1

FEEDFORWARD NEURAL NETWORKS: AN INTRODUCTION

Simon Haykin

A *neural network* is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects (Haykin 1998):

1. Knowledge is acquired by the network through a learning process.
2. Interconnection strengths known as synaptic weights are used to store the knowledge.

Basically, learning is a process by which the free parameters (i.e., synaptic weights and bias levels) of a neural network are adapted through a continuing process of stimulation by the environment in which the network is embedded. The type of learning is determined by the manner in which the parameter changes take place. In a general sense, the learning process may be classified as follows:

- Learning with a teacher, also referred to as supervised learning
- Learning without a teacher, also referred to as unsupervised learning

1.1 SUPERVISED LEARNING

This form of learning assumes the availability of a labeled (i.e., ground-truthed) set of training data made up of N input—output examples:

$$T = \{(\mathbf{x}_i, d_i)\}_{i=1}^{N} \tag{1.1}$$

where $\mathbf{x}_i$ = input vector of ith example
d_i = desired (target) response of ith example, assumed to be scalar for convenience of presentation
N = sample size

Given the training sample T, the requirement is to compute the free parameters of the neural network so that the actual output y_i of the neural network due to $\mathbf{x}_i$ is close enough to d_i for all i in a statistical sense. For example, we may use the mean-square error

$$E(n) = \frac{1}{N}\sum_{i=1}^{N}(d_i - y_i)^2 \tag{1.2}$$

as the index of performance to be minimized.

1.1.1 Multilayer Perceptrons and Back-Propagation Learning

The back-propagation algorithm has emerged as the workhorse for the design of a special class of layered feedforward networks known as *multilayer perceptrons* (MLP). As shown in Fig. 1.1, a multilayer perceptron has an input layer of source nodes and an output layer of neurons (i.e., computation nodes); these two layers connect the network to the outside world. In addition to these two layers, the multilayer perceptron usually has one or more layers of hidden neurons, which are so called because these neurons are not directly accessible. The hidden neurons extract important features contained in the input data.

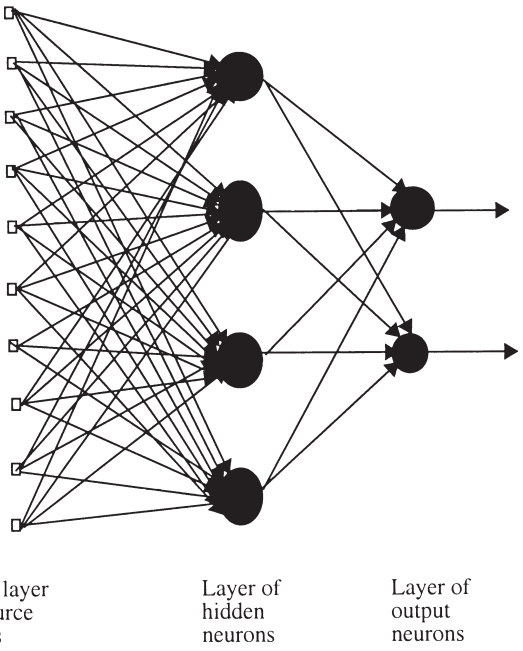

Figure 1.1 Fully connected feedforward with one hidden layer and one output layer.

The training of an MLP is usually accomplished by using a *back-propagation (BP) algorithm* that involves two phases (Werbos 1974; Rumelhart et al. 1986):

- *Forward Phase*. During this phase the free parameters of the network are fixed, and the input signal is propagated through the network of Fig. 1.1 layer by layer. The forward phase finishes with the computation of an error signal

$$e_i = d_i - y_i \tag{1.3}$$

  where d_i is the desired response and y_i is the actual output produced by the network in response to the input $\mathbf{x}_i$.
- *Backward Phase.* During this second phase, the error signal e_i is propagated through the network of Fig. 1.1 in the backward direction, hence the name of the algorithm. It is during this phase that adjustments are applied to the free parameters of the network so as to minimize the error e_i in a statistical sense.

Back-propagation learning may be implemented in one of two basic ways, as summarized here:

1. *Sequential mode* (also referred to as the on-line mode or stochastic mode): In this mode of BP learning, adjustments are made to the free parameters of the network on an example-by-example basis. The sequential mode is best suited for pattern classification.
2. *Batch mode*: In this second mode of BP learning, adjustments are made to the free parameters of the network on an epoch-by-epoch basis, where each epoch consists of the entire set of training examples. The batch mode is best suited for nonlinear regression.

The back-propagation learning algorithm is simple to implement and computationally efficient in that its complexity is linear in the synaptic weights of the network. However, a major limitation of the algorithm is that it does not always converge and can be excruciatingly slow, particularly when we have to deal with a difficult learning task that requires the use of a large network.

We may try to make back-propagation learning perform better by invoking the following list of heuristics:

- Use neurons with antisymmetric activation functions (e.g., hyperbolic tangent function) in preference to nonsymmetric activation functions (e.g., logistic function). Figure 1.2 shows examples of these two forms of activation functions.
- Shuffle the training examples after the presentation of each epoch; an epoch involves the presentation of the entire set of training examples to the network.
- Follow an easy-to-learn example with a difficult one.
- Preprocess the input data so as to remove the mean and decorrelate the data.
- Arrange for the neurons in the different layers to learn at essentially the same rate. This may be attained by assigning a learning rate parameter to neurons in the last layers that is smaller than those at the front end.
- Incorporate prior information into the network design whenever it is available.

One other heuristic that deserves to be mentioned relates to the size of the training set, N, for a pattern classification task. Given a multilayer perceptron with a total number of synaptic weights including bias levels, denoted by W, a rule of thumb for selecting N is

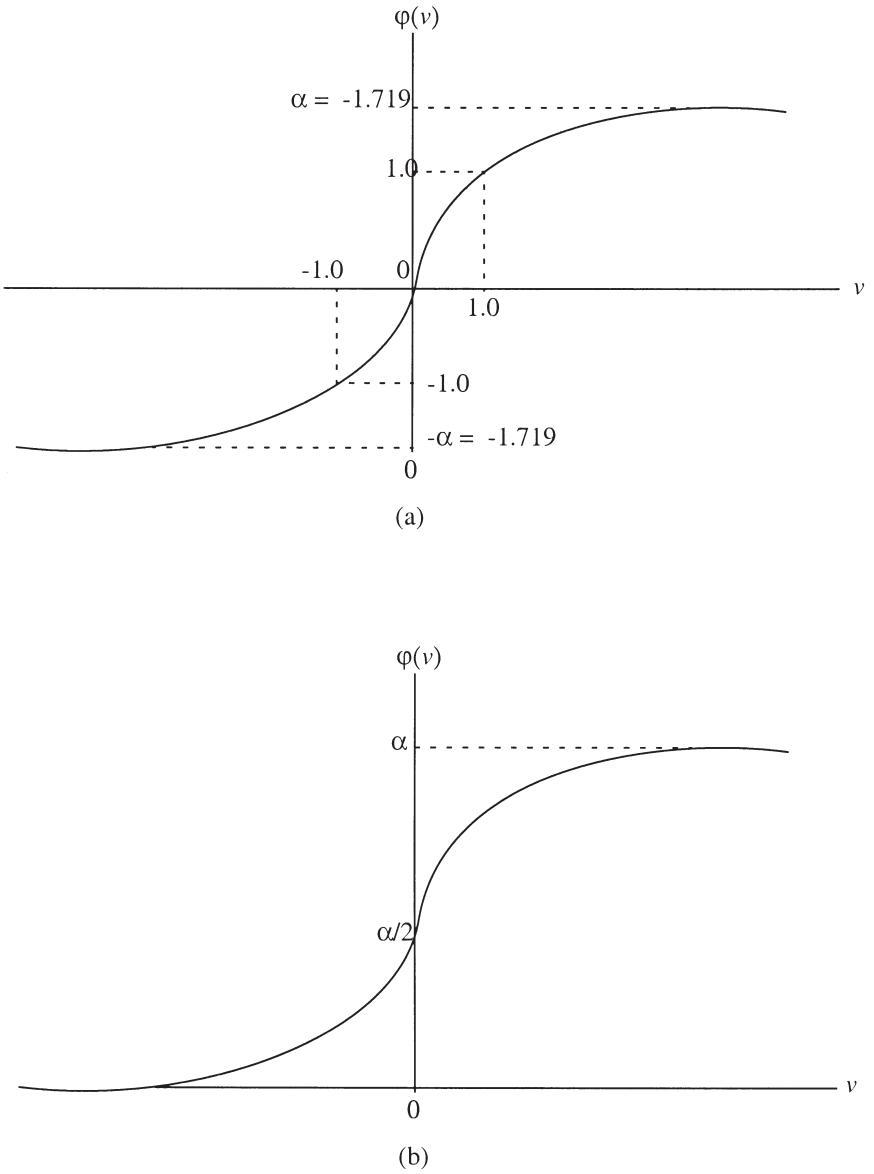

Figure 1.2 (*a*) Antisymmetric activation function. (*b*) Nonsymmetric activation function.

$$N = O\left(\frac{W}{\varepsilon}\right) \tag{1.4}$$

where O denotes "the order of," and ε denotes the fraction of classification errors permitted on test data. For example, with an error of 10% the number of training examples needed should be about 10 times the number of synaptic weights in the network.

Supposing that we have chosen a multilayer perceptron to be trained with the back-propagation algorithm, how do we determine when it is "best" to stop the training session? How do we select the size of individual hidden layers of the MLP? The answers to these important questions may be gotten though the use of a statistical technique known as *cross-validation*, which proceeds as follows (Haykin 1999):

- The set of training examples is split into two parts:
  - Estimation subset used for training of the model
  - Validation subset used for evaluating the model performance
- The network is finally tuned by using the entire set of training examples and then tested on test data not seen before.

1.1.2 Radial-Basis Function Networks

Another popular layered feedforward network is the radial-basis function (RBF) network which has important universal approximation properties (Park and Sandberg 1993), and whose structure is shown in Fig. 13. RBF networks use memory-based learning for their design. Specifically, learning is viewed as a curve-fitting problem in high-dimensional space (Broomhead and Lowe 1989; Poggio and Girosi 1990):

1. Learning is equivalent to finding a surface in a multidimensional space that provides a best fit to the training data.
2. Generalization (i.e., response of the network to input data not seen before) is equivalent to the use of this multidimensional surface to interpolate the test data.

RBF networks differ from multilayer perceptrons in some fundamental respects:

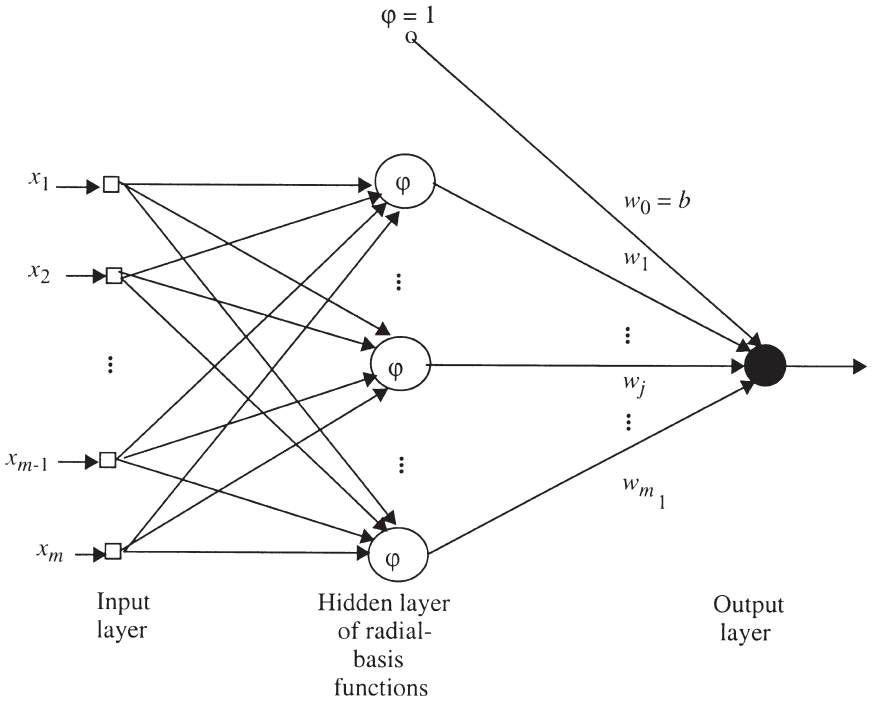

Figure 1.3 Radial-basis function network.

- RBF networks are local approximators, whereas multilayer perceptrons are global approximators.
- RBF networks have a single hidden layer, whereas multilayer perceptrons can have any number of hidden layers.
- The output layer of a RBF network is always linear, whereas in a multilayer perceptron it can be linear or nonlinear.
- The activation function of the hidden layer in an RBF network computes the Euclidean distance between the input signal vector and parameter vector of the network, whereas the activation function of a multilayer perceptron computes the inner product between the input signal vector and the pertinent synaptic weight vector.

The use of a linear output layer in an RBF network may be justified in light of *Cover's theorem* on the separability of patterns. According to this theorem, provided that the transformation from the input space to the feature (hidden) space is nonlinear and the

dimensionality of the feature space is high compared to that of the input (data) space, then there is a high likelihood that a nonseparable pattern classification task in the input space is transformed into a linearly separable one in the feature space. Another analytical basis for the use of RBF networks (and multilayer perceptrons) in classification problems is provided by the results in Chapter 2, where (as a special case) it is shown that a large family of classification problems in $\mathbb{R}^n$ can be solved using nonlinear static networks.

Design methods for RBF networks include the following:

1. Random selection of fixed centers (Broomhead and Lowe 1998)
2. Self-organized selection of centers (Moody and Darken 1989)
3. Supervised selection of centers (Poggio and Girosi 1990)
4. Regularized interpolation exploiting the connection between an RBF network and the Watson–Nadaraya regression kernel (Yee 1998).

1.2 UNSUPERVISED LEARNING

Turning next to unsupervised learning, adjustment of synaptic weights may be carried through the use of neurobiological principles such as Hebbian learning and competitive learning. In this section we will describe specific applications of these two approaches.

1.2.1 Principal Components Analysis

According to *Hebb's postulate of learning*, the change in synaptic weight Δw_{ji} of a neural network is defined by

$$\Delta w_{ji} = \eta x_i y_j \tag{1.5}$$

where η = learning-rate parameter
x_i = input (presynaptic) signal
y_i = output (postsynaptic) signal

Principal component analysis (PCA) networks use a modified form of this self-organized learning rule. To begin with, consider a linear neuron designed to operate as a maximum eigenfilter; such a neuron is referred to as *Oja's neuron* (Oja 1982). It is characterized as follow:

$$\Delta w_{ji} = \eta y_j (x_i - y_j w_{ji}) \tag{1.6}$$

where the term $-\eta y_j^2 w_{ji}$ is added to stabilize the learning process. As the number of iterations approaches infinity, we find the following:

1. The synaptic weight vector of neuron j approaches the eigenvector associated with the largest eigenvalue $\lambda_{\max}$ of the correlation matrix of the input vector (assumed to be of zero mean).
2. The variance of the output of neuron j approaches the largest eigenvalue $\lambda_{\max}$.

The generalized Hebbian algorithm (GHA), due to Sanger (1989), is a straightforward generalization of Oja's neuron for the extraction of any desired number of principal components.

1.2.2 Self-Organizing Maps

In a self-organizing map (SOM), due to Kohonen (1997), the neurons are placed at the nodes of a lattice, and they become selectively tuned to various input patterns (vectors) in the course of a competitive learning process. The process is characterized by the formation of a topographic map in which the spatial locations (i.e., coordinates) of the neurons in the lattice correspond to intrinsic features of the input patterns. Figure 1.4 illustrates the basic idea of a self-organizing map, assuming the use of a two-dimensional lattice of neurons as the network structure.

In reality, the SOM belongs to the class of vector-coding algorithms (Luttrell, 1989). That is, a fixed number of codewords are placed into a higher-dimensional input space, thereby facilitating data compression.

An integral feature of the SOM algorithm is the neighborhood function centered around a neuron that wins the competitive process. The neighborhood function starts by enclosing the entire lattice initially and is then allowed to shrink gradually until it encompasses the winning neuron.

The algorithm exhibits two distinct phases in its operation:

1. *Ordering phase*, during which the topological ordering of the weight vectors takes place
2. *Convergence phase*, during which the computational map is fine tuned

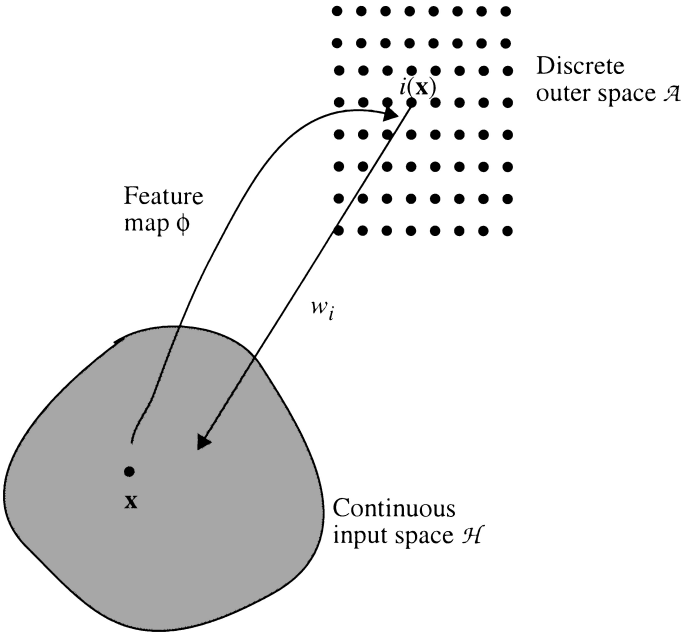

Figure 1.4 Illustration of relationship between feature map ϕ and weight vector $\mathbf{w}_i$ of winning neuron i.

The SOM algorithm exhibits the following properties:

1. Approximation of the continuous input space by the weight vectors of the discrete lattice.
2. Topological ordering exemplified by the fact that the spatial location of a neuron in the lattice corresponds to a particular feature of the input pattern.
3. The feature map computed by the algorithm reflects variations in the statistics of the input distribution.
4. SOM may be viewed as a nonlinear form of principal components analysis.

Refinements to the SOM algorithm are discussed in Van Hulle (2000).

1.3 TEMPORAL PROCESSING USING FEEDFORWARD NETWORKS

The material just described is concerned basically with the approximation of systems without dynamics (i.e., with static systems). At

about the same time as the appearance of the early universal-approximation theorems for static neural networks there began (Sandberg 1991a) a corresponding study (see Chapter 2) of the neural network approximation of approximately-finite-memory maps and myopic maps. It was found that large classes of these maps can be uniformly approximated arbitrarily well by the maps of certain simple nonlinear structures using, for example, sigmoidal nonlinearities or radial basis functions. The approximating networks are two-stage structures comprising a linear preprocessing stage followed by a memoryless nonlinear network. Much is now known about the properties of these networks, and examples of these properties are given in the following.

From another perspective, *time* is an essential dimension of learning. We may incorporate time into the design of a neural network implicitly or explicitly. A straightforward method of implicit representation of time[1] is to add a *short-term memory structure* in the input layer of a static neural network (e.g., multilayer perceptron). The resulting configuration is sometimes called a *focused time-lagged feedforward network (TLFN)*.

The short-term memory structure may be implemented in one of two forms, as described here:

1. *Tapped-Delay-Line (TDL) Memory*. This is the most commonly used form of short-term memory. It consists of p unit delays with $(p + 1)$ terminals, as shown in Fig. 1.5, which may be viewed as a single input–multiple output network. Figure 1.6 shows a focused TLFN network using the combination of a TDL memory and multilayer perceptron. In Figs. 1.5 and 1.6, the unit-delay is denoted by z^{-1}.

   The *memory depth* of a TDL memory is fixed at p, and its *memory resolution* is fixed at unity, giving a *depth resolution constant* of p.

2. *Gamma Memory*. We may exercise control over the memory depth by building a feedback loop around each unit delay, as illustrated in Fig. 1.7 (deVries and Principe 1992). In effect, the unit delay z^{-1} of the standard TDL memory is replaced by the transfer function

[1] Another practical way of accounting for time in a neural network is to employ feedback at the local or global level. Neural networks so configured are referred to as recurrent networks. For detailed treatment of recurrent networks, see Haykin (1999).

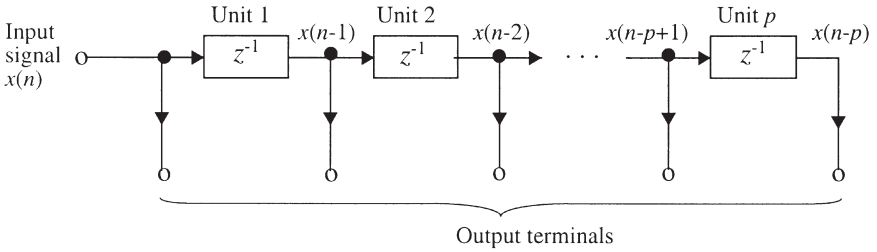

Figure 1.5 Ordinary tapped-delay line memory of order *p*.

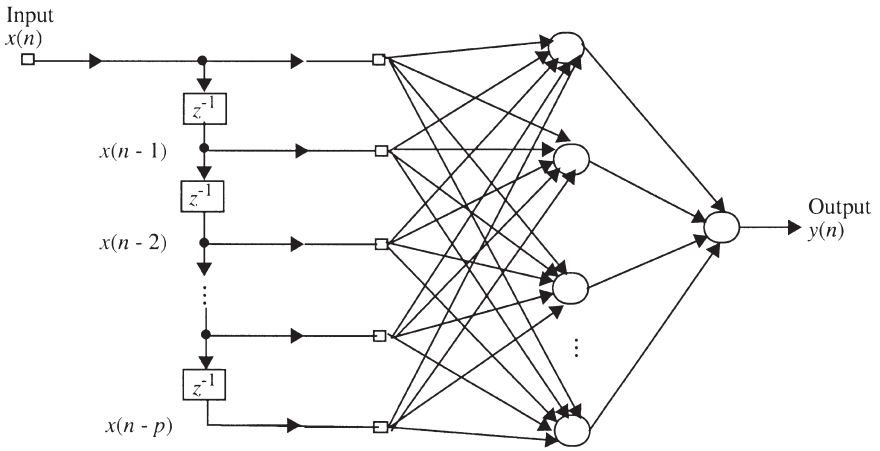

Figure 1.6 Focused time-lagged feedforward network (TLFN); the bias levels have been omitted for convenience of presentation.

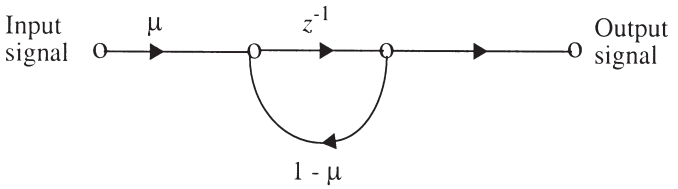

Figure 1.7 Signal-flow graph for one section of gamma memory.

$$G(z) = \frac{\mu z^{-1}}{1-(1-\mu)z^{-1}}$$
$$= \frac{\mu}{z-(1-\mu)}$$

where μ is an adjustable parameter. 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 we restrict the choice of μ to the following range of values:

$$0 < \mu < 2$$

The overall impulse response of the gamma memory, consisting of p sections, is the inverse z transform of the overall transfer function

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

Denoting the impulse response by $g_p(n)$, we have

$$g_p(z) = \binom{n-1}{p-1}\mu^p(1-\mu)^{n-p} \qquad n \geq p$$

where (:) is a binomial coefficient. 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 name "gamma memory."

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 over the TDL memory, but at the expense of memory resolution.

With regard to the utility of gamma networks—which are particular cases of the family of two-stage structures comprising a linear preprocessing stage followed by a memoryless nonlinear network—experimental results have been reported which indicate that the structure is useful. In fact, it is known (Sandberg and Xu 1997) that

for a large class of discrete-time dynamic system maps H, and for any choice of μ in the interval (0, 1), there is a focused gamma network that approximates H uniformly arbitrarily well. It is known that tapped-delay-line networks (i.e., networks with $\mu = 1$) also have the universal approximation property (Sandberg 1991b).

Focused TLFNs using ordinary tapped-delay memory or gamma memory are limited to stationary environments. To deal with nonstationary dynamical processes, we may use distributed TLFNs where the effect of time is distributed at the synaptic level throughout the network. One way in which this may be accomplished is to use finite-duration impulse response (FIR) filters to implement the synaptic connections of an MLP; Fig. 1.8 shows an FIR model of a synapse. The training of a distributed TLFN is naturally a more difficult proposition than the training of a focused TLFN. Whereas we may use the ordinary back-propagation algorithm to train a focused TLFN, we have to extend the back-propagation algorithm to cope with the replacement of a synaptic weight in the ordinary MLP by a synaptic weight vector. This extension is referred to as the temporal back-propagation algorithm due to Wan (1994).

1.4 CONCLUDING REMARKS

In this chapter, we briefly reviewed the feedforward type of neural networks, which are exemplified by multilayer perceptrons (MLPs), radial-basis function (RBF) networks, principal component analysis (PCA) networks, and self-organizing maps (SOMs). The training of MLPs and RBF networks proceeds in a supervised manner, whereas the training of PCA networks and SOMs proceeds in an unsupervised manner.

Feedforward networks by themselves are nonlinear static networks. They can be made to operate as nonlinear dynamical systems

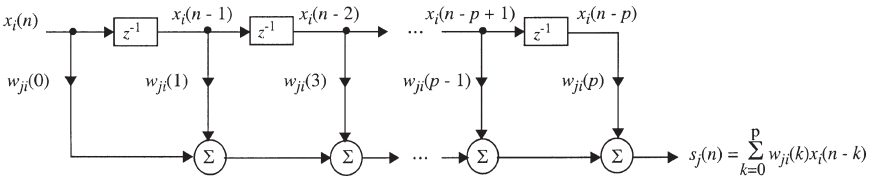

Figure 1.8 Finite-duration impulse response (FIR) filter.

by incorporating short-term memory into their input layer. Two important examples of short-term memory are the standard tapped-delay-line and the gamma memory that provides control over attainable memory depth. The attractive feature of nonlinear dynamical systems built in this way is that they are inherently stable.

BIBLIOGRAPHY

Anderson, J. A., 1995, *Introduction to Neural Networks* (Cambridge, MA: MIT Press).

Barlow, H. B., 1989, "Unsupervised learning," *Neural Computation*, vol. 1, pp. 295–311.

Becker, S., and G. E. Hinton, 1982, "A self-organizing neural network that discovers surfaces in random-dot stereograms," *Nature (London)*, vol. 355, pp. 161–163.

Broomhead, D. S., and D. Lowe, 1988, "Multivariable functional interpolation and adaptive networks," *Complex Systems*, vol. 2, pp. 321–355.

Comon, P., 1994, "Independent component analysis: A new concept?" *Signal Processing*, vol. 36, pp. 287–314.

deVries, B., and J. C. Principe, 1992, "The gamma model—A new neural model for temporal processing," *Neural Networks*, vol. 4, pp. 565–576.

Haykin, S., 1999, *Neural Networks: A Comprehensive Foundation*, 2nd ed. (Englewood Cliffs, NJ: Prentice-Hall).

Moody and Darken, 1989, "Fast learning in networks of locally-tuned processing unites," *Neural Computation*, vol. 1, pp. 281–294.

Oja, E., 1982, "A simplified neuron model as a principal component analyzer," *J. Math. Biol.*, vol. 15, pp. 267–273.

Park, J., and Sandberg, I. W., 1993, "Approximation and radial-basis function networks," *Neural computation*, vol. 5, pp. 305–316.

Poggio, T., and F. Girosi, 1990, "Networks for approximation and learning," *Proc. IEEE*, vol. 78, pp. 1481–1497.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams, 1986, "Learning internal representations by error propagation," in D. E. Rumelhart and J. L. McCleland, eds. (Cambridge, MA: MIT Press), vol. 1, Chapter 8.

Sandberg, I. W., 1991a, "Structure theorems for nonlinear systems," *Multidimensional Sys. Sig. Process.* vol. 2, pp. 267–286. (Errata in 1992, vol. 3, p. 101.)

Sandberg, I. W., 1991b, "Approximation theorems for discrete-time systems," *IEEE Trans. Circuits Sys.* vol. 38, no. 5, pp. 564–566, May 1991.

Sandberg, I. W., and Xu, L., 1997, "Uniform approximation and gamma networks," *Neural Networks*, vol. 10, pp. 781–784.

Van Hulle, M. M., 2000, *Faithful Representations and Topographic Maps: From Distortion-to-Information-Based Self Organization* (New York: Wiley).

Wan, E. A., 1994, "Time series prediction by using a connectionist network with internal delay lines," in A. S. Weigend and N. A. Gershenfield, eds., *Time Series Prediction: Forecasting the Future and Understanding the Past* (Reading, MA: Addison-Wesley), pp. 195–217.

Werbos, P. J., 1974, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," Ph.D. Thesis, Harvard University, Cambridge, MA.

Werbos, P. J., 1990, "Backpropagation through time: What it does and how to do it," *Proc. IEEE*, vol. 78, pp. 1550–1560.

Yee, P. V., 1998, "*Regularized radial basis function networks: Theory and applications to probability estimation, classification, and time series prediction*," Ph.D. Thesis, McMaster University, Hamilton, Ontario.