

- 12.12 The Q -learning algorithm is sometimes referred to as an adaptive form of the value iteration policy. Justify the validity of this description.
- 12.13 Construct a signal-flow graph for the approximate Q -learning algorithm summarized in Table 12.4.
- 12.14 The approximate Q -learning algorithm summarized in Table 12.4 assumes lack of knowledge of the state-transition probabilities. Reformulate this algorithm assuming the availability of these probabilities.

CHAPTER 13

Temporal Processing Using Feedforward Networks

13.1 INTRODUCTION

Time constitutes an essential ingredient of the learning process. It can be continuous or discrete. Whatever its form, time is an ordered entity that is basic to many of the cognitive tasks encountered in practice such as vision, speech, signal processing, and motor control. It is through the incorporation of time into the operation of a neural network that it is enabled to follow statistical variations in nonstationary processes such as speech signals, radar signals, signals picked up from the engine of an automobile, and fluctuations in stock market prices, just to mention a few. The question is: How do we build time into the operation of a neural network? The answer to this fundamental question lies in one of two possible ways:

- *Implicit representation.* Time is represented by the effect it has on signal processing in an implicit manner.¹ For example, the input signal is *uniformly sampled*, and the sequence of synaptic weights of each neuron connected to the input layer of the network is *convolved* with a different sequence of input samples. In so doing, the temporal structure of the input signal is embedded in the spatial structure of the network.
- *Explicit representation.* Time is given its own particular representation.² For example, the echo-location system of a bat operates by emitting a short frequency modulated (FM) signal, so that the same intensity level is maintained for each frequency channel restricted to a very short period within the FM sweep. Multiple comparisons between several different frequencies encoded by an array of auditory receptors are made for the purpose of extracting accurate distance (range) information about a target (Suga and Kanwal, 1995). When an echo is received from the target with an unknown delay, a neuron (in the auditory system) with a matching delay line responds, thereby providing an estimate of the range to the target.

In this chapter we are concerned with the implicit representation of time, whereby a “static” neural network (e.g., multilayer perceptron) is provided with *dynamic* properties. This, in turn, makes the network responsive to the temporal structure of information-bearing signals.

For a neural network to be dynamic, it must be given *memory*. As pointed out in Chapter 2, memory may be divided into “short-term” and “long-term” memory, depending on the retention time. Long-term memory is built into a neural network through supervised learning, whereby the information content of the training data set is stored (in part or in full) in the synaptic weights of the network. However, if the task at hand has a temporal dimension, we need some form of short-term memory to make the network dynamic. One simple way of building short-term memory into the structure of a neural network is through the use of *time delays*, which can be implemented at the synaptic level inside the network or at the input layer of the network. The use of time delays in neural networks is neurobiologically motivated, since it is well known that signal delays are omnipresent in the brain and play an important role in neurobiological information processing (Braitenberg, 1967, 1977, 1986; Miller, 1987).

Organization of the Chapter

The material in this chapter is organized in three parts. The first part, consisting of Sections 13.2 and 13.3, deals with network structures and models. In Section 13.2 we present a discussion of memory structures, followed by Section 13.3 on a description of two different network architectures for temporal processing of signals.

The second part of the chapter, consisting of Sections 13.4 to 13.6, deals with a class of neural networks known as focused time lagged feedforward networks; the term “focused” refers to the fact that the short-term memory is located entirely at the front end of the network. A computer experiment on this structure is described in Section 13.6.

The third part of the chapter, consisting of Sections 13.7 to 13.9, deals with distributed time lagged feedforward networks, where delay lines are distributed throughout the network. Section 13.7 describes spatio-temporal models of a neuron, followed by a discussion in Section 13.8 on the second class of neural networks just mentioned. In Section 13.9 we describe the “temporal” back-propagation algorithm for the supervised training of distributed time lagged feedforward networks.

The chapter concludes with some final remarks in Section 13.10.

SHORT-TERM MEMORY STRUCTURES

The primary role of memory is to *transform a static network into a dynamic one*. In particular, by embedding memory into the structure of a static network such as an ordinary multilayer perceptron, the output of the network becomes a function of time. This approach for building a nonlinear dynamical system is straightforward because it provides for a clear separation of responsibilities: The static network accounts for nonlinearity, and the memory accounts for time.

Short-term memory³ can be implemented in continuous time or in discrete time. Continuous time is denoted by t , and discrete time is denoted by n . The resistance-

capacitance circuit in Fig. 13.1 is an example of continuous-time memory, which is characterized by an impulse response (i.e., memory trace) $h(t)$ that decays exponentially with time t . This circuit is responsible for memory, at the synaptic level, in an analog implementation of the additive model of a neuron to be described later in the chapter. In this section we are concerned mainly with discrete-time memory.

A useful tool for dealing with discrete-time systems is the *z-transform*. Let $\{x(n)\}$ denote a discrete-time sequence, which may extend into the infinite past. The *z-transform* of this sequence, denoted by $X(z)$, is defined by

$$X(z) = \sum_{n=-\infty}^{\infty} x(n)z^{-n} \quad (13.1)$$

where z^{-1} is the *unit delay operator*; that is, z^{-1} operating on $x(n)$ yields its delayed version $x(n-1)$. Suppose $x(n)$ is applied to a discrete-time system of impulse response $h(n)$. The output of the system, $y(n)$, is defined by the *convolution sum*

$$y(n) = \sum_{k=-\infty}^{\infty} h(k)x(n-k) \quad (13.2)$$

For $x(n)$ equal to the unit impulse, $y(n)$ reduces to the *impulse response* $h(n)$ of the system. An important property of the *z-transform* is that *convolution in the time domain is transformed into multiplication in the z-domain* (Oppenheim and Schaffer, 1989; Haykin and Van Veen, 1998). If we denote the *z-transform* of the sequences $\{h(n)\}$ and $\{y(n)\}$ by $H(z)$ and $Y(z)$, respectively, application of the *z-transform* to Eq. (13.2) yields

$$Y(z) = H(z)X(z) \quad (13.3)$$

or equivalently

$$H(z) = \frac{Y(z)}{X(z)} \quad (13.4)$$

The function $H(z)$ is called the *transfer function* of the system.

Figure 13.2 shows a block diagram of a discrete-time memory consisting of p identical sections connected in cascade; hereafter p is referred to as the *order* of the

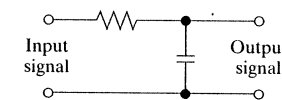


FIGURE 13.1 Resistance-capacitance circuit.

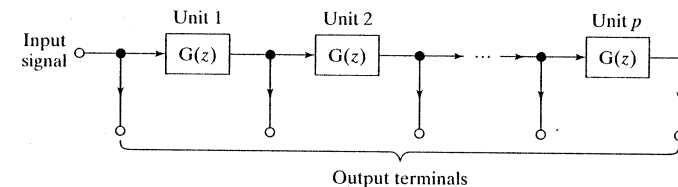


FIGURE 13.2 Generalized tapped delay line memory of order p .

memory. Each delay section, viewed as an operator, is characterized by a transfer function $G(z)$, as indicated in the figure. Equivalently, each section may be described in terms of impulse response $g(n)$, which has the following two properties:

- It is *causal*, that is, $g(n) = 0$ for $n < 0$.
- It is *normalized*, which means that $\sum_{n=0}^{\infty} |g(n)| = 1$.

Henceforth $g(n)$ is referred to as the *generating kernel* of the discrete-time memory.

In light of Fig. 13.2, we may formally define a *discrete-time memory* as a linear time invariant, single input–multiple output (SIMO) system whose generating kernel satisfies these properties. The junction points, to which the output terminals of the memory are connected, are commonly called *taps*. Note that for a memory of order p , there are $p + 1$ taps with one tap belonging to the input.

The attributes of a memory structure are measured in terms of depth and resolution. Let $g_p(n)$ denote the overall impulse response of the memory, defined as p successive convolutions of $g(n)$, or equivalently, the inverse z -transform of $G^p(z)$. *Memory depth*, denoted by D , is defined as the first time moment of $g_p(n)$, as shown by

$$D = \sum_{n=0}^{\infty} n g_p(n) \quad (13.5)$$

A memory of low depth D only holds its information content for a relatively short period of time, whereas a high depth memory holds its information content much further into the past. *Memory resolution*, denoted by R , is defined as the number of taps in the memory structure per unit time. A memory of high resolution R is able to hold information about the input sequence at a fine level, whereas a low resolution memory can only do so at a much coarser level. For a fixed number of taps, the product of memory depth and memory resolution is a constant equal to the memory order p .

Different choices of the generating kernel $g_p(n)$ naturally result in different values for the depth D and resolution R , as illustrated in the following two memory structures.

Tapped delay line memory. Figure 13.3 shows the block diagram of the simplest and most commonly used form of short-term memory called a *tapped delay line memory*. It consists of p unit delay operators, each of which is characterized by $G(z) = z^{-1}$. That is, the generating kernel is $g(n) = \delta(n - 1)$, where $\delta(n)$ is the unit impulse:

$$\delta(n) = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases} \quad (13.6)$$

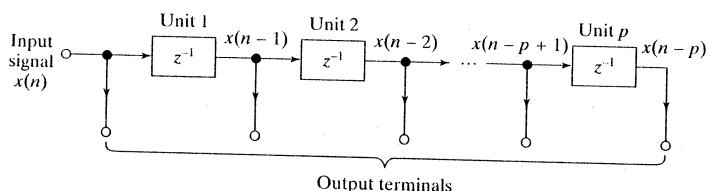


FIGURE 13.3 Ordinary tapped delay line memory of order p .

The overall impulse response of the tapped delay line in Fig. 13.3 is $g_p(n) = \delta(n - p)$. Substituting this $g_p(n)$ in Eq. (13.5) yields the memory depth $D = p$, which is intuitively satisfying. From Fig. 13.3 we see that there is only one tap per unit time; hence, $R = 1$. Thus the memory depth of a tapped delay line increases linearly with its order p , but its memory depth is fixed at unity; the depth-resolution product is constant at p .

We need an additional degree of freedom in order to exercise control over memory depth. Such a provision is made possible by an alternative to the tapped delay line considered next.

Gamma memory. Figure 13.4 shows the signal-flow graph of the basic functional block $G(z)$ used in a memory structure called the *gamma memory* (deVries and Principe, 1992). Specifically, each section of this memory structure consists of a feedback loop with unit delay z^{-1} and adjustable parameter μ . The transfer function of each such section is

$$\begin{aligned} G(z) &= \frac{\mu z^{-1}}{1 - (1 - \mu)z^{-1}} \\ &= \frac{\mu}{z - (1 - \mu)} \end{aligned} \quad (13.7)$$

For stability, the only pole of $G(z)$ at $z = 1 - \mu$ must lie inside the unit circle in the z -plane. This in turn requires that

$$0 < \mu < 2 \quad (13.8)$$

The generating kernel of the gamma memory is the inverse z -transform of $G(z)$, that is,

$$g(n) = \mu(1 - \mu)^{n-1}, \quad n \geq 1 \quad (13.9)$$

The condition of Eq. (13.8) ensures that $g(n)$ decays exponentially to zero as n approaches infinity.

The overall impulse response of the gamma memory is the inverse z -transform of the overall transfer function

$$G_p(z) = \left(\frac{\mu}{z - (1 - \mu)} \right)^p$$

That is,

$$g_p(n) = \binom{n-1}{p-1} \mu^p (1 - \mu)^{n-p}, \quad n \geq p \quad (13.10)$$

where $\binom{n-1}{p-1}$ is a binomial coefficient defined by $\binom{n-1}{p-1} = \frac{(n-1) \cdots (n-p+1)}{p!}$ for integer values of n and p . The overall impulse response $g_p(n)$ for varying p represents a discrete version of the integrand of the *gamma function* (deVries and Principe, 1992), hence the

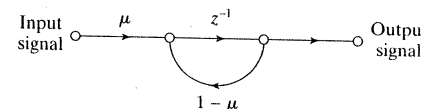


FIGURE 13.4 Signal-flow graph for one section of gamma memory.

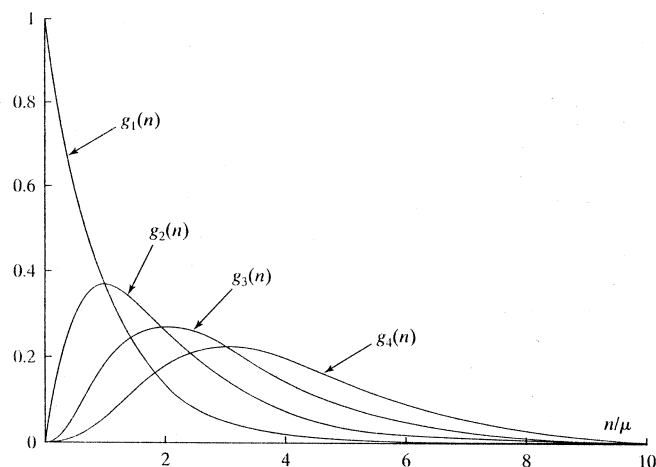


FIGURE 13.5 Family of impulse responses for the gamma memory for order $p = 1, 2, 3, 4$ and $\mu = 0.7$.

name of the memory. Figure 13.5 shows a family of impulse responses $g_p(n)$, normalized with respect to μ , for $p = 1, 2, 3, 4$ and $\mu = 0.7$. Note that the time axis in Fig. 13.5 is scaled by the parameter μ . This scaling has the effect of positioning the peak value of $g_p(n)$ at $n = p$.

The depth of the gamma memory is p/μ and its resolution is μ , for a depth-resolution product of p . Accordingly, by choosing μ to be less than unity, the gamma memory provides improvement in depth (but sacrifices resolution) over the tapped delay line for a specified order p . When $\mu = 1$, these quantities reduce to the respective values assumed by the tapped delay line. Thus the gamma memory includes the tapped delay line as a special case. This observation is also readily ascertained by setting $\mu = 1$ in Eq. (13.9). If μ is greater than 1 but less than 2, then $(1 - \mu)$ in this equation becomes negative but with an absolute value less than 1.

13.3 NETWORK ARCHITECTURES FOR TEMPORAL PROCESSING

Network architectures for temporal processing take more than one form, just as memory structures do. In this section we will describe two feedforward network architectures that have enriched the literature on temporal processing in their own individual ways.

NETalk

NETalk, devised by Sejnowski and Rosenberg (1987), was the first demonstration of a massively parallel distributed network that converts English speech to phonemes; a *phoneme* is a basic linguistic unit. Figure 13.6 shows a schematic diagram of the NETalk system, which is based on a multilayer perceptron with an input layer of 203 sensory nodes, a hidden layer of 80 neurons, and an output layer of 26 neurons. All the

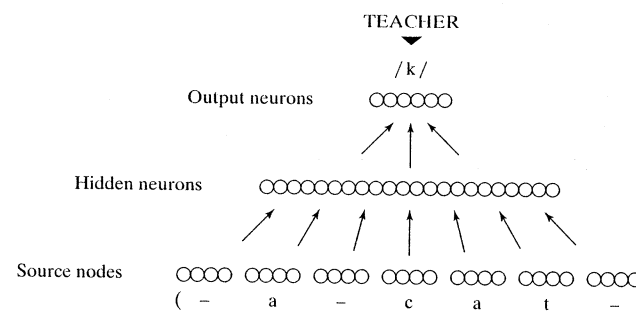


FIGURE 13.6 Schematic diagram of the NETalk network architecture.

neurons used sigmoid (logistic) activation functions. The synaptic connections in the network were specified by a total of 18,629 weights, including a variable threshold for each neuron; threshold is the negative of bias. The standard back-propagation algorithm was used to train the network.

The network had seven groups of nodes in the input layer, with each group encoding one letter of the input text. Strings of seven letters were thus presented to the input layer at any one time. The desired response for the training process was specified as the correct phoneme associated with the center (i.e., fourth) letter in the seven-letter window. The other six letters (three on either side of the center letter) provided a partial *context* for each decision made by the network. The text was stepped through the window on a letter-by-letter basis. At each step in the process, the network computed a phoneme, and after each word the synaptic weights of the network were adjusted according to how closely the computed pronunciation matched the correct one.

The performance of NETalk exhibited some similarities with observed human performance, as summarized here (Sejnowski and Rosenberg, 1987).

- The training followed a power law.
- The more words the network learned, the better it was at generalizing and correctly pronouncing new words.
- The performance of the network degraded very slowly as synaptic connections in the network were damaged.
- Relearning after damage to the network was much faster than learning during the original training.

NETalk was a brilliant illustration in miniature of many aspects of learning, starting out with considerable “innate” knowledge of its input patterns and then gradually acquiring competence at converting English speech to phonemes through practice. However, it did not lead to practical applications.

Time-delay Neural Network

A popular neural network that uses ordinary time delays to perform temporal processing is the so-called *time delay neural network* (TDNN), which was first described in Lang and Hinton (1988) and Waibel et al. (1989). The TDNN is a multilayer feedforward

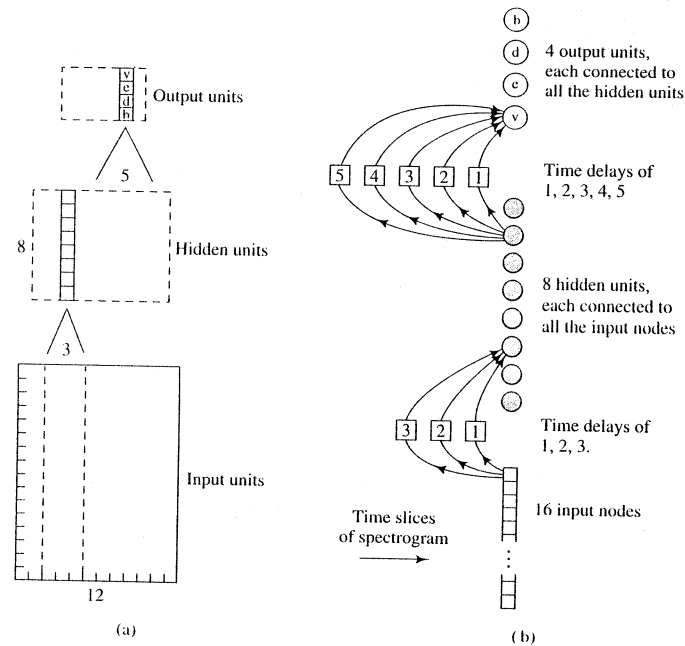


FIGURE 13.7 (a) A network whose hidden neurons and output neurons are replicated across time. (b) Time delay neural network (TDNN) representation. (From K. J. Lang and G. E. Hinton, 1988, with permission)

network whose hidden neurons and output neurons are *replicated across time*. It was devised to capture explicitly the concept of time symmetry as encountered in the recognition of an isolated word (phoneme) using a spectrogram. A *spectrogram* is a two-dimensional image in which the vertical dimension corresponds to frequency and the horizontal dimension corresponds to time; the intensity (darkness) of the image corresponds to signal energy (Rabiner and Schafer, 1978). Figure 13.7a illustrates a single hidden layer version of the TDNN (Lang and Hinton, 1988). The input layer consists of 192 (16 by 12) sensory nodes encoding the spectrogram; the hidden layer contains 10 copies of 8 hidden neurons; and the output layer contains 6 copies of 4 output neurons. The various replicas of a hidden neuron apply the same set of synaptic weights to narrow (three-time-step) windows of the spectrogram; similarly, the various replicas of an output neuron apply the same set of synaptic weights to narrow (five-time-step) windows of the pseudospectrogram computed by the hidden layer. Figure 13.7b presents a *time delay* interpretation of the replicated neural network of Fig. 13.7a—hence the name “time delay neural network”. This network has a total of 544 synaptic weights. Lang and Hinton (1988) used the TDNN for the recognition of four isolated words: “bee”, “dee”, “ee”, and “vee”, which accounts for the use of four output neurons

in Fig. 13.7. A recognition score of 93 percent was obtained on test data different from the training data. In a more elaborate study reported by Waibel et al. (1989), a TDNN with two hidden layers was used for the recognition of three isolated words: “bee”, “dee”, and “gee”. In performance evaluation involving the use of test data from three speakers, the TDNN achieved an average recognition score of 98.5 percent.

The TDNN appears to work best for classifying a temporal pattern that consists of a sequence of fixed dimensional feature vectors such as phonemes. In a practical speech recognizer, however, it is unrealistic to assume that the speech signal can be accurately segmented into its constituent phonemes. Rather, it is essential to adequately model the super-segmented temporal structure of speech patterns. In particular, the speech recognizer has to deal with word and sentence segments that vary significantly in their duration and nonlinear temporal structure. To model these natural characteristics of speech signals, the traditional approach in the speech recognition field has been to use a state transition structure like the hidden Markov model (Rabiner, 1989; Jelinek, 1997). Basically, a *hidden Markov model* (HMM) represents a stochastic process generated by an underlying Markov chain, and a set of observation distributions associated with its hidden states; see note 11 in Chapter 11. Many hybrids of TDNN and HMM have been studied in the literature.⁴

13.4 FOCUSED TIME LAGGED FEEDFORWARD NETWORKS

The prototypical use of a static neural network (e.g., multilayer perceptron, radial-basis function network) is in *structural pattern recognition*. In contrast, *temporal pattern recognition* requires processing of patterns that evolve over time, with the response at a particular instant of time depending not only on the present value of the input but also on its past values. Figure 13.8 shows the block diagram of a *nonlinear filter* built on a static neural network (Mozier, 1994). The network is stimulated through a short-term memory. Specifically, given an input signal consisting of the present value $x(n)$ and the p past values $x(n-1), \dots, x(n-p)$ stored in a delay line memory of order p , for example, the free parameters of the neural network are adjusted to minimize the mean-square error between the output of the network, $y(n)$, and the desired response $d(n)$.

The structure of Fig. 13.8 can be implemented at the level of a single neuron or a network of neurons. These two cases are illustrated in Figs. 13.9 and 13.10 respectively.

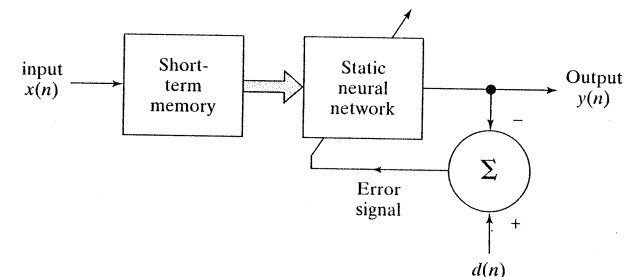


FIGURE 13.8 Nonlinear filter built on a static neural network.

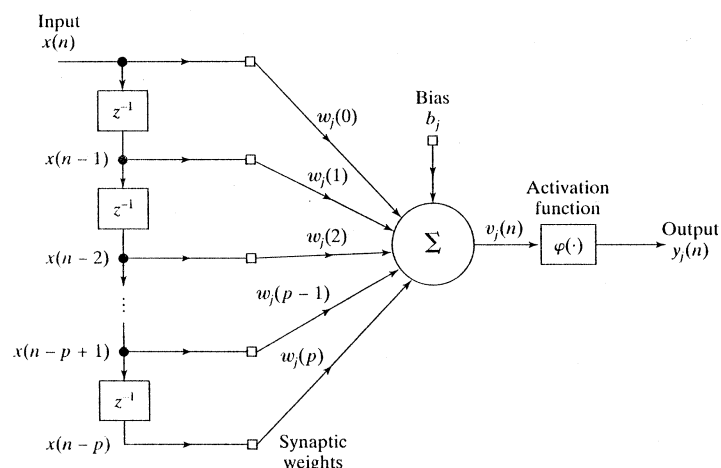


FIGURE 13.9 Focused neuronal filter.

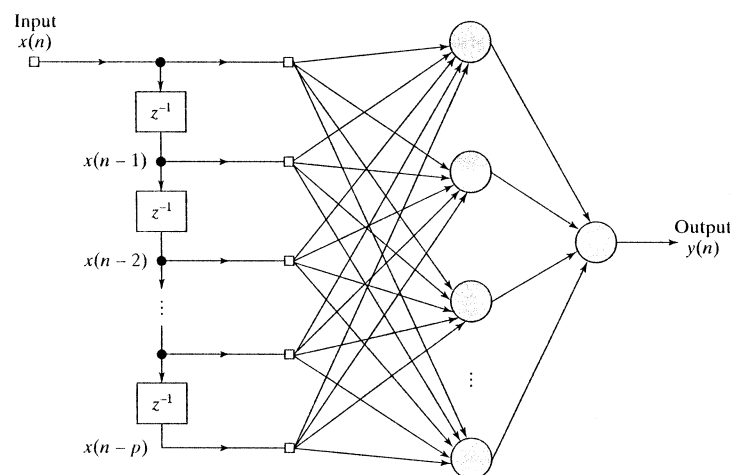


FIGURE 13.10 Focused time lagged feedforward network (TLFN); the bias levels have been omitted for convenience of presentation.

To simplify the presentation, we have used a tapped delay line memory as the short-term memory structure in Figs. 13.9 and 13.10. Clearly, both of these figures could be generalized by using a unit with transfer function $G(z)$ in place of z^{-1} .

The temporal processing unit of Fig. 13.9 is composed of a tapped delay line memory with its taps connected to the synapses of a neuron. The tapped delay line memory captures temporal information contained in the input signal and the neuron embeds that information in its own synaptic weights. The processing unit of Fig. 13.9 is

called a *focused neuronal filter*, focused in the sense that the entire memory structure is located at the input end of the unit. The output of the filter, in response to the input $x(n)$ and its past values $x(n-1), \dots, x(n-p)$, is given by

$$y_j(n) = \varphi \left(\sum_{l=0}^p w_j(l)x(n-l) + b_j \right) \quad (13.11)$$

where $\varphi(\cdot)$ is the activation function of neuron j , the $w_j(l)$ are its synaptic weights, and b_j is the bias. Note that the input to the activation function consists of a bias plus the *convolution* of sequences of input samples and synaptic weights of the neuron.

Turning next to Fig. 13.10, referred to as a *focused time lagged feedforward network* (TLFN), here we have a more powerful nonlinear filter consisting of a tapped delay line memory of order p and a multilayer perceptron. To train the filter, we may use the standard back-propagation algorithm described in Chapter 4. At time n , the “temporal pattern” applied to the input layer of the network is the signal vector

$$x(n) = [x(n), x(n-1), \dots, x(n-p)]^T$$

which may be viewed as a description of the *state* of the nonlinear filter at time n . An epoch consists of a sequence of states (patterns), the number of which is determined by the memory order p and the size N of the training sample.

The output of the nonlinear filter, assuming that the multilayer perceptron has a single hidden layer as shown in Fig. 13.10, is given by

$$\begin{aligned} y(n) &= \sum_{j=1}^{m_1} w_j y_j(n) \\ &= \sum_{j=1}^{m_1} w_j \varphi \left(\sum_{l=0}^p w_j(l)x(n-l) + b_j \right) + b_o \end{aligned} \quad (13.12)$$

where the output neuron in the focused TLFN is assumed to be linear; the synaptic weights of the output neuron are denoted by the set $\{w_j\}_{j=1}^{m_1}$, where m_1 is the size of the hidden layer, and the bias is denoted by b_o .

13.5 COMPUTER EXPERIMENT

In this computer experiment, we investigate the use of the focused TLFN in Fig. 13.10 to simulate a time series representing a difficult frequency modulated signal:

$$x(n) = \sin(n + \sin(n^2)), \quad n = 0, 1, 2, \dots$$

The network was used as a *one-step predictor* with $x(n+1)$ providing the desired response for an input consisting of the set $\{x(n-l)\}_{l=0}^p$. The composition of the network and its parameters were as follows:

Order of tapped delay line memory, p :	20
Hidden layer, m_1 :	10 neurons
Activation function of hidden neurons:	logistic
Output layer:	1 neuron
Activation function of output neuron:	linear
Learning-rate parameter (both layers):	0.01
Momentum constant:	none

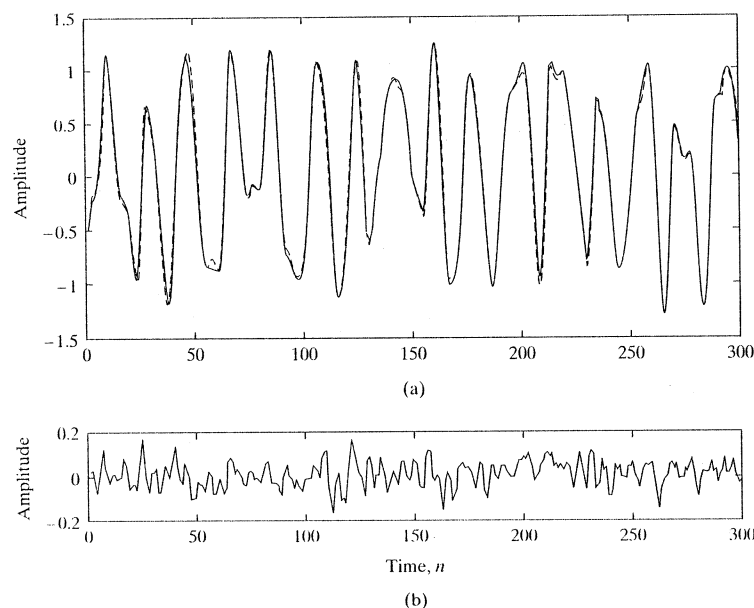


FIGURE 13.11 Results of computer experiment on one-step prediction. (a) Superposition of actual (continuous) and predicted (dashed) waveforms. (b) Waveform of prediction error.

The data set used to train the network consisted of 500 randomized patterns, with each pattern consisting of 20 time-ordered samples selected from the time series $\{x(n)\}$.

Figure 13.11a shows a superposition of the one-step prediction performed by the network on test data (not seen before) and the actual waveform. Figure 13.11b displays the waveform of the prediction error defined as the difference between the actual and predicted waveforms. The mean-square value of the prediction error is 1.2×10^{-3} .

13.6 UNIVERSAL MYOPIC MAPPING THEOREM

The nonlinear filter of Fig. 13.9 may be generalized as shown in Fig. 13.12. This generic dynamic structure consists of two functional blocks. The block labeled $\{h_i\}_{i=1}^L$ represents *multiple convolutions* in the time domain, that is, a bank of *linear filters* operating in parallel. The h_i are drawn from a large set of real-valued kernels, each one of which represents the impulse response of a linear filter. The block labeled $\mathcal{N}$ represents a static (i.e., memoryless) nonlinear feedforward network such as an ordinary multilayer perceptron. The structure of Fig. 13.12 is a *universal dynamic mapper*. In Sandberg and Xu (1997a) it is shown that any shift-invariant *myopic map* can be uniformly approximated arbitrarily well by a structure of the form depicted in Fig. 13.12 under mild conditions. The requirement that a map be myopic is equivalent to “uniform fading

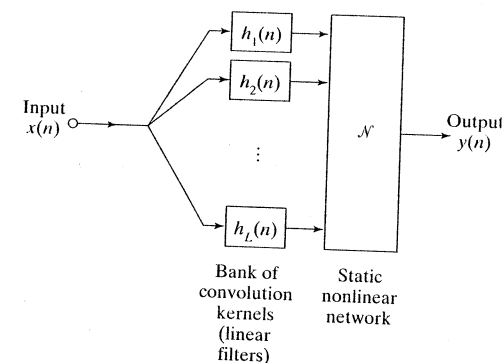


FIGURE 13.12 Generic structure for universal myopic mapping theorem.

memory”; it is assumed here that the map is *causal*, which means that an output signal is produced by the map at time $n \geq 0$ only when the input signal is applied at time $n = 0$. By “shift invariant” we mean: If $y(n)$ is the output of the map due to an input $x(n)$, then the output of the map due to the shifted input $x(n - n_0)$ is $y(n - n_0)$ where the time shift n_0 is an integer. In Sandberg and Xu (1997b), it is further shown that for any single-variable, shift-invariant, causal, uniformly-fading-memory map, there is a gamma memory and static neural network, the combination of which approximates the map uniformly and arbitrarily well.

We may now formally state the *universal myopic mapping theorem*⁵ as follows (Sandberg and Xu, 1997a, 1997b):

Any shift-invariant myopic dynamic map can be uniformly approximated arbitrarily well by a structure consisting of two functional blocks: a bank of linear filters feeding a static neural network.

The structure embodied in this theorem may take the form of a focused TLFN. It is also noteworthy that this theorem holds when the input and output signals are functions of a finite number of variables as in image processing, for example.

The universal myopic mapping theorem has profound practical implications. Not only does it provide mathematical justification for NETalk and its possible gamma memory extension, but it also lays down the framework for the design of more elaborate models of nonlinear dynamical processes. The multiple convolutions at the front end of the structure in Fig. 13.12 may be implemented using linear filters with a finite-duration impulse response (FIR) or infinite-duration impulse response (IIR). As for the static neural network, it can be implemented using a multilayer perceptron, radial-basis function network, or support vector machine trained by means of the algorithms described in Chapters 4, 5, and 6. In other words, we may naturally build on the material presented in those chapters on supervised learning to build nonlinear filters or models of nonlinear dynamic processes. Most importantly, the structure of Fig. 13.12 is *inherently stable*, provided that the linear filters are themselves stable. We thus have a clear-cut separation of roles as to how to take care of short-term memory and memoryless nonlinearity.

13.7 SPATIO-TEMPORAL MODELS OF A NEURON

The focused neuronal filter in Fig. 13.9 has an interesting interpretation as described here. The combination of unit delay elements and associated synaptic weights may be viewed as a *finite-duration impulse response (FIR) filter* of order p , as shown in Fig. 13.13a; the FIR filter is one of the basic building blocks in digital signal processing (Oppenheim and Schaffer, 1989; Haykin and Van Veen, 1998). Accordingly, the focused neuronal filter of Fig. 13.9 is, in fact, a nonlinear FIR filter, as shown in Fig. 13.13b. We may build on this representation and thereby extend the processing power of the neuron in a spatial sense through the use of multiple inputs, m_0 in number, as depicted in Fig. 13.14. The spatio-temporal model of Fig. 13.14 is referred to as a *multiple input neuronal filter*.

Yet another way of describing the model in Fig. 13.14 is to think of it as a *distributed neuronal filter*, in the sense that the filtering action is distributed across different points in space. The spatio-temporal characterization of the model is represented as follows:

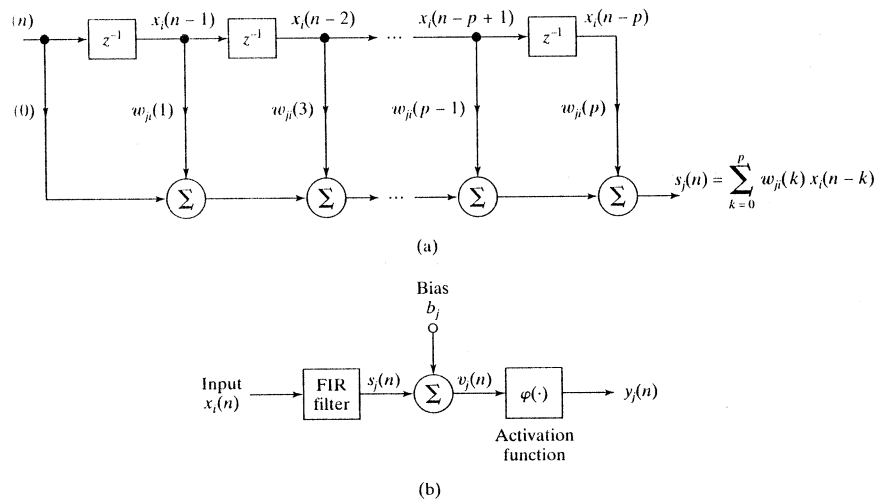


FIGURE 13.13 (a) Finite-duration impulse response (FIR) filter. (b) Interpretation of neuronal filter as a nonlinear FIR filter.

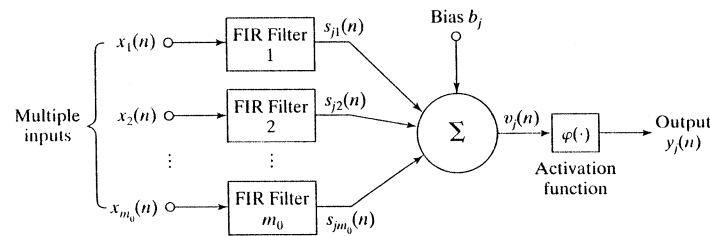


FIGURE 13.14 Multiple input neuronal filter.

- The neuron has m_0 “primary” synapses, each of which consists of a linear discrete-time filter implemented in the form of an FIR filter of order p ; the primary synapses account for the spatial dimension of signal processing.
- Each primary synapse has $(p + 1)$ “secondary” synapses that are connected to its respective input and the memory taps of its FIR filter, thereby accounting for the temporal dimension of signal processing.

The synaptic structure of the neuronal filter in Fig. 13.14 is thus tree-like, as depicted in Fig. 13.15. The total number of synaptic weights in the structure is $m_0(p + 1)$.

In mathematical terms, we may describe the spatio-temporal processing performed by the neuronal filter in Fig. 13.14 by expressing its output, $y_j(n)$, as

$$y_j(n) = \varphi \left(\sum_{i=1}^{m_0} \sum_{l=0}^p w_{ji}(l) x_i(n-l) + b_j \right) \quad (13.13)$$

where $w_{ji}(l)$ is the weight of the l th secondary synapse belonging to the i th primary synapse, $x_i(n)$ is the input applied to the i th primary synapse at time n , and b_j is the bias applied to the neuron. The induced local field $v_j(n)$ of the neuron, that is, the argument of the activation function φ in Eq. (13.13) may be viewed as the discrete time “approximation” to the continuous-time formula

$$v_j(t) = \sum_{i=1}^{m_0} \int_{-\infty}^t h_{ji}(\lambda) x_i(t-\lambda) d\lambda + b_j \quad (13.14)$$

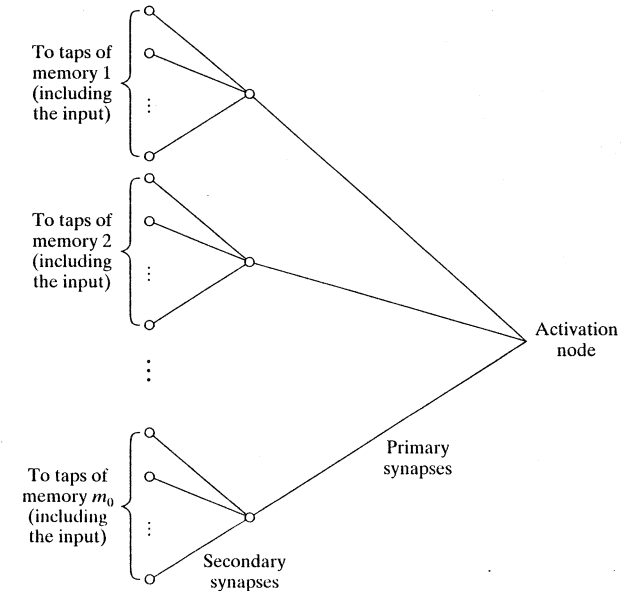


FIGURE 13.15 Tree-like description of the synaptic structure of a multiple input neuronal filter.

The integral in Eq. (13.14) is the *convolution* of the continuous-time input signal $x_i(t)$ and the impulse response $h_{ji}(t)$ characterizing a linear continuous-time filter representing synapse i . Equation (13.14) is the most general way in which the spatio-temporal behavior of the induced local field of a neuron can be described.

Additive Model

Equation (13.14) provides the basis for another commonly used spatio-temporal model of a neuron. Specifically, suppose we simplify the temporal behavior of the neuron by using a scaling parameter to determine the sign and strength of a "typical" synaptic impulse response, in which case we write

$$h_{ji}(t) = w_{ji} \cdot h_j(t) \quad \text{for all } i \quad (13.15)$$

where $h_j(t)$ models the temporal characteristics of a typical postsynaptic potential, and w_{ji} is a scalar that determines its sign (excitatory or inhibitory) and the overall strength of the connection between neuron j and input i (Shamma, 1989). Thus using Eq. (13.15) in (13.14) and by interchanging the order of integration and summation, we obtain

$$\begin{aligned} v_j(t) &= \int_{-\infty}^t h_j(\lambda) \left(\sum_{i=1}^{m_0} w_{ji} x_i(t - \lambda) \right) d\lambda + b_j \\ &= h_j(t) * \left(\sum_{i=1}^{m_0} w_{ji} x_i(t) \right) + b_j \end{aligned} \quad (13.16)$$

where the asterisk denotes convolution. The form of the common impulse response $h_j(t)$ depends on the amount of detail required. A popular choice is an exponential function defined by

$$h_j(t) = \frac{1}{\tau_j} \exp\left(-\frac{t}{\tau_j}\right) \quad (13.17)$$

where τ_j is a *time constant* that is a characteristic of neuron j . The time function $h_j(t)$ of Eq. (13.17) is recognized as the impulse response of a simple circuit consisting of resistor R_j and capacitor C_j connected in parallel and fed from a current source; that is,

$$\tau_j = R_j C_j \quad (13.18)$$

Accordingly, we may use Eqs. (13.16) and (13.17) to formulate the model shown in Fig. 13.16. In physical terms, the synaptic weights $w_{j1}, w_{j2}, \dots, w_{jm_0}$ are represented by conductances (i.e., reciprocals of resistances), and the respective inputs $x_1(t), x_2(t), \dots, x_{m_0}(t)$ are represented by potentials (i.e., voltages). The summing junction is characterized by a low input resistance, unity current gain, and high output resistance; that is, it acts as a summing node for incoming currents. The total current fed into the resistance-capacitance (RC) circuit is therefore

$$\sum_{i=1}^{m_0} w_{ji} x_i(t) + I_j$$

where the first (summation) term is due to the stimuli $x_1(t), x_2(t), \dots, x_{m_0}(t)$ acting on the synaptic weights (conductances) $w_{j1}, w_{j2}, \dots, w_{jm_0}$, respectively, and the second term is the current source I_j representing the externally applied bias b_j .

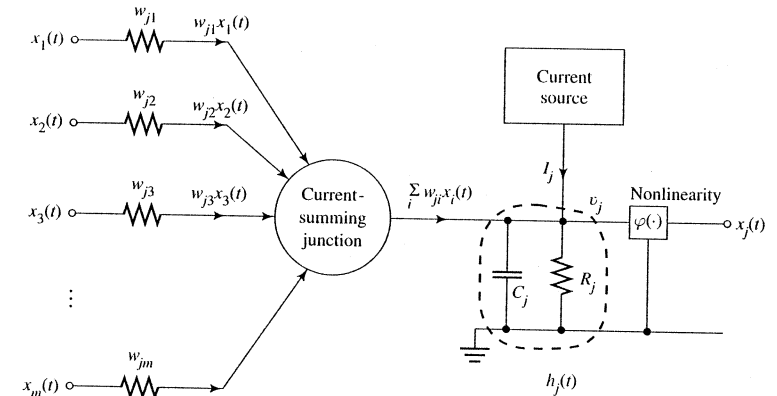


FIGURE 13.16 Additive model of a neuron.

In the neural network literature, the neuronal model shown in Fig. 13.16 is commonly referred to as the *additive model*. This model may be viewed as a lumped circuit approximation of the distributed transmission line model of a biological dendritic neuron (Rall, 1989). The low-pass nature of the RC circuit in Fig. 13.16 may also be justified by the fact that a biological synapse is itself a low-pass filter to an excellent approximation (Scott, 1977).

13.8 DISTRIBUTED TIME LAGGED FEEDFORWARD NETWORKS

The universal myopic mapping algorithm, which provides the mathematical justification for focused TLFNs, is limited to maps that are shift invariant. The implication of this limitation is that the use of focused TLFNs is only suitable for use in stationary (i.e., time-invariant) environments. We may overcome this limitation by using a *distributed time lagged feedforward network (TLFN)*, distributed in the sense that the implicit influence of time is distributed throughout the network. The construction of such a network is based on the multiple input neuronal filter of Fig. 13.14 as the spatio-temporal model of a neuron.

Let $w_{ji}(l)$ denote the weight connected to the l th tap of the FIR filter modeling the synapse that connects the output of neuron i to neuron j . The index l ranges from 0 to p , where p is the order of the FIR filter. According to this model, the signal $s_{ji}(n)$ appearing at the output of the i th synapse of neuron j is given by the *convolution sum*

$$s_{ji}(n) = \sum_{l=0}^p w_{ji}(l) x_i(n - l) \quad (13.19)$$

where n denotes discrete time. We may rewrite Eq. (13.19) in matrix form by introducing the following definitions for the *state vector* and *weight vector* for synapse i , respectively:

$$\mathbf{x}_i(n) = [x_i(n), x_i(n-1), \dots, x_i(n-p)]^T \quad (13.20)$$

$$\mathbf{w}_{ji} = [w_{ji}(0), w_{ji}(1), \dots, w_{ji}(p)]^T \quad (13.21)$$

We may thus express the (scalar) signal $s_{ji}(n)$ as the inner product of the vectors $\mathbf{w}_{ji}(n)$ and $\mathbf{x}_i(n)$; that is,

$$s_{ji}(n) = \mathbf{w}_{ji}^T \mathbf{x}_i(n) \quad (13.22)$$

Equation (13.22) defines the output $s_{ji}(n)$ of the i th synapse of neuron j in the model of Fig. 13.14 in response to the input vector $\mathbf{x}_i(n)$ where $i = 1, 2, \dots, m_0$. The vector $\mathbf{x}_i(n)$ is referred to as a "state" in that it represents the condition of the i th synapse at time n . Hence, summing the contributions of the complete set of m_0 synapses depicted in this model (i.e., summing over the index i), we may describe the output $y_j(n)$ of neuron j by the following pair of equations:

$$v_j(n) = \sum_{i=1}^{m_0} s_{ji}(n) + b_j = \sum_{i=1}^{m_0} \mathbf{w}_{ji}^T \mathbf{x}_i(n) + b_j \quad (13.23)$$

$$y_j(n) = \varphi(v_j(n)) \quad (13.24)$$

where $v_j(n)$ denotes the induced local field of neuron j , b_j is the externally applied bias, and $\varphi(\cdot)$ denotes the nonlinear activation function of the neuron. It is assumed that the same form of nonlinearity is used for all the neurons in the network. Note that if the weight vector $\mathbf{w}_{ji}$ and the state vector $\mathbf{x}_i(n)$ are replaced by the scalars w_{ji} and x_i , respectively, and if the operation of inner product is correspondingly replaced by ordinary multiplication, the dynamic model of a neuron described in Eqs. (13.23) and (13.24) reduces to the static model of the ordinary multilayer perceptron described in Chapter 4.

TEMPORAL BACK-PROPAGATION ALGORITHM

To train a distributed TLFN network, we need a supervised learning algorithm in which the actual response of each neuron in the output layer is compared with a desired (target) response at each time instant. Assume that neuron j lies in the output layer with its actual response denoted by $y_j(n)$ and that the desired response for this neuron is denoted by $d_j(n)$, both of which are measured at time n . We may then define an *instantaneous value* for the sum of squared errors produced by the network as follows:

$$\mathcal{E}(n) = \frac{1}{2} \sum_j e_j^2(n) \quad (13.25)$$

where the index j refers to a neuron in the output layer only, and $e_j(n)$ is the error signal defined by

$$e_j(n) = d_j(n) - y_j(n) \quad (13.26)$$

The goal is to minimize a *cost function* defined as the value of $\mathcal{E}(n)$ computed over all time:

$$\mathcal{E}_{\text{total}} = \sum_n \mathcal{E}(n) \quad (13.27)$$

The algorithm we have in mind for computing an estimate of the optimum weight vector that attains this goal is based on an approximation to the method of steepest descent.

An obvious way of proceeding with this development is to differentiate the cost function of Eq. (13.27) with respect to the weight vector $\mathbf{w}_{ji}$, and so write

$$\frac{\partial \mathcal{E}_{\text{total}}}{\partial \mathbf{w}_{ji}} = \sum_n \frac{\partial \mathcal{E}(n)}{\partial \mathbf{w}_{ji}} \quad (13.28)$$

To proceed further with the instantaneous gradient approach, we *unfold the network in time*. The strategy here is first to try to remove all the time delays in the network by expanding it into an equivalent but larger "static" network, and then to apply the standard back-propagation algorithm to compute the instantaneous error gradients. Unfortunately, such an approach is handicapped by the following negative attributes:

- A loss of a sense of symmetry between the forward propagation of states and the backward propagation of terms needed to calculate instantaneous error gradients
- No nice recursive formula for propagating the error terms
- Need for global bookkeeping to keep track of which static weights are actually the same in the equivalent network obtained by unfolding the distributed TLFN

Although using instantaneous gradient estimates is the obvious approach to developing a temporal version of back propagation, from a practical standpoint it is *not* the way to proceed.

To overcome the problems associated with the instantaneous gradient approach, we propose to proceed as follows (Wan, 1990, 1994). First, we recognize that the expansion of the total error gradient into a sum of instantaneous error gradients, as shown in Eq. (13.28), is not unique. In particular, we may consider an alternative way of expressing the partial derivative of the cost function $\mathcal{E}_{\text{total}}$ with respect to the weight vector $\mathbf{w}_{ji}(n)$ by writing

$$\frac{\partial \mathcal{E}_{\text{total}}}{\partial \mathbf{w}_{ji}} = \sum_n \frac{\partial \mathcal{E}_{\text{total}}}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial \mathbf{w}_{ji}} \quad (13.29)$$

where the time index n runs only over $v_j(n)$. We may interpret the partial derivative $\partial \mathcal{E}_{\text{total}} / \partial v_j(n)$ as the change in the cost function $\mathcal{E}_{\text{total}}$ produced by a change in the induced local field v_j of neuron j at time n . However, it is important to note that

$$\frac{\partial \mathcal{E}_{\text{total}}}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial \mathbf{w}_{ji}} \neq \frac{\partial \mathcal{E}(n)}{\partial \mathbf{w}_{ji}}$$

It is only when we take the sum over all n , as in Eqs. (13.28) and (13.29), that the equality holds.

Given the expansion of Eq. (13.29), we may now use the idea of gradient descent in weight space. In particular, we postulate a recursion for updating the tap-weight vector $\mathbf{w}_{ji}(n)$ as shown by

$$\mathbf{w}_{ji}(n+1) = \mathbf{w}_{ji}(n) - \eta \frac{\partial \mathcal{E}_{\text{total}}}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial \mathbf{w}_{ji}} \quad (13.30)$$

where η is the *learning-rate parameter*. From the defining equation (13.23), we find that for any neuron j in the network, the partial derivative of the induced local field $v_j(n)$ with respect to the weight vector $\mathbf{w}_{ji}(n)$ is given by

$$\frac{\partial v_j(n)}{\partial \mathbf{w}_{ji}(n)} = \mathbf{x}_i(n) \quad (13.31)$$

where $\mathbf{x}_i(n)$ is the input vector applied to synapse i of neuron j . Moreover, we may define the *local gradient* for neuron j as

$$\delta_j(n) = -\frac{\partial \mathcal{E}_{\text{total}}}{\partial v_j(n)} \quad (13.32)$$

Accordingly, we may rewrite Eq. (13.30) in the familiar form

$$\mathbf{w}_{ji}(n+1) = \mathbf{w}_{ji}(n) + \eta \delta_j(n) \mathbf{x}_i(n) \quad (13.33)$$

As in the derivation of the standard back-propagation algorithm described in Chapter 4, the explicit form of the local gradient $\delta_j(n)$ depends on whether or not neuron j lies in the output layer or in a hidden layer of the network. These two different cases are considered next.

CASE 1. Neuron j is an output unit

For the output layer, we simply have

$$\begin{aligned} \delta_j(n) &= \frac{\partial \mathcal{E}_{\text{total}}}{\partial v_j(n)} \\ &= -\frac{\partial \mathcal{E}(n)}{\partial v_j(n)} \\ &= e_j(n) \varphi'(v_j(n)) \end{aligned} \quad (13.34)$$

where $e_j(n)$ is the error signal measured at the output of neuron j , and $\varphi'(\cdot)$ is the derivative of the activation function $\varphi(\cdot)$ with respect to its argument.

CASE 2. Neuron j is a hidden unit

For neuron j located in a hidden layer, we define $\mathcal{A}$ as the set of all neurons whose inputs are fed by neuron j in a forward manner. Let $v_r(n)$ denote the induced local field of neuron r that belongs to the set $\mathcal{A}$. We may then write

$$\begin{aligned} \delta_j(n) &= -\frac{\partial \mathcal{E}_{\text{total}}}{\partial v_j(n)} \\ &= -\sum_{r \in \mathcal{A}} \sum_k \frac{\partial \mathcal{E}_{\text{total}}}{\partial v_r(k)} \frac{\partial v_r(k)}{\partial v_j(n)} \end{aligned} \quad (13.35)$$

where we have used the index k in place of n in those positions that are of particular concern. Using the definition of Eq. (13.32) (with index r used in place of j) in Eq. (13.35), we may thus write

$$\begin{aligned} \delta_j(n) &= \sum_{r \in \mathcal{A}} \sum_n \delta_r(k) \frac{\partial v_r(k)}{\partial v_j(n)} \\ &= \sum_{r \in \mathcal{A}} \sum_n \delta_r(k) \frac{\partial v_r(k)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \end{aligned} \quad (13.36)$$

where $y_j(n)$ is the output of neuron j . Next we recognize that the partial derivative $\partial y_j(n) / \partial v_j(n)$ is equal to $\varphi'(v_j(n))$, referring to neuron j that lies outside the set $\mathcal{A}$. We may therefore take this term outside the double summation and rewrite Eq. (13.36) as

$$\delta_j(n) = \varphi'(v_j(n)) \sum_{r \in \mathcal{A}} \sum_k \delta_r(k) \frac{\partial v_r(k)}{\partial y_j(n)} \quad (13.37)$$

As defined previously, $v_r(n)$ denotes the induced local field of neuron r fed by the output of neuron j . Hence, adapting the meaning of Eqs. (13.19) and (13.23) to the situation at hand, we may express $v_r(k)$ as

$$v_r(k) = \sum_{j=0}^{m_0} \sum_{l=0}^p w_{rj}(l) y_j(n-l) \quad (13.38)$$

In Eq. (13.38) we have included the bias b_r applied to neuron r as the term corresponding to $j=0$ by defining

$$w_{r0}(l) = b_r \quad \text{and} \quad y_0(n-l) = 1 \quad \text{for all } l \text{ and } n \quad (13.39)$$

The index p defining the upper limit of the inner summation in Eq. (13.38) is the order of each synaptic filter of neuron r , and every other neuron in the layer in question. The index m_0 , defining the upper limit of the outer summation in Eq. (13.38), is the total number of primary synapses belonging to neuron r . Recognizing that the convolution sum with respect to l is commutative, we may rewrite Eq. (13.38) in the equivalent form

$$v_r(k) = \sum_{j=0}^{m_0} \sum_{l=0}^p y_j(l) w_{rj}(n-l) \quad (13.40)$$

Differentiating Eq. (13.40) with respect to y_j , we thus obtain

$$\frac{\partial v_r(k)}{\partial y_j(n)} = \begin{cases} w_{rj}(k-l), & n \leq k \leq n+p \\ 0, & \text{otherwise} \end{cases} \quad (13.41)$$

In light of Eq. (13.41), the partial derivatives $\partial v_r(k) / \partial y_j(n)$ in Eq. (13.37), for which n is outside the range $n \leq k \leq n+p$, evaluate to zero. For the case of a hidden neuron j , the use of Eq. (13.41) in (13.37) yields

$$\begin{aligned} \delta_j(n) &= \varphi'(v_j(n)) \sum_{r \in \mathcal{A}} \sum_{k=n}^{n+p} \delta_r(k) w_{rj}(k-l) \\ &= \varphi'(v_j(n)) \sum_{r \in \mathcal{A}} \sum_{l=0}^p \delta_r(n+l) w_{rj}(n) \end{aligned} \quad (13.42)$$

Define a new $(p+1)$ -by-1 vector

$$\Delta_r(n) = [\delta_r(n), \delta_r(n+1), \dots, \delta_r(n+p)]^T \quad (13.43)$$

Earlier we defined the weight vector $\mathbf{w}_{ji}$ as in Eq. (13.21). By using matrix notation we may therefore rewrite Eq. (13.42) in the compact form

$$\delta_j(n) = \varphi'(v_j(n)) \sum_{r \in \mathcal{A}} \Delta_r^T(n) \mathbf{w}_{rj} \quad (13.44)$$

where $\Delta_r^T(n) \mathbf{w}_{rj}$ is the inner product of the vectors $\Delta_r(n)$ and $\mathbf{w}_{rj}$, both of which have dimension $(p + 1)$. Equation (13.44) completes the evaluation of $\delta_j(n)$ for neuron j in the hidden layer.

We are now ready to summarize the weight update equation for *temporal back propagation* as the following pair of relations (Wan, 1990, 1994):

$$\mathbf{w}_{ji}(n + 1) = \mathbf{w}_{ji}(n) + \eta \delta_j(n) \mathbf{x}_i(n) \quad (13.45)$$

$$\delta_j(n) = \begin{cases} e_j(n) \varphi'(v_j(n)), & \text{neuron } j \text{ in the output layer} \\ \varphi'(v_j(n)) \sum_{r \in \mathcal{A}} \Delta_r^T(n) \mathbf{w}_{rj}, & \text{neuron } j \text{ in a hidden layer} \end{cases} \quad (13.46)$$

which may be readily generalized to any number of hidden layers. We immediately observe that these relations represent a *vector generalization* of the standard back-propagation algorithm. If we replace the input vector $\mathbf{x}_i(n)$, the weight vector $\mathbf{w}_{rj}$, and the local gradient vector Δ_r by their scalar counterparts, the temporal back-propagation algorithm reduces to the standard form of the back-propagation algorithm derived in Chapter 4.

To compute $\delta_j(n)$ for neuron j located in a hidden layer, we *propagate* the δ s from the next layer backward through those synaptic filters whose excitation is derived from neuron j , in accordance with Eq. (13.44). This backward-propagation mechanism is illustrated in Fig. 13.17. The local gradient $\delta_j(n)$ is thus formed not by simply taking a weighted sum but by backward filtering through each primary synapse. In particular, for each new set of input and desired response vectors, the forward filters are incremented one time step and the backward filters are incremented one time step.

We now see the practical benefits gained by using the temporal back-propagation algorithm described herein:

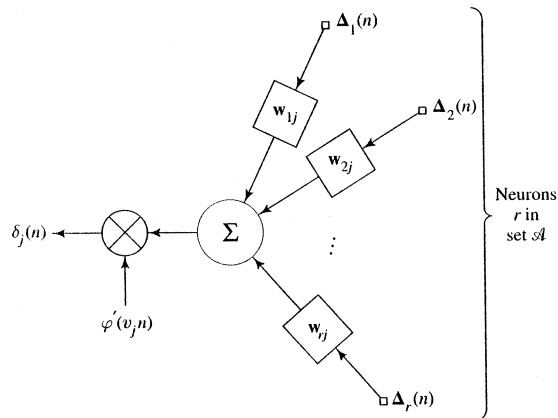


FIGURE 13.17 Back propagation of local gradients through a distributed TLFN.

1. The symmetry between the forward propagation of states and the backward propagation of error terms is preserved, and the sense of parallel distributed processing is thereby maintained.
2. Each unique weight of synaptic filter is used only once in the calculation of the δ s; there is no redundant use of terms experienced in the instantaneous gradient method.

In deriving the temporal back-propagation algorithm⁶ described in Eqs. (13.45) and (13.46), it is assumed that the synaptic filter weights are fixed for all gradient calculations. This is clearly not a valid assumption during actual adaptation. Accordingly discrepancies in performance will arise between the temporal back-propagation algorithm and the temporal version obtained using the instantaneous gradient method. However, these discrepancies are usually of a minor nature. For a small learning-rate parameter η , the differences in the learning characteristics of these two algorithms are negligible for all practical purposes.

Causality Constraints

Careful examination of Eq. (13.42) reveals that the computation of $\delta_j(n)$ is *noncausal* because it requires knowledge of future values of the δ s and the $\mathbf{w}$ s. To make this computation causal, we first note that the exact time reference used for adaptation is unimportant. Moreover, the synaptic structures employed in the network are all FIR filters. Accordingly, causality requires the use of additional buffering to store internal states of the network. In what follows we thus require that the adaptation of all weight vectors be based only on the current and past values of error signals. We may therefore immediately set up $\delta_j(n)$ for neuron j in the output layer and so adapt the synaptic filter weights in that layer. For the next layer back (i.e., one hidden layer back from the output layer), causality constraints imply that for neuron j in this layer the computation of the local gradient

$$\delta_j(n - p) = \varphi'(v_j(n - p)) \sum_{r \in \mathcal{A}} \Delta_r^T(n - p) \mathbf{w}_{rj} \quad (13.47)$$

is based only on current and past values of the vector Δ_r ; that is,

$$\Delta_r(n - p) = [\delta_r(n - p), \delta_r(n + 1 - p), \dots, \delta_r(n)]^T \quad (13.48)$$

Equation (13.47) is obtained from the second line of Eq. (13.46) simply by replacing the time index n by $n - p$, where p is the order of each synaptic FIR filter. As pointed out earlier, the states $\mathbf{x}_i(n - p)$ must be stored so that we may compute the product $\delta_j(n - p) \mathbf{x}_i(n - p)$ for the adaptation of the weight vector connecting neuron j in the last hidden layer to neuron i one layer farther back. For a network with multiple hidden layers, we may continue the operation described here for one more layer back (i.e., two layers back from the output layer) simply by making the time shift twice as long. The operation is continued in this fashion until all the computation layers in the network are accounted for.

We may formulate the *causal* form of the temporal back-propagation algorithm as summarized in Table 13.1.

TABLE 13.1 Summary of the Temporal Back-Propagation Algorithm

1. Propagate the input signal through the network in the forward direction, layer by layer. Determine the error signal $e_j(n)$ for neuron j in the output layer by subtracting its actual output from the corresponding desired response. Also record the state vector for each synapse in the network.
2. For neuron j in the output layer compute

$$\delta_j(n) = e_j(n)\varphi'_j(n)$$

$$\mathbf{w}_{ji}(n+1) = \mathbf{w}_{ji}(n) + \eta\delta_j(n)\mathbf{x}_i(n)$$

where $\mathbf{x}_i(n)$ is the state of synapse i of a hidden neuron connected to output neuron j .

3. For neuron j in a hidden layer, compute

$$\delta_j(n-lp) = \varphi'(v_j(n-lp)) \sum_{r \in \mathcal{A}} \Delta_r^T(n-lp)\mathbf{w}_{jr}$$

$$\mathbf{w}_{ji}(n+1) = \mathbf{w}_{ji}(n) + \eta\delta_j(n-lp)\mathbf{x}_i(n-lp)$$

where p is the order of each synaptic FIR filter, and the index l identifies the hidden layer in question. Specifically, for networks with multiple hidden layers, $l=1$ corresponds to one layer back from the output layer, $l=2$ corresponds to two layers back from the output layer, and so on.

Although this algorithm is less aesthetically pleasing than the noncausal form described in Eqs. (13.45) and (13.46), basically the two forms of the algorithm differ from each other only in terms of a simple change of indices.

Summarizing, then, we may state the following:

- The δ s are propagated through the layers of the network backward and continuously *without* added delays. This kind of propagation forces the internal values of the δ s to be shifted in time.
- To correct for this time shift, the states (i.e., the values of $\mathbf{x}_i(n)$) are stored so as to form the proper product terms needed for adaptation of the weight vectors. In other words, added storage delays are required only for the states, whereas the backward propagation of the δ s is performed without delays.
- The backward propagation of the δ s remains symmetric with respect to the forward propagation of the states.
- The order of computations is linear in the number of synaptic weights in the network as in the instantaneous gradient approach.

The distributed TLFN is naturally a more elaborate structure than the focused TLFN described in Section 13.4. Moreover, the temporal back-propagation algorithm needed to train the distributed TLFN is more computationally demanding than the standard back-propagation algorithm that is adequate to train the focused TLFN. In the final analysis, the use of one or the other of these two approaches is determined by whether the temporal processing task that needs to be solved pertains to a stationary environment or a nonstationary one.⁷

13.10 SUMMARY AND DISCUSSION

The need for temporal processing arises in numerous applications that include the following:

- *Prediction and modeling* of time series (Box and Jenkins, 1976; Haykin, 1996)
- *Noise cancelation*, where the requirement is to use a primary sensor (supplying a desired signal contaminated with noise) and a reference sensor (supplying a correlated version of the noise) to cancel the effect of noise (Widrow and Stearns, 1985; Haykin, 1996)
- *Adaptive equalization* of an unknown communication channel (Proakis, 1989; Haykin, 1996)
- *Adaptive control* (Narendra and Annaswamy, 1989)
- *System identification* (Ljung, 1987)

We already have well-developed theories for solving these problems when the system under study or the underlying physical mechanism of interest is linear; see the books cited above. However, when the system or physical mechanism is nonlinear, we have a more difficult task on our hands. It is in situations of this kind that neural networks have the potential to provide a viable solution and thereby make a significant difference in their application.

In the context of neural networks, we have two candidate networks for temporal processing:

- *Time lagged feedforward networks*
- *Recurrent networks*

Discussion of recurrent networks is taken up in the next two chapters. In this chapter we describe two classes of time lagged feedforward networks (TLFNs): *focused* and *distributed*. In a focused TLFN, the short-term memory is located entirely at the front end of a static network, which makes it straightforward to design. The training of the focused TLFN is accomplished by using the standard back-propagation algorithm, assuming that a multilayer perceptron is used for implementing the static neural network. In the *universal myopic mapping theorem* due to Sanberg and Xu (1997a, 1997b), we have an existence theorem in the sense that it provides the mathematical justification for the approximation of an arbitrary myopic map (i.e., a causal map with uniform-fading memory) by using a cascade of two functional blocks: a bank of linear filters and a static neural network. Such a structure may be implemented by using the focused TLFN, thereby providing a physical realization of this theorem.

The other class of TLFNs, that is, distributed TLFNs, relies on the use of a spatio-temporal model of a neuron, namely, a multiple-input neuronal filter. This model uses finite-duration impulse response (FIR) filters as synaptic filters. As such, the multiple input neuronal filter provides a powerful functional block in its own right for spatio-temporal signal processing built around a single neuron. To train it, we may use the least-mean-square (LMS) algorithm described in Chapter 3. However, to train a distributed TLFN, we require an elaborate learning algorithm exemplified by the temporal back-propagation algorithm described in Section 13.9. A distinctive feature of distributed TLFNs is the way in which the implicit representation of time is distributed

throughout the network, hence the ability to cope with nonstationary (i.e., time-varying) environments. In contrast, in a focused TLFN, by definition, the implicit representation of time is concentrated at the front end of the network, which therefore limits its practical use to stationary (i.e., time-invariant) environments.

ES AND REFERENCES

1. For an essay on the role of time in neural processing, see the classic paper entitled "Finding Structure in Time" by Elman (1990).
2. In Hopfield (1995), a method for the explicit representation of time in neural processing is described. In particular, analog information is represented by using the timing of action potentials with respect to an ongoing collective oscillatory pattern of activity, for which neurobiological evidence is cited; action potentials are described in Chapter 1.
3. For a review of short-term memory structures and their role in temporal processing, see Mozer (1994).
4. For a discussion of hybrids of TDNN and HMM for speech recognition, see Bourlard and Morgan (1994), Katagiri and McDermott (1996), and Bengio (1996).

Some TDNN-HMM hybrids combine the use of a TDNN frame coder (i.e., mapping "acoustic feature detector" into a "phoneme code") and an HMM word/sentence path finder (i.e., mapping "phoneme symbol" into "word/sentence class"), where the coder and path finder are designed separately. In some advanced TDNN-HMM hybrids the squared error loss function for the entire system is used so that a loss related to word/sentence error count can be minimized. An example of this latter scheme is the multi-state TDNN described in Haffner et al. (1991) and Haffner (1994). A simple hybrid of separately designed modules often causes a mismatch between the training and testing performances of the design. The multi-state TDNN performs better in this regard.

In a fundamental sense, recurrent networks (discussed in Chapter 15) have a larger capability of modeling the temporal structure of speech signals than "replica" networks like the TDNN. However, since the nonstationarity and nonlinearity of speech signals are considerable, even recurrent networks may not be sufficient for accurate speech recognition by themselves.

5. For the origins of the universal myopic mapping theorem, see Sandberg (1991).
6. For an alternative diagrammatic derivation of the temporal back-propagation algorithm, see Wan and Beaufays (1996).
7. In Wan (1994), the temporal back-propagation algorithm was used to perform nonlinear prediction on a nonstationary time series exhibiting chaotic pulsations of an NH_3 laser. This particular time series was part of the Santa Fe Institute Time-Series Competition that was held in the United States in 1992. Wan's solution for this temporal processing task won the competition from a diverse list of submissions that included standard recurrent and feedforward neural networks as well as many traditional linear techniques (Wan, 1994). Chaos is discussed in Chapter 14.

PROBLEMS

Focused time lagged feedforward networks (TLFNs)

- 13.1 Summarize the main attributes of a focused TLFN used to model a nonlinear dynamical process.

- 13.2 The focused TLFN depicted in Fig. 13.10 uses a tapped delay line memory for implementing short-term memory. What are the benefits and shortcomings of a focused TLFN that uses a gamma memory for implementing the short-term memory?
- 13.3 In Chapter 2 we describe qualitatively a dynamic approach for implementing a nonlinear adaptive filter. The method involves the use of a static neural network whose stimulation is derived by feeding the input data through a sliding window. The window is moved on the arrival of each new data sample, with the oldest sample inside the window dropped to make room for the new sample. Discuss how a focused TLFN can be used to implement this form of continuous learning.

Spatio-temporal models of a neuron

- 13.4 Consider a neuronal filter whose induced local field $v_i(t)$ is defined by Eq. (13.16). Suppose that the time function $h_i(t)$ in this equation is replaced by the shifted unit impulse

$$h_i(t) = \delta(t - \tau_j)$$

where τ_j is a fixed delay. Describe the way in which the neuronal filter is changed by this modification.

- 13.5 Using the LMS algorithm, formulate a learning algorithm for the multiple-input neuronal filter of Fig. 13.14.

Temporal back-propagation

- 13.6 Figure P13.6 illustrates the use of a *Gaussian-shaped time window* as a method for temporal processing (Bodenhausen and Waibel, 1991). The time window associated with synapse i of neuron j is denoted by $\theta(n, \tau_{ji}, \sigma_{ji})$, where τ_{ji} and σ_{ji} are measures of *time delay* and *width* of the windows, respectively, as shown by

$$\theta(n, \tau_{ji}, \sigma_{ji}) = \frac{1}{\sqrt{2\pi}\sigma_{ji}} \exp\left(-\frac{1}{2\sigma_{ji}^2} (n - \tau_{ji})^2\right)$$

The output of neuron j is thus modeled as

$$y_j(n) = \varphi\left(\sum_{i=0}^{m_0} w_{ji} u_i(n)\right)$$

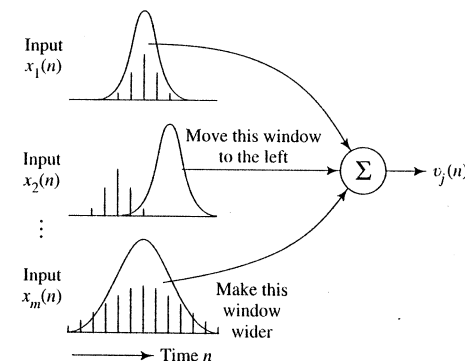


FIGURE P13.6

where $u_i(n)$ is the convolution of the input $x(n)$ and the time window $\theta(n, \tau_{ji}, \sigma_{ji})$. The weight w_{ji} and time delay τ_{ji} of synapse i belonging to neuron j are all to be learned in a supervised manner.

This learning may be accomplished by using the standard back-propagation algorithm. Demonstrate this learning process by deriving update equations for w_{ji} , τ_{ji} , and σ_{ji} .

- 13.7** The material presented in Section 13.9 on the temporal back-propagation algorithm deals with synaptic FIR filters of equal length. How could you handle the case of synaptic FIR filters of different lengths?
- 13.8** Discuss how the temporal back-propagation algorithm may be used for the training of a distributed TLFN for single-step prediction.
- 13.9** The discrepancies between the constrained (causal) and unconstrained (noncausal) forms of the temporal back-propagation algorithm are analogous to the standard least-mean-square (LMS) versus delayed LMS algorithm; the LMS algorithm is discussed in Chapter 3. Expand on this analogy.

Computer experiment

- 13.10** In this problem we use the standard back-propagation algorithm to solve a difficult nonlinear prediction problem and compare its performance with that of the LMS algorithm. The time series to be considered is created using a discrete *Volterra model* that has the form

$$x(n) = \sum_i g_i v(n-i) + \sum_i \sum_j g_{ij} v(n-i)v(n-j) + \dots$$

where $g_i, g_{ij}, \dots$ are the Volterra coefficients, the $v(n)$ are samples of a white, independently distributed Gaussian noise sequence, and $x(n)$ is the resulting Volterra model output. The first summation term is the familiar moving average (MA) time series model, and the remaining summation terms are nonlinear components of ever-increasing order. In general, the estimation of the Volterra coefficients is considered to be difficult, primarily because of their nonlinear relationship to the data.

In this problem we consider the simple example

$$x(n) = v(n) + \beta v(n-1)v(n-2)$$

The time series has zero mean, is uncorrelated, and therefore has a white spectrum. However, the time series samples are not independent of each other, and therefore a higher-order predictor can be constructed. The variance of the model output is given by

$$\sigma_x^2 = \sigma_v^2 + \beta^2 \sigma_v^4$$

where σ_v^2 is the white noise variance.

- (a) Construct a multilayer perceptron with an input layer of 6 nodes, a hidden layer of 16 neurons, and a single output neuron. A tapped delay line memory is used to feed the input layer of the network. The hidden neurons use sigmoid activation functions limited to the interval $[0, 1]$, whereas the output neuron operates as a linear combiner. The network is trained with the standard back-propagation algorithm having the following description:

Learning-rate parameter	$\eta = 0.001$
Momentum constant	$\alpha = 0.6$
Total number of samples processed	100,000
Number of samples per epoch	1,000
Total number of epochs	100

The white noise variance σ_v^2 is set equal to unity. Hence, with $\beta = 0.5$, we find that the output variance of the predictor is $\sigma_x^2 = 1.25$.

Compute the learning curve of the nonlinear predictor, with the variance of the predictor output $x(n)$ plotted as a function of the number of epochs of training samples up to 200 epochs. For the preparation of each epoch used to perform the training, explore the following two modes:

- (i) The time ordering of the training sample is maintained from one epoch to the next in exactly the same form as it is generated.
- (ii) The ordering of the training sample is randomized from one pattern (state) to another.

Also, use cross-validation (described in Chapter 4) with a validation set of 1000 samples to monitor the learning behavior of the predictor.

- (b) Repeat the experiment using the LMS algorithm designed to perform a linear prediction on an input of six samples. The learning-rate parameter of the algorithm is set equal to $\eta = 10^{-5}$.
- (c) Repeat the entire experiment for $\beta = 1$, $\sigma_v^2 = 2$, and then for $\beta = 2$, $\sigma_v^2 = 5$.

The results of each experiment should reveal that initially the back-propagation algorithm and the LMS algorithm follow essentially a similar path, and then the back-propagation algorithm continues to improve, finally producing a prediction variance close to the prescribed value of σ_x^2 .

Neurodynamics

1 INTRODUCTION

In the previous chapter on temporal processing, we studied short-time memory structures and how to operate a static neural network (e.g., multilayer perceptron) as a dynamic mapper by stimulating it via a memory structure. Another important way in which time can be built into the operation of a neural network in an implicit manner is through the use of *feedback*. There are two basic ways of applying feedback to a neural network: *local feedback* at the level of a single neuron inside the network, and *global feedback* encompassing the whole network. Local feedback is a relatively simple matter to deal with, but global feedback has much more profound implications. In the neural network literature, neural networks with one or more feedback loops are referred to as *recurrent networks*. In this chapter and the next one, we focus attention on recurrent networks that use global feedback.

Feedback is like a two-edged sword, in that when it is applied improperly it can produce harmful effects. In particular, the application of feedback can cause a system that is originally stable to become unstable. Our primary interest in this chapter is in the stability of recurrent networks. Other aspects of recurrent networks are considered in the next chapter.

The subject of neural networks viewed as nonlinear dynamical systems, with particular emphasis on the *stability* problem, is referred to as *neurodynamics* (Hirsch, 1989). An important feature of the stability (or instability) of a nonlinear dynamical system is that it is a property of the whole system. As a corollary, *the presence of stability always implies some form of coordination between the individual parts of the system* (Ashby, 1960). It appears that the study of neurodynamics began in 1938 with the work of Nicholas Rashevsky, in whose visionary mind the application of dynamics to biology came into view for the first time.

The stability of a nonlinear dynamical system is a difficult issue to deal with. When we speak of the stability problem, those with an engineering background usually think in terms of the *bounded input–bounded output (BIBO) stability criterion*. According to this criterion, stability means that the output of a system must *not* grow without bound as a result of a bounded input, initial condition, or unwanted disturbance (Brogan, 1985). The BIBO stability criterion is well suited for a linear dynamical

system. However, it is useless to apply it to neural networks because all such nonlinear dynamical systems are BIBO stable because of the saturating nonlinearity built into the constitution of a neuron.

When we speak of stability in the context of a nonlinear dynamical system, we usually mean *stability in the sense of Lyapunov*. In a celebrated *mémoire* dated 1892, Lyapunov (a Russian mathematician and engineer) presented the fundamental concepts of the stability theory known as the *direct method of Lyapunov*.¹ This method is widely used for the stability analysis of linear and nonlinear systems, both time-invariant and time-varying. As such it is directly applicable to the stability analysis of neural networks. Indeed, much of the material presented in this chapter is concerned with the direct method of Lyapunov. However, its application is no easy task.

The study of neurodynamics may follow one of two routes, depending on the application of interest:

- *Deterministic neurodynamics*, in which the neural network model has a deterministic behavior. In mathematical terms, it is described by a set of *nonlinear differential equations* that define the exact evolution of the model as a function of time (Grossberg, 1967; Cohen and Grossberg, 1983; Hopfield, 1984).
- *Statistical neurodynamics*, in which the neural network model is perturbed by the presence of noise. In this case, we have to deal with *stochastic nonlinear differential equations*, thereby expressing the solution in probabilistic terms (Amari et al., 1972; Peretto, 1984; Amari, 1990). The combination of stochasticity and nonlinearity makes the subject more difficult to handle.

In this chapter we restrict ourselves to deterministic neurodynamics.

Organization of the Chapter

The material in this chapter is organized in three parts. In the first part of the chapter, consisting of Sections 14.2 through 14.6, we provide introductory material. Section 14.2 introduces some fundamental concepts in dynamical systems, followed by a discussion of the stability of equilibrium points in Section 14.3. In Section 14.4 we describe various types of attractors that arise in the study of dynamical systems. In Section 14.5 we revisit the additive model of a neuron that was derived in Chapter 13. In Section 14.6 we discuss the manipulation of attractors as a neural network paradigm.

The second part of the chapter, consisting of Sections 14.7 through 14.11, deals with associative memories. Section 14.7 is devoted to a detailed discussion of Hopfield models and the use of discrete Hopfield models as a content-addressable memory. Section 14.8 presents a computer experiment on this application of the Hopfield network. In Section 14.9 we present the Cohen–Grossberg theorem for nonlinear dynamical systems that includes the Hopfield network and other associative memories as special cases. In Section 14.10, we describe another neurodynamical model known as the brain state-in-a-box model that is well suited for clustering. Section 14.11 presents a computer experiment on this second model.

The last part of the chapter, consisting of Sections 14.12 through 14.14, deals with the topic of chaos. Section 14.12 discusses the invariant characteristics of a chaotic process, followed by a discussion of the closely related topic of dynamic reconstruction