercises

E14.1 Put the network of Figure 14.1 into the schematic form, which we introduced in Problem P14.1.

E14.2 Consider the network in Figure 14.4, with weight values $iw_{1,1} = 2$ and $lw_{1,1}(1) = 0.5$. If $a(0) = 4$, and $p(1) = 2$, $p(2) = 3$, $p(3) = 2$, find $a(1)$, $a(2)$, $a(3)$.

E14.3 Consider the network in Figure P14.3, with $D = 2$, $S^1 = 1$, $R = 1$, $\mathbf{IW}^{1,1} = [1]$, $\mathbf{LW}^{1,1}(1) = [0.5]$, $\mathbf{LW}^{1,1}(2) = [0.2]$, $\mathbf{a}^1(0) = [2]$ and $\mathbf{a}^1(-1) = [1]$. If $\mathbf{p}^1(1) = [1]$, $\mathbf{p}^1(2) = [2]$ and $\mathbf{p}^1(3) = [-1]$, find $\mathbf{a}^1(1)$, $\mathbf{a}^1(2)$ and $\mathbf{a}^1(3)$.

E14.4 Consider the network in Figure E14.1. Define the network architecture, by showing U, X, I_m, $DI_{m,1}$, $DL_{m,l}$, L_m^f, L_m^b, $E_{LW}^U(x)$, $E_{LW}^X(u)$. Also, select a simulation order and indicate the dimension of each weight matrix.

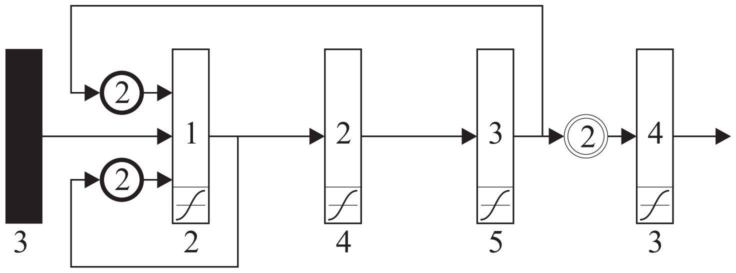


Figure E14.1 Dynamic Network for Exercise E14.4

E14.5 Write out the RTRL equations for the network in Figure E14.2

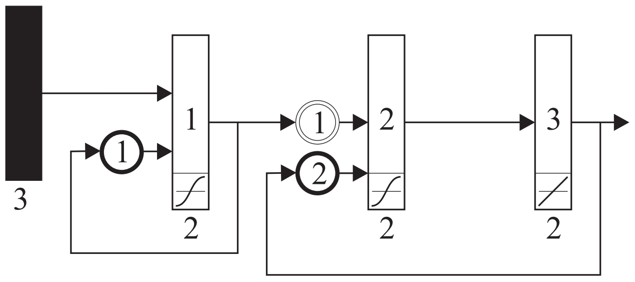


Figure E14.2 Dynamic Network for Exercise E14.5

E14.6 Write out the BPTT equations for the network in Figure E14.2.

E14.7 Write out the equations of operation for the network in Figure E14.3. Assume that all weights have a value of 0.5, and that all biases have a value of 0.

i. Assume that the initial network output is $a(0) = 0.5$, and that the initial network input is $p(1) = 1$. Solve for $a(1)$.

ii. Describe any problems that you would have in simulating this network. Will you be able to apply the BPTT and RTRL algorithms to compute the gradients for this network? What test should you apply to recurrent networks to determine if they can be simulated and trained?

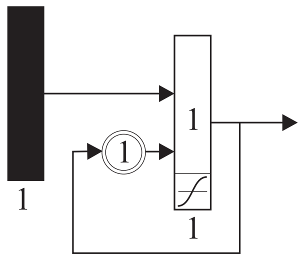


Figure E14.3 Dynamic Network for Exercise E14.7

E14.8 Consider the network in Figure E14.4.

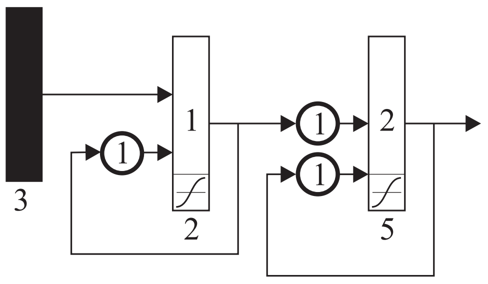


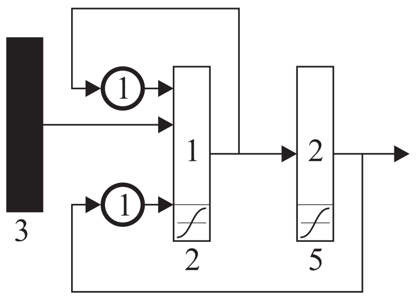
Figure E14.4 Dynamic Network for Exercise E14.8

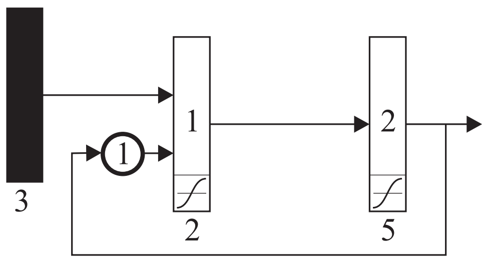
i. Write out the equations for computing the network response.

ii. Write out the BPTT equations for the network.

iii. Write out the RTRL equations for the network.

E14.9 Repeat E14.8 for the following networks.

i. 

ii. 

E14.10 Consider the network in Figure E14.5.

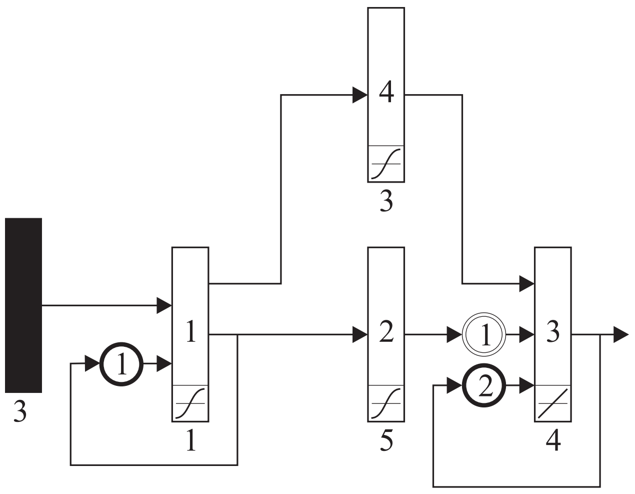


Figure E14.5 Recurrent Network for Exercise E14.10

i. Define the network architecture, by showing U, X, I_m, $DI_{m,1}$, $DL_{m,l}$, L_m^f, L_m^b, $E_{LW}^U(x)$, $E_{LW}^X(u)$.

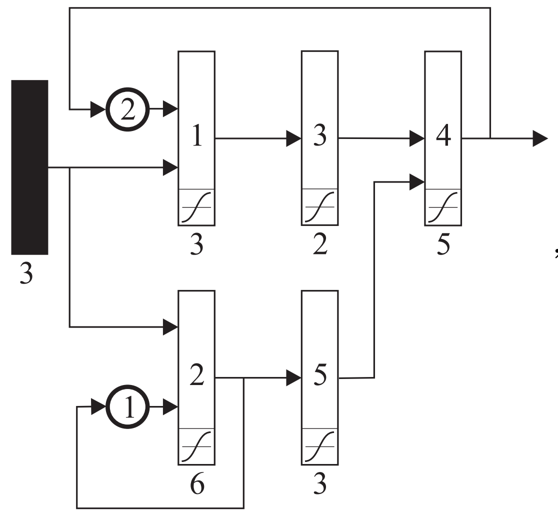
ii. Select a simulation order and write out the equations needed to determine the network response.

iii. Which $S^{u,x}(t)$ will need to be calculated (i.e., for which u and which x)?

iv. Write out Eq. (14.63) specifically for the term $\partial F/\partial \mathbf{a}^3(t)$ and Eq. (14.34) specifically for the term $\partial \mathbf{a}^3(t)/\partial \mathbf{x}^T$. (Expand the summation to show exactly which terms are included.)

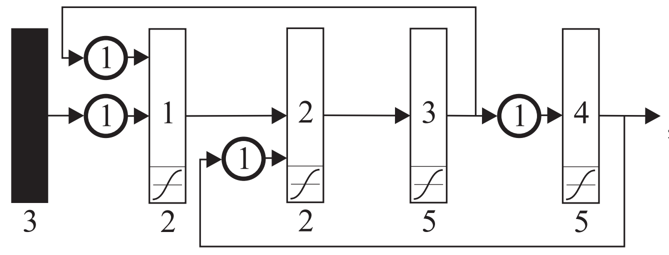
E14.11 Repeat E14.10 for the following networks, except, in part iv., change $\mathbf{a}^3$ to the indicated layer.

i.



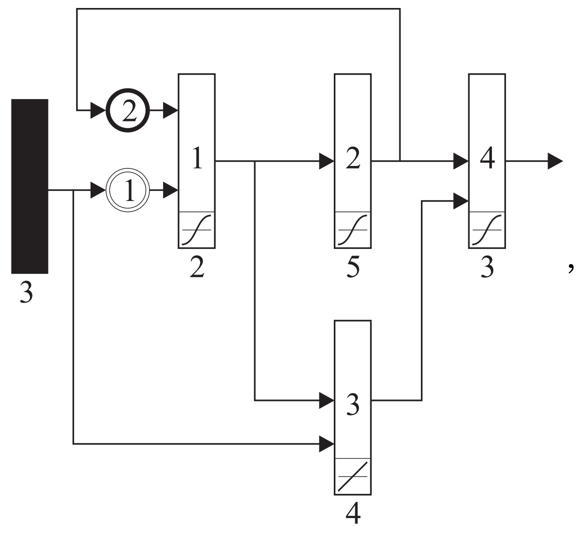
, $\mathbf{a}^4$.

ii.

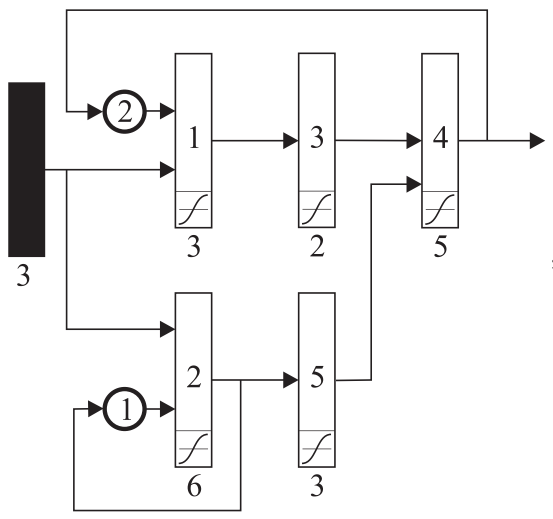


, $\mathbf{a}^4$.

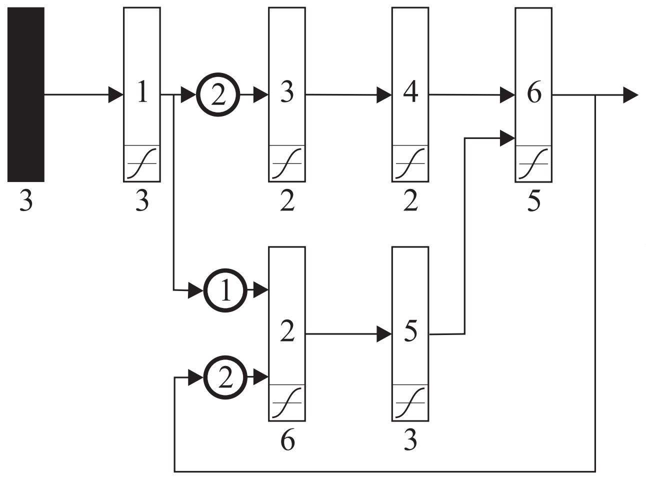
iii.



, $\mathbf{a}^2$.

iv.


, $\mathbf{a}^2$.

v.


, $\mathbf{a}^1$.

E14.12 One of the advantages of the RTRL algorithm is that the gradient can be computed at the same time as the network response, since it is computed by starting at the first time point, and then working forward through time. Therefore, RTRL allows a convenient framework for on-line implementation. Suppose that you are implementing the RTRL algorithm, and you update the weights of the network at each time step.

i. Discuss the accuracy of the gradient calculation if the weights are updated at each time step of a steepest descent algorithm.

ii. Implement the RTRL algorithm in MATLAB for the network in Figure 14.4. Generate training data by using the network with the two weights set equal to 0.5. Use the same input sequence shown in Figure 14.10, and use the network responses $a(t)$ as the targets. Using this data set, train the network using the steepest descent algorithm, with a learning rate of $\alpha = 0.1$. Set the initial weights to zero. Update the weights at each time step. Save the gradient at the end of the eighth time step.

iii. Repeat part ii., but do not update the weights. Compute the gradient at the end of the eighth time step. Compare this gradient with the one you obtained in part ii. Explain any differences

E14.13 Consider again the recurrent network of Figure 14.4. Assume that $F(\mathbf{x})$ is the sum squared error for the first two time points.

i. Let $a(0) = 0$. Find $a(1)$ and $a(2)$ as a functions of $p(1)$, $p(2)$, and the weights in the network.

ii. Find the sum squared error for the first two time steps as an explicit function of $p(1)$, $p(2)$, $t(1)$, $t(2)$, and the network weights.

iii. Using part (ii), find $\frac{\partial F}{\partial lw_{1,1}(1)}$.

iv. Compare the results of part (iii) with the results determined by RTRL on page 14-22 and by BPTT on page 14-30.

E14.14 In the process of deriving the RTRL algorithm in Exercise E14.5, you should have produced the following expression

$$\frac{\partial^e \mathbf{a}^3(t)}{\partial vec(\mathbf{LW}^{2,1}(1))^T} = [\mathbf{a}^1(t-1)]^T \otimes \mathbf{S}^{3,2}(t)$$

If

$$\mathbf{a}^1(1) = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \text{ and } \mathbf{S}^{3,2}(2) = \begin{bmatrix} 2 & 3 \\ 4 & -5 \end{bmatrix},$$

Find $\frac{\partial^e \mathbf{a}^3(2)}{\partial vec(\mathbf{LW}^{2,1}(1))^T}$ and indicate $\frac{\partial^e a_1^3(2)}{\partial vec(lw_{1,2}^{2,1}(1))^T}$.

E14.15 Each layer of a standard LDDN network has a summing junction, which combines contributions from inputs, other layers, and the bias, as in Eq. (14.1), which is repeated here:

$$\mathbf{n}^m(t) = \sum_{l \in L_m^f} \sum_{d \in DL_{m,l}} \mathbf{LW}^{m,l}(d)\mathbf{a}^l(t-d) + \sum_{l \in I_m} \sum_{d \in DI_{m,l}} \mathbf{IW}^{m,l}(d)\mathbf{p}^l(t-d) + \mathbf{b}^m.$$

If, instead of *summing*, the net input was computed as a *product* of the contributions, how would the RTRL and BPTT algorithms change?

E14.16 As discussed in Exercise E14.15, the contribution to the net input from other layers is computed as product of a layer matrix with a layer output, as in

$$\mathbf{LW}^{m,l}(d)\mathbf{a}^l(t-d).$$

If, instead of multiplying the layer matrix times the layer output, we were to compute the distance between each row of the weight matrix and the layer output, as in

$$n_i = -\| {}_i\mathbf{w} - \mathbf{a} \| .$$

How would the RTRL and BPTT algorithms change?

E14.17 Find and compare the computational complexity for the BPTT and RTRL algorithms applied to the sample network in Figure E14.6 as a function of the number of neurons in Layer 2 (S^2), the number of delays in the tapped delay line (D) and the length of the training sequence (Q).

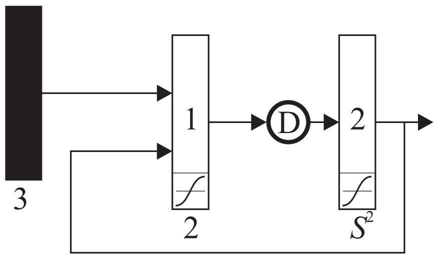


Figure E14.6 Recurrent Network for Exercise E14.17

E14.18 Consider again the network of Figure 14.4. Let the input weight of the network $iw_{1,1} = 1$. Assume that the initial network output is $a(0) = 0$.

i. Write the network output at time t as a function only of the layer weight $lw_{1,1}(1)$, and the input sequence. (The result should be a polynomial in $lw_{1,1}(1)$.)

```
» 2 + 2
ans =
     4
```

ii. Find the network output at time $t = 8$, using $lw_{1,1}(1) = -1.4$ and the following input sequence:

$$p(t) = \{3, 1, 1, 6, 3, 5, 1, 6\} .$$

iii. With $lw_{1,1}(1) = -1.4$, the network should be unstable, since this weight in the feedback loop is greater than one in magnitude. The output would generally grow with time for an unstable network. (This applies to linear networks.) In ii., you should have found a small value for $a(8)$. Can you explain this result? (Hint: Investigate the roots of the polynomial you found in part i. You can use the MATLAB command `roots`.) How might this result be related to the spurious valleys in the error surface discussed on page 14-32?