CHAPTER 1

Introduction

1.1 WHAT IS A NEURAL NETWORK?

Work on artificial neural networks, commonly referred to as "neural networks," has been motivated right from its inception by the recognition that the human brain computes in an entirely different way from the conventional digital computer. The brain is a highly *complex, nonlinear, and parallel computer* (information-processing system). It has the capability to organize its structural constituents, known as *neurons*, so as to perform certain computations (e.g., pattern recognition, perception, and motor control) many times faster than the fastest digital computer in existence today. Consider, for example, human *vision*, which is an information-processing task (Marr, 1982; Levine, 1985; Churchland and Sejnowski, 1992). It is the function of the visual system to provide a *representation* of the environment around us and, more important, to supply the information we need to *interact* with the environment. To be specific, the brain routinely accomplishes perceptual recognition tasks (e.g., recognizing a familiar face embedded in an unfamiliar scene) in approximately 100–200 ms, whereas tasks of much lesser complexity may take days on a conventional computer.

For another example, consider the *sonar* of a bat. Sonar is an active echo-location system. In addition to providing information about how far away a target (e.g., a flying insect) is, a bat sonar conveys information about the relative velocity of the target, the size of the target, the size of various features of the target, and the azimuth and elevation of the target (Suga, 1990a, b). The complex neural computations needed to extract all this information from the target echo occur within a brain the size of a plum. Indeed, an echo-locating bat can pursue and capture its target with a facility and success rate that would be the envy of a radar or sonar engineer.

How, then, does a human brain or the brain of a bat do it? At birth, a brain has great structure and the ability to build up its own rules through what we usually refer to as "experience." Indeed, experience is built up over time, with the most dramatic development (i.e., hard-wiring) of the human brain taking place during the first two years from birth; but the development continues well beyond that stage.

A "developing" neuron is synonymous with a plastic brain: *Plasticity* permits the developing nervous system to adapt to its surrounding environment. Just as plasticity appears to be essential to the functioning of neurons as information-processing units in

the human brain, so it is with neural networks made up of artificial neurons. In its most general form, a *neural network* is a machine that is designed to *model* the way in which the brain performs a particular task or function of interest; the network is usually implemented by using electronic components or is simulated in software on a digital computer. Our interest in this book is confined largely to an important class of neural networks that perform useful computations through a process of *learning*. To achieve good performance, neural networks employ a massive interconnection of simple computing cells referred to as "neurons" or "processing units." We may thus offer the following definition of a neural network viewed as an adaptive machine[1]:

> *A neural network is a massively parallel distributed processor made up of simple processing units, which has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:*
>
> 1. *Knowledge is acquired by the network from its environment through a learning process.*
> 2. *Interneuron connection strengths, known as synaptic weights, are used to store the acquired knowledge.*

The procedure used to perform the learning process is called a *learning algorithm*, the function of which is to modify the synaptic weights of the network in an orderly fashion to attain a desired design objective.

The modification of synaptic weights provides the traditional method for the design of neural networks. Such an approach is the closest to linear adaptive filter theory, which is already well established and successfully applied in many diverse fields (Widrow and Stearns, 1985; Haykin, 1996). However, it is also possible for a neural network to modify its own topology, which is motivated by the fact that neurons in the human brain can die and that new synaptic connections can grow.

Neural networks are also referred to in literature as *neurocomputers*, *connectionist networks*, *parallel distributed processors*, etc. Throughout the book we use the term "neural networks"; occasionally the term "neurocomputer" or "connectionist network" is used.

Benefits of Neural Networks

It is apparent that a neural network derives its computing power through, first, its massively parallel distributed structure and, second, its ability to learn and therefore generalize. *Generalization* refers to the neural network producing reasonable outputs for inputs not encountered during training (learning). These two information-processing capabilities make it possible for neural networks to solve complex (large-scale) problems that are currently intractable. In practice, however, neural networks cannot provide the solution by working individually. Rather, they need to be integrated into a consistent system engineering approach. Specifically, a complex problem of interest is *decomposed* into a number of relatively simple tasks, and neural networks are assigned a subset of the tasks that *match* their inherent capabilities. It is important to recognize, however, that we have a long way to go (if ever) before we can build a computer architecture that mimics a human brain.

The use of neural networks offers the following useful properties and capabilities:

1. *Nonlinearity*. An artificial neuron can be linear or nonlinear. A neural network, made up of an interconnection of nonlinear neurons, is itself nonlinear. Moreover,

the nonlinearity is of a special kind in the sense that it is *distributed* throughout the network. Nonlinearity is a highly important property, particularly if the underlying physical mechanism responsible for generation of the input signal (e.g., speech signal) is inherently nonlinear.

2. *Input–Output Mapping.* A popular paradigm of learning called *learning with a teacher or supervised learning* involves modification of the synaptic weights of a neural network by applying a set of labeled *training samples* or *task examples*. Each example consists of a unique *input signal* and a corresponding *desired response*. The network is presented with an example picked at random from the set, and the synaptic weights (free parameters) of the network are modified to minimize the difference between the desired response and the actual response of the network produced by the input signal in accordance with an appropriate statistical criterion. The training of the network is repeated for many examples in the set until the network reaches a steady state where there are no further significant changes in the synaptic weights. The previously applied training examples may be reapplied during the training session but in a different order. Thus the network learns from the examples by constructing an *input–output mapping* for the problem at hand. Such an approach brings to mind the study of *nonparametric statistical inference*, which is a branch of statistics dealing with model-free estimation, or, from a biological viewpoint, *tabula rasa* learning (Geman et. al., 1992); the term "nonparametric" is used here to signify the fact that no prior assumptions are made on a statistical model for the input data. Consider, for example, a *pattern classification* task, where the requirement is to assign an input signal representing a physical object or event to one of several prespecified categories (classes). In a nonparametric approach to this problem, the requirement is to "estimate" arbitrary decision boundaries in the input signal space for the pattern-classification task using a set of examples, and to do so *without* invoking a probabilistic distribution model. A similar point of view is implicit in the supervised learning paradigm, which suggests a close analogy between the input–output mapping performed by a neural network and nonparametric statistical inference.

3. *Adaptivity.* Neural networks have a built-in capability to *adapt* their synaptic weights to changes in the surrounding environment. In particular, a neural network trained to operate in a specific environment can be easily *retrained* to deal with minor changes in the operating environmental conditions. Moreover,when it is operating in a *nonstationary* environment (i.e., one where statistics change with time), a neural network can be designed to change its synaptic weights in real time. The natural architecture of a neural network for pattern classification, signal processing, and control applications, coupled with the adaptive capability of the network, make it a useful tool in adaptive pattern classification, adaptive signal processing, and adaptive control. As a general rule, it may be said that the more adaptive we make a system, all the time ensuring that the system remains stable, the more robust its performance will likely be when the system is required to operate in a nonstationary environment. It should be emphasized, however, that adaptivity does not always lead to robustness; indeed, it may do the very opposite. For example, an adaptive system with short time constants may change rapidly and therefore tend to respond to spurious disturbances, causing a drastic degradation in system performance. To realize the full benefits of adaptivity, the principal time constants of the system should be long enough for the system to ignore spurious disturbances and yet short enough to respond to meaningful changes in the

environment; the problem described here is referred to as the *stability–plasticity dilemma* (Grossberg, 1988b).

4. *Evidential Response.* In the context of pattern classification, a neural network can be designed to provide information not only about which particular pattern to *select*, but also about the *confidence* in the decision made. This latter information may be used to reject ambiguous patterns, should they arise, and thereby improve the classification performance of the network.

5. *Contextual Information.* Knowledge is represented by the very structure and activation state of a neural network. Every neuron in the network is potentially affected by the global activity of all other neurons in the network. Consequently, contextual information is dealt with naturally by a neural network.

6. *Fault Tolerance.* A neural network, implemented in hardware form, has the potential to be inherently *fault tolerant*, or capable of robust computation, in the sense that its performance degrades gracefully under adverse operating conditions. For example, if a neuron or its connecting links are damaged, recall of a stored pattern is impaired in quality. However, due to the distributed nature of information stored in the network, the damage has to be extensive before the overall response of the network is degraded seriously. Thus, in principle, a neural network exhibits a graceful degradation in performance rather than catastrophic failure. There is some empirical evidence for robust computation, but usually it is uncontrolled. In order to be assured that the neural network is in fact fault tolerant, it may be necessary to take corrective measures in designing the algorithm used to train the network (Kerlirzin and Vallet, 1993).

7. *VLSI Implementability.* The massively parallel nature of a neural network makes it potentially fast for the computation of certain tasks. This same feature makes a neural network well suited for implementation using *very-large-scale-integrated* (VLSI) technology. One particular beneficial virtue of VLSI is that it provides a means of capturing truly complex behavior in a highly hierarchical fashion (Mead, 1989).

8. *Uniformity of Analysis and Design.* Basically, neural networks enjoy universality as information processors. We say this in the sense that the same notation is used in all domains involving the application of neural networks. This feature manifests itself in different ways:

- Neurons, in one form or another, represent an ingredient *common* to all neural networks.
- This commonality makes it possible to *share* theories and learning algorithms in different applications of neural networks.
- Modular networks can be built through a *seamless integration of modules.*

9. *Neurobiological Analogy.* The design of a neural network is motivated by analogy with the brain, which is a living proof that fault tolerant parallel processing is not only physically possible but also fast and powerful. Neurobiologists look to (artificial) neural networks as a research tool for the interpretation of neurobiological phenomena. On the other hand, engineers look to neurobiology for new ideas to solve problems more complex than those based on conventional hard-wired design techniques. These two viewpoints are illustrated by the following two respective examples:

- In Anastasio (1993), linear system models of the vestibulo-ocular reflex are compared to neural network models based on *recurrent networks* that are described in Section 1.6 and discussed in detail in Chapter 15. The *vestibulo-ocular reflex (VOR)* is part of the oculomotor system. The function of VOR is to maintain visual (i.e., retinal) image stability by making eye rotations that are opposite to head rotations. The VOR is mediated by premotor neurons in the vestibular nuclei that receive and process head rotation signals from vestibular sensory neurons and send the results to the eye muscle motor neurons. The VOR is well suited for modeling because its input (head rotation) and its output (eye rotation) can be precisely specified. It is also a relatively simple reflex and the neurophysiological properties of its constituent neurons have been well described. Among the three neural types, the premotor neurons (reflex interneurons) in the vestibular nuclei are the most complex and therefore most interesting. The VOR has previously been modeled using lumped, linear system descriptors and control theory. These models were useful in explaining some of the overall properties of the VOR, but gave little insight into the properties of its constituent neurons. This situation has been greatly improved through neural network modeling. Recurrent network models of VOR (programmed using an algorithm called real-time recurrent learning that is described in Chapter 15) can reproduce and help explain many of the static, dynamic, nonlinear, and distributed aspects of signal processing by the neurons that mediate the VOR, especially the vestibular nuclei neurons (Anastasio, 1993).
- The *retina,* more than any other part of the brain, is where we begin to put together the relationships between the outside world represented by a visual sense, its *physical image* projected onto an array of receptors, and the first *neural images.* The retina is a thin sheet of neural tissue that lines the posterior hemisphere of the eyeball. The retina's task is to convert an optical image into a neural image for transmission down the optic nerve to a multitude of centers for further analysis. This is a complex task, as evidenced by the synaptic organization of the retina. In all vertebrate retinas the transformation from optical to neural image involves three stages (Sterling, 1990):
  - **(i)** Photo transduction by a layer of receptor neurons.
  - **(ii)** Transmission of the resulting signals (produced in response to light) by chemical synapses to a layer of bipolar cells.
  - **(iii)** Transmission of these signals, also by chemical synapses, to output neurons that are called ganglion cells.

  At both synaptic stages (i.e., from receptor to bipolar cells, and from bipolar to ganglion cells), there are specialized laterally connected neurons called *horizontal cells* and *amacrine cells,* respectively. The task of these neurons is to modify the transmission across the synaptic layers. There are also centrifugal elements called *inter-plexiform cells;* their task is to convey signals from the inner synaptic layer back to the outer one. A few researchers have built electronic chips that mimic the structure of the retina (Mahowald and Mead, 1989; Boahen and Ardreou, 1992; Boahen, 1996). These electronic chips are called *neuromorphic* integrated circuits, a term coined by Mead (1989). A neuromorphic imaging sensor consists of an array of photoreceptors combined with analog circuitry at each

picture element (pixel). It emulates the retina in that it can adapt locally to changes in brightness, detect edges, and detect motion. The neurobiological analogy, exemplified by neuromorphic integrated circuits is useful in another important way: It provides a hope and belief, and to a certain extent an existence of proof, that physical understanding of neurobiological structures could have a productive influence on the art of electronics and VLSI technology.

With inspiration from neurobiology in mind, it seems appropriate that we take a brief look at the human brain and its structural levels of organization.

1.2 HUMAN BRAIN

The human nervous system may be viewed as a three-stage system, as depicted in the block diagram of Fig. 1.1 (Arbib, 1987). Central to the system is the *brain*, represented by the *neural (nerve) net*, which continually receives information, perceives it, and makes appropriate decisions. Two sets of arrows are shown in the figure. Those pointing from left to right indicate the *forward* transmission of information-bearing signals through the system. The arrows pointing from right to left signify the presence of *feedback* in the system. The *receptors* convert stimuli from the human body or the external environment into electrical impulses that convey information to the neural net (brain). The *effectors* convert electrical impulses generated by the neural net into discernible responses as system outputs.

The struggle to understand the brain has been made easier because of the pioneering work of Ramón y Cajál (1911), who introduced the idea of *neurons* as structural constituents of the brain. Typically, neurons are five to six orders of magnitude slower than silicon logic gates; events in a silicon chip happen in the nanosecond (10^{-9} s) range, whereas neural events happen in the millisecond (10^{-3} s) range. However, the brain makes up for the relatively slow rate of operation of a neuron by having a truly staggering number of neurons (nerve cells) with massive interconnections between them. It is estimated that there are approximately 10 billion neurons in the human cortex, and 60 trillion synapses or connections (Shepherd and Koch, 1990). The net result is that the brain is an enormously efficient structure. Specifically, the *energetic efficiency* of the brain is approximately 10^{-16} joules (J) per operation per second, whereas the corresponding value for the best computers in use today is about 10^{-6} joules per operation per second (Faggin, 1991).

Synapses are elementary structural and functional units that mediate the interactions between neurons. The most common kind of synapse is a *chemical synapse*, which operates as follows. A presynaptic process liberates a *transmitter* substance that diffuses across the synaptic junction between neurons and then acts on a postsynaptic process. Thus a synapse converts a presynaptic electrical signal into a chemical signal and then

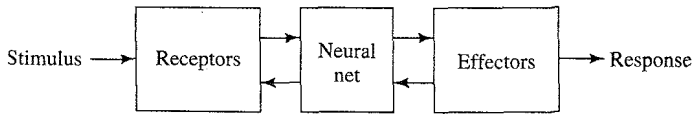

FIGURE 1.1 Block diagram representation of nervous system.

back into a postsynaptic electrical signal (Shepherd and Koch, 1990). In electrical terminology, such an element is said to be a *nonreciprocal two-port device*. In traditional descriptions of neural organization, it is assumed that a synapse is a simple connection that can impose *excitation* or *inhibition*, but not both on the receptive neuron.

Earlier we mentioned that plasticity permits the developing nervous system to adapt to its surrounding environment (Eggermont, 1990; Churchland and Sejnowski, 1992). In an adult brain, plasticity may be accounted for by two mechanisms: the creation of new synaptic connections between neurons, and the modification of existing synapses. *Axons*, the transmission lines, and *dendrites*, the receptive zones, constitute two types of cell filaments that are distinguished on morphological grounds; an axon has a smoother surface, fewer branches, and greater length, whereas a dendrite (so called because of its resemblance to a tree) has an irregular surface and more branches (Freeman, 1975). Neurons come in a wide variety of shapes and sizes in different parts of the brain. Figure 1.2 illustrates the shape of a *pyramidal cell*, which is one of the most common types of cortical neurons. Like many other types of neurons, it receives most of its inputs through dendritic spines; see the segment of dendrite in the insert in Fig. 1.2 for detail. The pyramidal cell can receive 10,000 or more synaptic contacts and it can project onto thousands of target cells.

The majority of neurons encode their outputs as a series of brief voltage pulses. These pulses, commonly known as *action potentials* or *spikes,* originate at or close to the cell body of neurons and then propagate across the individual neurons at constant velocity and amplitude. The reasons for the use of action potentials for communication among neurons are based on the physics of axons. The axon of a neuron is very long and thin and is characterized by high electrical resistance and very large capacitance. Both of these elements are distributed across the axon. The axon may therefore be modeled as an RC transmission line, hence the common use of "cable equation" as the terminology for describing signal propagation along an axon. Analysis of this propagation mechanism reveals that when a voltage is applied at one end of the axon it decays exponentially with distance, dropping to an insignificant level by the time it reaches the other end. The action potentials provide a way to circumvent this transmission problem (Anderson, 1995).

In the brain there are both small-scale and large-scale anatomical organizations, and different functions take place at lower and higher levels. Figure 1.3 shows a hierarchy of interwoven levels of organization that has emerged from the extensive work done on the analysis of local regions in the brain (Shepherd and Koch, 1990; Churchland and Sejnowski, 1992). The *synapses* represent the most fundamental level, depending on molecules and ions for their action. At the next levels we have neural microcircuits, dendritic trees, and then neurons. A *neural microcircuit* refers to an assembly of synapses organized into patterns of connectivity to produce a functional operation of interest. A neural microcircuit may be likened to a silicon chip made up of an assembly of transistors. The smallest size of microcircuits is measured in micrometers (μm), and their fastest speed of operation is measured in milliseconds. The neural microcircuits are grouped to form *dendritic subunits* within the *dendritic trees* of individual neurons. The whole *neuron*, about 100 μm in size, contains several dendritic subunits. At the next level of complexity we have *local circuits* (about 1 mm in size) made up of neurons with similar or different properties; these neural assemblies perform

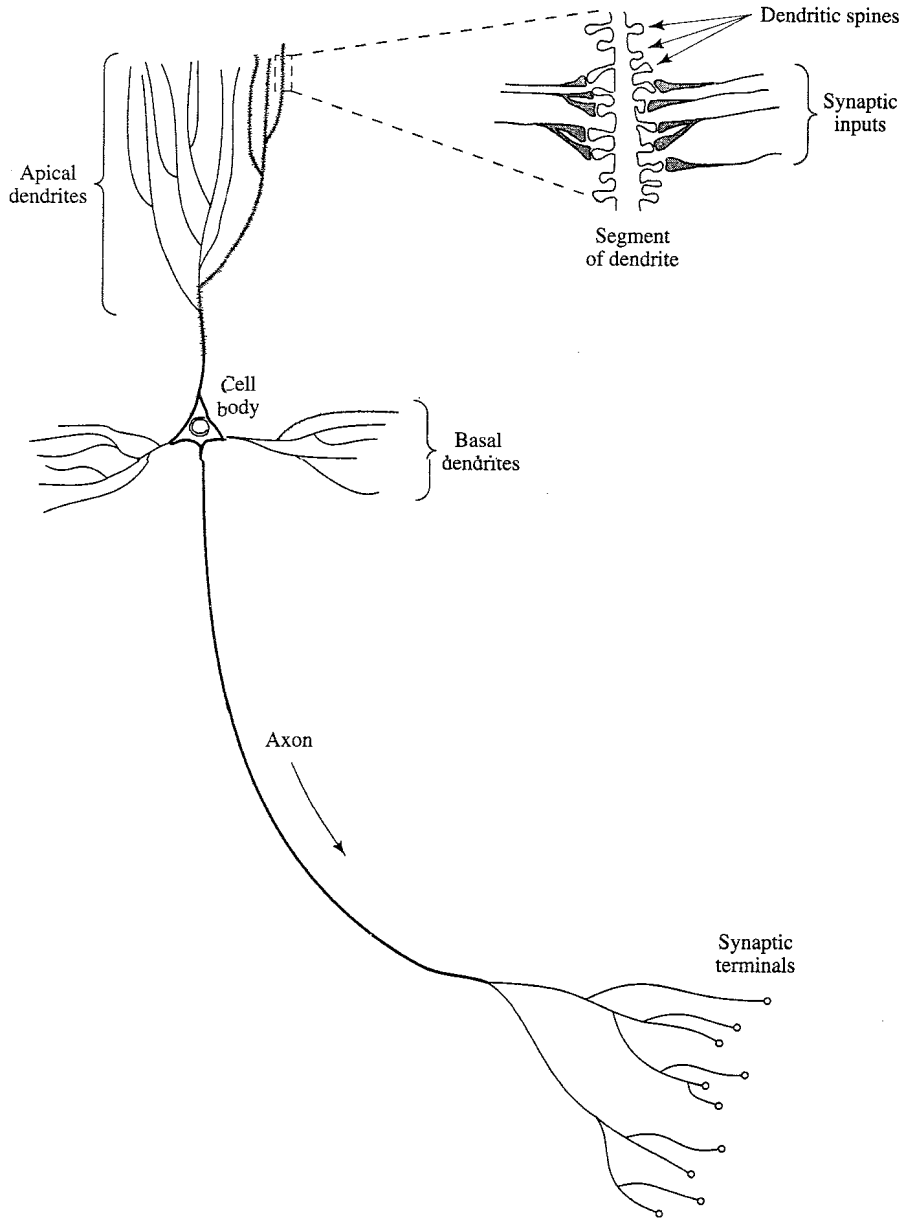

FIGURE 1.2 The pyramidal cell.

operations characteristic of a localized region in the brain. This is followed by *interregional circuits* made up of pathways, columns, and topographic maps, which involve multiple regions located in different parts of the brain.

Topographic maps are organized to respond to incoming sensory information. These maps are often arranged in sheets, as in the *superior colliculus*, where the visual,

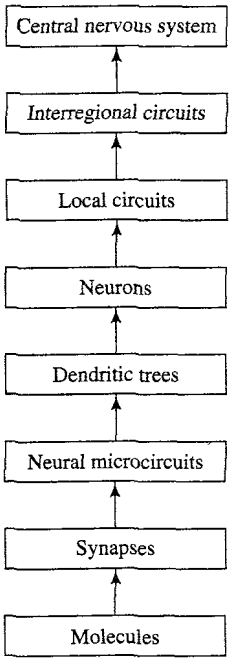

FIGURE 1.3 Structural organization of levels in the brain.

auditory, and somatosensory maps are stacked in adjacent layers in such a way that stimuli from corresponding points in space lie above or below each other. Figure 1.4 presents a cytoarchitectural map of the cerebral cortex as worked out by Brodmann (Brodal, 1981). This figure shows clearly that different sensory inputs (motor, somatosensory, visual, auditory, etc.) are mapped onto corresponding areas of the cerebral cortex in an orderly fashion. At the final level of complexity, the topographic maps and other interregional circuits mediate specific types of behavior in the *central nervous system.*

It is important to recognize that the structural levels of organization described herein are a unique characteristic of the brain. They are nowhere to be found in a digital computer, and we are nowhere close to re-creating them with artificial neural networks. Nevertheless, we are inching our way toward a hierarchy of computational levels similar to that described in Fig. 1.3. The artificial neurons we use to build our neural networks are truly primitive in comparison to those found in the brain. The neural networks we are presently able to design are just as primitive compared to the local circuits and the interregional circuits in the brain. What is really satisfying, however, is the remarkable progress that we have made on so many fronts during the past two decades. With neurobiological analogy as the source of inspiration, and the wealth of theoretical and technological tools that we are bringing together, it is certain that in another decade our understanding of artificial neural networks will be much more sophisticated than it is today.

Our primary interest in this book is confined to the study of artificial neural networks from an engineering perspective.[2] We begin the study by describing the models of (artificial) neurons that form the basis of the neural networks considered in subsequent chapters of the book.

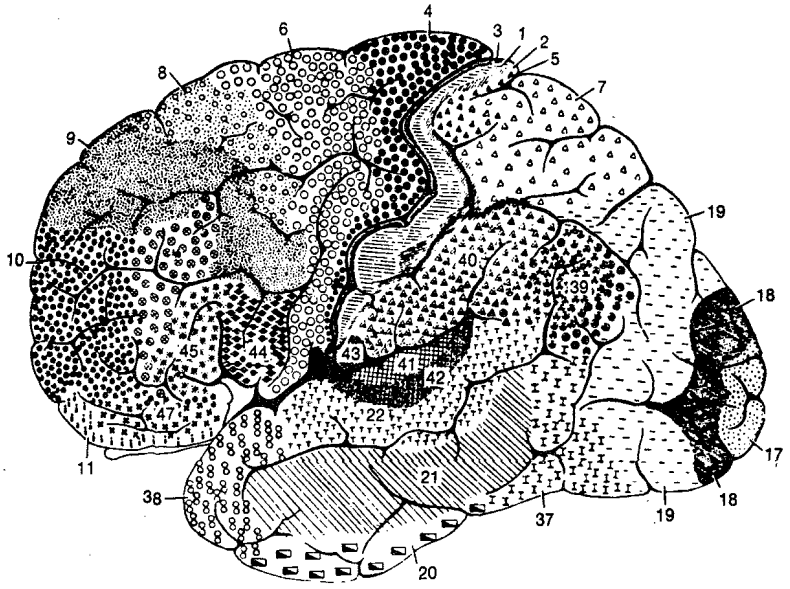

FIGURE 1.4 Cytoarchitectural map of the cerebral cortex. The different areas are identified by the thickness of their layers and types of cells within them. Some of the most important specific areas are as follows. Motor cortex: motor strip, area 4; premotor area, area 6; frontal eye fields, area 8. Somatosensory cortex: areas 3, 1, 2. Visual cortex: areas 17, 18, 19. Auditory cortex: area 41 and 42. (From A. Brodal, 1981; with permission of Oxford University Press.)

1.3 MODELS OF A NEURON

A *neuron* is an information-processing unit that is fundamental to the operation of a neural network. The block diagram of Fig. 1.5 shows the *model* of a neuron, which forms the basis for designing (artificial) neural networks. Here we identify three basic elements of the neuronal model:

1. A set of *synapses* or *connecting links*, each of which is characterized by a *weight* or *strength* of its own. Specifically, a signal x_j at the input of synapse j connected to neuron k is multiplied by the synaptic weight w_{kj}. It is important to make a note of the manner in which the subscripts of the synaptic weight w_{kj} are written. The first subscript refers to the neuron in question and the second subscript refers to the input end of the synapse to which the weight refers. Unlike a synapse in the brain, the synaptic weight of an artificial neuron may lie in a range that includes negative as well as positive values.
2. An *adder* for summing the input signals, weighted by the respective synapses of the neuron; the operations described here constitute a *linear combiner.*
3. An *activation function* for limiting the amplitude of the output of a neuron. The activation function is also referred to as a *squashing function* in that it squashes (limits) the permissible amplitude range of the output signal to some finite value.

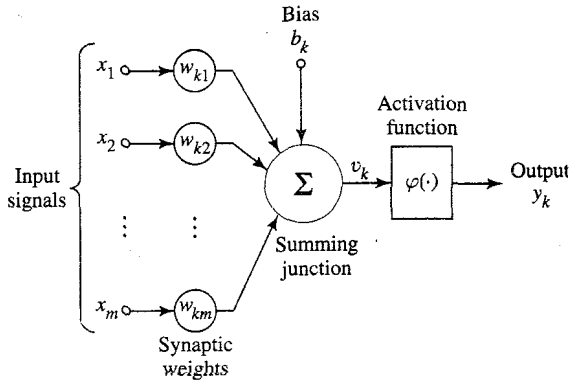

FIGURE 1.5 Nonlinear model of a neuron.

Typically, the normalized amplitude range of the output of a neuron is written as the closed unit interval [0,1] or alternatively [−1,1].

The neuronal model of Fig. 1.5 also includes an externally applied *bias,* denoted by b_k. The bias b_k has the effect of increasing or lowering the net input of the activation function, depending on whether it is positive or negative, respectively.

In mathematical terms, we may describe a neuron k by writing the following pair of equations:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{1.1}$$

and

$$y_k = \varphi(u_k + b_k) \tag{1.2}$$

where $x_1, x_2, \ldots, x_m$ are the input signals; $w_{k1}, w_{k2}, \ldots, w_{km}$ are the synaptic weights of neuron k; u_k is the *linear combiner output* due to the input signals; b_k is the bias; $\varphi(\cdot)$ is the *activation function*; and y_k is the output signal of the neuron. The use of bias b_k has the effect of applying an *affine transformation* to the output u_k of the linear combiner in the model of Fig. 1.5, as shown by

$$v_k = u_k + b_k \tag{1.3}$$

In particular, depending on whether the bias b_k is positive or negative, the relationship between the *induced local field* or *activation potential* v_k of neuron k and the linear combiner output u_k is modified in the manner illustrated in Fig. 1.6; hereafter the term "induced local field" is used. Note that as a result of this affine transformation, the graph of v_k versus u_k no longer passes through the origin.

The bias b_k is an external parameter of artificial neuron k. We may account for its presence as in Eq. (1.2). Equivalently, we may formulate the combination of Eqs. (1.1) to (1.3) as follows:

$$v_k = \sum_{j=0}^{m} w_{kj} x_j \tag{1.4}$$

and

$$y_k = \varphi(v_k) \tag{1.5}$$

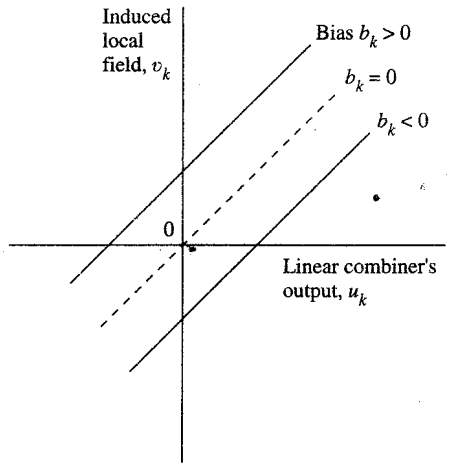

FIGURE 1.6 Affine transformation produced by the presence of a bias; note that $v_k = b_k$ at $u_k = 0$.

In Eq. (1.4) we have added a new synapse. Its input is

$$x_0 = +1 \tag{1.6}$$

and its weight is

$$w_{k0} = b_k \tag{1.7}$$

We may therefore reformulate the model of neruon k as in Fig. 1.7. In this figure, the effect of the bias is accounted for by doing two things: (1) adding a new input signal fixed at $+1$, and (2) adding a new synaptic weight equal to the bias b_k. Although the models of Figs. 1.5 and 1.7 are different in appearance, they are mathematically equivalent.

Types of Activation Function

The activation function, denoted by $\varphi(v)$, defines the output of a neuron in terms of the induced local field v. Here we identify three basic types of activation functions:

1. *Threshold Function.* For this type of activation function, described in Fig. 1.8a, we have

$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases} \tag{1.8}$$

In engineering literature, this form of a threshold function is commonly referred to as a *Heaviside function.* Correspondingly, the output of neuron k employing such a threshold function is expressed as

$$y_k = \begin{cases} 1 & \text{if } v_k \geq 0 \\ 0 & \text{if } v_k < 0 \end{cases} \tag{1.9}$$

where v_k is the induced local field of the neuron; that is,

$$v_k = \sum_{j=1}^{m} w_{kj} x_j + b_k \tag{1.10}$$

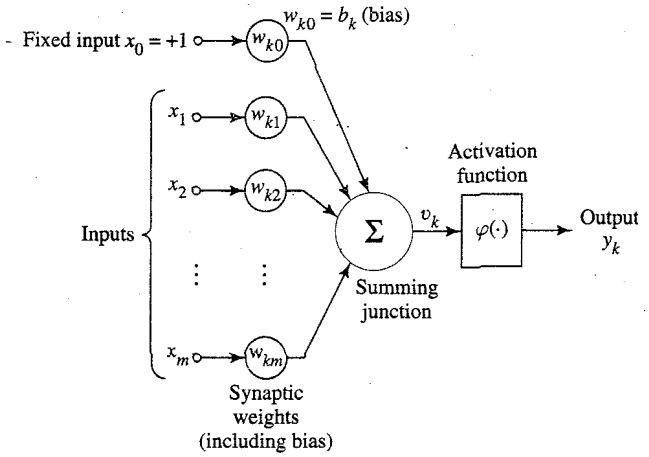

FIGURE 1.7 Another nonlinear model of a neuron.

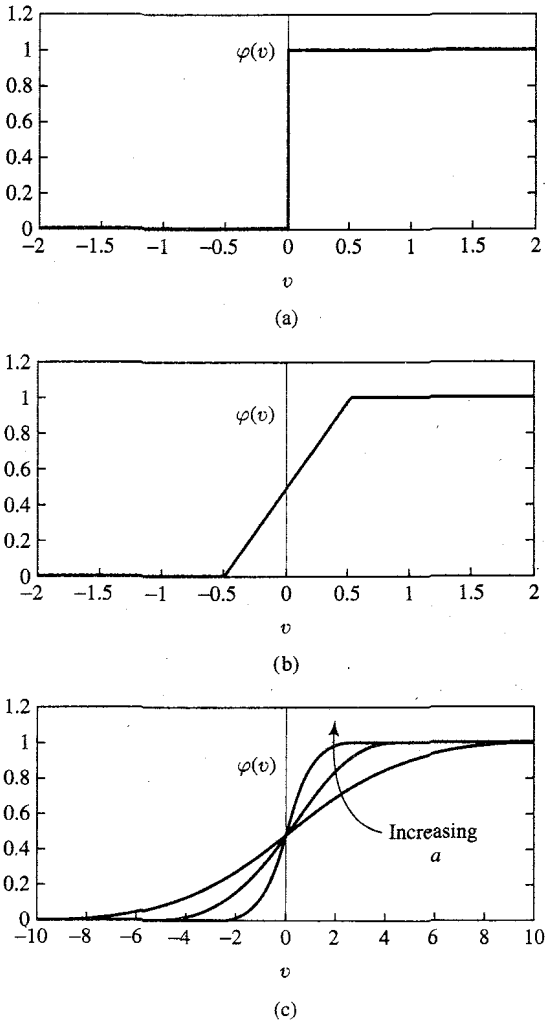

FIGURE 1.8 (a) Threshold function. (b) Piecewise-linear function. (c) Sigmoid function for varying slope parameter a.

Such a neuron is referred to in the literature as the *McCulloch–Pitts model*, in recognition of the pioneering work done by McCulloch and Pitts (1943). In this model, the output of a neuron takes on the value of 1 if the induced local field of that neuron is nonnegative, and 0 otherwise. This statement describes the *all-or-none property* of the McCulloch–Pitts model.

2. *Piecewise-Linear Function.* For the piecewise-linear function described in Fig. 1.8b we have

$$\varphi(v) = \begin{cases} 1, & v \geq +\frac{1}{2} \\ v, & +\frac{1}{2} > v > -\frac{1}{2} \\ 0, & v \leq -\frac{1}{2} \end{cases} \tag{1.11}$$

where the amplification factor inside the linear region of operation is assumed to be unity. This form of an activation function may be viewed as an *approximation* to a nonlinear amplifier. The following two situations may be viewed as special forms of the piecewise-linear function:

- A *linear combiner* arises if the linear region of operation is maintained without running into saturation.
- The piecewise-linear function reduces to a *threshold function* if the amplification factor of the linear region is made infinitely large.

3. *Sigmoid Function.* The sigmoid function, whose graph is s-shaped, is by far the most common form of activation function used in the construction of artifical neural networks. It is defined as a strictly increasing function that exhibits a graceful balance between linear and nonlinear behavior.[3] An example of the sigmoid function is the *logistic function*,[4] defined by

$$\varphi(v) = \frac{1}{1 + \exp(-av)} \tag{1.12}$$

where a is the *slope parameter* of the sigmoid function. By varying the parameter a, we obtain sigmoid functions of different slopes, as illustrated in Fig. 1.8c. In fact, the slope at the origin equals $a/4$. In the limit, as the slope parameter approaches infinity, the sigmoid function becomes simply a threshold function. Whereas a threshold function assumes the value of 0 or 1, a sigmoid function assumes a continuous range of values from 0 to 1. Note also that the sigmoid function is differentiable, whereas the threshold function is not. (Differentiability is an important feature of neural network theory, as described in Chapter 4.)

The activation functions defined in Eqs. (1.8), (1.11), and (1.12) range from 0 to +1. It is sometimes desirable to have the activation function range from −1 to +1, in which case the activation function assumes an antisymmetric form with respect to the origin; that is, the activation function is an odd function of the induced local field. Specifically, the threshold function of Eq. (1.8) is now defined as

$$\varphi(v) = \begin{cases} 1 & \text{if } v > 0 \\ 0 & \text{if } v = 0 \\ -1 & \text{if } v < 0 \end{cases} \tag{1.13}$$

which is commonly referred to as the *signum function*. For the corresponding form of a sigmoid function we may use the *hyperbolic tangent function*, defined by

$$\varphi(v) = \tanh(v) \tag{1.14}$$

Allowing an activation function of the sigmoid type to assume negative values as prescribed by Eq. (1.14) has analytic benefits (as shown in Chapter 4).

Stochastic Model of a Neuron

The neuronal model described in Fig. 1.7 is deterministic in that its input-output behavior is precisely defined for all inputs. For some applications of neural networks, it is desirable to base the analysis on a stochastic neuronal model. In an analytically tractable approach, the activation function of the McCulloch–Pitts model is given a probabilistic interpretation. Specifically, a neuron is permitted to reside in only one of two states: +1 or −1, say. The decision for a neuron to *fire* (i.e., switch its state from "off" to "on") is probabilistic. Let x denote the state of the neuron, and $P(v)$ denote the *probability* of firing, where v is the induced local field of the neuron. We may then write

$$x = \begin{cases} +1 & \text{with probability } P(v) \\ -1 & \text{with probability } 1 - P(v) \end{cases}$$

A standard choice for $P(v)$ is the sigmoid-shaped function (Little, 1974):

$$P(v) = \frac{1}{1 + \exp(-v/T)} \tag{1.15}$$

where T is a *pseudotemperature* that is used to control the noise level and therefore the uncertainty in firing. It is important to realize, however, that T is *not* the physical temperature of a neural network, be it a biological or an artificial neural network. Rather, as already stated, we should think of T merely as a parameter that controls the thermal fluctuations representing the effects of synaptic noise. Note that when $T \to 0$, the stochastic neuron described by Eq. (1.15) reduces to a noiseless (i.e., deterministic) form, namely the McCulloch–Pitts model.

1.4 NEURAL NETWORKS VIEWED AS DIRECTED GRAPHS

The *block diagram* of Fig. 1.5 or that of Fig. 1.7 provides a functional description of the various elements that constitute the model of an artificial neuron. We may simplify the appearance of the model by using the idea of signal-flow graphs without sacrificing any of the functional details of the model. Signal-flow graphs with a well-defined set of rules were originally developed by Mason (1953, 1956) for linear networks. The presence of nonlinearity in the model of a neuron limits the scope of their application to neural networks. Nevertheless, signal-flow graphs do provide a neat method for the portrayal of the flow of signals in a neural network, which we pursue in this section.

A *signal-flow graph* is a network of directed *links* (*branches*) that are interconnected at certain points called *nodes*. A typical node j has an associated *node signal* x_j. A typical directed link originates at node j and terminates on node k; it has an associated

transfer function or *transmittance* that specifies the manner in which the signal y_k at node k depends on the signal x_j at node j. The flow of signals in the various parts of the graph is dictated by three basic rules:

Rule 1. A signal flows along a link only in the direction defined by the arrow on the link.

Two different types of links may be distinguished:

- *Synaptic links*, whose behavior is governed by a *linear* input–output relation. Specifically, the node signal x_j is multiplied by the synaptic weight w_{kj} to produce the node signal y_k, as illustrated in Fig. 1.9a.
- *Activation links*, whose behavior is governed in general by a *nonlinear* input–output relation. This form of relationship is illustrated in Fig 1.9b, where $\varphi(\cdot)$ is the nonlinear activation function.

Rule 2. A node signal equals the algebraic sum of all signals entering the pertinent node via the incoming links.

This second rule is illustrated in Fig. 1.9c for the case of *synaptic convergence* or *fan-in*.

Rule 3. The signal at a node is transmitted to each outgoing link originating from that node, with the transmission being entirely independent of the transfer functions of the outgoing links.

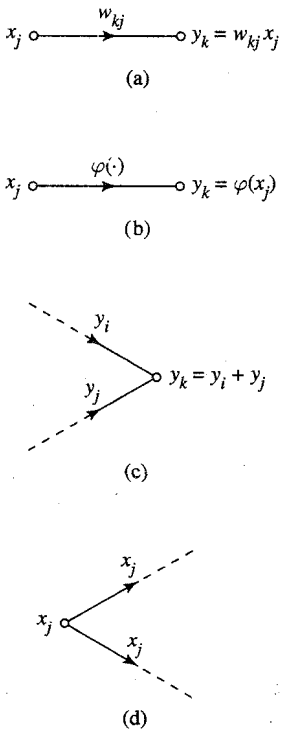

FIGURE 1.9 Illustrating basic rules for the construction of signal-flow graphs.

This third rule is illustrated in Fig. 1.9d for the case of *synaptic divergence* or *fan-out.*

For example, using these rules we may construct the signal-flow graph of Fig 1.10 as the model of a neuron, corresponding to the block diagram of Fig. 1.7. The representation shown in Fig. 1.10 is clearly simpler in appearance than that of Fig. 1.7, yet it contains all the functional details depicted in the latter diagram. Note that in both figures, the input $x_0 = +1$ and the associated synaptic weight $w_{k0} = b_k$, where b_k is the bias applied to neuron k.

Indeed, based on the signal-flow graph of Fig. 1.10 as the model of a neuron, we may now offer the following mathematical definition of a neural network:

> *A neural network is a directed graph consisting of nodes with interconnecting synaptic and activation links, and is characterized by four properties:*
>
> 1. *Each neuron is represented by a set of linear synaptic links, an externally applied bias, and a possibly nonlinear activation link. The bias is represented by a synaptic link connected to an input fixed at +1.*
> 2. *The synaptic links of a neuron weight their respective input signals.*
> 3. *The weighted sum of the input signals defines the induced local field of the neuron in question.*
> 4. *The activation link squashes the induced local field of the neuron to produce an output.*

The state of the neuron may be defined in terms of its induced local field or its output signal.

A directed graph so defined is *complete* in the sense that it describes not only the signal flow from neuron to neuron, but also the signal flow inside each neuron. When, however, the focus of attention is restricted to signal flow from neuron to neuron, we may use a reduced form of this graph by omitting the details of signal flow inside the individual neurons. Such a directed graph is said to be *partially complete.* It is characterized as follows:

1. *Source nodes* supply input signals to the graph.
2. Each neuron is represented by a single node called a *computation node.*
3. The *communication links* interconnecting the source and computation nodes of the graph carry no weight; they merely provide directions of signal flow in the graph.

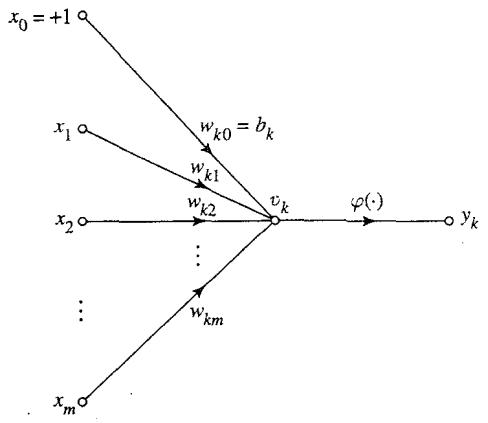

FIGURE 1.10 Signal-flow graph of a neuron.

A partially complete directed graph defined in this way is referred to as an *architectural graph,* describing the layout of the neural network. It is illustrated in Fig. 1.11 for the simple case of a single neuron with m source nodes and a single node fixed at +1 for the bias. Note that the computation node representing the neuron is shown shaded, and the source node is shown as a small square. This convention is followed throughout the book. More elaborate examples of architectural layouts are presented in Section 1.6.

To sum up, we have three graphical representations of a neural network:

- Block diagram, providing a functional description of the network.
- Signal-flow graph, providing a complete description of signal flow in the network.
- Architectural graph, describing the network layout.

1.5 FEEDBACK

Feedback is said to exist in a dynamic system whenever the output of an element in the system influences in part the input applied to that particular element, thereby giving rise to one or more closed paths for the transmission of signals around the system. Indeed, feedback occurs in almost every part of the nervous system of every animal (Freeman, 1975). Moreover, it plays a major role in the study of a special class of neural networks known as *recurrent networks.* Figure 1.12 shows the signal-flow graph of a *single-loop feedback system,* where the input signal $x_j(n)$, internal signal $x'_j(n)$, and output signal $y_k(n)$ are functions of the discrete-time variable n. The system is assumed to be *linear,* consisting of a forward path and a feedback path that are characterized by the "operators" A and B, respectively. In particular, the output of the forward channel determines in part its own output through the feedback channel. From Fig 1.12 we readily note the following input–output relationships:

$$y_k(n) = A[x'_j(n)] \tag{1.16}$$

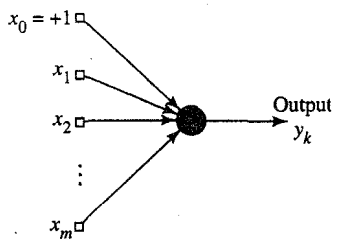

FIGURE 1.11 Architectural graph of a neuron.

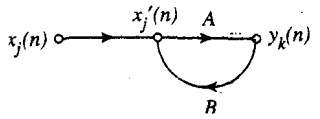

FIGURE 1.12 Signal-flow graph of a single-loop feedback system.

$$x_j'(n) = x_j(n) + B[y_k(n)] \tag{1.17}$$

where the square brackets are included to emphasize that A and B act as operators. Eliminating $x_j'(n)$ between Eqs. (1.16) and (1.17), we get

$$y_k(n) = \frac{A}{1 - AB}[x_j(n)] \tag{1.18}$$

We refer to $A/(1 - AB)$ as the *closed-loop operator* of the system, and to AB as the *open-loop operator*. In general, the open-loop operator is noncommutative in that $BA \neq AB$.

Consider, for example, the single-loop feedback system shown in Fig. 1.13, for which A is a fixed weight, w; and B is a *unit-delay operator*, z^{-1}, whose output is delayed with respect to the input by one time unit. We may then express the closed-loop operator of the system as

$$\begin{aligned}\frac{A}{1 - AB} &= \frac{w}{1 - wz^{-1}} \\ &= w(1 - wz^{-1})^{-1}\end{aligned}$$

Using the binomial expansion for $(1 - wz^{-1})^{-1}$, we may rewrite the closed-loop operator of the system as

$$\frac{A}{1 - AB} = w\sum_{l=0}^{\infty} w^l z^{-l} \tag{1.19}$$

Hence, substituting Eq. (1.19) in (1.18), we get

$$y_k(n) = w\sum_{l=0}^{\infty} w^l z^{-l}[x_j(n)] \tag{1.20}$$

where again we have included square brackets to emphasize the fact that z^{-1} is an operator. In particular, from the definition of z^{-1} we have

$$z^{-l}[x_j(n)] = x_j(n - l) \tag{1.21}$$

where $x_j(n - l)$ is a sample of the input signal delayed by l time units. Accordingly, we may express the output signal $y_k(n)$ as an infinite weighted summation of present and past samples of the input signal $x_j(n)$, as shown by

$$y_k(n) = \sum_{l=0}^{\infty} w^{l+1} x_j(n - l) \tag{1.22}$$

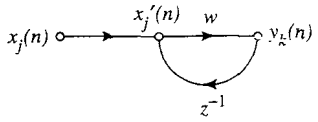

FIGURE 1.13 Signal-flow graph of a first-order, infinite-duration impulse response (IIR) filter.

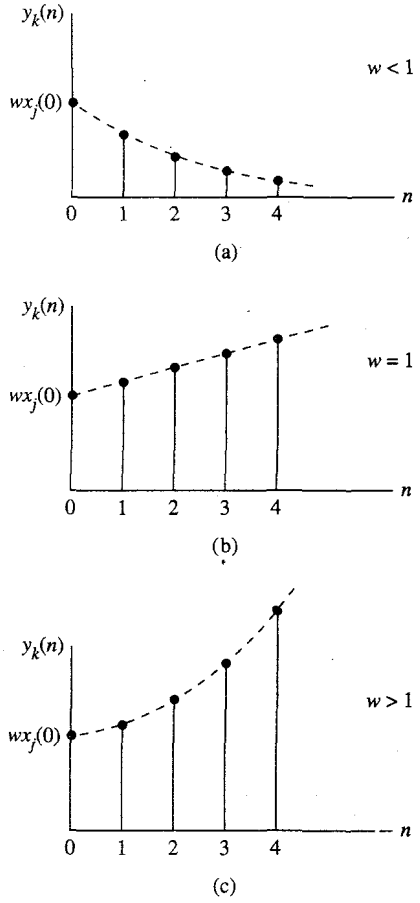

FIGURE 1.14 Time response of Fig. 1.13 for three different values of forward weight w. (a) Stable. (b) Linear divergence. (c) Exponential divergence.

We now see clearly that the dynamic behavior of the system is controlled by the weight w. In particular, we may distinguish two specific cases:

1. $|w| < 1$, for which the output signal $y_k(n)$ is exponentially *convergent*; that is, the system is *stable*. This is illustrated in Fig. 1.14a for a positive w.
2. $|w| \geq 1$, for which the output signal $y_k(n)$ is *divergent*; that is, the system is *unstable*. If $|w| = 1$ the divergence is linear as in Fig. 1.14b, and if $|w| > 1$ the divergence is exponential as in Fig 1.14c.

Stability features prominently in the study of feedback systems.

The case of $|w| < 1$ corresponds to a system with *infinite memory* in the sense that the output of the system depends on samples of the input extending into the infinite past. Moreover, the memory is *fading* in that the influence of a past sample is reduced exponentially with time n.

The analysis of the dynamic behavior of neural networks involving the application of feedback is unfortunately complicated by virute of the fact that the processing units used for the construction of the network are usually *nonlinear*. Further consideration of this issue is deferred to the latter part of the book.

1.6 NETWORK ARCHITECTURES

The manner in which the neurons of a neural network are structured is intimately linked with the learning algorithm used to train the network. We may therefore speak of learning algorithms (rules) used in the design of neural networks as being *structured*. The classification of learning algorithms is considered in the next chapter, and the development of different learning algorithms is taken up in subsequent chapters of the book. In this section we focus our attention on network architectures (structures).

In general, we may identify three fundamentally different classes of network architectures:

1. Single-Layer Feedforward Networks

In a *layered* neural network the neurons are organized in the form of layers. In the simplest form of a layered network, we have an *input layer* of source nodes that projects onto an *output layer* of neurons (computation nodes), but not vice versa. In other words, this network is strictly a *feedforward* or *acyclic* type. It is illustrated in Fig. 1.15 for the case of four nodes in both the input and output layers. Such a network is called a *single-layer network*, with the designation "single-layer" referring to the output layer of computation nodes (neurons). We do not count the input layer of source nodes because no computation is performed there.

2. Multilayer Feedforward Networks

The second class of a feedforward neural network distinguishes itself by the presence of one or more *hidden layers*, whose computation nodes are correspondingly called *hidden neurons* or *hidden units*. The function of hidden neurons is to intervene between the external input and the network output in some useful manner. By adding one or

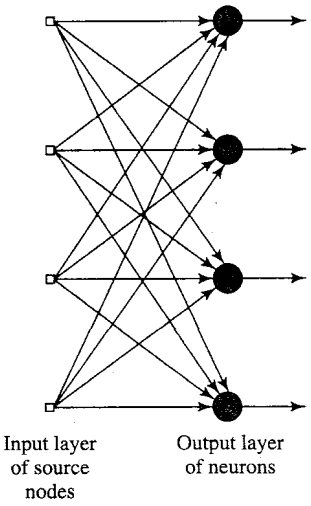

FIGURE 1.15 Feedforward or acyclic network with a single layer of neurons.

more hidden layers, the network is enabled to extract higher-order statistics. In a rather loose sense the network acquires a *global* perspective despite its local connectivity due to the extra set of synaptic connections and the extra dimension of neural interactions (Churchland and Sejnowski, 1992). The ability of hidden neurons to extract higher-order statistics is particularly valuable when the size of the input layer is large.

The source nodes in the input layer of the network supply respective elements of the activation pattern (input vector), which constitute the input signals applied to the neurons (computation nodes) in the second layer (i.e., the first hidden layer). The output signals of the second layer are used as inputs to the third layer, and so on for the rest of the network. Typically the neurons in each layer of the network have as their inputs the output signals of the preceding layer only. The set of output signals of the neurons in the output (final) layer of the network constitutes the overall response of the network to the activation pattern supplied by the source nodes in the input (first) layer. The architectural graph in Fig. 1.16 illustrates the layout of a multilayer feedforward neural network for the case of a single hidden layer. For brevity the network in Fig. 1.16 is referred to as a 10-4-2 network because it has 10 source nodes, 4 hidden neurons, and 2 output neurons. As another example, a feedforward network with m source nodes, h_1 neurons in the first hidden layer, h_2 neurons in the second hidden layer, and q neurons in the output layer is referred to as an m-h_1-h_2-q network.

The neural network in Fig. 1.16 is said to be *fully connected* in the sense that every node in each layer of the network is connected to every other node in the adjacent forward layer. If, however, some of the communication links (synaptic connections) are missing from the network, we say that the network is *partially connected.*

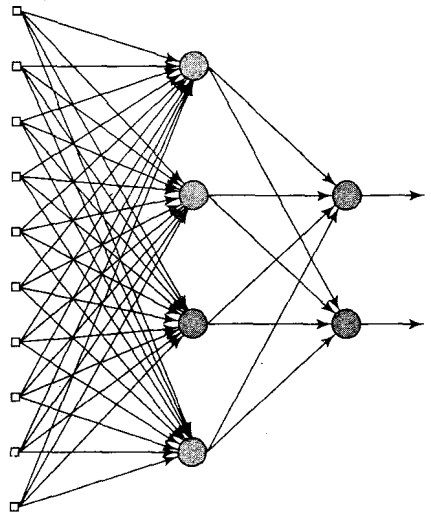

FIGURE 1.16 Fully connected feedforward or acyclic network with one hidden layer and one output layer.

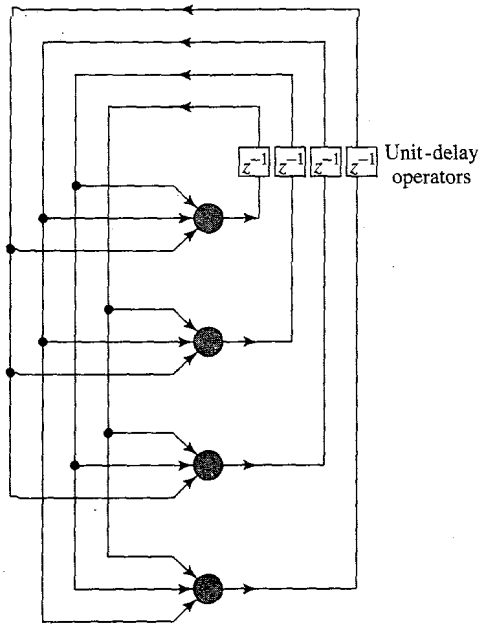

FIGURE 1.17 Recurrent network with no self-feedback loops and no hidden neurons.

3. Recurrent Networks

A *recurrent neural network* distinguishes itself from a feedforward neural network in that it has at least one *feedback* loop. For example, a recurrent network may consist of a single layer of neurons with each neuron feeding its output signal back to the inputs of all the other neurons, as illustrated in the architectural graph in Fig. 1.17. In the structure depicted in this figure there are *no* self-feedback loops in the network; self-feedback refers to a situation where the output of a neuron is fed back into its own input. The recurrent network illustrated in Fig. 1.17 also has *no* hidden neurons. In Fig. 1.18 we illustrate another class of recurrent networks with hidden neurons. The feedback connections shown in Fig. 1.18 originate from the hidden neurons as well as from the output neurons.

The presence of feedback loops, whether in the recurrent structure of Fig. 1.17 or that of Fig. 1.18, has a profound impact on the learning capability of the network and on its performance. Moreover, the feedback loops involve the use of particular branches composed of *unit-delay elements* (denoted by z^{-1}), which result in a nonlinear dynamical behavior, assuming that the neural network contains nonlinear units.

1.7 KNOWLEDGE REPRESENTATION

In Section 1.1 we used the term "knowledge" in the definition of a neural network without an explicit description of what we mean by it. We now take care of this matter by offering the following generic definition (Fischler and Firschein, 1987):

> *Knowledge refers to stored information or models used by a person or machine to interpret, predict, and appropriately respond to the outside world.*

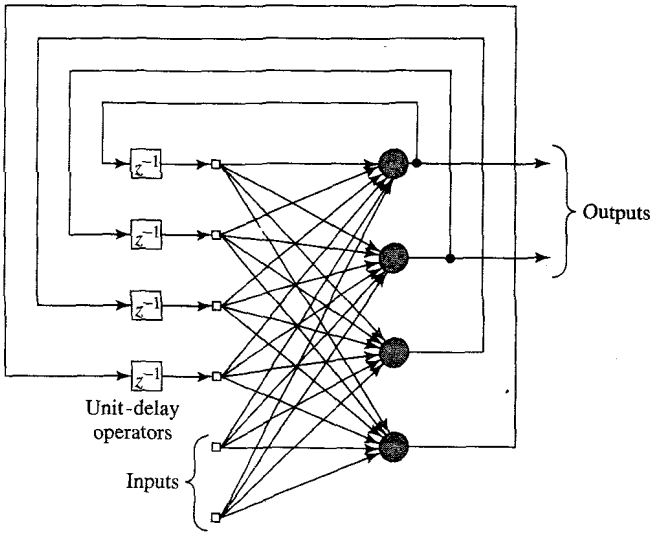

FIGURE 1.18 Recurrent network with hidden neurons.

The primary characteristics of *knowledge representation* are twofold: (1) what information is actually made explicit; and (2) how the information is physically encoded for subsequent use. By the very nature of it, therefore, knowledge representation is goal directed. In real-world applications of "intelligent" machines, it can be said that a good solution depends on a good representation of knowledge (Woods, 1986). So it is with neural networks that represent a special class of intelligent machines. Typically, however, the possible forms of representation from the inputs to internal network parameters are highly diverse, which tends to make the development of a satisfactory solution by means of a neural network a real design challenge.

A major task for a neural network is to learn a model of the world (environment) in which it is embedded and to maintain the model sufficiently consistent with the real world so as to achieve the specified goals of the application of interest. Knowledge of the world consists of two kinds of information:

1. The known world state, represented by facts about what is and what has been known; this form of knowledge is referred to as *prior information.*
2. Observations (measurements) of the world, obtained by means of sensors designed to probe the environment in which the neural network is supposed to operate. Ordinarily these observations are inherently noisy, being subject to errors due to sensor noise and system imperfections. In any event, the observations so obtained provide the pool of information from which the *examples* used to train the neural network are drawn.

The examples can be *labeled* or *unlabeled.* In labeled examples, each example representing an *input signal* is paired with a corresponding *desired response* (i.e., target output). On the other hand, unlabeled examples consist of different realizations of the input signal by itself. In any event, a set of examples, labeled or otherwise, represents knowledge about the environment of interest that a neural network can learn through training.

A set of input–output pairs, with each pair consisting of an input signal and the corresponding desired response, is referred to as a *set of training data* or *training sample.* To illustrate how such a data set can be used, consider, for example, the *handwritten digit recognition problem.* In this problem, the input signal consists of an image with black or white pixels, with each image representing one of 10 digits that are well separated from the background. The desired response is defined by the "identity" of the particular digit whose image is presented to the network as the input signal. Typically, the training sample consists of a large variety of handwritten digits that are representative of a real-world situation. Given such a set of examples, the design of a neural network may proceed as follows:

- First, an appropriate architecture is selected for the neural network, with an input layer consisting of source nodes equal in number to the pixels of an input image, and an output layer consisting of 10 neurons (one for each digit). A subset of examples is then used to train the network by means of a suitable algorithm. This phase of the network design is called *learning.*
- Second, the recognition performance of the trained network is *tested* with data not seen before. Specifically, an input image is presented to the network, but this time it is not told the indentity of the digit to which that particular image belongs. The performance of the network is then assessed by comparing the digit recognition reported by the network with the actual identity of the digit in question. This second phase of the network operation is called *generalization*, a term borrowed from psychology.

Herein lies a fundamental difference between the design of a neural network and that of its classical information-processing counterpart (pattern classifier). In the latter case, we usually proceed by first formulating a mathematical model of environmental observations, validating the model with real data, and then building the design on the basis of the model. In contrast, the design of a neural network is based directly on real-life data, with the *data set being permitted to speak for itself.* Thus, the neural network not only provides the implicit model of the environment in which it is embedded, but also performs the information-processing function of interest.

The examples used to train a neural network may consist of both *positive* and *negative* examples. For instance, in a passive sonar detection problem, positive examples pertain to input training data that contain the target of interest (e.g., a submarine). Now, in a passive sonar environment, the possible presence of marine life in the test data is known to cause occasional false alarms. To alleviate this problem, negative examples (e.g., echos from marine life) are included in the training data to teach the network not to confuse marine life with the target.

In a neural network of specified architecture, knowledge representation of the surrounding environment is defined by the values taken on by the free parameters (i.e., synaptic weights and biases) of the network. The form of this knowledge representation constitutes the very design of the neural network, and therefore holds the key to its performance.

The subject of knowledge representation inside an artificial network is, however, very complicated. Nevertheless, there are four rules for knowledge representation that are of a general commonsense nature (Anderson, 1988).

Rule 1. Similar inputs from similar classes should usually produce similar representations inside the network, and should therefore be classified as belonging to the same category.

There are a plethora of measures for determining the "similarity" between inputs. A commonly used measure of similarity is based on the concept of Euclidian distance. To be specific, let $\mathbf{x}_i$ denote an m-by-1 vector

$$\mathbf{x}_i = [x_{i1}, x_{i2}, \ldots, x_{im}]^T$$

all of whose elements are real; the superscript T denotes matrix *transposition*. The vector $\mathbf{x}_i$ defines a point in an m-dimensional space called *Euclidean space* and denoted by $\mathbb{R}^m$. The *Euclidean distance* between a pair of m-by-1 vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ is defined by

$$\begin{aligned} d(\mathbf{x}_i, \mathbf{x}_j) &= \|\mathbf{x}_i - \mathbf{x}_j\| \\ &= \left[\sum_{k=1}^{m} (x_{ik} - x_{jk})^2 \right]^{1/2} \end{aligned} \tag{1.23}$$

where x_{ik} and x_{jk} are the kth elements of the input vectors $\mathbf{x}_i$ and $\mathbf{x}_j$, respectively. Correspondingly, the similarity between the inputs represented by the vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ is defined as the *reciprocal* of the Euclidean distance $d(\mathbf{x}_i, \mathbf{x}_j)$. The closer the individual elements of the input vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ are to each other, the smaller the Euclidean distance $d(\mathbf{x}_i, \mathbf{x}_j)$ will be, and therefore the greater the similarity between the vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ will be. Rule 1 states that if the vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ are similar, they should be assigned to the same category (class).

Another measure of similarity is based on the idea of a *dot product* or *inner product* that is also borrowed from matrix algebra. Given a pair of vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ of the same dimension, their inner product is $\mathbf{x}_i^T \mathbf{x}_j$ written in expanded form as follows:

$$\begin{aligned} (\mathbf{x}_i, \mathbf{x}_j) &= \mathbf{x}_i^T \mathbf{x}_j \\ &= \sum_{k=1}^{m} x_{ik} x_{jk} \end{aligned} \tag{1.24}$$

The inner product $(\mathbf{x}_i, \mathbf{x}_j)$ divided by $\|\mathbf{x}_i\| \, \|\mathbf{x}_j\|$ is the cosine of the angle subtended between the vectors $\mathbf{x}_i$ and $\mathbf{x}_j$.

The two measures of similarity defined here are indeed intimately related to each other, as illustrated in Fig. 1.19. The Euclidean distance $\|\mathbf{x}_i - \mathbf{x}_j\|$ between the vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ is related to the "projection" of the vector $\mathbf{x}_i$ onto the vector $\mathbf{x}_j$. Figure 1.19 shows

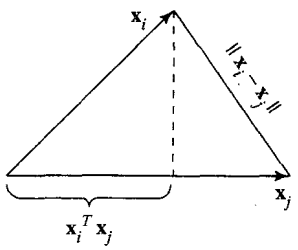

FIGURE 1.19 Illustrating the relationship between inner product and Euclidean distance as measures of similiarity between patterns.

clearly that the smaller the Euclidean distance $\|\mathbf{x}_i - \mathbf{x}_j\|$ and therefore the more similar the vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ are, the larger the inner product $\mathbf{x}_i^T\mathbf{x}_j$ will be.

To put this relationship on a formal basis, we first normalize the vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ to have unit length, that is,

$$\|\mathbf{x}_i\| = \|\mathbf{x}_j\| = 1$$

We may then use Eq. (1.23) to write

$$\begin{aligned} d^2(\mathbf{x}_i, \mathbf{x}_j) &= (\mathbf{x}_i - \mathbf{x}_j)^T(\mathbf{x}_i - \mathbf{x}_j) \\ &= 2 - 2\mathbf{x}_i^T\mathbf{x}_j \end{aligned} \tag{1.25}$$

Equation (1.25) shows that minimization of the Euclidean distance $d(\mathbf{x}_i,\mathbf{x}_j)$ corresponds to maximization of the inner product $(\mathbf{x}_i,\mathbf{x}_j)$ and, therefore, the similarity between the vectors $\mathbf{x}_i$ and $\mathbf{x}_j$.

The Euclidean distance and inner product described here are defined in deterministic terms. What if the vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ are drawn from two different populations (pools) of data? To be specific, suppose that the difference between these two populations lies solely in their mean vectors. Let $\boldsymbol{\mu}_i$ and $\boldsymbol{\mu}_j$ denote the mean values of the vectors $\mathbf{x}_i$ and $\mathbf{x}_j$, respectively. That is,

$$\boldsymbol{\mu}_i = E[\mathbf{x}_i] \tag{1.26}$$

where E is the statistical expectation operator. The mean vector $\boldsymbol{\mu}_j$ is similarly defined. For a measure of the distance between these two populations, we may use the *Mahalanobis distance* denoted by d_{ij}. The squared value of this distance from $\mathbf{x}_i$ to $\mathbf{x}_j$ is defined by (Duda and Hart, 1973):

$$d_{ij}^2 = (\mathbf{x}_i - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}_j - \boldsymbol{\mu}_j) \tag{1.27}$$

where $\boldsymbol{\Sigma}^{-1}$ is the inverse of the covariance matrix $\boldsymbol{\Sigma}$. It is assumed that the covariance matrix is the same for both populations, as shown by

$$\begin{aligned} \boldsymbol{\Sigma} &= E[(\mathbf{x}_i - \boldsymbol{\mu}_i)(\mathbf{x}_i - \boldsymbol{\mu}_i)^T] \\ &= E[(\mathbf{x}_j - \boldsymbol{\mu}_j)(\mathbf{x}_j - \boldsymbol{\mu}_j)^T] \end{aligned} \tag{1.28}$$

For the special case when $\mathbf{x}_j = \mathbf{x}_i$, $\boldsymbol{\mu}_i = \boldsymbol{\mu}_j = \boldsymbol{\mu}$ and $\boldsymbol{\Sigma} = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix, the Mahalanobis distance reduces to the Euclidean distance between the sample vector $\mathbf{x}_i$ and the mean vector $\boldsymbol{\mu}$.

Rule 2. Items to be categorized as separate classes should be given widely different representations in the network.

The second rule is the exact opposite of Rule 1.

Rule 3. If a particular feature is important, then there should be a large number of neurons involved in the representation of that item in the network.

Consider, for example, a radar application involving the detection of a target (e.g., aircraft) in the presence of clutter (i.e., radar reflections from undesirable targets such as buildings, trees, and weather formations). The detection performance of such a radar system is measured in terms of two probabilities:

- *Probability of detection*, defined as the probability that the system decides that a target is present when it is.
- *Probability of false alarm*, defined as the probability that the system decides that a target is present when it is not.

According to the *Neyman-Pearson criterion*, the probability of detection is maximized, subject to the constraint that the probability of false alarm does not exceed a prescribed value (Van Trees, 1968). In such an application, the actual presence of a target in the received signal represents an important feature of the input. Rule 3, in effect, states that there should be a large number of neurons involved in making the decision that a target is present when it actually is. By the same token, there should be a very large number of neurons involved in making the decision that the input consists of clutter only when it actually does. In both situations the large number of neurons assures a high degree of accuracy in decision making and tolerance with respect to faulty neurons.

Rule 4. Prior information and invariances should be built into the design of a neural network, thereby simplifying the network design by not having to learn them.

Rule 4 is particularly important because proper adherence to it results in a neural network with a *specialized (restricted) structure.* This is highly desirable for several reasons (Russo, 1991):

1. Biological visual and auditory networks are known to be very specialized.
2. A neural network with specialized structure usually has a smaller number of free parameters available for adjustment than a fully connected network. Consequently, the specialized network requires a smaller data set for training, learns faster, and often generalizes better.
3. The rate of information transmission through a specialized network (i.e., the network throughput) is accelerated.
4. The cost of building a specialized network is reduced because of its smaller size, compared to its fully connected counterpart.

How to Build Prior Information into Neural Network Design

An important issue that has to be addressed, of course, is how to develop a specialized structure by building prior information into its design. Unfortunately, there are currently no well-defined rules for doing this; rather, we have some *ad-hoc* procedures that are known to yield useful results. In particular, we may use a combination of two techniques (LeCun et al., 1990a):

1. *Restricting the network architecture* through the use of local connections known as *receptive fields*.[5]
2. *Constraining the choice of synaptic weights* through the use of *weight-sharing*.[6]

These two techniques, particularly the latter one, have a profitable side benefit: the number of free parameters in the network is reduced significantly.

To be specific, consider the partially connected feedforward network of Fig. 1.20. This network has a restricted architecture by construction. The top six source nodes

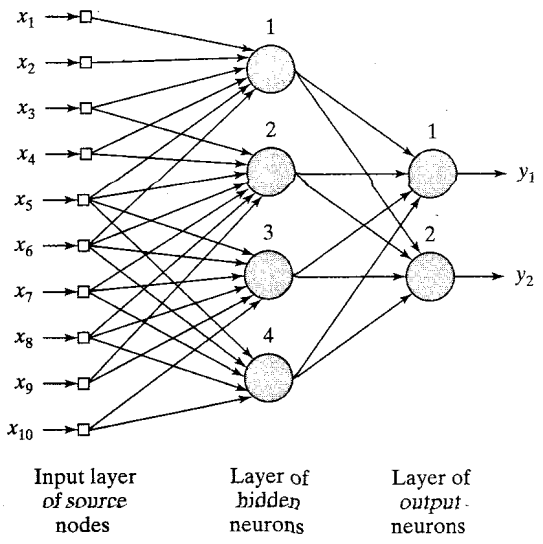

FIGURE 1.20 Illustrating the combined use of a receptive field and weight-sharing. All four hidden neurons share the same set of weights for their synaptic connections.

constitute the receptive field for hidden neuron 1 and so on for the other hidden neurons in the network. To satisfy the weight-sharing constraint, we merely have to use the same set of synaptic weights for each one of the neurons in the hidden layer of the network. Then, for the example shown in Fig. 1.20 with six local connections per hidden neuron and a total of four hidden neurons, we may express the induced local field of hidden neuron j as follows:

$$v_j = \sum_{i=1}^{6} w_i x_{i+j-1}, \quad j = 1, 2, 3, 4 \tag{1.29}$$

where $\{w_i\}_{i=1}^{6}$ constitute the same set of weights shared by all four hidden neurons, and x_k is the signal picked up from source node $k = i + j - 1$. Equation (1.29) is in the form of a *convolution sum*. It is for this reason that a feedforward network using local connections and weight-sharing in the manner described herein is referred to as a *convolutional network*.

The issue of building prior information into the design of a neural network pertains to one part of Rule 4; the remaining part of the rule involves the issue of invariances.

How to Build Invariances into Neural Network Design

Consider the following physical phenomena:

- When an object of interest rotates, the image of the object as perceived by an observer usually changes in a corresponding way.
- In a coherent radar that provides amplitude as well as phase information about its surrounding environment, the echo from a moving target is shifted in frequency due to the Doppler effect that arises due to the radial motion of the target in relation to the radar.

- The utterance from a person may be spoken in a soft or loud voice, and in a slow or quick manner.

In order to build an object recognition system, a radar target recognition system and a speech recognition system for dealing with these phenomena, respectively, the system must be capable of coping with a range of *transformations* of the observed signal (Barnard and Casasent, 1991). Accordingly, a primary requirement of pattern recognition is to design a classifier that is *invariant* to such transformations. In other words, a class estimate represented by an output of the classifier must not be affected by transformations of the observed signal applied to the classifier input.

There are at least three techniques for rendering classifier-type neural networks invariant to transformations (Barnard and Casasent, 1991):

1. *Invariance by Structure.* Invariance may be imposed on a neutral network by structuring its design appropriately. Specifically, synaptic connections between the neurons of the network are created so that transformed versions of the same input are forced to produce the same output. Consider, for example, the classification of an input image by a neural network that is required to be independent of in-plane rotations of the image about its center. We may impose rotational invariance on the network structure as follows. Let w_{ji} be the synaptic weight of neuron j connected to pixel i in the input image. If the condition $w_{ji} = w_{jk}$ is enforced for all pixels i and k that lie at equal distances from the center of the image, then the neural network is invariant to in-plane rotations. However, in order to maintain rotational invariance, the synaptic weight w_{ji} has to be duplicated for every pixel of the input image at the same radial distance from the origin. This points to a shortcoming of invariance by structure: The number of synaptic connections in the neural network becomes prohibitively large even for images of moderate size.

2. *Invariance by Training.* A neural network has a natural ability for pattern classification. This ability may be exploited directly to obtain transformation invariance as follows. The network is trained by presenting it a number of different examples of the same object, with the examples being chosen to correspond to different transformations (i.e., different aspect views) of the object. Provided that the number of examples is sufficiently large, and if the the network is trained to learn to discriminate the different aspect views of the object, we may then expect the network to generalize correctly to transformations other than those shown to it. However, from an engineering perspective, invariance by training has two disadvantages. First, when a neural network has been trained to recognize an object in an invariant fashion with respect to known transformations, it is not obvious that this training will also enable the network to recognize other objects of different classes invariantly. Second, the computational demand imposed on the network may be too severe to cope with, especially if the dimensionality of the feature space is high.

3. *Invariant Feature Space.* The third technique of creating an invariant classifier-type neural network is illustrated in Fig. 1.21. It rests on the premise that it may be pos-

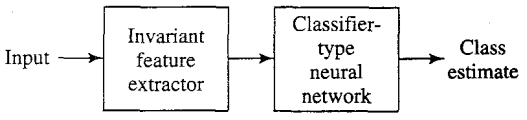

FIGURE 1.21 Block diagram of an invariant feature-space type of system.

sible to extract *features* that characterize the essential information content of an input data set, and which are invariant to transformations of the input. If such features are used, then the network as a classifier is relieved from the burden of having to delineate the range of transformations of an object with complicated decision boundaries. Indeed, the only differences that may arise between different instances of the same object are due to unavoidable factors such as noise and occlusion. The use of an invariant feature space offers three distinct advantages. First, the number of features applied to the network may be reduced to realistic levels. Second, the requirements imposed on network design are relaxed. Third, invariance for all objects with respect to known transformations is assured (Barnard and Casasent, 1991). However, this approach requires prior knowledge of the problem for it to work.

In conclusion, the use of an invariant-feature space as described may offer a most suitable technique for neural classifiers.

To illustrate the idea of invariant-feature space, consider the example of a coherent radar system used for air surveillance, where the targets of interest include aircraft, weather systems, flocks of migrating birds, and ground objects. The radar echoes from these targets possess different spectral characteristics. Moreover, experimental studies have shown that such radar signals can be modeled fairly closely as an *autoregressive* (AR) *process* of moderate order (Haykin and Deng, 1991). An AR model is a special form of regressive model defined for complex-valued data by

$$x(n) = \sum_{i=1}^{M} a_i^* x(n - i) + e(n) \tag{1.30}$$

where the $\{a_i\}_{i=1}^{M}$ are the *AR coefficients*, M is the *model order*, $x(n)$ is the *input*, and $e(n)$ is the *error* described as white noise. Basically, the AR model of Eq. (1.30) is represented by a *tapped-delay-line filter* as illustrated in Fig. 1.22a for $M = 2$. Equivalently, it may be represented by a *lattice filter* as shown in Fig. 1.22b, the coefficients of which are called *reflection coefficients*. There is a one-to-one correspondence between the AR coefficients of the model in Fig. 1.22a and the reflection coefficients of the model in Fig. 1.22b. The two models depicted assume that the input $x(n)$ is complex valued, as in the case of a coherent radar, in which case the AR coefficients and the reflection coefficients are all complex valued. The asterisk in Eq. (1.30) and Fig. 1.22 signifies *complex conjugation*. For now, it suffices to say that the coherent radar data may be described by a set of *autoregressive coefficients*, or by a corresponding set of *reflection coefficients*. The latter set of coefficients has a computational advantage in that efficient algorithms exist for their computation directly from the input data. The feature extraction problem, however, is complicated by the fact that moving objects produce varying Doppler frequencies that depend on their radial velocities measured with respect to the radar, and that tend to obscure the spectral content of the reflection coefficients as feature discriminants. To overcome this difficulty, we must build *Doppler invariance* into the computation of the reflection coefficients. The phase angle of the first reflection coefficient turns out to be equal to the Doppler frequency of the radar signal. Accordingly, Doppler frequency *normalization* is applied to all coefficients so as to remove the mean Doppler shift. This is done by defining a new set of reflection

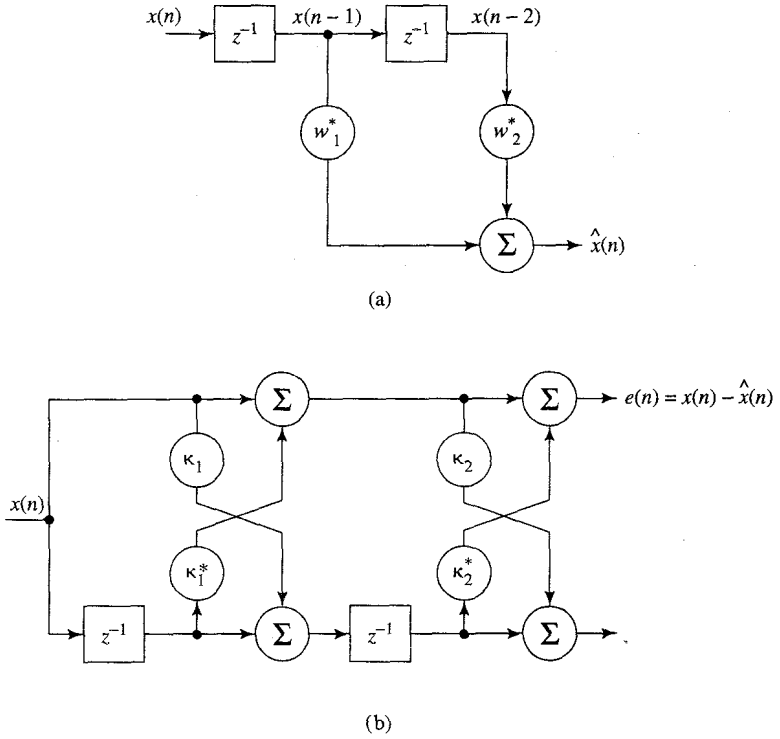

FIGURE 1.22 Autoregressive model of order 2: (a) tapped-delay-line model; (b) lattice filter model. (The asterisk denotes complex conjugation.)

coefficients $\{\kappa'_m\}$ related to the set of ordinary reflection coefficients $\{\kappa_m\}$ computed from the input data as follows:

$$\kappa'_m = \kappa_m e^{-jm\theta} \quad \text{for } m = 1, 2, \ldots, M \tag{1.31}$$

where θ is the phase angle of the first reflection coefficient. The operation described in Eq. (1.31) is referred to as *heterodyning*. A set of *Doppler-invariant radar features* is thus represented by the normalized reflection coefficients $\kappa'_1, \kappa'_2, \ldots, \kappa'_M$, with κ'_1 being the only real-valued coefficient in the set. As mentioned previously, the major categories of radar targets of interest in air surveillance are weather, birds, aircraft, and ground. The first three targets are moving, whereas the last one is not. The heterodyned spectral parameters of radar echoes from ground have echoes similar in characteristic to those from aircraft. A ground echo can be discriminated from an aircraft echo because of its small Doppler shift. Accordingly, the radar classifier includes a postprocessor as shown in Fig. 1.23, which operates on the classified results (encoded labels) for the purpose of identifying the ground class (Haykin and Deng, 1991). Thus, the *preprocessor* in Fig. 1.23 takes care of Doppler shift-invariant feature extraction at the classifier input, whereas the *postprocessor* uses the stored Doppler signature to distinguish between aircraft and ground returns.

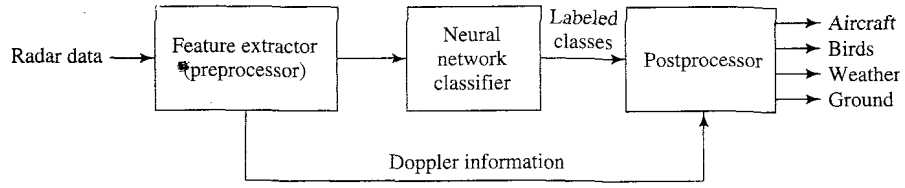

FIGURE 1.23 Doppler shift-invariant classifier of radar signals.

A much more fascinating example of knowledge representation in a neural network is found in the biological sonar system of echo-locating bats. Most bats use *frequency-modulated* (FM or "chirp") signals for the purpose of acoustic imaging; in an FM signal the instantaneous frequency of the signal varies with time. Specifically, the bat uses its mouth to broadcast short-duration FM sonar signals and uses its auditory system as the sonar receiver. Echoes from targets of interest are represented in the auditory system by the activity of neurons that are selective to different combinations of acoustic parameters. There are three principal neural dimensions of the bat's auditory representation (Simmons, 1991; Simmons and Saillant, 1992):

- *Echo frequency*, which is encoded by "place" originating in the frequency map of the cochlea; it is preserved throughout the entire auditory pathway as an orderly arrangement across certain neurons tuned to different frequencies.
- *Echo amplitude*, which is encoded by other neurons with different dynamic ranges; it is manifested both as amplitude tuning and as the number of discharges per stimulus.
- *Echo delay*, which is encoded through neural computations (based on cross-correlation) that produce delay-selective responses; it is manifested as target-range tuning.

The two principal characteristics of a target echo for image-forming purposes are *spectrum* for target shape, and *delay* for target range. The bat perceives "shape" in terms of the arrival time of echoes from different reflecting surfaces (glints) within the target. For this to occur, *frequency* information in the echo spectrum is converted into estimates of the *time* structure of the target. Experiments conducted by Simmons and coworkers on the big brown bat, *Eptesicus fuscus*, critically identify this conversion process as consisting of parallel time-domain and frequency-to-time-domain transforms whose converging outputs create the common delay of range axis of a perceived image of the target. It appears that the unity of the bat's perception is due to certain properties of the transforms themselves, despite the separate ways in which the auditory time representation of the echo delay and frequency representation of the echo spectrum are initially performed. Moreover, feature invariances are built into the sonar image-forming process so as to make it essentially independent of the target's motion and the bat's own motion.

Returning to the main theme of this section, namely, that of knowledge representation in a neural network, this issue is directly related to that of network architecture described in Section 1.6. Unfortunately, there is no well-developed theory for optimizing the architechture of a neural network required to interact with an environment of

interest, or for evaluating the way in which changes in the network architecture affect the representation of knowledge inside the network. Indeed, satisfactory answers to these issues are usually found through an exhaustive experimental study, with the designer of the neural network becoming an essential part of the structural learning loop.

No matter how the design is performed, knowledge about the problem domain of interest is acquired by the network in a comparatively straightforward and direct manner through training. The knowledge so acquired is represented in a compactly distributed form as weights across the synaptic connections of the network. While this form of knowledge representation enables the neural network to adapt and generalize, unfortunately the neural network suffers from the inherent inability to explain, in a comprehensive manner, the computational process by which the network makes a decision or reports its output. This can be a serious limitation, particularly in those applications where safety is of prime concern, as in air traffic control or medical diagnosis, for example. In applications of this kind, it is not only highly desirable but also absolutely essential to provide some form of *explanation capability*. One way in which this provision can be made is to integrate a neural network and artificial intelligence into a hybrid system, as discussed in the next section.

1.8 ARTIFICIAL INTELLIGENCE AND NEURAL NETWORKS

The goal of *artificial intelligence* (AI) is the development of paradigms or algorithms that require machines to perform cognitive tasks, at which humans are currently better. This statement on AI is adopted from Sage, 1990. Note that it is not the only accepted definition of AI.

An AI system must be capable of doing three things: (1) store knowledge, (2) apply the knowledge stored to solve problems, and (3) acquire new knowledge through experience. An AI system has three key components: representation, reasoning, and learning (Sage, 1990), as depicted in Fig. 1.24.

1. *Representation.* The most distinctive feature of AI is probably the pervasive use of a language of *symbol* structures to represent both general knowledge about a problem domain of interest and specific knowledge about the solution to the problem. The symbols are usually formulated in familiar terms, which makes the symbolic repre-

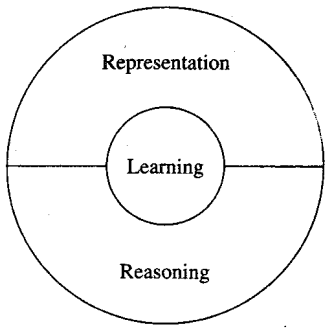

FIGURE 1.24 Illustrating the three key components of an AI system.

sentations of AI relatively easy to understand by a human user. Indeed, the clarity of symbolic AI makes it well suited for human–machine communication.

"Knowledge," as used by AI researchers, is just another term for data. It may be of a declarative or procedural kind. In a *declarative* representation, knowledge is represented as a static collection of facts, with a small set of general procedures used to manipulate the facts. A characteristic feature of declarative representations is that they appear to possess a meaning of their own in the eyes of the human user, independent of their use within the AI system. In a *procedural* representation, on the other hand, knowledge is embodied in an executable code that acts out the meaning of the knowledge. Both kinds of knowledge, declarative and procedural, are needed in most problem domains of interest.

2. *Reasoning.* In its most basic form, *reasoning* is the ability to solve problems. For a system to qualify as a reasoning system it must satisfy certain conditions (Fischler and Firschein,1987):

- The system must be able to express and solve a broad range of problems and problem types.
- The system must be able to make *explicit* and *implicit* information known to it.
- The system must have a *control* mechanism that determines which operations to apply to a particular problem, when a solution to the problem has been obtained, or when further work on the problem should be terminated.

Problem solving may be viewed as a *searching* problem. A common way to deal with "search" is to use *rules*, *data*, and *control* (Nilsson, 1980). The rules operate on the data, and the control operates on the rules. Consider, for example, the "traveling salesman problem," where the requirement is to find the shortest tour that goes from one city to another, with all the cities on the tour being visited only once. In this problem the data are made up of the set of possible tours and their costs in a weighted graph, the rules define the ways to proceed from city to city, and the control decides which rules to apply and when to apply them.

In many situations encountered in practice (e.g., medical diagnosis), the available knowledge is incomplete or inexact. In such situations, *probabilistic reasoning* procedures are used, thereby permitting AI systems to deal with uncertainty (Russell and Norvig, 1995; Pearl, 1988).

3. *Learning.* In the simple model of machine learning depicted in Fig. 1.25, the environment supplies some information to a *learning element*. The learning element then uses this information to make improvements in a *knowledge base*, and finally the

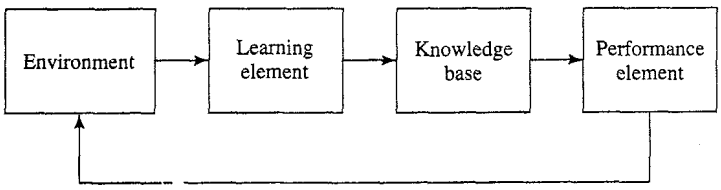

FIGURE 1.25 Simple model of machine learning.

performance element uses the knowledge base to perform its task. The kind of information supplied to the machine by the environment is usually imperfect, with the result that the learning element does not know in advance how to fill in missing details or to ignore details that are unimportant. The machine therefore operates by guessing, and then receiving *feedback* from the performance element. The feedback mechanism enables the machine to evaluate its hypotheses and revise them if necessary.

Machine learning may involve two rather different kinds of information processing: inductive and deductive. In *inductive* information processing, general patterns and rules are determined from raw data and experience. In *deductive* information processing, however, general rules are used to determine specific facts. Similarity-based learning uses induction, whereas the proof of a theorem is a deduction from known axioms and other existing theorems. Explanation-based learning uses both induction and deduction.

The importance of knowledge bases and the difficulties experienced in learning have led to the development of various methods for augmenting knowledge bases. Specifically, if there are experts in a given field, it is usually easier to obtain the compiled experience of the experts than to try to duplicate and direct experience that gave rise to the expertise. This, indeed, is the idea behind *expert systems*.

Having familiarized ourselves with symbolic AI machines, how would we compare them to neural networks as cognitive models? For this comparison, we follow three subdivisions: level of explanation, style of processing, and representational structure (Memmi, 1989).

1. *Level of Explanation.* In classical AI, the emphasis is on building *symbolic representations* that are presumably so called because they stand for something. From the viewpoint of cognition, AI assumes the existence of mental representations, and it models cognition as the *sequential processing* of symbolic representations (Newell and Simon, 1972).

The emphasis in neural networks, on the other hand, is on the development of *parallel distributed processing* (PDP) *models*. These models assume that information processing takes place through the interaction of a large number of neurons, each of which sends excitatory and inhibitory signals to other neurons in the network (Rumelhart and McClelland, 1986). Moreover, neural networks place great emphasis on neurobiological explanation of cognitive phenomena.

2. *Processing Style.* In classical AI, the processing is *sequential*, as in typical computer programming. Even when there is no predetermined order (scanning the facts and rules of an expert system, for example), the operations are performed in a step-by-step manner. Most probably, the inspiration for sequential processing comes from the sequential nature of natural language and logical inference, as much as from the structure of the von Neumann machine. We should not forget that classical AI was born shortly after the von Neumann machine, during the same intellectual era.

In contrast, *parallelism* is not only conceptually essential to the processing of information in neural networks, but also the source of their flexibility. Moreover, parallelism may be massive (hundreds of thousands of neurons), which gives neural networks a remarkable form of robustness. With the computation spread over many neurons, it usually does not matter much if the states of some neurons in the network deviate from their expected values. Noisy or incomplete inputs may still be recognized, a damaged network may still be able to function satisfactorily, and learning does not

have to be perfect. Performance of the network degrades gracefully within a certain range. The network is made even more robust by virtue of the "coarse coding" (Hinton, 1981), where each feature is spread over several neurons.

3. *Representational Structure.* With a language of thought pursued as a model for classical AI, we find that symbolic representations possess a *quasi-linguistic structure.* Like expressions of natural language, the expressions of classical AI are generally complex, built in a systematic fashion from simple symbols. Given a limited stock of symbols, meaningful new expressions may be composed by virtue of the *compositionality* of symbolic expressions and the analogy between syntactic structure and semantics.

The nature and structure of representations is, however, a crucial problem for neural networks. In the March 1988 Special Issue of the journal *Cognition*, Fodor and Pylyshyn make some potent criticisms about the computational adequacy of neural networks in dealing with cognition and linguistics. They argue that neural networks are on the wrong side of two basic issues in cognition: the nature of *mental representations*, and the nature of *mental processes*. According to Fodor and Pylyshyn, for classical AI theories but *not* neural networks:

- Mental representations characteristically exhibit a combinatorial constituent structure and combinatorial semantics.
- Mental processes are characteristically sensitive to the combinatorial structure of the representations on which they operate.

In summary, we may describe symbolic AI as the formal manipulation of a language of algorithms and data representations in a *top-down* fashion. We may describe neural networks, however, as parallel distributed processors with a natural ability to learn, and which usually operate in a *bottom-up* fashion. For the implementation of cognitive tasks, it therefore appears that rather than seek solutions based on symbolic AI or neural networks alone, a more potentially useful approach would be to build *structured connectionist models* or *hybrid systems* that integrate them together. By so doing, we are able to combine the desirable features of adaptivity, robustness, and uniformity offered by neural networks with the representation, inference, and universality that are inherent features of symbolic AI (Feldman, 1992; Waltz, 1997). Indeed, it is with this objective in mind that several methods have been developed for the extraction of rules from trained neural networks. In addition to the understanding of how symbolic and connectionist approaches can be integrated for building intelligent machines, there are several other reasons for the extraction of rules from neural networks (Andrews and Diederich, 1996):

- To validate neural network components in software systems by making the internal states of the neural network accessible and understandable to users.
- To improve the generalization performance of neural networks by (1) identifying regions of the input space where the training data are not adequately represented, or (2) indicating the circumstances where the neural network may fail to generalize.
- To discover salient features of the input data for data exploration (mining).
- To provide a means for traversing the boundary between the connectionist and symbolic approaches to the development of intelligent machines.
- To satisfy the critical need for safety in a special class of systems where safety is a mandatory condition.

1.9 HISTORICAL NOTES

We conclude this introductory chapter on neural networks with some historical notes.[7]

The modern era of neural networks began with the pioneering work of McCulloch and Pitts (1943). McCulloch was a psychiatrist and neuroanatomist by training; he spent some 20 years thinking about the representation of an event in the nervous system. Pitts was a mathematical prodigy, who joined McCulloch in 1942. According to Rall (1990), the 1943 paper by McCulloch and Pitts arose within a neural modeling community that had been active at the University of Chicago for at least five years prior to 1943, under the leadership of Rashevsky.

In their classic paper, McCulloch and Pitts describe a logical calculus of neural networks that united the studies of neurophysiology and mathematical logic. Their formal model of a neuron was assumed to follow an "all-or-none" law. With a sufficient number of such simple units, and synaptic connections set properly and operating synchronously, McCulloch and Pitts showed that a network so constituted would, in principle, compute any computable function. This was a very significant result and with it, it is generally agreed that the disciplines of neural networks and of artificial intelligence were born.

The 1943 paper by McCulloch and Pitts was widely read at the time and still is. It influenced von Neumann to use idealized switch-delay elements derived from the McCulloch–Pitts neuron in the construction of the EDVAC (Electronic Discrete Variable Automatic Computer) that developed out of the ENIAC (Electronic Numerical Integrator and Computer) (Aspray and Burks, 1986). The ENIAC was the first general purpose electronic computer, which was built at the Moore School of Electrical Engineering of the University of Pennsylvania from 1943 to 1946. The McCulloch–Pitts theory of formal neural networks featured prominently in the second of four lectures delivered by von Neumann at the University of Illinois in 1949.

In 1948, Wiener's famous book *Cybernetics* was published, describing some important concepts for control, communications, and statistical signal processing. The second edition of the book was published in 1961, adding new material on learning and self-organization. In Chapter 2 of both editions of this book, Wiener appears to grasp the physical significance of statistical mechanics in the context of the subject matter, but it was left to Hopfield (more than 30 years later) to bring the linkage between statistical mechanics and learning systems to full fruition.

The next major development in neural networks came in 1949 with the publication of Hebb's book *The Organization of Behavior*, in which an explicit statement of a physiological learning rule for *synaptic modification* was presented for the first time. Specifically, Hebb proposed that the connectivity of the brain is continually changing as an organism learns differing functional tasks, and that *neural assemblies* are created by such changes. Hebb followed up an early suggestion by Ramón y Cajál and introduced his now famous *postulate of learning*, which states that the effectiveness of a variable synapse between two neurons is increased by the repeated activation of one neuron by the other across that synapse. Hebb's book was immensely influential among psychologists, but unfortunately it had little or no impact on the engineering community.

Hebb's book has been a source of inspiration for the development of computational models of *learning and adaptive systems*. The paper by Rochester, Holland,

Haibt, and Duda (1956) is perhaps the first attempt to use computer simulation to test a well-formulated neural theory based on Hebb's postulate of learning; the simulation results reported in that paper clearly show that inhibition must be added for the theory to actually work. In that same year, Uttley (1956) demonstrated that a neural network with modifiable synapses may learn to classify simple sets of binary patterns into corresponding classes. Uttley introduced the so-called *leaky integrate and fire neuron*, which was later formally analyzed by Caianiello (1961). In later work, Uttley (1979) hypothesized that the effectiveness of a variable synapse in the nervous system depends on the statistical relationship between the fluctuating states on either side of that synapse, thereby linking up with Shannon's information theory.

In 1952, Ashby's book, *Design for a Brain: The Origin of Adaptive Behavior*, was published, which is just as fascinating to read today as it must have been then. The book was concerned with the basic notion that adaptive behavior is not inborn but rather learned, and that through learning the behavior of an animal (system) usually changes for the better. The book emphasized the dynamic aspects of the living organism as a machine and the related concept of stability.

In 1954, Minsky wrote a "neural network" doctorate thesis at Princeton University, which was entitled "Theory of Neural-Analog Reinforcement Systems and Its Application to the Brain-Model Problem." In 1961, an excellent early paper by Minsky on AI entitled "Steps Toward Artificial Intelligence," was published; this latter paper contains a large section on what is now termed neural networks. In 1967 Minsky's book, *Computation: Finite and Infinite Machines*, was published. This clearly written book extended the 1943 results of McCulloch and Pitts and put them in the context of automata theory and the theory of computation.

Also in 1954, the idea of a *nonlinear adaptive filter* was proposed by Gabor, one of the early pioneers of communication theory, and the inventor of holography. He went on to build such a machine with the aid of collaborators, the details of which are described in Gabor et al. (1960). Learning was accomplished by feeding samples of a stochastic process into the machine, together with the target function that the machine was expected to produce.

In the 1950s work on *associative memory* was initiated by Taylor (1956). This was followed by the introduction of the *learning matrix* by Steinbuch (1961); this matrix consists of a planar network of switches interposed between arrays of "sensory" receptors and "motor" effectors. In 1969, an elegant paper on nonholographic associative memory by Willshaw, Buneman, and Longuet-Higgins was published. This paper presents two ingenious network models: a simple optical system realizing a correlation memory, and a closely related neural network suggested by the optical memory. Other significant contributions to the early development of associative memory include papers by Anderson (1972), Kohonen (1972), and Nakano (1972), who independently and in the same year introduced the idea of a *correlation matrix memory* based on the *outer product* learning rule.

Von Neumann was one of the great figures in science in the first half of the twentieth century. The *von Neumann architecture*, basic to the design of a digital computer, is named in his honor. In 1955 he was invited by Yale University to give the Silliman Lectures during 1956. He died in 1957, and the unfinished manuscript of the

Silliman Lectures was published later as a book, *The Computer and the Brain* (1958). This book is interesting because it suggests what von Neumann might have done had he lived; he had started to become aware of the profound differences between brains and computers.

An issue of particular concern in the context of neural networks is that of designing a reliable network with neurons that may be viewed as unreliable components. This important problem was solved by von Neumann (1956) using the idea of redundancy, which motivated Winograd and Cowan (1963) to suggest the use of a *distributed* redundant representation for neural networks. Winograd and Cowan showed how a large number of elements could collectively represent an individual concept, with a corresponding increase in robustness and parallelism.

Some 15 years after the publication of McCulloch and Pitt's classic paper, a new approach to the pattern recognition problem was introduced by Rosenblatt (1958) in his work on the *perceptron*, a novel method of supervised learning. The crowning achievement of Rosenblatt's work was the so-called *perceptron convergence theorem*, the first proof for which was outlined by Rosenblatt (1960b); proofs of the theorem also appeared in Novikoff (1963) and others. In 1960, Widrow and Hoff introduced the *least mean-square (LMS) algorithm* and used it to formulate the *Adaline* (adaptive linear element). The difference between the perceptron and the Adaline lies in the training procedure. One of the earliest trainable layered neural networks with multiple adaptive elements was the Madaline (multiple-adaline) structure proposed by Widrow and his students (Widrow, 1962). In 1967, Amari used the stochastic gradient method for adaptive pattern classification. In 1965, Nilsson's book, *Learning Machines*, was published, which is still the best-written exposition of linearly separable patterns in hypersurfaces. During the classical period of the perceptron in the 1960s, it seemed as if neural networks could do anything. But then came the book by Minsky and Papert (1969), who used mathematics to demonstrate that there are fundamental limits on what single-layer perceptrons can compute. In a brief section on multilayer perceptrons, they stated that there was no reason to assume that any of the limitations of single-layer perceptrons could be overcome in the multilayer version.

An important problem encountered in the design of a multilayer perceptron is the *credit assignment problem* (i.e., the problem of assigning credit to hidden neurons in the network). The terminology "credit assignment" was first used by Minsky (1961), under the title "Credit Assignment Problem for Reinforcement Learning Systems." By the late 1960s, most of the ideas and concepts necessary to solve the perceptron credit assignment problem were already formulated, as were many of the ideas underlying the recurrent (attractor neural) networks that are now referred to as Hopfield networks. However, we had to wait until the 1980s for the solutions of these basic problems to emerge. According to Cowan (1990), there were three reasons for this lag of more than 10 years:

- One reason was technological—there were no personal computers or workstations for experimentation. For example, when Gabor developed his nonlinear learning filter, it took his research team an additional six years to build the filter with analog devices (Gabor, 1954; Gabor et al., 1960).

- The other reason was in part psychological, in part financial. The 1969 monograph by Minsky and Papert certainly did not encourage anyone to work on perceptrons, or agencies to support the work on them.
- The analogy between neural networks and lattice spins was premature. The *spin-glass model* by Sherrington and Kirkpatrick was not invented until 1975.

These factors contributed in one way or another to the dampening of continued interest in neural networks in the 1970s. Many of the researchers, except for those in psychology and the neurosciences, deserted the field during that decade. Indeed, only a handful of the early pioneers maintained their commitment to neural networks. From an engineering perspective, we may look back on the 1970s as a decade of dormancy for neural networks.

An important activity that did emerge in the 1970s was *self-organizing maps* using competitive learning. The computer simulation work done by von der Malsburg (1973) was perhaps the first to demonstrate self-organization. In 1976 Willshaw and von der Malsburg published the first paper on the formation of self-organizing maps, motivated by topologically ordered maps in the brain.

In the 1980s major contributions to the theory and design of neural networks were made on several fronts, and with it there was a resurgence of interest in neural networks.

Grossberg (1980), building on his earlier work on competitive learning (Grossberg, 1972, 1976a, b), established a new principle of self-organization known as *adaptive resonance theory* (ART). Basically, the theory involves a bottom-up recognition layer and a top-down generative layer. If the input pattern and learned feedback pattern match, a dynamical state called "adaptive resonance" (i.e., amplification and prolongation of neural activity) takes place. This *principle of forward/backward projections* has been rediscovered by other investigators under different guises.

In 1982, Hopfield used the idea of an energy function to formulate a new way of understanding the computation performed by recurrent networks with symmetric synaptic connections. Moreover, he established the isomorphism between such a recurrent network and an *Ising model* used in statistical physics. This analogy paved the way for a deluge of physical theory (and physicists) to enter neural modeling, thereby transforming the field of neural networks. This particular class of neural networks with feedback attracted a great deal of attention in the 1980s, and in the course of time it has come to be known as *Hopfield networks*. Although Hopfield networks may not be realistic models for neurobiological systems, the principle they embody, namely that of storing information in dynamically stable networks, is profound. The origin of this principle may in fact be traced back to pioneering work of many other investigators:

- Cragg and Tamperley (1954, 1955) made the observation that just as neurons can be "fired" (activated) or "not fired" (quiescent), so can atoms in a lattice have their spins pointing "up" or "down."
- Cowan (1967) introduced the "sigmoid" firing characteristic and the smooth firing condition for a neuron that was based on the logistic function.
- Grossberg (1967, 1968) introduced the *additive model* of a neuron, involving nonlinear difference/differential equations, and explored the use of the model as a basis for short-term memory.

- Amari (1972) independently introduced the additive model of a neuron, and used it to study the dynamic behavior of randomly connected neuron-like elements.
- Wilson and Cowan (1972) derived coupled nonlinear differential equations for the dynamics of spatially localized populations containing both excitatory and inhibitory model neurons.
- Little and Shaw (1975) described a *probabilistic model* of a neuron, either firing or not firing an action potential, and used the model to develop a theory of short-term memory.
- Anderson, Silverstein, Ritz, and Jones (1977) proposed the *brain-state-in-a-box (BSB) model*, consisting of a simple associative network coupled to nonlinear dynamics.

It is therefore not surprising that the publication of Hopfield's paper in 1982 generated a great deal of controversy. Nevertheless, it is in the same paper that the principle of storing information in dynamically stable networks is first made explicit. Moreover, Hopfield showed that he had the insight from the spin-glass model in statistical mechanics to examine the special case of recurrent networks with symmetric connectivity, thereby guaranteeing their convergence to a stable condition. In 1983, Cohen and Grossberg established a general principle for assessing the stability of a *content-addressable memory* that includes the continuous-time version of the Hopfield network as a special case. A distinctive feature of an attractor neural network is the natural way in which *time*, an essential dimension of learning, manifests itself in the nonlinear dynamics of the network. In this context, the Cohen–Grossberg theorem is of profound importance.

Another important development in 1982 was the publication of Kohonen's paper on self-organizing maps (Kohonen, 1982) using a one- or two-dimensional lattice structure, which was different in some respects from the earlier work by Willshaw and von der Malsburg. Kohonen's model has received far more attention in an analytic context and with respect to applications in the literature, than the Willshaw–von der Malsburg model, and has become the benchmark against which other innovations in this field are evaluated.

In 1983, Kirkpatrick, Gelatt, and Vecchi described a new procedure called *simulated annealing*, for solving combinatorial optimization problems. Simulated annealing is rooted in statistical mechanics. It is based on a simple technique that was first used in computer simulation by Metropolis et al. (1953). The idea of simulated annealing was later used by Ackley, Hinton, and Sejnowski (1985) in the development of a stochastic machine known as the *Boltzmann machine*, which was the *first* successful realization of a multilayer neural network. Although the Boltzmann machine learning algorithm proved not as computationally efficient as the back-propagation algorithm, it broke the psychological logjam by showing that the speculation in Minsky and Papert (1969) was incorrectly founded. The Boltzmann machine also laid the groundwork for the subsequent development of *sigmoid belief networks* by Neal (1992), which accomplished two things: (1) significant improvement in learning, and (2) linking neural networks to belief networks (Pearl, 1988). A further improvement in the learning performance of sigmoid belief networks was made by Saul, Jakkolla, and

Jordan (1996) by using mean-field theory, a technique also rooted in statistical mechanics.

A paper by Barto, Sutton, and Anderson on *reinforcement learning* was published in 1983. Although they were not the first to use reinforcement learning (Minsky considered it in his 1954 Ph.D. thesis, for example), this paper has generated a great deal of interest in reinforcement learning and its application in control. Specifically, they demonstrated that a reinforcement learning system could learn to balance a broomstick (i.e., a pole mounted on a cart) in the absence of a helpful teacher. The system required only a failure signal that occurs when the pole falls past a critical angle from the vertical, or when the cart reaches the end of a track. In 1996, the book *Neuro-dynamic Programming* by Bertsekas and Tsitsiklis was published. This book put reinforcement on a proper mathematical basis by linking it with Bellman's dynamic programming.

In 1984 Braitenberg's book, *Vehicles: Experiments in Synthetic Psychology*, was published. In this book, Braitenberg advocates the *principle of goal-directed, self-organized performance:* the understanding of a complex process is best achieved by a synthesis of putative elementary mechanisms, rather than by a top-down analysis. Under the guise of science fiction, Braitenberg illustrates this important principle by describing various machines with simple internal architecture. The properties of the machines and their behavior are inspired by facts about animal brains, a subject he studied directly or indirectly for more than 20 years.

In 1986 the development of the *back-propagation algorithm* was reported by Rumelhart, Hinton, and Williams (1986). In that same year, the celebrated two-volume book, *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, edited by Rumelhart and McClelland, was published. This latter book has been a major influence in the use of back-propagation learning, which has emerged as the most popular learning algorithm for the training of multilayer perceptrons. In fact, back-propagation learning was discovered independently in two other places about the same time (Parker, 1985; LeCun, 1985). After the discovery of the back-propagation algorithm in the mid-1980s, it turned out that the algorithm had been described earlier by Werbos in his Ph.D. thesis at Harvard University in August 1974; Werbos's Ph.D. thesis was the first documented description of efficient reverse-mode gradient computation that was applied to general network models with neural networks arising as a special case. The basic idea of back propagation may be traced further back to the book *Applied Optimal Control* by Bryson and Ho (1969). In Section 2.2 entitled "Multistage Systems" of that book, a derivation of back propagation using a Lagrangian formalism is described. In the final analysis, however, much of the credit for the back-propagation algorithm has to be given to Rumelhart, Hinton, and Williams (1986) for proposing its use for machine learning and for demonstrating how it could work.

In 1988 Linsker described a new principle for self-organization in a perceptual network (Linsker, 1988a). The principle is designed to preserve maximum information about input activity patterns, subject to such constraints as synaptic connections and synapse dynamic range. A similar suggestion had been made independently by several vision researchers. However, it was Linsker who used abstract concepts rooted in information theory (originated by Shannon in 1948) to formulate the *maximum*

mutual information (Infomax) principle. Linsker's paper reignited interest in the application of information theory to neural networks. In particular, the application of information theory to the *blind source separation problem* by Bell and Sejnowski (1995) has prompted many researchers to explore other information-theoretic models for solving a broad class of problems known collectively as *blind deconvolution*.

Also in 1988, Broomhead and Lowe described a procedure for the design of layered feedforward networks using *radial basis functions* (RBF), which provide an alternative to multilayer perceptrons. The basic idea of radial basis functions goes back at least to the *method of potential functions* that was originally proposed by Bashkirov, Braverman, and Muchnik (1964), and the theoretical properties of which were developed by Aizerman, Braverman, and Rozonoer (1964a, b). A description of the method of potential functions is presented in the classic book, *Pattern Classification and Scene Analysis*, by Duda and Hart (1973). Nevertheless, the paper by Broomhead and Lowe has led to a great deal of research effort linking the design of neural networks to an important area in numerical analysis and also linear adaptive filters. In 1990, Poggio and Girosi (1990a) further enriched the theory of RBF networks by applying Tikhonov's regularization theory.

In 1989, Mead's book, *Analog VLSI and Neural Systems*, was published. This book provides an unusual mix of concepts drawn from neurobiology and VLSI technology. Above all, it includes chapters on silicon retina and silicon cochlea, written by Mead and coworkers, which are vivid examples of Mead's creative mind.

In the early 1990s, Vapnik and coworkers invented a computationally powerful class of supervised learning networks, called *support vector machines*, for solving pattern recognition, regression, and density estimation problems (Boser, Guyon, and Vapnik, 1992; Cortes and Vapnik, 1995; Vapnik, 1995, 1998). This new method is based on results in the theory of learning with finite sample sizes. A novel feature of support vector machines is the natural way in which the *Vapnik-Chervonenkis (VC) dimension* is embodied in their design. The VC dimension provides a measure for the capacity of a neural network to learn from a set of examples (Vapnik and Chervonenkis, 1971; Vapnik, 1982).

It is now well established that *chaos* constitutes a key aspect of physical phenomena. A question raised by many is: Is there a key role for chaos in the study of neural networks? In a biological context, Freeman (1995) believes that the answer to this question is in the affirmative. According to Freeman, patterns of neural activity are not imposed from outside the brain; rather, they are constructed from within. In particular, chaotic dynamics offers a basis for describing the conditions that are required for emergence of self-organized patterns in and among populations of neurons.

Perhaps more than any other publication, the 1982 paper by Hopfield and the 1986 two-volume book by Rumelhart and McLelland were the most influential publications responsible for the resurgence of interest in neural networks in the 1980s. Neural networks have certainly come a long way from the early days of McCulloch and Pitts. Indeed, they have established themselves as an interdisciplinary subject with deep roots in the neurosciences, psychology, mathematics, the physical sciences, and engineering. Needless to say, they are here to stay, and will continue to grow in theory, design, and applications.

NOTES AND REFERENCES

1. This definition of a neural network is adapted from Aleksander and Morton (1990).
2. For a complementary perspective on neural networks with emphasis on neural modeling, cognition, and neurophysiological considerations, see Anderson (1995). For a highly readable account of the computational aspects of the brain, see Churchland and Sejnowski (1992). For more detailed descriptions of neural mechanisms and the human brain, see Kandel and Schwartz (1991), Shepherd (1990a, b), Koch and Segev (1989), Kuffler et al. (1984), and Freeman (1975).
3. For a thorough account of sigmoid functions and related issues, see Menon et al. (1996).
4. The logistic function, or more precisely the *logistic distribution function*, derives its name from a transcendental "law of logistic growth" that resulted in a huge literature. Measured in appropriate units, all growth processes were supposed to be represented by the logistic distribution function

$$F(t) = \frac{1}{1 + e^{\alpha t - \beta}}$$

where t represents time, and α and β are constants. It turned out, however, that not only the logistic distribution but also the Gaussian and other distributions can apply to the same data with the same or better goodness of fit (Feller, 1968).

5. According to Kuffler et al. (1984), the term "receptive field" was coined originally by Sherrington (1906) and reintroduced by Hartline (1940). In the context of a visual system, the receptive field of a neuron refers to the restricted area on the retinal surface, which influences the discharges of that neuron due to light.
6. It appears that the weight-sharing technique was originally described in Rumelhart et al. (1986b).
7. The historical notes presented here are largely (but not exclusively) based on the following sources: (1) the paper by Saarinen et al. (1992); (2) the chapter contribution by Rall (1990); (3) the paper by Widrow and Lehr (1990); (4) the papers by Cowan (1990) and Cowan and Sharp (1988); (5) the paper by Grossberg (1988c); (6) the two-volume book on neurocomputing (Anderson et al., 1990; Anderson and Rosenfeld, 1988); (7) the chapter contribution of Selfridge et al. (1988); (8) the collection of papers by von Neumann on computing and computer theory (Aspray and Burks, 1986); (9) the handbook on brain theory and neural networks edited by Arbib (1995); (10) Chapter 1 of the book by Russell and Norvig (1995); and (11) the article by Taylor (1997).

PROBLEMS

Models of a neuron

1.1 An example of the logistic function is defined by

$$\varphi(v) = \frac{1}{1 + \exp(-av)}$$

whose limiting values are 0 and 1. Show that the derivative of $\varphi(v)$ with respect to v is given by

$$\frac{d\varphi}{dv} = a\varphi(v)[1 - \varphi(v)]$$

What is the value of this derivative at the origin?