



Abbreviations and Symbols

ABBREVIATIONS

AI	artificial intelligence
APEX	adaptive principal components extraction
AR	autoregressive
BBTT	back propagation through time
BM	Boltzmann machine
BP	back propagation
b/s	bits per second
BOSS	bounded, one-sided saturation
BSB	brain-state-in-a-box
BSS	Blind source (signal) separation
CART	classification and regression tree
cmm	correlation matrix memory
CV	cross-validation
DEKF	decoupled extended Kalman filter
DFA	deterministic finite-state automata
DSP	digital signal processor
EKF	extended Kalman filter
EM	expectation-maximization
FIR	finite-duration impulse response
FM	frequency-modulated (signal)
GEKF	global extended Kalman filter
GCV	generalized cross-validation
GHA	generalized Hebbian algorithm
GSLC	generalized sidelobe canceler

HME	hierarchical mixture of experts
HMM	hidden Markov model
Hz	hertz
ICA	independent components analysis
Infomax	maximum mutual information
KR	kernel regression
LMS	least-mean-square
LR	likelihood ratio
LTP	long-term potentiation
LTD	long-term depression
LR	likelihood ratio
LVQ	learning vector quantization
MCA	minor components analysis
MDL	minimum description length
ME	mixture of experts
MFT	mean-field theory
MIMO	multiple input–multiple output
ML	maximum likelihood
MLP	multilayer perceptron
MRAC	model reference adaptive control
NARMA	nonlinear autoregressive moving average
NARX	nonlinear autoregressive with exogenous inputs
NDP	neuron-dynamic programming
NW	Nadaraya–Watson (estimator)
NWKR	Nadaraya–Watson kernel regression
OBD	optimal brain damage
OBS	optimal brain surgeon
OCR	optical character recognition
ODE	ordinary differential equation
PAC	probably approximately correct
PCA	principal components analysis
pdf	probability density function
pmf	probability mass function
RBF	radial basis function
RMLP	recurrent multilayer perceptron
RTRL	real-time recurrent learning
SIMO	single input–multiple output
SISO	single input–single output
SNR	signal-to-noise ratio
SOM	self-organizing map

SRN	simple recurrent network (also referred to as Elman's recurrent network)
SVD	singular value decomposition
SVM	support vector machine
TDNN	time-delay neural network
TLFN	time lagged feedforward network
VC	Vapnik–Chervononkis (dimension)
VLSI	very-large-scale integration
XOR	exclusive OR

IMPORTANT SYMBOLS

a	action
$\mathbf{a}^T \mathbf{b}$	inner product of vectors \mathbf{a} and \mathbf{b}
$\mathbf{a} \mathbf{b}^T$	outer product of vectors \mathbf{a} and \mathbf{b}
$\binom{l}{m}$	binomial coefficient
$A \cup B$	unions of A and B
B	inverse of temperature
b_k	bias applied to neuron k
$\cos(\mathbf{a}, \mathbf{b})$	cosine of the angle between vectors \mathbf{a} and \mathbf{b}
D	depth of memory
D_{flg}	Kullback–Leibler divergence between probability density functions f and g
$\tilde{\mathbf{D}}$	adjoint of operator \mathbf{D}
E	energy function
E_i	energy of state i in statistical mechanics
E	statistical expectation operator
$\langle E \rangle$	average energy
erf	error function
erfc	complimentary error function
\exp	exponential
\mathcal{E}_{av}	average squared error or sum of squared errors
$\mathcal{E}(n)$	instantaneous value of the sum of squared errors
$\mathcal{E}_{\text{total}}$	total sum of error squares
F	free energy
$f_{\mathbf{x}}(\mathbf{x})$	probability density function of random vector \mathbf{X}
\mathcal{F}^*	subset (network) with the smallest minimum empirical risk
\mathbf{H}	Hessian matrix
\mathbf{H}^{-1}	inverse of matrix \mathbf{H}
i	square root of -1 , also denoted by j
\mathbf{I}	identity matrix
\mathbf{I}	Fisher's information matrix
J	mean-square error

\mathbf{J}	Jacobian matrix
$\mathbf{K}(n, n-1)$	error covariance matrix in Kalman filter theory
$\mathbf{K}^{1/2}$	square root of matrix \mathbf{K}
$\mathbf{K}^{T/2}$	transpose of square root of matrix \mathbf{K}
k_B	Boltzmann constant
\log	logarithm
$L(\mathbf{w})$	log-likelihood function of weight vector \mathbf{w}
$\mathcal{L}(\mathbf{w})$	log-likelihood function of weight vector \mathbf{w} based on a single example
\mathbf{M}_c	controllability matrix
\mathbf{M}_o	observability matrix
n	discrete time
p_i	probability of state i in statistical mechanics
p_{ij}	transition probability from state i to state j
\mathbf{P}	stochastic matrix
P_c	probability of correct classification
P_e	probability of error
$P(e \mathcal{C})$	conditional probability of error e given that the input is drawn from class \mathcal{C}
p_α^+	probability that the visible neurons of a Boltzmann machine are in state α , given that the network is in its clamped condition (i.e., positive phase)
p_α^-	probability that the visible neurons of a Boltzmann machine are in state α , given that the network is in its free-running condition (i.e., negative phase)
$\hat{r}_x(j, k; n)$	estimate of autocorrelation function of $x_j(n)$ and $x_k(n)$
$\hat{r}_{dx}(k; n)$	estimate of cross-correlation function of $d(n)$ and $x_k(n)$
\mathbf{R}	correlation matrix of an input vector
t	continuous time
T	temperature
\mathcal{T}	training set (sample)
tr	trace of a matrix operator
var	variance operator
$V(\mathbf{x})$	Lyapunov function of state vector \mathbf{x}
v_j	induced local field or activation potential of neuron j
\mathbf{w}_o	optimum value of synaptic weight vector
w_{kj}	synaptic weight of synapse j belonging to neuron k
\mathbf{w}^*	optimum weight vector
$\bar{\mathbf{x}}$	equilibrium value of state vector \mathbf{x}
$\langle x_j \rangle$	average of state x_j in a "thermal" sense
\hat{x}	estimate of x , signified by the use of a caret (hat)
$ x $	absolute value (magnitude) of x
x^*	complex conjugate of x , signified by asterisk as superscript
$\ \mathbf{x}\ $	Euclidean norm (length) of vector \mathbf{x}
\mathbf{x}^T	transpose of vector \mathbf{x} , signified by the superscript T
z^{-1}	unit delay operator
Z	partition function

$\delta_j(n)$	local gradient of neuron j at time n
Δw	small change applied to weight w
∇	gradient operator
∇^2	Laplacian operator
$\nabla_w J$	gradient of J with respect to w
$\nabla \cdot \mathbf{F}$	divergence of vector \mathbf{F}
η	learning-rate parameter
κ	cumulant
μ	policy
θ_k	threshold applied to neuron k (i.e., negative of bias b_k)
λ	regularization parameter
λ_k	k th eigenvalue of a square matrix
$\varphi_k(\cdot)$	nonlinear activation function of neuron k
\in	symbol for “belongs to”
\cup	symbol for “union of”
\cap	symbol for “intersection of”
$*$	symbol for convolution
$+$	superscript symbol for pseudoinverse of a matrix

Open and closed intervals

- The open interval (a, b) of a variable x signifies that $a < x < b$.
- The closed interval $[a, b]$ of a variable x signifies that $a \leq x \leq b$.
- The closed-open interval $[a, b)$ of a variable x signifies that $a \leq x < b$; likewise for the open-closed interval $(a, b]$.

Minima and Maxima

- The symbol $\arg \min_{\mathbf{w}} f(\mathbf{w})$ signifies the minimum of the function $f(\mathbf{w})$ with respect to the argument vector \mathbf{w} .
- The symbol $\arg \max_{\mathbf{w}} f(\mathbf{w})$ signifies the maximum of the function $f(\mathbf{w})$ with respect to the argument vector \mathbf{w} .

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 machine¹:

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 *feed-back* 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

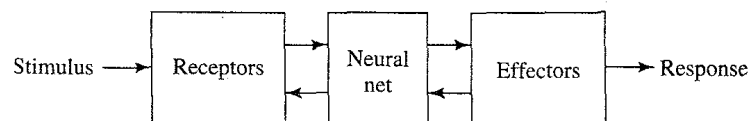


FIGURE 1.1 Block diagram representation of nervous system.

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

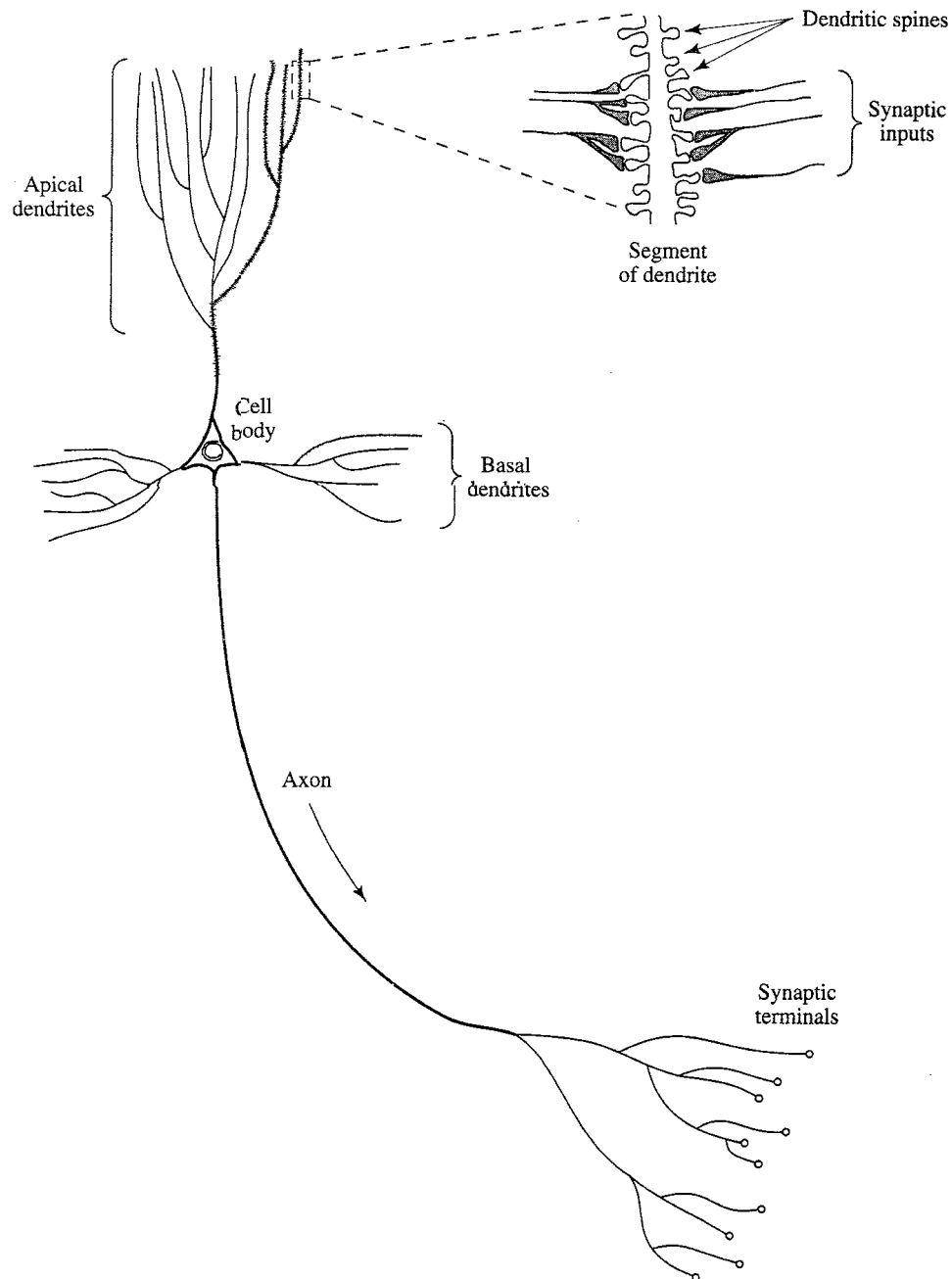


FIGURE 1.2 The pyramidal cell.

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,

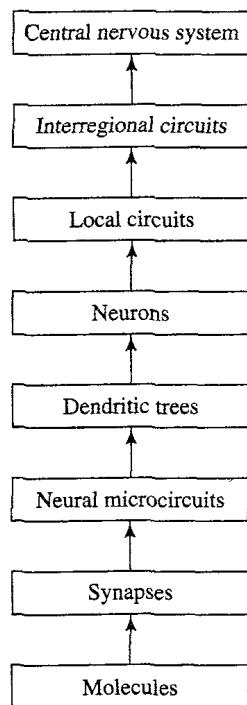


FIGURE 1.3 Structural organization of levels in the brain.

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.

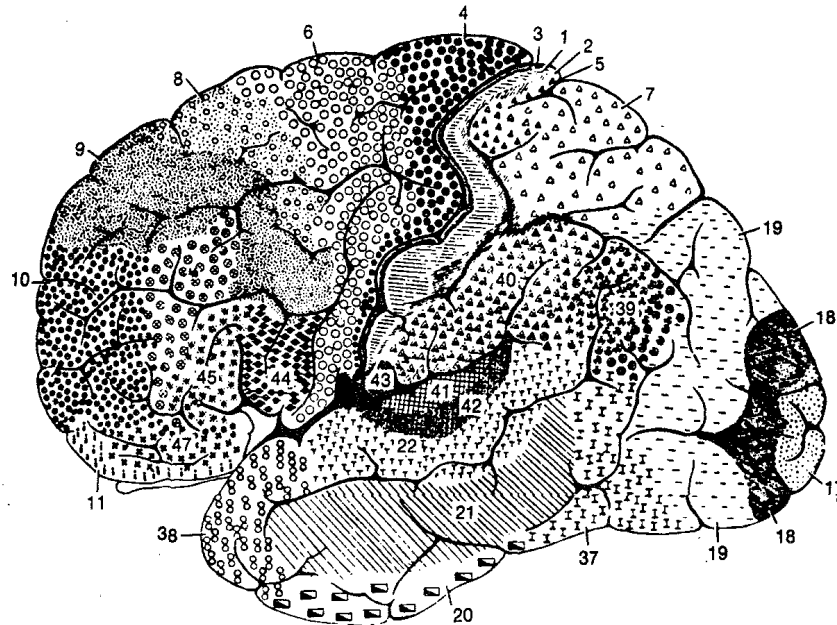


FIGURE 1.4 Cytoarchitectural map of the cerebral cortex. The different areas are identified by the thickness of their layers and types of cells within them. Some of the most important specific areas are as follows. Motor cortex: motor strip, area 4; premotor area, area 6; frontal eye fields, area 8. 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.

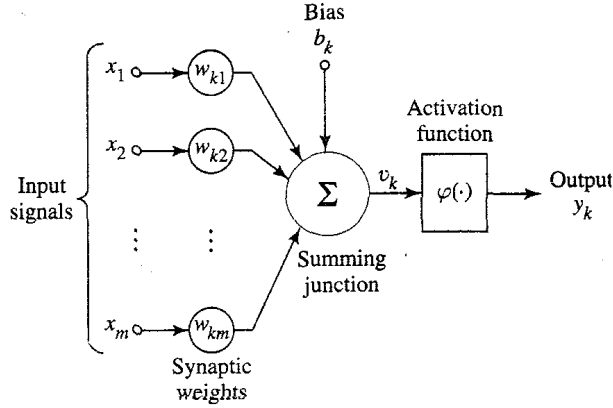


FIGURE 1.5 Nonlinear model of a neuron.

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:

$$u_k = \sum_{j=1}^m w_{kj} x_j \quad (1.1)$$

and

$$y_k = \varphi(u_k + b_k) \quad (1.2)$$

where x_1, x_2, \dots, x_m are the input signals; $w_{k1}, w_{k2}, \dots, w_{km}$ 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

$$v_k = u_k + b_k \quad (1.3)$$

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:

$$v_k = \sum_{j=0}^m w_{kj} x_j \quad (1.4)$$

and

$$y_k = \varphi(v_k) \quad (1.5)$$

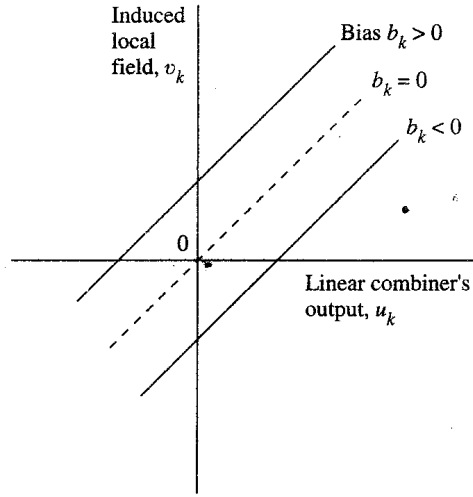


FIGURE 1.6 Affine transformation produced by the presence of a bias; note that $v_k = b_k$ at $u_k = 0$.

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

$$x_0 = +1 \quad (1.6)$$

and its weight is

$$w_{k0} = b_k \quad (1.7)$$

We may therefore reformulate the model of neuron 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

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

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

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

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

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

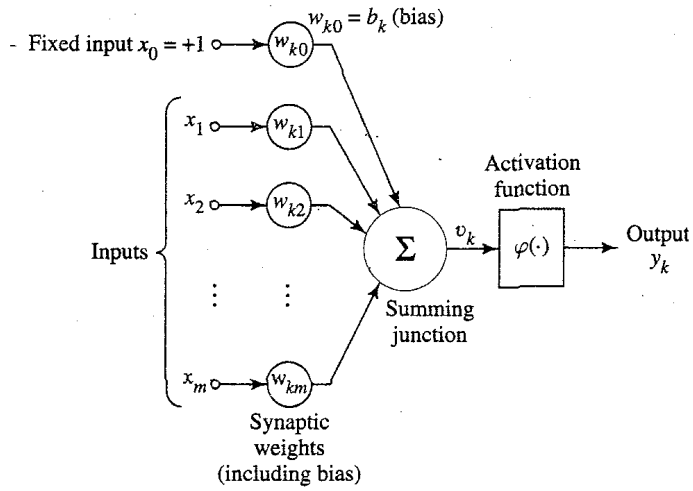


FIGURE 1.7 Another nonlinear model of a neuron.

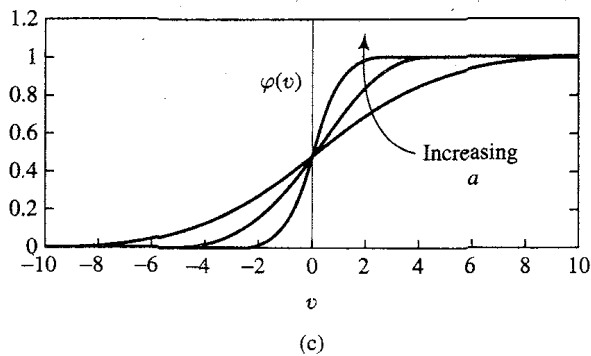
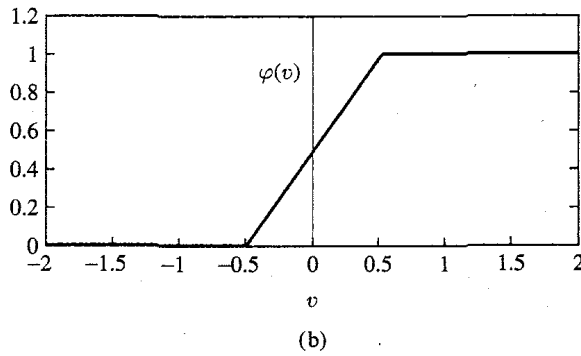
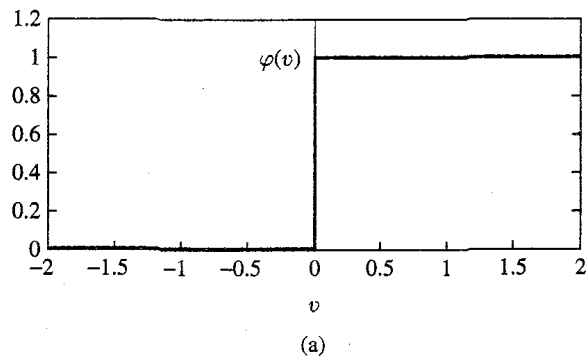


FIGURE 1.8 (a) Threshold function. (b) Piecewise-linear function. (c) Sigmoid function for varying slope parameter a .

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

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

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 artificial 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

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

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

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

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

$$\varphi(v) = \tanh(v) \quad (1.14)$$

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):

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

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 $T \rightarrow 0$, 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.

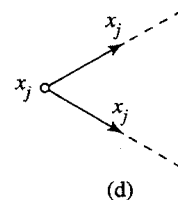
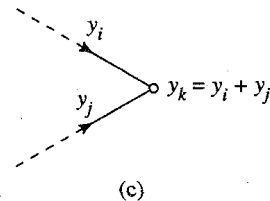
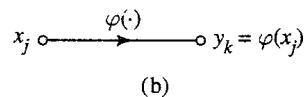
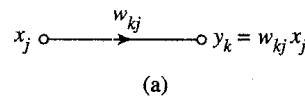


FIGURE 1.9 Illustrating basic rules for the construction of signal-flow graphs.

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.

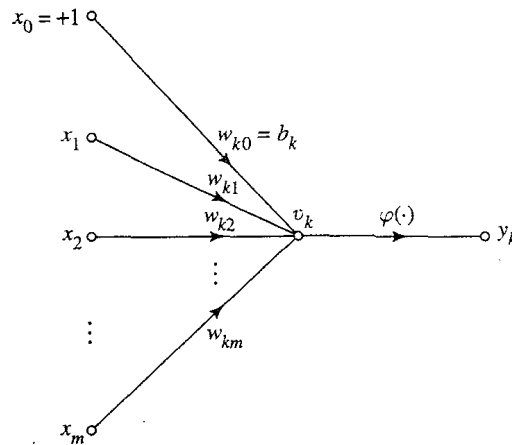


FIGURE 1.10 Signal-flow graph of a neuron.

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:

$$y_k(n) = A[x'_j(n)] \quad (1.16)$$

FIGURE 1.11 Architectural graph of a neuron.

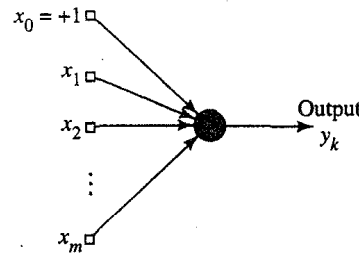
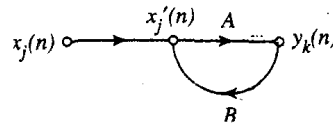


FIGURE 1.12 Signal-flow graph of a single-loop feedback system.



$$x'_j(n) = x_j(n) + B[y_k(n)] \quad (1.17)$$

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

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

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

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

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

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

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

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

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

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

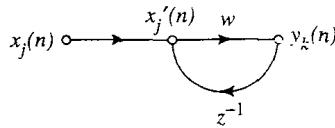


FIGURE 1.13 Signal-flow graph of a first-order, infinite-duration impulse response (IIR) filter.

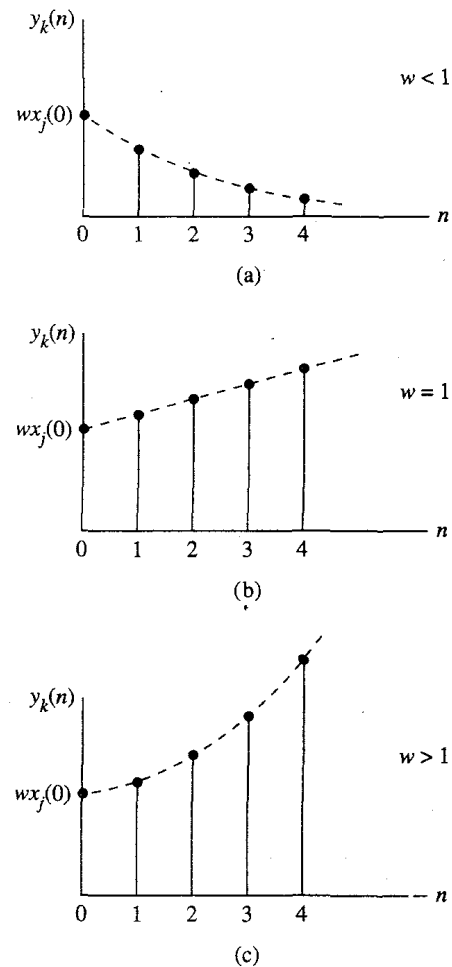


FIGURE 1.14 Time response of Fig. 1.13 for three different values of forward weight w . (a) Stable. (b) Linear divergence. (c) Exponential divergence.

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 virtue 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

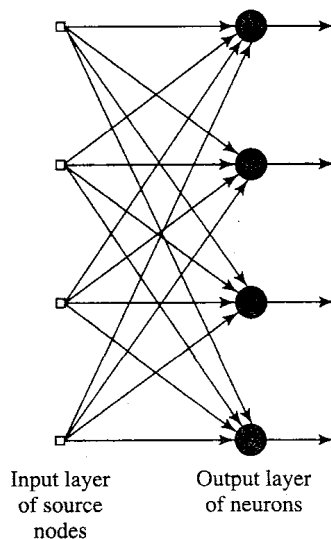


FIGURE 1.15 Feedforward or acyclic network with a single layer of neurons.

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*.

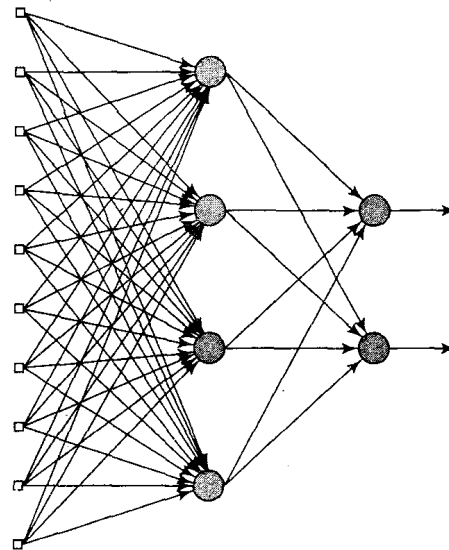


FIGURE 1.16 Fully connected feedforward or acyclic network with one hidden layer and one output layer.

Input layer
of source
nodes

Layer of
hidden
neurons

Layer of
output
neurons

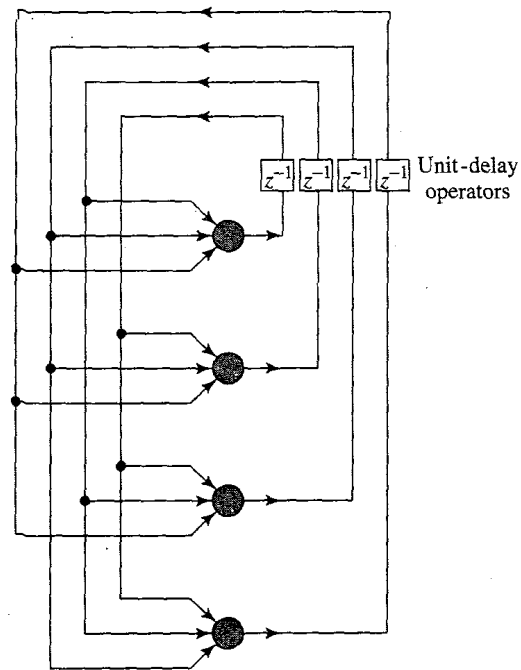


FIGURE 1.17 Recurrent network with no self-feedback loops and no hidden neurons.

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.

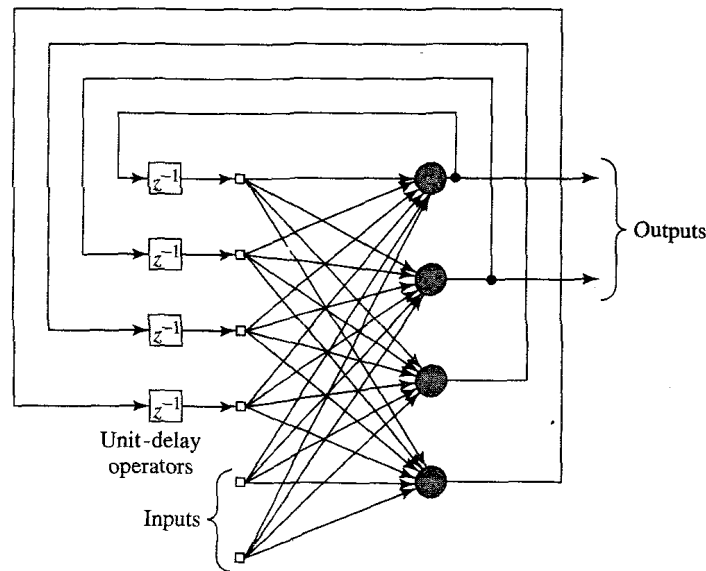


FIGURE 1.18 Recurrent network with hidden neurons.

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 identity 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 Euclidean distance. To be specific, let \mathbf{x}_i denote an m -by-1 vector

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

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

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

where x_{ik} and x_{jk} are the k th 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).

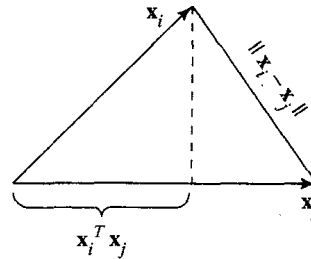
Another measure of similarity is based on the idea of a *dot product* or *inner product* that is also borrowed from matrix algebra. Given a pair of vectors \mathbf{x}_i and \mathbf{x}_j of the same dimension, their inner product is $\mathbf{x}_i^T \mathbf{x}_j$ written in expanded form as follows:

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

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 similarity 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

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

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,

$$\boldsymbol{\mu}_i = E[\mathbf{x}_i] \quad (1.26)$$

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):

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

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

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

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

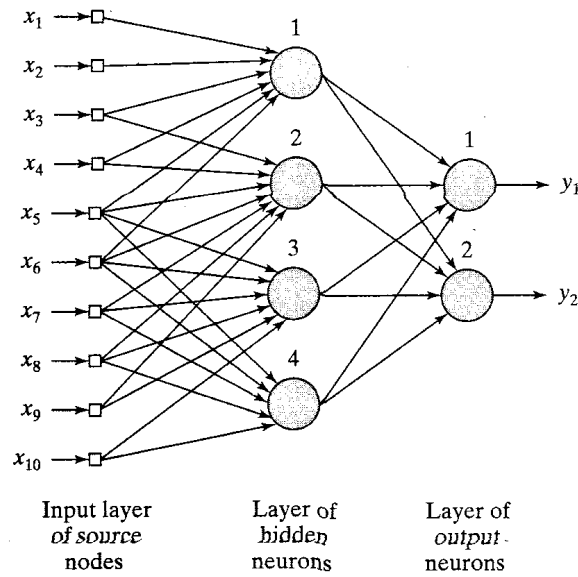


FIGURE 1.20 Illustrating the combined use of a receptive field and weight-sharing. All four hidden neurons share the same set of weights for their synaptic connections.

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:

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

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.

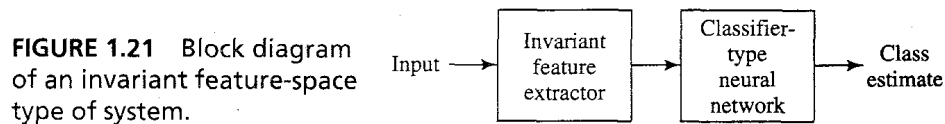
In order to build an object recognition system, a radar target recognition system and a speech recognition system for dealing with these phenomena, respectively, the system must be capable of coping with a range of *transformations* of the observed signal (Barnard and Casasent, 1991). Accordingly, a 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 neural 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 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-



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

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

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

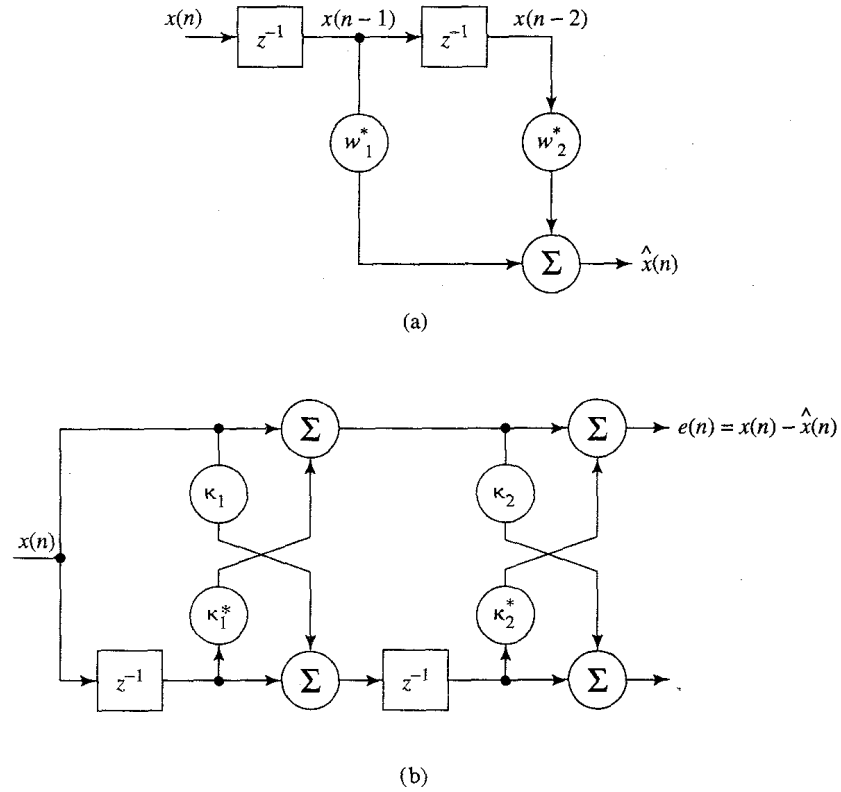


FIGURE 1.22 Autoregressive model of order 2: (a) tapped-delay-line model; (b) lattice filter model. (The asterisk denotes complex conjugation.)

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

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

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 $\kappa'_1, \kappa'_2, \dots, \kappa'_M$, 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.

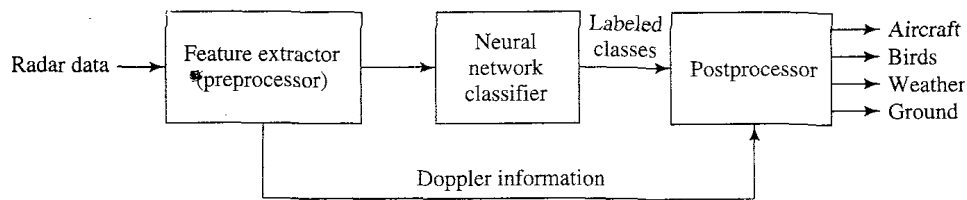


FIGURE 1.23 Doppler shift-invariant classifier of radar signals.

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 architecture 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-

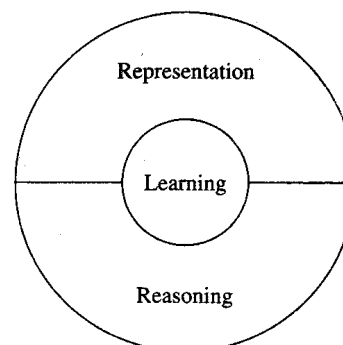


FIGURE 1.24 Illustrating the three key components of an AI system.

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

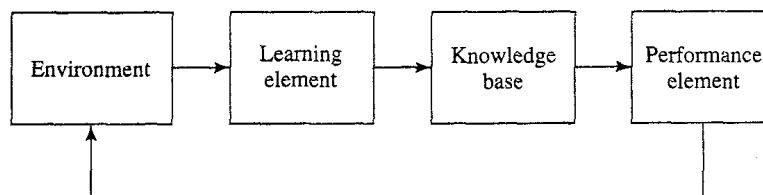


FIGURE 1.25 Simple model of machine learning.

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 *Neurodynamic 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 McClelland 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?

1.2 An odd sigmoid function is defined by

$$\begin{aligned}\varphi(v) &= \frac{1 - \exp(-av)}{1 + \exp(-av)} \\ &= \tanh\left(\frac{av}{2}\right)\end{aligned}$$

where \tanh denotes a hyperbolic tangent. The limiting values of this second sigmoid function are -1 and $+1$. Show that the derivative of $\varphi(v)$ with respect to v is given by

$$\frac{d\varphi}{dv} = \frac{a}{2}[1 - \varphi^2(v)]$$

What is the value of this derivative at the origin? Suppose that the slope parameter a is made infinitely large. What is the resulting form of $\varphi(v)$?

1.3 Yet another odd sigmoid function is the algebraic sigmoid:

$$\varphi(v) = \frac{v}{\sqrt{1 + v^2}}$$

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

$$\frac{d\varphi}{dv} = \frac{\varphi^3(v)}{v^3}$$

What is the value of this derivative at the origin?

1.4 Consider the following two functions:

$$\begin{aligned}\text{(i)} \quad \varphi(v) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^v \exp\left(-\frac{x^2}{2}\right) dx \\ \text{(ii)} \quad \varphi(v) &= \frac{2}{\pi} \tan^{-1}(v)\end{aligned}$$

Explain why both of these functions fit the requirements of a sigmoid function. How do these two functions differ from each other?

1.5 Which of the five sigmoid functions described in Problems 1.1 to 1.4 would qualify as a cumulative (probability) distribution function? Justify your answer.

1.6 Consider the pseudolinear activation function $\varphi(v)$ shown in Fig. P1.6.

- (a) Formulate $\varphi(v)$ as a function of v .
- (b) What happens to $\varphi(v)$ if a is allowed to approach zero?

1.7 Repeat Problem 1.6 for the pseudolinear activation function $\varphi(v)$ shown in Fig. P1.7.

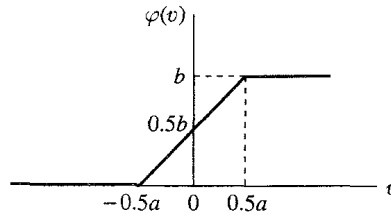


FIGURE P1.6

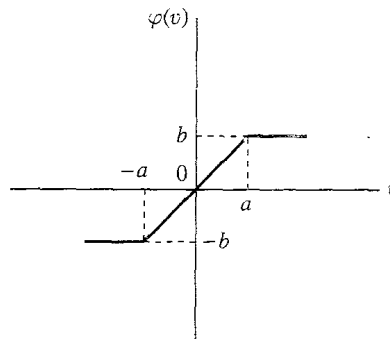


FIGURE P1.7

- 1.8** A neuron has an activation function $\varphi(v)$ defined by the logistic function of Problem 1.1, where v is the induced local field, and the slope parameter a is available for adjustment. Let x_1, x_2, \dots, x_m denote the input signals applied to the source nodes of the neuron, and b denote the bias. For convenience of presentation, we would like to absorb the slope parameter a in the induced local field v by writing

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

How would you modify the inputs x_1, x_2, \dots, x_m to produce the same output as before? Justify your answer.

- 1.9** A neuron j receives inputs from four other neurons whose activity levels are 10, -20 , 4, and -2 . The respective synaptic weights of neuron j are 0.8, 0.2, -1.0 , and -0.9 . Calculate the output of neuron j for the following two situations:
- (a) The neuron is linear.
 - (b) The neuron is represented by a McCulloch–Pitts model. Assume that the bias applied to the neuron is zero.
- 1.10** Repeat Problem 1.9 for a neuron model based on the logistic function

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

- 1.11** (a) Show that the McCulloch–Pitts formal model of a neuron may be approximated by a sigmoidal neuron (i.e., neuron using a sigmoid activation function with large synaptic weights).
- (b) Show that a linear neuron may be approximated by a sigmoidal neuron with small synaptic weights.

Network architectures

- 1.12** A fully connected feedforward network has 10 source nodes, 2 hidden layers, one with 4 neurons and the other with 3 neurons, and a single output neuron. Construct an architectural graph of this network.
- 1.13** (a) Figure P1.13 shows the signal-flow graph of a 2-2-2-1 feedforward network. The function $\varphi(\cdot)$ denotes a logistic function. Write the input–output mapping defined by this network.
- (b) Suppose that the output neuron in the signal-flow graph of Fig. P1.13 operates in its linear region. Write the input–output mapping defined by this new network.

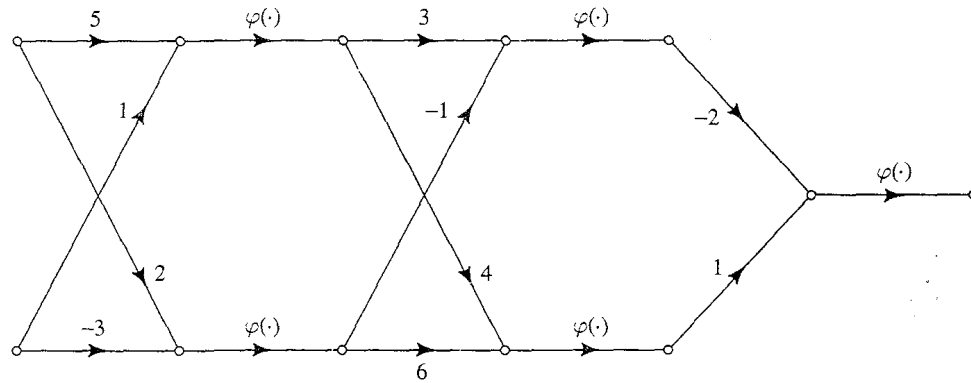


FIGURE P1.13

- 1.14 The network described in Fig. P1.13 has no biases. Suppose that biases equal to -1 and $+1$ are applied to the top and bottom neurons of the first hidden layer, and biases equal to $+1$ and -2 are applied to the top and bottom neurons of the second hidden layer. Write the new form of the input-output mapping defined by the network.
- 1.15 Consider a multilayer feedforward network, all the neurons of which operate in their linear regions. Justify the statement that such a network is equivalent to a single-layer feedforward network.
- 1.16 Construct a fully recurrent network with 5 neurons, but with no self-feedback.
- 1.17 Figure P1.17 shows the signal-flow graph of a recurrent network made up of two neurons. Write the nonlinear difference equation that defines the evolution of $x_1(n)$ or that of $x_2(n)$. These two variables define the outputs of the top and bottom neurons, respectively. What is the order of this equation?
- 1.18 Figure P1.18 shows the signal-flow graph of a recurrent network consisting of two neurons with self-feedback. Write the coupled system of two first-order nonlinear difference equations that describe the operation of the system.
- 1.19 A recurrent network has 3 source nodes, 2 hidden neurons, and 4 output neurons. Construct an architectural graph that describes such a network.

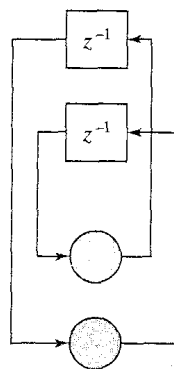


FIGURE P1.17

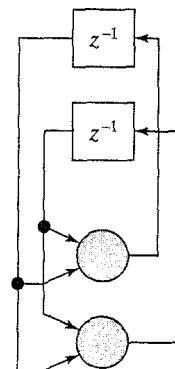


FIGURE P1.18

Knowledge representation

- 1.20** A useful form of preprocessing is based on the *autoregressive (AR) model* described by the difference equation (for real-valued data)

$$y(n) = w_1 y(n-1) + w_2 y(n-2) + \cdots + w_M y(n-M) + v(n)$$

where $y(n)$ is the model output; $v(n)$ is a sample drawn from a white-noise process of zero mean and some prescribed variance; w_1, w_2, \dots, w_M are the *AR* model coefficients; and M is the model order. Show that the use of this model provides two forms of geometric invariance: (a) scale, and (b) time translation. How could these two invariances be used in neural networks?

- 1.21** Let \mathbf{x} be an input vector, and $\mathbf{s}(\alpha, \mathbf{x})$ be a transformation operator acting on \mathbf{x} and depending on some parameter α . The operator $\mathbf{s}(\alpha, \mathbf{x})$ satisfies two requirements:

- $\mathbf{s}(0, \mathbf{x}) = \mathbf{x}$
- $\mathbf{s}(\alpha, \mathbf{x})$ is differentiable with respect to α .

The *tangent vector* is defined by the partial derivative $\partial \mathbf{s}(\alpha, \mathbf{x}) / \partial \alpha$ (Simard et al., 1992).

Suppose that \mathbf{x} represents an image, and α is a rotation parameter. How would you compute the tangent vector for the case when α is small? The tangent vector is locally invariant with respect to rotation of the original image; why?

Learning Processes

2.1 INTRODUCTION

The property that is of primary significance for a neural network is the ability of the network to *learn* from its environment, and to *improve* its performance through learning. The improvement in performance takes place over time in accordance with some prescribed measure. A neural network learns about its environment through an interactive process of adjustments applied to its synaptic weights and bias levels. Ideally, the network becomes more knowledgeable about its environment after each iteration of the learning process.

There are too many activities associated with the notion of “learning” to justify defining it in a precise manner. Moreover, the process of learning is a matter of viewpoint, which makes it all the more difficult to agree on a precise definition of the term. For example, learning as viewed by a psychologist is quite different from learning in a classroom sense. Recognizing that our particular interest is in neural networks, we use a definition of learning that is adapted from Mendel and McClaren (1970).

We define learning in the context of neural networks as:

Learning is a process by which the free parameters of a neural network are adapted through a process of stimulation by the environment in which the network is embedded. The type of learning is determined by the manner in which the parameter changes take place.

This definition of the learning process implies the following sequence of events:

1. The neural network is *stimulated* by an environment.
2. The neural network *undergoes changes* in its free parameters as a result of this stimulation.
3. The neural network *responds in a new way* to the environment because of the changes that have occurred in its internal structure.

A prescribed set of well-defined rules for the solution of a learning problem is called a *learning algorithm*.¹ As one would expect, there is no unique learning algorithm for the design of neural networks. Rather, we have a “kit of tools” represented by a diverse variety of learning algorithms, each of which offers advantages of its own. Basically, learning algorithms differ from each other in the way in which the adjust-

ment to a synaptic weight of a neuron is formulated. Another factor to be considered is the manner in which a neural network (learning machine), made up of a set of interconnected neurons, relates to its environment. In this latter context we speak of a *learning paradigm* that refers to a *model* of the environment in which the neural network operates.

Organization of the Chapter

The chapter is organized in four interrelated parts. In the first part, consisting of Sections 2.2 through 2.6, we discuss five basic learning rules: error-correction learning, memory-based learning, Hebbian learning, competitive learning, and Boltzmann learning. Error-correction learning is rooted in optimum filtering. Memory-based learning operates by memorizing the training data explicitly. Hebbian learning and competitive learning are both inspired by neurobiological considerations. Boltzmann learning is different because it is based on ideas borrowed from statistical mechanics.

The second part of the chapter explores learning paradigms. Section 2.7 discusses the credit-assignment problem, which is basic to the learning process. Sections 2.8 and 2.9 present overviews of the two fundamental learning paradigms: (1) learning *with* a teacher, and (2) learning *without* a teacher.

The third part of the chapter, consisting of Sections 2.10 through 2.12, examines the issues of learning tasks, memory, and adaptation.

The final part of the chapter, consisting of Sections 2.13 through 2.15, deals with probabilistic and statistical aspects of the learning process. Section 2.13 discusses the bias/variance dilemma. Section 2.14 discusses statistical learning theory, based on the notion of VC-dimension that provides a measure of machine capacity. Section 2.14 introduces another important concept: probably approximately correct (PAC) learning, which provides a conservative model for the learning process.

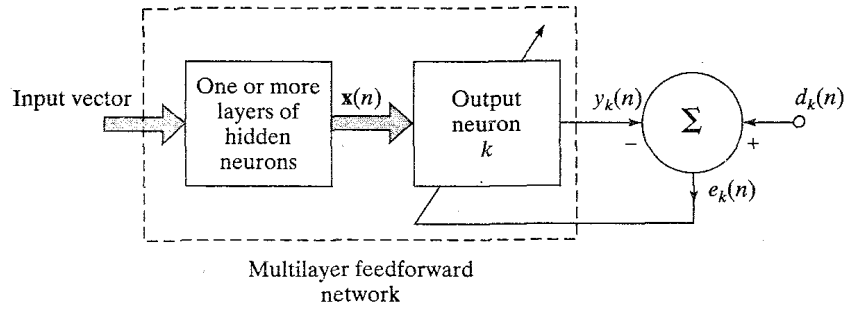
The chapter concludes with some final remarks in Section 2.16.

2.2 ERROR-CORRECTION LEARNING

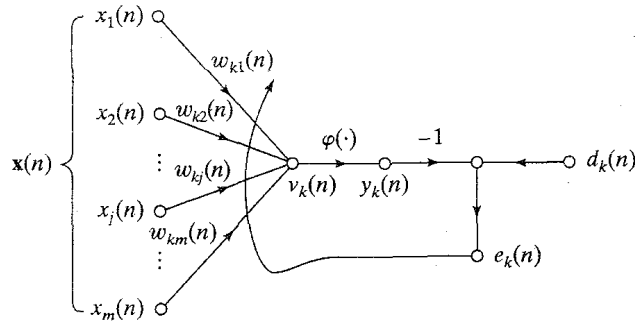
To illustrate our first learning rule, consider the simple case of a neuron k constituting the only computational node in the output layer of a feedforward neural network, as depicted in Fig. 2.1a. Neuron k is driven by a *signal vector* $\mathbf{x}(n)$ produced by one or more layers of hidden neurons, which are themselves driven by an input vector (stimulus) applied to the source nodes (i.e., input layer) of the neural network. The argument n denotes discrete time, or more precisely, the time step of an iterative process involved in adjusting the synaptic weights of neuron k . The *output signal* of neuron k is denoted by $y_k(n)$. This output signal, representing the only output of the neural network, is compared to a *desired response* or *target output*, denoted by $d_k(n)$. Consequently, an *error signal*, denoted by $e_k(n)$, is produced. By definition, we thus have

$$e_k(n) = d_k(n) - y_k(n) \quad (2.1)$$

The error signal $e_k(n)$ actuates a *control mechanism*, the purpose of which is to apply a sequence of corrective adjustments to the synaptic weights of neuron k . The corrective adjustments are designed to make the output signal $y_k(n)$ come closer to the desired



(a) Block diagram of a neural network, highlighting the only neuron in the output layer



(b) Signal-flow graph of output neuron

FIGURE 2.1 Illustrating error-correction learning.

response $d_k(n)$ in a step-by-step manner. This objective is achieved by minimizing a *cost function* or *index of performance*, $\mathcal{E}(n)$, defined in terms of the error signal $e_k(n)$ as:

$$\mathcal{E}(n) = \frac{1}{2} e_k^2(n) \quad (2.2)$$

That is, $\mathcal{E}(n)$ is the *instantaneous value of the error energy*. The step-by-step adjustments to the synaptic weights of neuron k are continued until the system reaches a *steady state* (i.e., the synaptic weights are essentially stabilized). At that point the learning process is terminated.

The learning process described herein is obviously referred to as *error-correction learning*. In particular, minimization of the cost function $\mathcal{E}(n)$ leads to a learning rule commonly referred to as the *delta rule* or *Widrow-Hoff rule*, named in honor of its originators (Widrow and Hoff, 1960). Let $w_{kj}(n)$ denote the value of synaptic weight w_{kj} of neuron k excited by element $x_j(n)$ of the signal vector $\mathbf{x}(n)$ at time step n . According to the delta rule, the adjustment $\Delta w_{kj}(n)$ applied to the synaptic weight w_{kj} at time step n is defined by

$$\Delta w_{kj}(n) = \eta e_k(n) x_j(n) \quad (2.3)$$

where η is a positive constant that determines the *rate of learning* as we proceed from one step in the learning process to another. It is therefore natural that we refer to η as the *learning-rate parameter*. In other words, the delta rule may be stated as:

The adjustment made to a synaptic weight of a neuron is proportional to the product of the error signal and the input signal of the synapse in question.

Keep in mind that the delta rule, as stated herein, presumes that the error signal is *directly measurable*. For this measurement to be feasible we clearly need a supply of desired response from some external source, which is directly accessible to neuron k . In other words, neuron k is *visible* to the outside world, as depicted in Fig. 2.1a. From this figure we also observe that error-correction learning is in fact *local* in nature. This is merely saying that the synaptic adjustments made by the delta rule are localized around neuron k .

Having computed the synaptic adjustment $\Delta w_{kj}(n)$, the updated value of synaptic weight w_{kj} is determined by

$$w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n) \quad (2.4)$$

In effect, $w_{kj}(n)$ and $w_{kj}(n+1)$ may be viewed as the *old* and *new* values of synaptic weight w_{kj} , respectively. In computational terms we may also write

$$w_{kj}(n) = z^{-1}[w_{kj}(n+1)] \quad (2.5)$$

where z^{-1} is the *unit-delay operator*. That is, z^{-1} represents a *storage element*.

Figure 2.1b shows a signal-flow graph representation of the error-correction learning process, focusing on the activity surrounding neuron k . The input signal x_j and induced local field v_k of neuron k are referred to as the *presynaptic* and *postsynaptic signals* of the j th synapse of neuron k , respectively. From Fig. 2.1b we see that error-correction learning is an example of a *closed-loop feedback system*. From control theory we know that the stability of such a system is determined by those parameters that constitute the feedback loops of the system. In our case we only have a single feedback loop, and one of those parameters of particular interest is the learning-rate parameter η . It is therefore important that η is carefully selected to ensure that the stability or convergence of the iterative learning process is achieved. The choice of η also has a profound influence on the accuracy and other aspects of the learning process. In short, the learning-rate parameter η plays a key role in determining the performance of error-correction learning in practice.

Error-correction learning is discussed in much greater detail in Chapter 3, which discusses single-layer feedforward networks and in Chapter 4, which details multilayer feedforward networks.

2.3 MEMORY-BASED LEARNING

In *memory-based learning*, all (or most) of the past experiences are explicitly stored in a large memory of correctly classified input-output examples: $\{(\mathbf{x}_i, d_i)\}_{i=1}^N$, where \mathbf{x}_i denotes an input vector and d_i denotes the corresponding desired response. Without loss of generality, we have restricted the desired response to be a scalar. For example, in a binary pattern classification problem there are two classes/hypotheses, denoted by \mathcal{C}_1 and \mathcal{C}_2 , to be considered. In this example, the desired response d_i takes the value 0 (or -1) for class \mathcal{C}_1 and the value 1 for class \mathcal{C}_2 . When classification of a test vector \mathbf{x}_{test} (not seen before) is required, the algorithm responds by retrieving and analyzing the training data in a “local neighborhood” of \mathbf{x}_{test} .

All memory-based learning algorithms involve two essential ingredients:

- Criterion used for defining the local neighborhood of the test vector \mathbf{x}_{test} .
- Learning rule applied to the training examples in the local neighborhood of \mathbf{x}_{test} .

The algorithms differ from each other in the way in which these two ingredients are defined.

In a simple yet effective type of memory-based learning known as the *nearest neighbor rule*,² the local neighborhood is defined as the training example that lies in the immediate neighborhood of the test vector \mathbf{x}_{test} . In particular, the vector

$$\mathbf{x}'_N \in \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\} \quad (2.6)$$

is said to be the nearest neighbor of \mathbf{x}_{test} if

$$\min_i d(\mathbf{x}_i, \mathbf{x}_{\text{test}}) = d(\mathbf{x}'_N, \mathbf{x}_{\text{test}}) \quad (2.7)$$

where $d(\mathbf{x}_i, \mathbf{x}_{\text{test}})$ is the Euclidean distance between the vectors \mathbf{x}_i and \mathbf{x}_{test} . The class associated with the minimum distance, that is, vector \mathbf{x}'_N , is reported as the classification of \mathbf{x}_{test} . This rule is independent of the underlying distribution responsible for generating the training examples.

Cover and Hart (1967) have formally studied the nearest neighbor rule as a tool for pattern classification. The analysis presented therein is based on two assumptions:

- The classified examples (\mathbf{x}_i, d_i) are *independently and identically distributed (iid)*, according to the joint probability distribution of the example (\mathbf{x}, d) .
- The sample size N is infinitely large.

Under these two assumptions, it is shown that the probability of classification error incurred by the nearest neighbor rule is bounded above by twice the *Bayes probability of error*, that is, the minimum probability of error over all decision rules. Bayes probability of error is discussed in Chapter 3. In this sense, it may be said that half the classification information in a training set of infinite size is contained in the nearest neighbor, which is a surprising result.

A variant of the nearest neighbor classifier is the *k-nearest neighbor classifier*, which proceeds as follows:

- Identify the k classified patterns that lie nearest to the test vector \mathbf{x}_{test} for some integer k .
- Assign \mathbf{x}_{test} to the class (hypothesis) that is most frequently represented in the k nearest neighbors to \mathbf{x}_{test} (i.e., use a majority vote to make the classification).

Thus the k -nearest neighbor classifier acts like an averaging device. In particular, it discriminates against a single outlier, as illustrated in Fig. 2.2 for $k = 3$. An *outlier* is an observation that is improbably large for a nominal model of interest.

In Chapter 5 we discuss another important type of memory-based classifier known as the radial-basis function network.

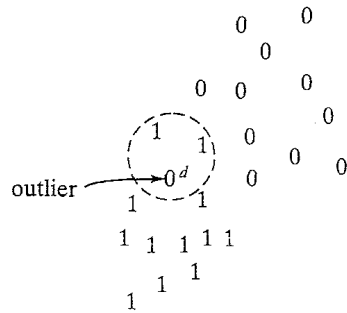


FIGURE 2.2 The area lying inside the dashed circle includes two points pertaining to class 1 and an outlier from class 0. The point d corresponds to the test vector \mathbf{x}_{test} . With $k = 3$, the k -nearest neighbor classifier assigns class 1 to point d even though it lies closest to the outlier.

2.4 HEBBIAN LEARNING

Hebb's postulate of learning is the oldest and most famous of all learning rules; it is named in honor of the neuropsychologist Hebb (1949). Quoting from Hebb's book, *The Organization of Behavior* (1949, p.62):

When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic changes take place in one or both cells such that A's efficiency as one of the cells firing B, is increased.

Hebb proposed this change as a basis of associative learning (at the cellular level), which would result in an enduring modification in the activity pattern of a spatially distributed "assembly of nerve cells."

This statement is made in a neurobiological context. We may expand and rephrase it as a two-part rule (Stent, 1973; Changeux and Danchin, 1976):

1. If two neurons on either side of a synapse (connection) are activated simultaneously (i.e., synchronously), then the strength of that synapse is selectively increased.
2. If two neurons on either side of a synapse are activated asynchronously, then that synapse is selectively weakened or eliminated.

Such a synapse is called a *Hebbian synapse*.³ (The original Hebb rule did not contain part 2.) More precisely, we define a Hebbian synapse as a synapse that uses a *time-dependent, highly local, and strongly interactive mechanism to increase synaptic efficiency as a function of the correlation between the presynaptic and postsynaptic activities*. From this definition we may deduce the following four key mechanisms (properties) that characterize a Hebbian synapse (Brown et al., 1990):

1. *Time-dependent mechanism.* This mechanism refers to the fact that the modifications in a Hebbian synapse depend on the exact time of occurrence of the presynaptic and postsynaptic signals.

2. *Local mechanism.* By its very nature, a synapse is the transmission site where information-bearing signals (representing ongoing activity in the presynaptic and postsynaptic units) are in *spatiotemporal* contiguity. This locally available information is used by a Hebbian synapse to produce a local synaptic modification that is input specific.

3. *Interactive mechanism.* The occurrence of a change in a Hebbian synapse depends on signals on both sides of the synapse. That is, a Hebbian form of learning depends on a “true interaction” between presynaptic and postsynaptic signals in the sense that we cannot make a prediction from either one of these two activities by itself. Note also that this dependence or interaction may be deterministic or statistical in nature.

4. *Conjunctional or correlational mechanism.* One interpretation of Hebb’s postulate of learning is that the condition for a change in synaptic efficiency is the conjunction of presynaptic and postsynaptic signals. Thus, according to this interpretation, the co-occurrence of presynaptic and postsynaptic signals (within a short interval of time) is sufficient to produce the synaptic modification. It is for this reason that a Hebbian synapse is sometimes referred to as a *conjunctional synapse*. For another interpretation of Hebb’s postulate of learning, we may think of the interactive mechanism characterizing a Hebbian synapse in statistical terms. In particular, the correlation over time between presynaptic and postsynaptic signals is viewed as being responsible for a synaptic change. Accordingly, a Hebbian synapse is also referred to as a *correlational synapse*. Correlation is indeed the basis of learning (Eggermont, 1990).

Synaptic Enhancement and Depression

The definition of a Hebbian synapse presented here does not include additional processes that may result in weakening of a synapse connecting a pair of neurons. Indeed, we may generalize the concept of a Hebbian modification by recognizing that positively correlated activity produces synaptic strengthening, and that either uncorrelated or negatively correlated activity produces synaptic weakening (Stent, 1973). Synaptic depression may also be of a noninteractive type. Specifically, the interactive condition for synaptic weakening may simply be noncoincident presynaptic or postsynaptic activity.

We may go one step further by classifying synaptic modifications as *Hebbian*, *anti-Hebbian*, and *non-Hebbian* (Palm, 1982). According to this scheme, a Hebbian synapse increases its strength with positively correlated presynaptic and postsynaptic signals, and decreases its strength when these signals are either uncorrelated or negatively correlated. Conversely, an anti-Hebbian synapse weakens positively correlated presynaptic and postsynaptic signals, and strengthens negatively correlated signals. In both Hebbian and anti-Hebbian synapses, however, the modification of synaptic efficiency relies on a mechanism that is time-dependent, highly local, and strongly interactive in nature. In that sense, an anti-Hebbian synapse is still Hebbian in nature, though not in function. A non-Hebbian synapse, on the other hand, does not involve a Hebbian mechanism of either kind.

Mathematical Models of Hebbian Modifications

To formulate Hebbian learning in mathematical terms, consider a synaptic weight w_{kj} of neuron k with presynaptic and postsynaptic signals denoted by x_j and y_k , respectively. The adjustment applied to the synaptic weight w_{kj} at time step n is expressed in the general form

$$\Delta w_{kj}(n) = F(y_k(n), x_j(n)) \quad (2.8)$$

where $F(\cdot, \cdot)$ is a function of both postsynaptic and presynaptic signals. The signals $x_j(n)$ and $y_k(n)$ are often treated as dimensionless. The formula of Eq. (2.8) admits many forms, all of which qualify as Hebbian. In what follows, we consider two such forms.

Hebb's hypothesis. The simplest form of Hebbian learning is described by

$$\Delta w_{kj}(n) = \eta y_k(n) x_j(n) \quad (2.9)$$

where η is a positive constant that determines the *rate of learning*. Equation (2.9) clearly emphasizes the correlational nature of a Hebbian synapse. It is sometimes referred to as the *activity product rule*. The top curve of Fig. 2.3 shows a graphical representation of Eq. (2.9) with the change Δw_{kj} plotted versus the output signal (postsynaptic activity) y_k . From this representation we see that the repeated application of the input signal (presynaptic activity) x_j leads to an increase in y_k and therefore *exponential growth* that finally drives the synaptic connection into saturation. At that point no information will be stored in the synapse and selectivity is lost.

Covariance hypothesis. One way of overcoming the limitation of Hebb's hypothesis is to use the *covariance hypothesis* introduced in Sejnowski (1977a, b). In this hypothesis, the presynaptic and postsynaptic signals in Eq. (2.9) are replaced by the departure of presynaptic and postsynaptic signals from their respective average values over a certain time interval. Let \bar{x} and \bar{y} denote the *time-averaged values* of the presynaptic signal x_j and postsynaptic signal y_k , respectively. According to the covariance hypothesis, the adjustment applied to the synaptic weight w_{kj} is defined by

$$\Delta w_{kj} = \eta(x_j - \bar{x})(y_k - \bar{y}) \quad (2.10)$$

where η is the learning-rate parameter. The average values \bar{x} and \bar{y} constitute presynaptic and postsynaptic thresholds, which determine the sign of synaptic modification. In particular, the covariance hypothesis allows for the following:

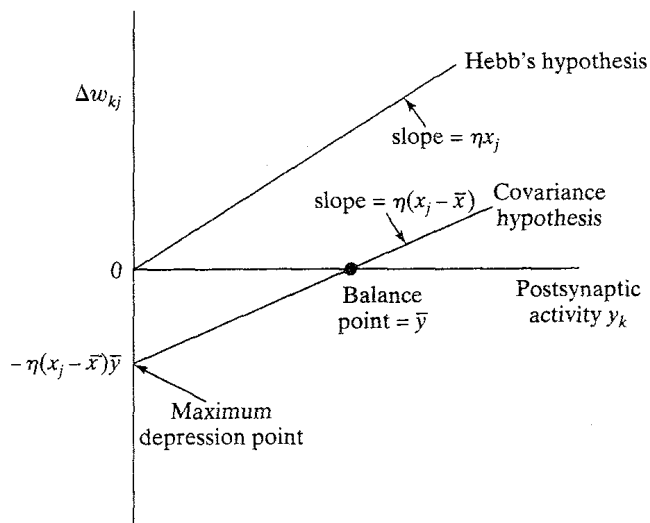


FIGURE 2.3 Illustration of Hebb's hypothesis and the covariance hypothesis.

- Convergence to a nontrivial state, which is reached when $x_k = \bar{x}$ or $y_j = \bar{y}$.
- Prediction of both synaptic *potentiation* (i.e., increase in synaptic strength) and synaptic *depression* (i.e., decrease in synaptic strength).

Figure 2.3 illustrates the difference between Hebb's hypothesis and the covariance hypothesis. In both cases the dependence of Δw_{kj} on y_k is linear; however, the intercept with the y_k -axis in Hebb's hypothesis is at the origin, whereas in the covariance hypothesis it is at $y_k = \bar{y}$.

We make the following important observations from Eq. (2.10):

1. Synaptic weight w_{kj} is enhanced if there are sufficient levels of presynaptic and postsynaptic activities, that is, the conditions $x_j > \bar{x}$ and $y_k > \bar{y}$ are both satisfied.
2. Synaptic weight w_{kj} is depressed if there is either
 - a presynaptic activation (i.e., $x_j > \bar{x}$) in the absence of sufficient postsynaptic activation (i.e., $y_k < \bar{y}$), or
 - a postsynaptic activation (i.e., $y_k > \bar{y}$) in the absence of sufficient presynaptic activation (i.e., $x_j < \bar{x}$).

This behavior may be regarded as a form of temporal competition between the incoming patterns.

There is strong physiological evidence⁴ for Hebbian learning in the area of the brain called the *hippocampus*. The hippocampus plays an important role in certain aspects of learning or memory. This physiological evidence makes Hebbian learning all the more appealing.

2.5 COMPETITIVE LEARNING

In *competitive learning*,⁵ as the name implies, the output neurons of a neural network compete among themselves to become active (fired). Whereas in a neural network based on Hebbian learning several output neurons may be active simultaneously, in competitive learning only a single output neuron is active at any one time. It is this feature that makes competitive learning highly suited to discover statistically salient features that may be used to classify a set of input patterns.

There are three basic elements to a competitive learning rule (Rumelhart and Zipser, 1985):

- A set of neurons that are all the same except for some randomly distributed synaptic weights, and which therefore *respond differently* to a given set of input patterns.
- A *limit* imposed on the "strength" of each neuron.
- A mechanism that permits the neurons to *compete* for the right to respond to a given subset of inputs, such that only *one* output neuron, or only one neuron per group, is active (i.e., "on") at a time. The neuron that wins the competition is called a *winner-takes-all neuron*.

Accordingly the individual neurons of the network learn to specialize on ensembles of similar patterns; in so doing they become *feature detectors* for different classes of input patterns.

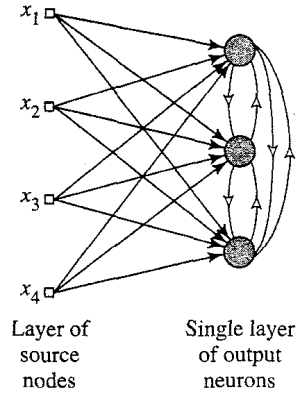


FIGURE 2.4 Architectural graph of a simple competitive learning network with feedforward (excitatory) connections from the source nodes to the neurons, and lateral (inhibitory) connections among the neurons; the lateral connections are signified by open arrows.

In the simplest form of competitive learning, the neural network has a single layer of output neurons, each of which is fully connected to the input nodes. The network may include feedback connections among the neurons, as indicated in Fig. 2.4. In the network architecture described herein, the feedback connections perform *lateral inhibition*,⁶ with each neuron tending to inhibit the neuron to which it is laterally connected. In contrast, the feedforward synaptic connections in the network of Fig. 2.4 are all *excitatory*.

For a neuron k to be the winning neuron, its induced local field v_k for a specified input pattern \mathbf{x} must be the largest among all the neurons in the network. The output signal y_k of winning neuron k is set equal to one; the output signals of all the neurons that lose the competition are set equal to zero. We thus write

$$y_k = \begin{cases} 1 & \text{if } v_k > v_j \text{ for all } j, j \neq k \\ 0 & \text{otherwise} \end{cases} \quad (2.11)$$

where the induced local field v_k represents the combined action of all the forward and feedback inputs to neuron k .

Let w_{kj} denote the synaptic weight connecting input node j to neuron k . Suppose that each neuron is allotted a *fixed* amount of synaptic weight (i.e., all synaptic weights are positive), which is distributed among its input nodes; that is,

$$\sum_j w_{kj} = 1 \quad \text{for all } k \quad (2.12)$$

A neuron then learns by shifting synaptic weights from its inactive to active input nodes. If a neuron does not respond to a particular input pattern, no learning takes place in that neuron. If a particular neuron wins the competition, each input node of that neuron relinquishes some proportion of its synaptic weight, and the weight relinquished is then distributed equally among the active input nodes. According to the standard *competitive learning rule*, the change Δw_{kj} applied to synaptic weight w_{kj} is defined by

$$\Delta w_{kj} = \begin{cases} \eta(x_j - w_{kj}) & \text{if neuron } k \text{ wins the competition} \\ 0 & \text{if neuron } k \text{ loses the competition} \end{cases} \quad (2.13)$$

where η is the learning-rate parameter. This rule has the overall effect of moving the synaptic weight vector \mathbf{w}_k of winning neuron k toward the input pattern \mathbf{x} .

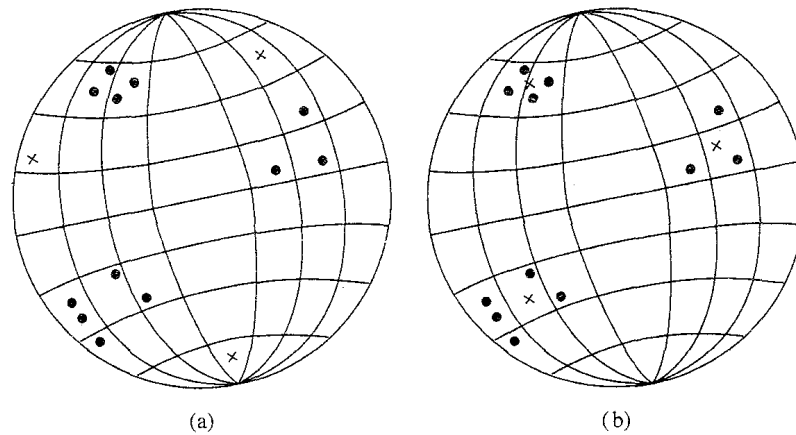


FIGURE 2.5 Geometric interpretation of the competitive learning process. The dots represent the input vectors, and the crosses represent the synaptic weight vectors of three output neurons. (a) Initial state of the network. (b) Final state of the network.

We may use the geometric analogy depicted in Fig. 2.5 to illustrate the essence of competitive learning (Rumelhart and Zipser, 1985). It is assumed that each input pattern (vector) \mathbf{x} has some constant Euclidean length so that we may view it as a point on an N -dimensional unit sphere where N is the number of input nodes. N also represents the dimension of each synaptic weight vector \mathbf{w}_k . It is further assumed that all neurons in the network are constrained to have the same Euclidean length (norm), as shown by

$$\sum_j w_{kj}^2 = 1 \quad \text{for all } k \quad (2.14)$$

When the synaptic weights are properly scaled they form a set of vectors that fall on the same N -dimensional unit sphere. In Fig. 2.5a we show three natural groupings (clusters) of the stimulus patterns represented by dots. This figure also includes a possible initial state of the network (represented by crosses) that may exist before learning. Figure 2.5b shows a typical final state of the network that results from the use of competitive learning. In particular, each output neuron has discovered a cluster of input patterns by moving its synaptic weight vector to the center of gravity of the discovered cluster (Rumelhart and Zipser, 1985; Hertz et al., 1991). This figure illustrates the ability of a neural network to perform *clustering* through competitive learning. However, for this function to be performed in a “stable” fashion the input patterns must fall into sufficiently distinct groupings to begin with. Otherwise the network may be unstable because it will no longer respond to a given input pattern with the same output neuron.

2.6 BOLTZMANN LEARNING

The Boltzmann learning rule, named in honor of Ludwig Boltzmann, is a stochastic learning algorithm derived from ideas rooted in statistical mechanics.⁷ A neural net-

work designed on the basis of the Boltzmann learning rule is called a *Boltzmann machine* (Ackley et al., 1985; Hinton and Sejnowski, 1986).

In a Boltzmann machine the neurons constitute a recurrent structure, and they operate in a binary manner since, for example, they are either in an “on” state denoted by +1 or in an “off” state denoted by −1. The machine is characterized by an *energy function*, E , the value of which is determined by the particular states occupied by the individual neurons of the machine, as shown by

$$E = -\frac{1}{2} \sum_j \sum_{k, j \neq k} w_{kj} x_k x_j \quad (2.15)$$

where x_j is the state of neuron j , and w_{kj} is the synaptic weight connecting neuron j to neuron k . The fact that $j \neq k$ means simply that none of the neurons in the machine has self-feedback. The machine operates by choosing a neuron at random—for example, neuron k —at some step of the learning process, then flipping the state of neuron k from state x_k to state $-x_k$ at some temperature T with probability

$$P(x_k \rightarrow -x_k) = \frac{1}{1 + \exp(-\Delta E_k/T)} \quad (2.16)$$

where ΔE_k is the *energy change* (i.e., the change in the energy function of the machine) resulting from such a flip. Notice that T is not a physical temperature, but rather a *pseudotemperature*, as explained in Chapter 1. If this rule is applied repeatedly, the machine will reach *thermal equilibrium*.

The neurons of a Boltzmann machine partition into two functional groups: *visible* and *hidden*. The visible neurons provide an interface between the network and the environment in which it operates, whereas the hidden neurons always operate freely. There are two modes of operation to be considered:

- *Clamped condition*, in which the visible neurons are all clamped onto specific states determined by the environment.
- *Free-running condition*, in which all the neurons (visible and hidden) are allowed to operate freely.

Let ρ_{kj}^+ denote the *correlation* between the states of neurons j and k , with the network in its clamped condition. Let ρ_{kj}^- denote the *correlation* between the states of neurons j and k with the network in its free-running condition. Both correlations are averaged over all possible states of the machine when it is in thermal equilibrium. Then, according to the *Boltzmann learning rule*, the change Δw_{kj} applied to the synaptic weight w_{kj} from neuron j to neuron k is defined by (Hinton and Sejnowski, 1986)

$$\Delta w_{kj} = \eta(\rho_{kj}^+ - \rho_{kj}^-), \quad j \neq k \quad (2.17)$$

where η is a learning-rate parameter. Note that both ρ_{kj}^+ and ρ_{kj}^- range in value from −1 to +1.

A brief review of statistical mechanics is presented in Chapter 11; in that chapter we also present a detailed treatment of the Boltzmann machine and other stochastic machines.

2.7 CREDIT-ASSIGNMENT PROBLEM

When studying learning algorithms for distributed systems, it is useful to consider the notion of *credit assignment* (Minsky, 1961). Basically, the credit-assignment problem is the problem of assigning *credit* or *blame* for overall outcomes to each of the internal decisions made by a learning machine and which contributed to those outcomes. (The credit assignment problem is also referred to as the *loading problem*, the problem of “loading” a given set of training data into the free parameters of the network.)

In many cases the dependence of outcomes on internal decisions is mediated by a sequence of actions taken by the learning machine. In other words, internal decisions affect which particular actions are taken, and then the actions, not the internal decisions, directly influence overall outcomes. In these situations, we may decompose the credit-assignment problem into two subproblems (Sutton, 1984):

1. The assignment of credit for outcomes to actions. This is called the *temporal credit-assignment problem* in that it involves the instants of time *when* the actions that deserve credit were actually taken.
2. The assignment of credit for actions to internal decisions. This is called the *structural credit-assignment problem* in that it involves assigning credit to the *internal structures* of actions generated by the system.

The structural credit-assignment problem is relevant in the context of a multicomponent learning machine when we must determine precisely which particular component of the system should have its behavior altered and by how much in order to improve overall system performance. On the other hand, the temporal credit-assignment problem is relevant when there are many actions taken by a learning machine that result in certain outcomes, and we must determine which of these actions were responsible for the outcomes. The combined temporal and structural credit-assignment problem faces any distributed learning machine that attempts to improve its performance in situations involving temporally extended behavior (Williams, 1988).

The credit-assignment problem, for example, arises when error-correction learning is applied to a multilayer feedforward neural network. The operation of each hidden neuron, as well as that of each output neuron in such a network is important to its correct overall operation on a learning task of interest. That is, in order to solve the prescribed task the network must assign certain forms of behavior to all of its neurons through the specification of error-correction learning. With this background in mind, consider the situation described in Fig. 2.1a. Since the output neuron k is visible to the outside world, it is possible to supply a desired response to this neuron. As far as the output neuron is concerned, it is a straightforward matter to adjust the synaptic weights of the output neuron in accordance with error-correction learning, as outlined in Section 2.2. But how do we assign credit or blame for the action of the hidden neurons when the error-correction learning process is used to adjust the respective synaptic weights of these neurons? The answer to this fundamental question requires more detailed attention; it is presented in Chapter 4, where algorithmic details of the design of multilayer feedforward neural networks are described.

2.8 LEARNING WITH A TEACHER

We now turn our attention to learning paradigms. We begin by considering *learning with a teacher*, which is also referred to as *supervised learning*. Figure 2.6 shows a block diagram that illustrates this form of learning. In conceptual terms, we may think of the teacher as having knowledge of the environment, with that knowledge being represented by a set of *input-output examples*. The environment is, however, *unknown* to the neural network of interest. Suppose now that the teacher and the neural network are both exposed to a training vector (i.e., example) drawn from the environment. By virtue of built-in knowledge, the teacher is able to provide the neural network with a desired response for that training vector. Indeed, the desired response represents the optimum action to be performed by the neural network. The network parameters are adjusted under the combined influence of the training vector and the error signal. The *error signal* is defined as the difference between the desired response and the actual response of the network. This adjustment is carried out iteratively in a step-by-step fashion with the aim of eventually making the neural network *emulate* the teacher; the emulation is presumed to be optimum in some statistical sense. In this way knowledge of the environment available to the teacher is transferred to the neural network through training as fully as possible. When this condition is reached, we may then dispense with the teacher and let the neural network deal with the environment completely by itself.

The form of supervised learning we have just described is the error-correction learning discussed previously in Section 2.2. It is a closed-loop feedback system, but the unknown environment is not in the loop. As a performance measure for the system we may think in terms of the mean-square error or the sum of squared errors over the training sample, defined as a function of the free parameters of the system. This function may be visualized as a multidimensional *error-performance surface* or simply *error surface*, with the free parameters as coordinates. The true error surface is *averaged* over all possible input-output examples. Any given operation of the system under the

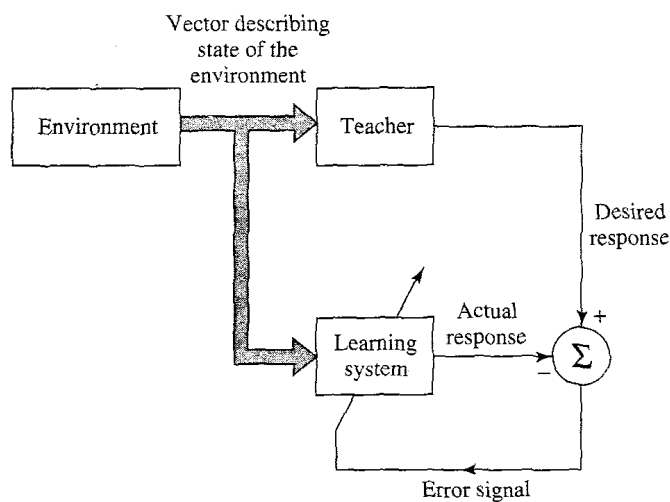


FIGURE 2.6 Block diagram of learning with a teacher.

teacher's supervision is represented as a point on the error surface. For the system to improve performance over time and therefore learn from the teacher, the operating point has to move down successively toward a minimum point of the error surface; the minimum point may be a local minimum or a global minimum. A supervised learning system is able to do this with the useful information it has about the *gradient* of the error surface corresponding to the current behavior of the system. The gradient of an error surface at any point is a vector that points in the direction of *steepest descent*. In fact, in the case of supervised learning from examples, the system may use an *instantaneous estimate* of the gradient vector, with the example indices presumed to be those of time. The use of such an estimate results in a motion of the operating point on the error surface that is typically in the form of a "random walk." Nevertheless, given an algorithm designed to minimize the cost function, an adequate set of input-output examples, and enough time permitted to do the training, a supervised learning system is usually able to perform such tasks as pattern classification and function approximation.

2.9 LEARNING WITHOUT A TEACHER

In supervised learning, the learning process takes place under the tutelage of a teacher. However, in the paradigm known as *learning without a teacher*, as the name implies, there is *no* teacher to oversee the learning process. That is to say, there are no labeled examples of the function to be learned by the network. Under this second paradigm, two subdivisions are identified:

1. Reinforcement learning/Neurodynamic programming

In *reinforcement learning*,⁸ the learning of an input-output mapping is performed through continued interaction with the environment in order to minimize a scalar index of performance. Figure 2.7 shows the block diagram of one form of a reinforcement learning system built around a *critic* that converts a *primary reinforcement signal* received from the environment into a higher quality reinforcement signal called the *heuristic reinforcement signal*, both of which are scalar inputs (Barto et al., 1983). The

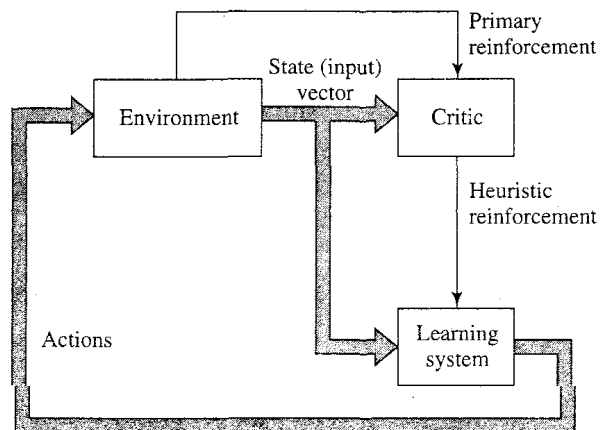


FIGURE 2.7 Block diagram of reinforcement learning.

system is designed to learn under *delayed reinforcement*, which means that the system observes a temporal sequence of stimuli (i.e., state vectors) also received from the environment, which eventually result in the generation of the heuristic reinforcement signal. The goal of learning is to minimize a *cost-to-go function*, defined as the expectation of the cumulative cost of *actions* taken over a sequence of steps instead of simply the immediate cost. It may turn out that certain actions taken earlier in that sequence of time steps are in fact the best determinants of overall system behavior. The function of the *learning machine*, which constitutes the second component of the system, is to *discover* these actions and to feed them back to the environment.

Delayed-reinforcement learning is difficult to perform for two basic reasons:

- There is no teacher to provide a desired response at each step of the learning process.
- The delay incurred in the generation of the primary reinforcement signal implies that the learning machine must solve a *temporal credit assignment problem*. By this we mean that the learning machine must be able to assign credit and blame individually to each action in the sequence of time steps that led to the final outcome, while the primary reinforcement may only evaluate the outcome.

Notwithstanding these difficulties, delayed-reinforcement learning is very appealing. It provides the basis for the system to interact with its environment, thereby developing the ability to learn to perform a prescribed task solely on the basis of the outcomes of its experience that result from the interaction.

Reinforcement learning is closely related to *dynamic programming*, which was developed by Bellman (1957) in the context of optimal control theory. Dynamic programming provides the mathematical formalism for sequential decision making. By casting reinforcement learning within the framework of dynamic programming, the subject matter becomes all the richer for it, as demonstrated in Bertsekas and Tsitsiklis (1996). An introductory treatment of dynamic programming and its relationship to reinforcement learning is presented in Chapter 12.

2. Unsupervised Learning

In *unsupervised* or *self-organized* learning there is no external teacher or critic to oversee the learning process, as indicated in Fig. 2.8. Rather, provision is made for a *task-independent measure* of the quality of representation that the network is required to learn, and the free parameters of the network are optimized with respect to that measure. Once the network has become tuned to the statistical regularities of the input data, it develops the ability to form internal representations for encoding features of the input and thereby to create new classes automatically (Becker, 1991).

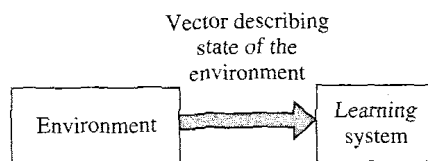


FIGURE 2.8 Block diagram of unsupervised learning.

To perform unsupervised learning we may use a competitive learning rule. For example, we may use a neural network that consists of two layers—an input layer and a competitive layer. The input layer receives the available data. The competitive layer consists of neurons that compete with each other (in accordance with a learning rule) for the “opportunity” to respond to features contained in the input data. In its simplest form, the network operates in accordance with a “winner-takes-all” strategy. As described in Section 2.5, in such a strategy the neuron with the greatest total input “wins” the competition and turns on; all the other neurons then switch off.

Different algorithms for unsupervised learning are described in Chapters 8 through 11.

2.10 LEARNING TASKS

In previous sections of this chapter we have discussed different learning algorithms and learning paradigms. In this section, we describe some basic learning tasks. The choice of a particular learning algorithm is influenced by the learning task that a neural network is required to perform. In this context we identify six learning tasks that apply to the use of neural networks in one form or another.

Pattern Association

An *associative memory* is a brainlike distributed memory that learns by *association*. Association has been known to be a prominent feature of human memory since Aristotle, and all models of cognition use association in one form or another as the basic operation (Anderson, 1995).

Association takes one of two forms: *autoassociation* or *heteroassociation*. In autoassociation, a neural network is required to *store* a set of patterns (vectors) by repeatedly presenting them to the network. The network is subsequently presented a partial description or distorted (noisy) version of an original pattern stored in it, and the task is to *retrieve (recall)* that particular pattern. Heteroassociation differs from autoassociation in that an arbitrary set of input patterns is *paired* with another arbitrary set of output patterns. Autoassociation involves the use of unsupervised learning, whereas the type of learning involved in heteroassociation is supervised.

Let \mathbf{x}_k denote a *key pattern* (vector) applied to an associative memory and \mathbf{y}_k denote a *memorized pattern* (vector). The pattern association performed by the network is described by

$$\mathbf{x}_k \rightarrow \mathbf{y}_k, \quad k = 1, 2, \dots, q \quad (2.18)$$

where q is the number of patterns stored in the network. The key pattern \mathbf{x}_k acts as a stimulus that not only determines the storage location of memorized pattern \mathbf{y}_k , but also holds the key for its retrieval.

In an autoassociative memory, $\mathbf{y}_k = \mathbf{x}_k$, so the input and output (data) spaces of the network have the same dimensionality. In a heteroassociative memory, $\mathbf{y}_k \neq \mathbf{x}_k$; hence, the dimensionality of the output space in this second case may or may not equal the dimensionality of the input space.

There are two phases involved in the operation of an associative memory:

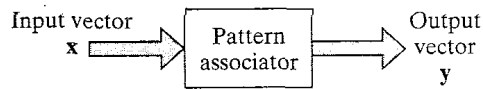


FIGURE 2.9 Input-output relation of pattern associator.

- *Storage phase*, which refers to the training of the network in accordance with Eq. (2.18).
- *Recall phase*, which involves the retrieval of a memorized pattern in response to the presentation of a noisy or distorted version of a key pattern to the network.

Let the stimulus (input) \mathbf{x} represent a noisy or distorted version of a key pattern \mathbf{x}_j . This stimulus produces a response (output) \mathbf{y} , as indicated in Fig. 2.9. For perfect recall, we should find that $\mathbf{y} = \mathbf{y}_j$, where \mathbf{y}_j is the memorized pattern associated with the key pattern \mathbf{x}_j . When $\mathbf{y} \neq \mathbf{y}_j$ for $\mathbf{x} = \mathbf{x}_j$, the associative memory is said to have made an *error* in recall.

The number q of patterns stored in an associative memory provides a direct measure of the *storage capacity* of the network. In designing an associative memory, the challenge is to make the storage capacity q (expressed as a percentage of the total number N of neurons used to construct the network) as large as possible and yet insist that a large fraction of the memorized patterns is recalled correctly.

Pattern Recognition

Humans are good at pattern recognition. We receive data from the world around us via our senses and are able to recognize the source of the data. We are often able to do so almost immediately and with practically no effort. For example, we can recognize the familiar face of a person even though that person has aged since our last encounter, identify a familiar person by his or her voice on the telephone despite a bad connection, and distinguish a boiled egg that is good from a bad one by smelling it. Humans perform pattern recognition through a learning process; so it is with neural networks.

Pattern recognition is formally defined as *the process whereby a received pattern/signal is assigned to one of a prescribed number of classes (categories)*. A neural network performs pattern recognition by first undergoing a training session, during which the network is repeatedly presented a set of input patterns along with the category to which each particular pattern belongs. Later, a new pattern is presented to the network that has not been seen before, but which belongs to the same population of patterns used to train the network. The network is able to identify the class of that particular pattern because of the information it has extracted from the training data. Pattern recognition performed by a neural network is statistical in nature, with the patterns being represented by points in a multidimensional *decision space*. The decision space is divided into regions, each one of which is associated with a class. The decision boundaries are determined by the training process. The construction of these boundaries is made statistical by the inherent variability that exists within and between classes.

In generic terms, pattern-recognition machines using neural networks may take one of two forms:

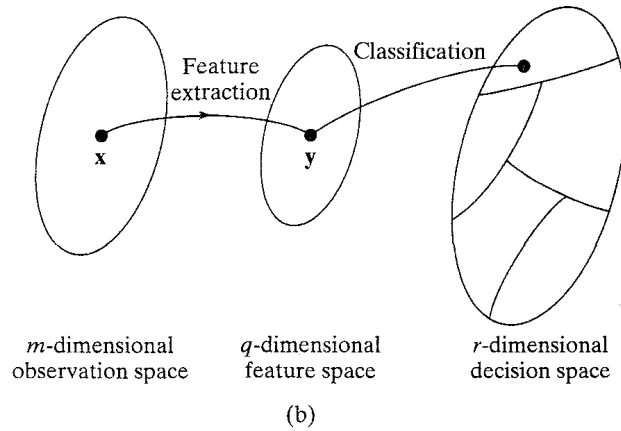
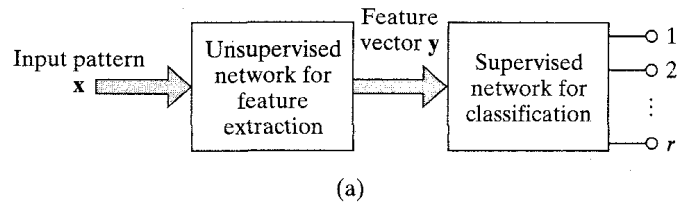


FIGURE 2.10 Illustration of the classical approach to pattern classification.

- The machine is split into two parts, an unsupervised network for *feature extraction* and a supervised network for *classification*, as shown in Fig. 2.10a. Such a method follows the traditional approach to statistical pattern recognition (Duda and Hart, 1973; Fukunaga, 1990). In conceptual terms, a pattern is represented by a set of m observables, which may be viewed as a point x in an m -dimensional *observation (data) space*. Feature extraction is described by a transformation that maps the point x into an intermediate point y in a q -dimensional *feature space* with $q < m$, as indicated in Fig. 2.10b. This transformation may be viewed as one of dimensionality reduction (i.e., data compression), the use of which is justified on the grounds that it simplifies the task of classification. The classification is itself described as a transformation that maps the intermediate point y into one of the classes in an r -dimensional decision space, where r is the number of classes to be distinguished.
- The machine is designed as a single multilayer feedforward network using a supervised learning algorithm. In this second approach, the task of feature extraction is performed by the computational units in the hidden layer(s) of the network.

Which of these two approaches is adopted in practice depends on the application of interest.

Function Approximation

The third learning task of interest is that of function approximation. Consider a nonlinear input-output mapping described by the functional relationship

$$d = f(x) \quad (2.19)$$

where the vector \mathbf{x} is the input and the vector \mathbf{d} is the output. The vector-valued function $\mathbf{f}(\cdot)$ is assumed to be unknown. To make up for the lack of knowledge about the function $\mathbf{f}(\cdot)$, we are given the set of labeled examples:

$$\mathcal{T} = \{(\mathbf{x}_i, \mathbf{d}_i)\}_{i=1}^N \quad (2.20)$$

The requirement is to design a neural network that approximates the unknown function $\mathbf{f}(\cdot)$ such that the function $\mathbf{F}(\cdot)$ describing the input–output mapping actually realized by the network is close enough to $\mathbf{f}(\cdot)$ in a Euclidean sense over all inputs, as shown by

$$\|\mathbf{F}(\mathbf{x}) - \mathbf{f}(\mathbf{x})\| < \epsilon \quad \text{for all } \mathbf{x} \quad (2.21)$$

where ϵ is a small positive number. Provided that the size N of the training set is large enough and the network is equipped with an adequate number of free parameters, then the approximation error ϵ can be made small enough for the task.

The approximation problem described here is a perfect candidate for supervised learning with \mathbf{x}_i playing the role of input vector and \mathbf{d}_i serving the role of desired response. We may turn this issue around and view supervised learning as an approximation problem.

The ability of a neural network to approximate an unknown input–output mapping may be exploited in two important ways:

- *System identification.* Let Eq. (2.19) describe the input–output relation of an unknown memoryless *multiple input-multiple output (MIMO)* system; by a “memoryless” system we mean a system that is time invariant. We may then use the set of labeled examples in Eq. (2.20) to train a neural network as a model of the system. Let \mathbf{y}_i denote the output of the neural network produced in response to an input vector \mathbf{x}_i . The difference between \mathbf{d}_i (associated with \mathbf{x}_i) and the network output \mathbf{y}_i provides the error signal vector \mathbf{e}_i , as depicted in Fig. 2.11. This error signal is in turn used to adjust the free parameters of the network to minimize the squared difference between the outputs of the unknown system and the neural network in a statistical sense, and is computed over the entire training set.
- *Inverse system.* Suppose next we are given a known memoryless MIMO system whose input–output relation is described by Eq. (2.19). The requirement in this

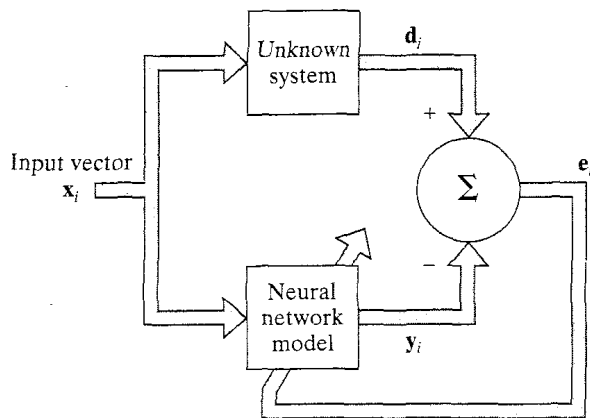


FIGURE 2.11 Block diagram of system identification.

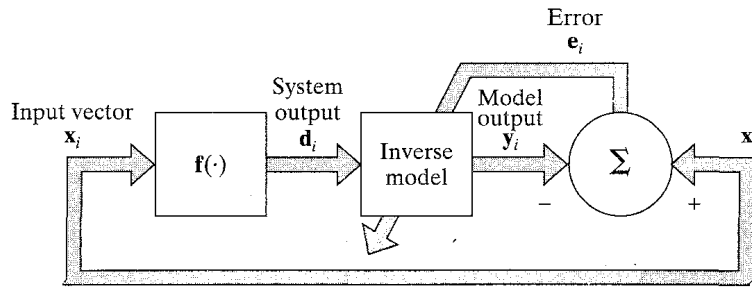


FIGURE 2.12 Block diagram of inverse system modeling.

case is to construct an *inverse system* that produces the vector \mathbf{x} in response to the vector \mathbf{d} . The inverse system may thus be described by

$$\mathbf{x} = \mathbf{f}^{-1}(\mathbf{d}) \quad (2.22)$$

where the vector-valued function $\mathbf{f}^{-1}(\cdot)$ denotes the inverse of $\mathbf{f}(\cdot)$. Note, however, that $\mathbf{f}^{-1}(\cdot)$ is not the reciprocal of $\mathbf{f}(\cdot)$; rather, the use of superscript -1 is merely a flag to indicate an inverse. In many situations encountered in practice, the vector-valued function $\mathbf{f}(\cdot)$ is much too complex, and inhibits a straightforward formulation of the inverse function $\mathbf{f}^{-1}(\cdot)$. Given the set of labeled examples in Eq. (2.20), we may construct a neural network approximation of $\mathbf{f}^{-1}(\cdot)$ by using the scheme shown in Fig. 2.12. In the situation described here, the roles of \mathbf{x}_i and \mathbf{d}_i are interchanged: the vector \mathbf{d}_i is used as the input and \mathbf{x}_i is treated as the desired response. Let the error signal vector \mathbf{e}_i denote the difference between \mathbf{x}_i and the actual output \mathbf{y}_i of the neural network produced in response to \mathbf{d}_i . As with the system identification problem, this error signal vector is used to adjust the free parameters of the neural network to minimize the squared difference between the outputs of the unknown inverse system and the neural network in a statistical sense, and is computed over the complete training set.

Control

The control of a *plant* is another learning task that can be done by a neural network; by a “plant” we mean a process or critical part of a system that is to be maintained in a controlled condition. The relevance of learning to control should not be surprising because, after all, the human brain is a computer (i.e., information processor), the outputs of which as a whole system are *actions*. In the context of control, the brain is living proof that it is possible to build a generalized controller that takes full advantage of parallel distributed hardware, can control many thousands of actuators (muscle fibers) in parallel, can handle nonlinearity and noise, and can optimize over a long-range planning horizon (Werbos, 1992).

Consider the *feedback control system* shown in Fig. 2.13. The system involves the use of unity feedback around a plant to be controlled; that is, the plant output is fed back directly to the input.⁹ Thus, the plant output \mathbf{y} is subtracted from a *reference signal* \mathbf{d} supplied from an external source. The error signal \mathbf{e} so produced is applied to a neural

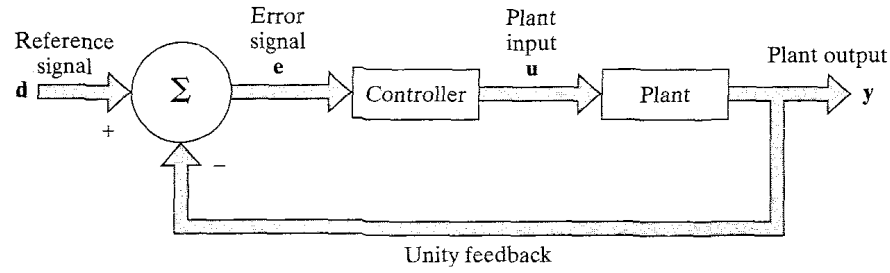


FIGURE 2.13 Block diagram of feedback control system.

controller for the purpose of adjusting its free parameters. The primary objective of the controller is to supply appropriate inputs to the plant to make its output \mathbf{y} track the reference signal \mathbf{d} . In other words, the controller has to invert the plant's input–output behavior.

We note that in Fig. 2.13 the error signal \mathbf{e} has to propagate through the neural controller before reaching the plant. Consequently, to perform adjustments on the free parameters of the plant in accordance with an error-correction learning algorithm we need to know the Jacobian matrix

$$\mathbf{J} = \left\{ \frac{\partial y_k}{\partial u_j} \right\} \quad (2.23)$$

where y_k is an element of the plant output \mathbf{y} and u_j is an element of the plant input \mathbf{u} . Unfortunately, the partial derivatives $\partial y_k / \partial u_j$ for various k and j depend on the operating point of the plant and are therefore not known. We may use one of two approaches to account for them:

- *Indirect learning.* Using actual input–output measurements on the plant, a neural network model is first constructed to produce a copy of it. This model is in turn used to provide an estimate of the Jacobian matrix \mathbf{J} . The partial derivatives constituting this Jacobian matrix are subsequently used in the error-correction learning algorithm for computing the adjustments to the free parameters of the neural controller (Nguyen and Widrow, 1989; Suykens et al., 1996; Widrow and Walach, 1996).
- *Direct learning.* The signs of the partial derivatives $\partial y_k / \partial u_j$ are generally known and usually remain constant over the dynamic range of the plant. This suggests that we may approximate these partial derivatives by their individual signs. Their absolute values are given a distributed representation in the free parameters of the neural controller (Saerens and Soquet, 1991; Schiffman and Geffers, 1993). The neural controller is thereby enabled to learn the adjustments to its free parameters directly from the plant.

Filtering

The term *filter* often refers to a device or algorithm used to extract information about a prescribed quantity of interest from a set of noisy data. The noise may arise from a variety of sources. For example, the data may have been measured by means of noisy

sensors or may represent an information-bearing signal that has been corrupted by transmission through a communication channel. Another example is that of a useful signal component corrupted by an interfering signal picked up from the surrounding environment. We may use a filter to perform three basic information processing tasks:

1. *Filtering*. This task refers to the extraction of information about a quantity of interest at discrete time n by using data measured up to and including time n .
2. *Smoothing*. This second task differs from filtering in that information about the quantity of interest need not be available at time n , and data measured later than time n can be used in obtaining this information. This means that in smoothing there is a *delay* in producing the result of interest. Since, in the smoothing process, we are able to use data obtained not only up to time n but also after time n , we expect smoothing to be more accurate than filtering in some statistical sense.
3. *Prediction*. This task is the forecasting side of information processing. The aim here is to derive information about what the quantity of interest will be like at some time $n + n_0$ in the future, for some $n_0 > 0$, by using data measured up to and including time n .

A filtering problem humans are familiar with is the *cocktail party problem*.¹⁰ We have a remarkable ability to focus on a speaker in the noisy environment of a cocktail party, despite the fact that the speech signal originating from that speaker is buried in an undifferentiated noise background due to other interfering conversations in the room. It is thought that some form of preattentive, preconscious analysis must be involved in resolving the cocktail party problem (Velmans, 1995). In the context of (artificial) neural networks, a similar filtering problem arises under the umbrella of *blind signal separation* (Comon, 1994; Bell and Sejnowski, 1995; Amari et al., 1996). To formulate the blind signal separation problem, consider a set of unknown source signals $\{s_i(n)\}_{i=1}^m$ that are mutually independent of each other. These signals are linearly mixed by an unknown sensor to produce the m -by-1 observation vector (see Fig. 2.14)

$$\mathbf{x}(n) = \mathbf{A}\mathbf{u}(n) \quad (2.24)$$

where

$$\mathbf{u}(n) = [u_1(n), u_2(n), \dots, u_m(n)]^T \quad (2.25)$$

$$\mathbf{x}(n) = [x_1(n), x_2(n), \dots, x_m(n)]^T \quad (2.26)$$

and \mathbf{A} is an unknown nonsingular *mixing matrix* of dimensions m -by- m . Given the observation vector $\mathbf{x}(n)$, the requirement is to recover the original signals $u_1(n)$, $u_2(n)$, ..., $u_m(n)$ in an unsupervised manner.

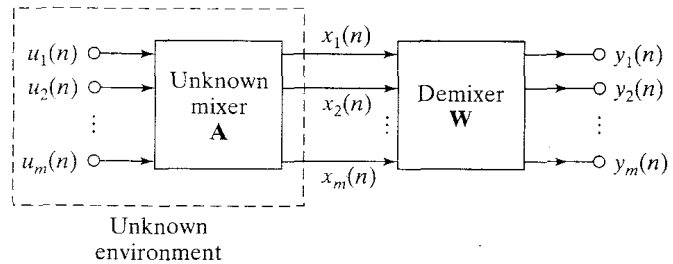


FIGURE 2.14 Block diagram of blind source separation.

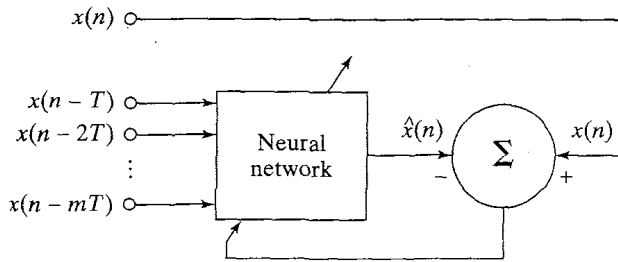


FIGURE 2.15 Block diagram of nonlinear prediction.

Turning now to the prediction problem, the requirement is to predict the present value $x(n)$ of a process, given past values of the process that are uniformly spaced in time as shown by $x(n-T)$, $x(n-2T)$, ..., $x(n-mT)$, where T is the sampling period and m is the prediction order. Prediction may be solved by using error-correction learning in an unsupervised manner since the training examples are drawn directly from the process itself, as depicted in Fig. 2.15, where $x(n)$ serves the purpose of desired response. Let $\hat{x}(n)$ denote the one-step prediction produced by the neural network at time n . The error signal $e(n)$ is defined as the difference between $x(n)$ and $\hat{x}(n)$, which is used to adjust the free parameters of the neural network. On this basis, prediction may be viewed as a form of *model building* in the sense that the smaller we make the prediction error in a statistical sense, the better the network serves as a model of the underlying physical process responsible for generating the data. When this process is *nonlinear*, the use of a neural network provides a powerful method for solving the prediction problem because of the nonlinear processing units that could be built into its construction. The only possible exception to the use of nonlinear processing units, however, is the output unit of the network: If the dynamic range of the time series $\{x(n)\}$ is unknown, the use of a linear output unit is the most reasonable choice.

Beamforming

Beamforming is a *spatial* form of filtering and is used to distinguish between the spatial properties of a target signal and background noise. The device used to do the beamforming is called a *beamformer*.

The task of beamforming is compatible with the use of a neural network, for which we have relevant cues from psychoacoustic studies of human auditory responses (Bregman, 1990) and studies of feature mapping in the cortical layers of auditory systems of echo-locating bats (Suga, 1990a; Simmons and Sillant, 1992). The echo-locating bat illuminates the surrounding environment by broadcasting short-duration frequency-modulated (FM) sonar signals, and then uses its auditory system (including a pair of ears) to focus attention on its prey (e.g., flying insect). The ears provide the bat with some form of spatial filtering (interferometry to be precise), which is then exploited by the auditory system to produce *attentional selectivity*.

Beamforming is commonly used in radar and sonar systems where the primary task is to detect and track a target of interest in the combined presence of receiver noise and interfering signals (e.g., jammers). This task is complicated by two factors.

- The target signal originates from an unknown direction.
- There is no *a priori* information available on the interfering signals.

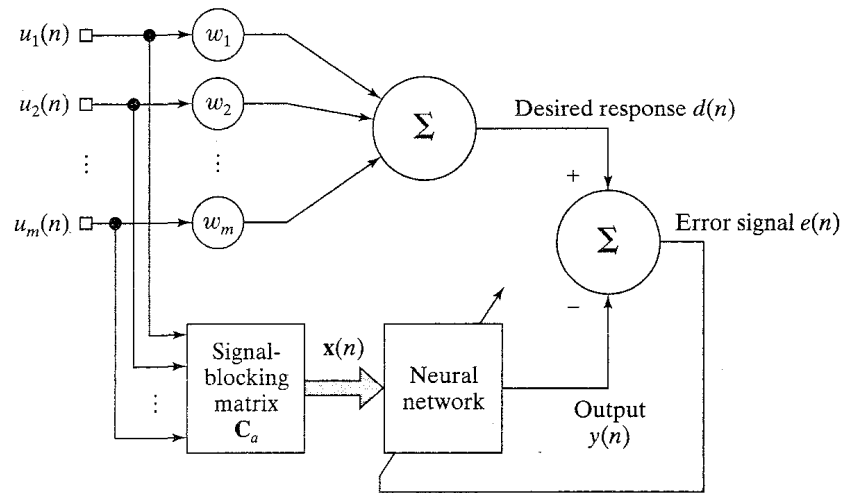


FIGURE 2.16 Block diagram of generalized sidelobe canceller.

One way of coping with situations of this kind is to use a *generalized sidelobe canceller* (GSLC), the block diagram of which is shown in Fig. 2.16. The system consists of the following components (Griffiths and Jim, 1982; Van Veen, 1992; Haykin, 1996):

- An *array of antenna elements*, which provides a means of sampling the observed signal at discrete points in space.
- A *linear combiner* defined by a set of fixed weights $\{w_i\}_{i=1}^m$, the output of which is a desired response. This linear combiner acts like a “spatial filter,” characterized by a radiation pattern (i.e., a polar plot of the amplitude of the antenna output versus the incidence angle of an incoming signal). The mainlobe of this radiation pattern is pointed along a prescribed direction, for which the GSLC is *constrained* to produce a distortionless response. The output of the linear combiner, denoted by $d(n)$, provides a desired response for the beamformer.
- A *signal-blocking matrix* C_a , the function of which is to cancel interference that leaks through the sidelobes of the radiation pattern of the spatial filter representing the linear combiner.
- A *neural network* with adjustable parameters, which is designed to accommodate statistical variations in the interfering signals.

The adjustments to the free parameters of the neural network are performed by an error-correcting learning algorithm that operates on the error signal $e(n)$, defined as the difference between the linear combiner output $d(n)$ and the actual output $y(n)$ of the neural network. Thus the GSLC operates under the supervision of the linear combiner that assumes the role of a “teacher.” As with ordinary supervised learning, notice that the linear combiner is outside the feedback loop acting on the neural network. A beamformer that uses a neural network for learning is called a *neural beamformer* or *neuro-beamformer*. This class of learning machines comes under the general heading of *attentional neurocomputers* (Hecht-Nielsen, 1990).

The diversity of the six learning tasks discussed here is testimony to the *universality* of neural networks as information-processing systems. In a fundamental sense,

these learning tasks are all problems of learning a *mapping* from (possibly noisy) examples of the mapping. Without the imposition of prior knowledge, each of the tasks is in fact *ill posed* in the sense of nonuniqueness of possible solution mappings. One method of making the solution well posed is to use the theory of regularization as described in Chapter 5.

2.11 MEMORY

Discussion of learning tasks, particularly the task of pattern association, leads us naturally to think about *memory*. In a neurobiological context, memory refers to the relatively enduring neural alterations induced by the interaction of an organism with its environment (Teyler, 1986). Without such a change there can be no memory. Furthermore, for the memory to be useful it must be accessible to the nervous system in order to influence future behavior. However, an activity pattern must initially be stored in memory through a *learning process*. Memory and learning are intricately connected. When a particular activity pattern is learned, it is stored in the brain where it can be recalled later when required. Memory may be divided into “short-term” and “long-term” memory, depending on the retention time (Arbib, 1989). *Short-term memory* refers to a compilation of knowledge representing the “current” state of the environment. Any discrepancies between knowledge stored in short-term memory and a “new” state are used to update the short-term memory. *Long-term memory*, on the other hand, refers to knowledge stored for a long time or permanently.

In this section we study an associative memory that offers the following characteristics:

- The memory is distributed.
- Both the stimulus (key) pattern and the response (stored) pattern of an associative memory consist of data vectors.
- Information is stored in memory by setting up a spatial pattern of neural activities across a large number of neurons.
- Information contained in a stimulus not only determines its storage location in memory but also an address for its retrieval.
- Although neurons do not represent reliable and low-noise computing cells, the memory exhibits a high degree of resistance to noise and damage of a diffusive kind.
- There may be interactions between individual patterns stored in memory. (Otherwise the memory would have to be exceptionally large for it to accommodate the storage of a large number of patterns in perfect isolation from each other.) There is therefore the distinct possibility for the memory to make *errors* during the recall process.

In a *distributed memory*, the basic issue of interest is the simultaneous or near-simultaneous activities of many different neurons, which are the result of external or internal stimuli. The neural activities form a spatial pattern inside the memory that contains information about the stimuli. The memory is therefore said to perform a distributed mapping that transforms an activity pattern in the input space into another activity pattern in the output space. We may illustrate some important properties of a

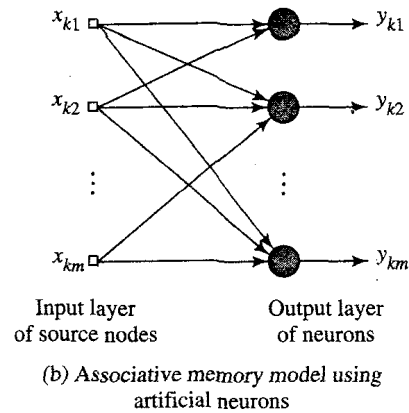
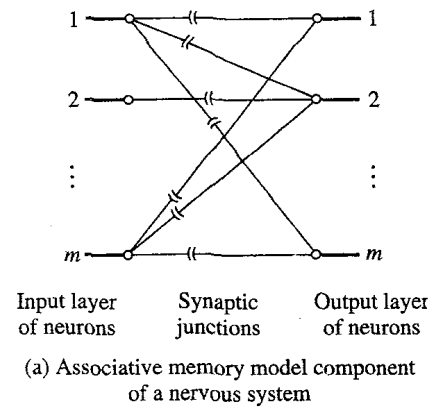


FIGURE 2.17 Associative memory models.

distributed memory mapping by considering an idealized neural network that consists of two layers of neurons. Figure 2.17a illustrates a network that may be regarded as a *model component of a nervous system* (Cooper, 1973; Scofield and Cooper, 1985). Each neuron in the input layer is connected to every one of the neurons in the output layer. The actual synaptic connections between the neurons are complex and redundant. In the model of Fig. 2.17a, a single ideal junction is used to represent the integrated effect of all the synaptic contacts between the dendrites of a neuron in the input layer and the axon branches of a neuron in the output layer. The level of activity of a neuron in the input layer may affect the level of activity of every other neuron in the output layer.

The corresponding situation for an artificial neural network is depicted in Fig. 2.17b. Here we have an input layer of source nodes and an output layer of neurons acting as computation nodes. In this case, the synaptic weights of the network are included as integral parts of the neurons in the output layer. The connecting links between the two layers of the network are simply wires.

In the following mathematical analysis, the neural networks in Figs. 2.17a and 2.17b are both assumed to be *linear*. The implication of this assumption is that each neuron acts as a linear combiner, as depicted in the signal-flow graph of Fig. 2.18. To proceed with the analysis, suppose that an activity pattern \mathbf{x}_k occurs in the input layer of the network and that an activity pattern \mathbf{y}_k occurs simultaneously in the output layer. The issue we wish to consider here is that of learning from the association

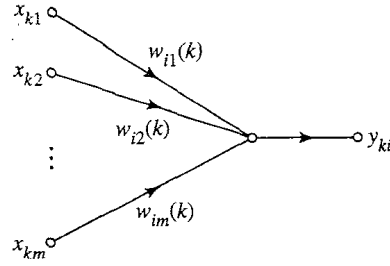


FIGURE 2.18 Signal-flow graph model of a linear neuron labeled i .

between the patterns \mathbf{x}_k and \mathbf{y}_k . The patterns \mathbf{x}_k and \mathbf{y}_k are represented by vectors, written in their expanded forms as:

$$\mathbf{x}_k = [x_{k1}, x_{k2}, \dots, x_{km}]^T$$

and

$$\mathbf{y}_k = [y_{k1}, y_{k2}, \dots, y_{km}]^T$$

For convenience of presentation we have assumed that the input space dimensionality (i.e., the dimension of vector \mathbf{x}_k) and the output space dimensionality (i.e., the dimension of vector \mathbf{y}_k) are the same, equal to m . From here on we refer to m as *network dimensionality* or simply *dimensionality*. Note that m equals the number of source nodes in the input layer or neurons in the output layer. For a neural network with a large number of neurons, which is typically the case, the dimensionality m can be large.

The elements of both \mathbf{x}_k and \mathbf{y}_k can assume positive and negative values. This is a valid proposition in an artificial neural network. It may also occur in a nervous system by considering the relevant physiological variable to be the difference between an actual activity level (e.g., firing rate of a neuron) and a nonzero spontaneous activity level.

With the networks of Fig. 2.17 assumed to be linear, the association of key vector \mathbf{x}_k with memorized vector \mathbf{y}_k may be described in matrix form as:

$$\mathbf{y}_k = \mathbf{W}(k)\mathbf{x}_k, \quad k = 1, 2, \dots, q \quad (2.27)$$

where $\mathbf{W}(k)$ is a weight matrix determined solely by the input–output pair $(\mathbf{x}_k, \mathbf{y}_k)$.

To develop a detailed description of the weight matrix $\mathbf{W}(k)$, consider Fig. 2.18 that shows a detailed arrangement of neuron i in the output layer. The output y_{ki} of neuron i due to the combined action of the elements of the key pattern \mathbf{x}_k applied as stimulus to the input layer is given by

$$y_{ki} = \sum_{j=1}^m w_{ij}(k)x_{kj}, \quad i = 1, 2, \dots, m \quad (2.28)$$

where the $w_{ij}(k)$, $j = 1, 2, \dots, m$, are the synaptic weights of neuron i corresponding to the k th pair of associated patterns. Using matrix notation, we may express y_{ki} in the equivalent form

$$y_{ki} = [w_{i1}(k), w_{i2}(k), \dots, w_{im}(k)] \begin{bmatrix} x_{k1} \\ x_{k2} \\ \vdots \\ x_{km} \end{bmatrix}, \quad i = 1, 2, \dots, m \quad (2.29)$$

The column vector on the right-hand side of Eq. (2.29) is recognized as the key vector \mathbf{x}_k . By substituting Eq. (2.29) in the definition of the m -by-1 stored vector \mathbf{y}_k , we get

$$\begin{bmatrix} y_{k1} \\ y_{k2} \\ \vdots \\ y_{km} \end{bmatrix} = \begin{bmatrix} w_{11}(k) & w_{12}(k) & \dots & w_{1m}(k) \\ w_{21}(k) & w_{22}(k) & \dots & w_{2m}(k) \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1}(k) & w_{m2}(k) & \dots & w_{mm}(k) \end{bmatrix} \begin{bmatrix} x_{k1} \\ x_{k2} \\ \vdots \\ x_{km} \end{bmatrix} \quad (2.30)$$

Equation (2.30) is the expanded form of the matrix transformation or mapping described in Eq. (2.27). In particular, the m -by- m weight matrix $\mathbf{W}(k)$ is defined by

$$\mathbf{W}(k) = \begin{bmatrix} w_{11}(k) & w_{12}(k) & \dots & w_{1m}(k) \\ w_{21}(k) & w_{22}(k) & \dots & w_{2m}(k) \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1}(k) & w_{m2}(k) & \dots & w_{mm}(k) \end{bmatrix} \quad (2.31)$$

The individual presentations of the q pairs of associated patterns $\mathbf{x}_k \rightarrow \mathbf{y}_k$, $k = 1, 2, \dots, q$, produce corresponding values of the individual matrix, namely, $\mathbf{W}(1), \mathbf{W}(2), \dots, \mathbf{W}(q)$. Recognizing that this pattern association is represented by the weight matrix $\mathbf{W}(k)$, we may define an m -by- m *memory matrix* that describes the summation of the weight matrices for the entire set of pattern associations as follows:

$$\mathbf{M} = \sum_{k=1}^q \mathbf{W}(k) \quad (2.32)$$

The memory matrix \mathbf{M} defines the overall connectivity between the input and output layers of the associative memory. In effect, it represents the *total experience* gained by the memory as a result of the presentations of q input-output patterns. Stated in another way, the memory matrix \mathbf{M} contains a piece of every input-output pair of activity patterns presented to the memory.

The definition of the memory matrix given in Eq. (2.32) may be restructured in the form of a recursion as shown by

$$\mathbf{M}_k = \mathbf{M}_{k-1} + \mathbf{W}(k), \quad k = 1, 2, \dots, q \quad (2.33)$$

where the initial value \mathbf{M}_0 is zero (i.e., the synaptic weights in the memory are all initially zero), and the final value \mathbf{M}_q is identically equal to \mathbf{M} as defined in Eq. (2.32). According to the recursive formula of Eq. (2.33), the term \mathbf{M}_{k-1} is the old value of the memory matrix resulting from $(k-1)$ pattern associations, and \mathbf{M}_k is the updated value in light of the increment $\mathbf{W}(k)$ produced by the k th association. Note, however, that when $\mathbf{W}(k)$ is added to \mathbf{M}_{k-1} , the increment $\mathbf{W}(k)$ loses its distinct identity among the mixture of contributions that form \mathbf{M}_k . In spite of the synaptic mixing of different associations, information about the stimuli may not have been lost, as demonstrated in the sequel. Notice also that as the number q of stored patterns increases, the influence of a new pattern on the memory as a whole is progressively reduced.

Correlation Matrix Memory

Suppose that the associative memory of Fig. 2.17b has learned the memory matrix \mathbf{M} through the associations of key and memorized patterns described by $\mathbf{x}_k \rightarrow \mathbf{y}_k$, where $k = 1, 2, \dots, q$. We may postulate $\hat{\mathbf{M}}$, denoting an *estimate* of the memory matrix \mathbf{M} in terms of these patterns as (Anderson, 1972, 1983; Cooper, 1973):

$$\hat{\mathbf{M}} = \sum_{k=1}^q \mathbf{y}_k \mathbf{x}_k^T \quad (2.34)$$

The term $\mathbf{y}_k \mathbf{x}_k^T$ represents the *outer product* of the key pattern \mathbf{x}_k and the memorized pattern \mathbf{y}_k . This outer product is an “estimate” of the weight matrix $\mathbf{W}(k)$ that maps the output pattern \mathbf{y}_k onto the input pattern \mathbf{x}_k . Since the pattern \mathbf{x}_k and \mathbf{y}_k are both m -by-1 vectors by assumption, it follows that their output product $\mathbf{y}_k \mathbf{x}_k^T$, and therefore the estimate $\hat{\mathbf{M}}$, is an m -by- m matrix. This dimensionality is in perfect agreement with that of the memory matrix \mathbf{M} defined in Eq. (2.32). The format of the summation of the estimate $\hat{\mathbf{M}}$ bears a direct relation to that of the memory matrix defined in that equation.

A typical term of the outer product $\mathbf{y}_k \mathbf{x}_k^T$ is written as $y_{ki} x_{kj}$, where x_{kj} is the output of source node j in the input layer, and y_{ki} is the output of neuron i in the output layer. In the context of synaptic weight $w_{ij}(k)$ for the k th association, source node j acts as a presynaptic node and neuron i in the output layer acts as a postsynaptic node. Hence, the “local” learning process described in Eq. (2.34) may be viewed as a *generalization of Hebb’s postulate of learning*. It is also referred to as the *outer product rule* in recognition of the matrix operation used to construct the memory matrix $\hat{\mathbf{M}}$. Correspondingly, an associative memory so designed is called a *correlation matrix memory*. Correlation, in one form or another, is indeed the basis of learning, association, pattern recognition, and memory recall in the human nervous system (Eggermont, 1990.)

Equation (2.34) may be reformulated in the equivalent form

$$\begin{aligned} \hat{\mathbf{M}} &= [\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_q] \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_q^T \end{bmatrix} \\ &= \mathbf{Y} \mathbf{X}^T \end{aligned} \quad (2.35)$$

where

$$\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_q] \quad (2.36)$$

and

$$\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_q] \quad (2.37)$$

The matrix \mathbf{X} is an m -by- q matrix composed of the entire set of key patterns used in the learning process; it is called the *key matrix*. The matrix \mathbf{Y} is an m -by- q matrix composed of the corresponding set of memorized patterns; it is called the *memorized matrix*.

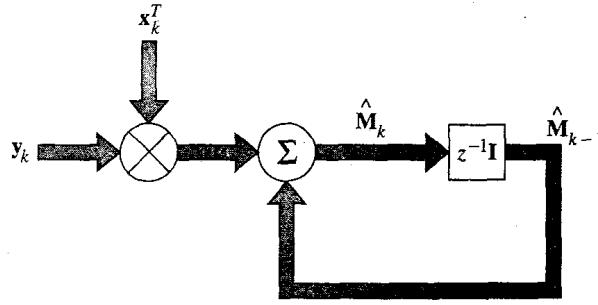


FIGURE 2.19 Signal-flow graph representation of Eq. (2.38).

Equation (2.35) may also be restructured in the form of a recursion as follows:

$$\hat{\mathbf{M}}_k = \hat{\mathbf{M}}_{k-1} + \mathbf{y}_k \mathbf{x}_k^T, \quad k = 1, 2, \dots, q \quad (2.38)$$

A signal-flow graph representation of this recursion is depicted in Fig. 2.19. According to this signal-flow graph and the recursive formula of Eq. (2.38), the matrix $\hat{\mathbf{M}}_{k-1}$ represents an old estimate of the memory matrix; and $\hat{\mathbf{M}}_k$ represents its updated value in the light of a new association performed by the memory on the patterns \mathbf{x}_k and \mathbf{y}_k . Comparing the recursion of Eq. (2.38) with that of Eq. (2.33), we see that the outer product $\mathbf{y}_k \mathbf{x}_k^T$ represents an estimate of the weight matrix $\mathbf{W}(k)$ corresponding to the k th association of key and memorized patterns, \mathbf{x}_k and \mathbf{y}_k .

Recall

The fundamental problem posed by the use of an associative memory is the address and recall of patterns stored in memory. To explain one aspect of this problem, let $\hat{\mathbf{M}}$ denote the memory matrix of an associative memory, which has been completely learned through its exposure to q pattern associations in accordance with Eq. (2.34). Let a key pattern \mathbf{x}_j be picked at random and reapplied as *stimulus* to the memory, yielding the *response*

$$\mathbf{y} = \hat{\mathbf{M}} \mathbf{x}_j \quad (2.39)$$

Substituting Eq. (2.34) in (2.39), we get

$$\begin{aligned} \mathbf{y} &= \sum_{k=1}^m \mathbf{y}_k \mathbf{x}_k^T \mathbf{x}_j \\ &= \sum_{k=1}^m (\mathbf{x}_k^T \mathbf{x}_j) \mathbf{y}_k \end{aligned} \quad (2.40)$$

where, in the second line, it is recognized that $\mathbf{x}_k^T \mathbf{x}_j$ is a scalar equal to the *inner product* of the key vectors \mathbf{x}_k and \mathbf{x}_j . We may rewrite Eq. (2.40) as

$$\mathbf{y} = (\mathbf{x}_j^T \mathbf{x}_j) \mathbf{y}_j + \sum_{\substack{k=1 \\ k \neq j}}^m (\mathbf{x}_k^T \mathbf{x}_j) \mathbf{y}_k \quad (2.41)$$

Let each of the key patterns $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_q$ be normalized to have unit energy; that is,

$$\begin{aligned} E_k &= \sum_{l=1}^m x_{kl}^2 \\ &= \mathbf{x}_k^T \mathbf{x}_k \\ &= 1, \quad k = 1, 2, \dots, q \end{aligned} \quad (2.42)$$

Accordingly, we may simplify the response of the memory to the stimulus (key pattern) \mathbf{x}_j as

$$\mathbf{y} = \mathbf{y}_j + \mathbf{v}_j \quad (2.43)$$

where

$$\mathbf{v}_j = \sum_{\substack{k=1 \\ k \neq j}}^m (\mathbf{x}_k^T \mathbf{x}_j) \mathbf{y}_k \quad (2.44)$$

The first term on the right-hand side of Eq. (2.43) represents the “desired” response \mathbf{y}_j ; it may therefore be viewed as the “signal” component of the actual response \mathbf{y} . The second term \mathbf{v}_j is a “noise vector” that arises because of the *crossstalk* between the key vector \mathbf{x}_j and all the other key vectors stored in memory. The noise vector \mathbf{v}_j is responsible for making errors on recall.

In the context of a linear signal space, we may define the *cosine of the angle* between a pair of vectors \mathbf{x}_j and \mathbf{x}_k as the inner product of \mathbf{x}_j and \mathbf{x}_k divided by the product of their individual Euclidean *norms* or *lengths* as shown by

$$\cos(\mathbf{x}_k, \mathbf{x}_j) = \frac{\mathbf{x}_k^T \mathbf{x}_j}{\|\mathbf{x}_k\| \|\mathbf{x}_j\|} \quad (2.45)$$

The symbol $\|\mathbf{x}_k\|$ signifies the Euclidean norm of vector \mathbf{x}_k , defined as the square root of the energy of \mathbf{x}_k :

$$\begin{aligned} \|\mathbf{x}_k\| &= (\mathbf{x}_k^T \mathbf{x}_k)^{1/2} \\ &= E_k^{1/2} \end{aligned} \quad (2.46)$$

Returning to the situation, note that the key vectors are normalized to have unit energy in accordance with Eq. (2.42). We may therefore reduce the definition of Eq. (2.45) to

$$\cos(\mathbf{x}_k, \mathbf{x}_j) = \mathbf{x}_k^T \mathbf{x}_j \quad (2.47)$$

We may then redefine the noise vector of Eq. (2.44) as

$$\mathbf{v}_j = \sum_{\substack{k=1 \\ k \neq j}}^m \cos(\mathbf{x}_k, \mathbf{x}_j) \mathbf{y}_k \quad (2.48)$$

We now see that if the key vectors are *orthogonal* (i.e., perpendicular to each other in a Euclidean sense), then

$$\cos(\mathbf{x}_k, \mathbf{x}_j) = 0, \quad k \neq j \quad (2.49)$$

and therefore the noise vector \mathbf{v}_j is identically zero. In such a case, the response \mathbf{y} equals \mathbf{y}_j . The *memory associates perfectly* if the key vectors form an *orthonormal set*; that is, if they satisfy the following pair of conditions:

$$\mathbf{x}_k^T \mathbf{x}_j = \begin{cases} 1, & k = j \\ 0, & k \neq j \end{cases} \quad (2.50)$$

Suppose now that the key vectors do form an orthonormal set, as prescribed in Eq. (2.50). What is then the limit on the *storage capacity* of the associative memory? Stated in another way, what is the largest number of patterns that can be reliably stored? The answer to this fundamental question lies in the rank of the memory matrix $\hat{\mathbf{M}}$. The *rank* of a matrix is defined as the number of independent columns (rows) of the matrix. That is, if r is the rank of such a rectangular matrix of dimensions l -by- m , we then have $r \leq \min(l, m)$. In the case of a correlation memory, the memory matrix $\hat{\mathbf{M}}$ is an m -by- m matrix, where m is the dimensionality of the input space. Hence the rank of the memory matrix $\hat{\mathbf{M}}$ is limited by the dimensionality m . We may thus formally state that the number of patterns that can be reliably stored in a correlation matrix memory can never exceed the input space dimensionality.

In real-life situations, we often find that the key patterns presented to an associative memory are neither orthogonal nor highly separated from each other. Consequently, a correlation matrix memory characterized by the memory matrix of Eq. (2.34) may sometimes get confused and make *errors*. That is, the memory occasionally recognizes and associates patterns never seen or associated before. To illustrate this property of an associative memory, consider a set of key patterns.

$$\{\mathbf{x}_{\text{key}}\}: \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_q$$

and a corresponding set of memorized patterns,

$$\{\mathbf{y}_{\text{mem}}\}: \mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_q$$

To express the closeness of the key patterns in a linear signal space, we introduce the concept of *community*. We define the community of the set of patterns $\{\mathbf{x}_{\text{key}}\}$ as the lower bound on the inner products $\mathbf{x}_k^T \mathbf{x}_j$ of any two patterns \mathbf{x}_j and \mathbf{x}_k in the set. Let $\hat{\mathbf{M}}$ denote the memory matrix resulting from the training of the associative memory on a set of key patterns represented by $\{\mathbf{x}_{\text{key}}\}$ and a corresponding set of memorized patterns $\{\mathbf{y}_{\text{mem}}\}$ in accordance with Eq. (2.34). The response of the memory, \mathbf{y} , to a stimulus \mathbf{x}_j selected from the set $\{\mathbf{x}_{\text{key}}\}$ is given by Eq. (2.39), where it is assumed that each pattern in the set $\{\mathbf{x}_{\text{key}}\}$ is a unit vector (i.e., a vector with unit energy). Let it be further assumed that

$$\mathbf{x}_k^T \mathbf{x}_j \geq \gamma \quad \text{for } k \neq j \quad (2.51)$$

If the lower bound γ is large enough, the memory may fail to distinguish the response \mathbf{y} from that of any other key pattern contained in the set $\{\mathbf{x}_{\text{key}}\}$. If the key patterns in this set have the form

$$\mathbf{x}_j = \mathbf{x}_0 + \mathbf{v} \quad (2.52)$$

where \mathbf{v} is a stochastic vector, it is likely that the memory will recognize \mathbf{x}_0 and associate with it a vector \mathbf{y}_0 rather than any of the actual pattern pairs used to train it in the

first place; \mathbf{x}_0 and \mathbf{y}_0 denote a pair of patterns never seen before. This phenomenon may be termed *animal logic*, which is not logic at all (Cooper, 1973).

2.12 ADAPTATION

In performing a task of interest, we often find that *space* is one fundamental dimension of the learning process; *time* is the other. The *spatiotemporal* nature of learning is exemplified by many of the learning tasks (e.g., control, beamforming) discussed in Section 2.10. Species ranging from insects to humans have an inherent capacity to represent the temporal structure of experience. Such a representation makes it possible for an animal to *adapt* its behavior to the temporal structure of an event in its behavioral space (Gallistel, 1990).

When a neural network operates in a *stationary* environment (i.e., an environment whose statistical characteristics do not change with time), the essential statistics of the environment can, in theory, be *learned* by the network under the supervision of a teacher. In particular, the synaptic weights of the network can be computed by having the network undergo a training session with a set of data that is representative of the environment. Once the training process has completed, the synaptic weights of the network should capture the underlying statistical structure of the environment, which would justify “freezing” their values thereafter. Thus a learning system relies on *memory*, in one form or another, to recall and exploit past experiences.

Frequently, however, the environment of interest is *nonstationary*, which means that the statistical parameters of the information-bearing signals generated by the environment vary with time. In situations of this kind, the traditional methods of supervised learning may prove to be inadequate because the network is not equipped with the necessary means to *track* the statistical variations of the environment in which it operates. To overcome this shortcoming, it is desirable for a neural network to continually *adapt* its free parameters to variations in the incoming signals in a *real-time* fashion. Thus an *adaptive system* responds to every distinct input as a novel one. In other words the learning process encountered in an adaptive system never stops, with learning going on while signal processing is being performed by the system. This form of learning is called *continuous learning* or *learning-on-the-fly*.

Linear adaptive filters, built around a linear combiner (i.e., a single neuron operating in its linear mode), are designed to perform continuous learning. Despite their simple structure (and perhaps because of it), they are widely used in such diverse applications as radar, sonar, communications, seismology, and biomedical signal processing. The theory of linear adaptive filters has reached a highly mature stage of development (Haykin, 1996; Widrow and Stearns, 1985). However, the same cannot be said about nonlinear adaptive filters.¹¹

With continuous learning as the property of interest and a neural network as the vehicle for its implementation, the question we need to address is: How can a neural network adapt its behavior to the varying temporal structure of the incoming signals in its behavioral space? One way of addressing this fundamental issue is to recognize that statistical characteristics of a nonstationary process usually change slowly enough for the process to be considered *pseudostationary* over a window of short enough duration. Examples include:

- The mechanism responsible for the production of a speech signal may be considered essentially stationary over a period of 10 to 30 milliseconds.
- Radar returns from an ocean surface remain essentially stationary over a period of several seconds.
- With long-range weather forecasting in mind, weather related data may be viewed as essentially stationary over a period of minutes.
- In the context of long-range trends extending into months and years, stock market data may be considered as essentially stationary over a period of days.

We may thus exploit the pseudostationary property of a stochastic process to extend the utility of a neural network by *retraining* it at some regular intervals to account for statistical fluctuations of the incoming data. Such an approach may, for example, be suitable for processing stock market data.

For a more refined *dynamic* approach to learning, we may proceed as follows:

- Select a window short enough for the input data to be considered pseudostationary, and use the data to train the network.
- When a new data sample is received, update the window by dropping the oldest data sample and shifting the remaining data samples back by one time unit to make room for the new sample.
- Use the updated data window to retrain the network.
- Repeat the procedure on a continuing basis.

We may thus build temporal structure into the design of a neural network by having the network undergo *continual training* with *time-ordered examples*. According to this dynamic approach, a neural network is viewed as a *nonlinear adaptive filter* that represents a generalization of linear adaptive filters. However, for this dynamic approach to nonlinear adaptive filters to be feasible, the resources available must be *fast* enough to complete all the described computations in one sampling period. Only then can the filter keep up with changes in the input.

2.13 STATISTICAL NATURE OF THE LEARNING PROCESS

The last part of the chapter deals with statistical aspects of learning. In this context we are not interested in the evolution of the weight vector \mathbf{w} as a neural network is cycled through a learning algorithm. We instead focus on the deviation between a “target” function $f(\mathbf{x})$ and the “actual” function $F(\mathbf{x}, \mathbf{w})$ realized by the neural network where the vector \mathbf{x} denotes the input signal. The deviation is expressed in statistical terms.

A neural network is merely one form in which *empirical knowledge* about a physical phenomenon or environment of interest may be encoded through training. By “empirical” knowledge we mean a set of measurements that characterizes the phenomenon. To be more specific, consider the example of a stochastic phenomenon described by a random vector \mathbf{X} consisting of a set of *independent variables*, and a random scalar D representing a *dependent variable*. The elements of the random vector \mathbf{X} may have different physical meanings of their own. The assumption that the dependent variable D is a scalar has been made merely to simplify the exposition without any loss of generality. Suppose also that we have N realizations of the random vector \mathbf{X} denoted by

$\{\mathbf{x}_i\}_{i=1}^N$, and a corresponding set of realizations of the random scalar D denoted by $\{d_i\}_{i=1}^N$. These realizations (measurements) constitute the training sample denoted by

$$\mathcal{T} = \{(\mathbf{x}_i, d_i)\}_{i=1}^N \quad (2.53)$$

Ordinarily we do *not* have knowledge of the exact functional relationship between \mathbf{X} and D , so we proceed by proposing the model (White, 1989a)

$$D = f(\mathbf{X}) + \epsilon \quad (2.54)$$

where $f(\cdot)$ is a *deterministic* function of its argument vector, and ϵ is a random *expectational error* that represents our “ignorance” about the dependence of D and \mathbf{X} . The statistical model described by Eq. (2.54) is called a *regressive model*; it is depicted in Fig. 2.20a. The expectational error ϵ is, in general, a random variable with zero mean and positive probability of occurrence. On this basis, the regressive model of Fig. 2.20a has two useful properties:

1. The mean value of the expectational error ϵ , given any realization \mathbf{x} , is zero; that is,

$$E[\epsilon|\mathbf{x}] = 0 \quad (2.55)$$

where E is the statistical expectation operator. As a corollary to this property, we may state that the *regression function* $f(\mathbf{x})$ is the *conditional mean of the model output* D , given that the input $\mathbf{X} = \mathbf{x}$, as shown by

$$f(\mathbf{x}) = E[D|\mathbf{x}] \quad (2.56)$$

This property follows directly from Eq. (2.54) in light of Eq. (2.55).

2. The expectational error ϵ is uncorrelated with the regression function $f(\mathbf{X})$; that is

$$E[\epsilon f(\mathbf{X})] = 0 \quad (2.57)$$

This property is the well-known *principle of orthogonality*, which states that all the information about D available to us through the input \mathbf{X} has been encoded

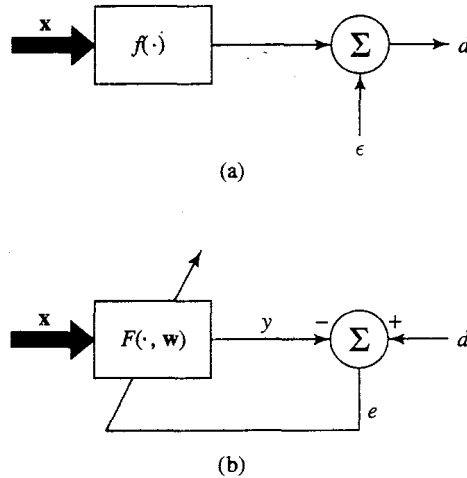


FIGURE 2.20 (a) Regressive model (mathematical). (b) Neural network model (physical).

into the regression function $f(\mathbf{X})$. Equation (2.57) is readily demonstrated by writing:

$$\begin{aligned} E[\epsilon f(\mathbf{X})] &= E[E[\epsilon f(\mathbf{X})|\mathbf{x}]] \\ &= E[f(\mathbf{X})E[\epsilon|\mathbf{x}]] \\ &= E[f(\mathbf{X}) \cdot 0] \\ &= 0 \end{aligned}$$

The regressive model of Fig. 2.20a is a “mathematical” description of a stochastic environment. Its purpose is to use the vector \mathbf{X} to explain or predict the dependent variable D . Figure 2.20b is the corresponding “physical” model of the environment. The purpose of this second model, based on a neural network, is to encode the empirical knowledge represented by the training sample \mathcal{T} into a corresponding set of synaptic weight vectors, \mathbf{w} , as shown by

$$\mathcal{T} \rightarrow \mathbf{w} \quad (2.58)$$

In effect, the neural network provides an “approximation” to the regressive model of Fig. 2.20a. Let the actual response of the neural network, produced in response to the input vector \mathbf{x} , be denoted by the random variable

$$Y = F(\mathbf{X}, \mathbf{w}) \quad (2.59)$$

where $F(\cdot, \mathbf{w})$ is the input–output function realized by the neural network. Given the training data \mathcal{T} of Eq. (2.53), the weight vector \mathbf{w} is obtained by minimizing the cost function

$$\mathcal{E}(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^N (d_i - F(\mathbf{x}_i, \mathbf{w}))^2 \quad (2.60)$$

where the factor $1/2$ has been used to be consistent with earlier notations and those used in subsequent chapters. Except for the scaling factor $1/2$, the cost function $\mathcal{E}(\mathbf{w})$ is the squared difference between the desired response d and the actual response y of the neural network, averaged over the entire training data set \mathcal{T} . The use of Eq. (2.60) as the cost function implies the use of “batch” training, by which we mean that the adjustments to the synaptic weights of the network are performed over the entire set of training examples rather than on an example-by-example basis.

Let the symbol $E_{\mathcal{T}}$ denote the *average operator* taken over the entire training sample \mathcal{T} . The variables or their functions that come under the average operator $E_{\mathcal{T}}$ are denoted by \mathbf{x} and d ; the pair (\mathbf{x}, d) represents an example in the training sample \mathcal{T} . In contrast, the statistical expectation operator E acts on the whole ensemble of random variables \mathbf{X} and D , which includes \mathcal{T} as a subset. The difference between the operators E and $E_{\mathcal{T}}$ should be carefully identified in what follows.

In light of the transformation described in Eq. (2.58), we may interchangeably use $F(\mathbf{x}, \mathbf{w})$ and $F(\mathbf{x}, \mathcal{T})$ and therefore rewrite Eq. (2.60) in the equivalent form

$$\mathcal{E}(\mathbf{w}) = \frac{1}{2} E_{\mathcal{T}}[(d - F(\mathbf{x}, \mathcal{T}))^2] \quad (2.61)$$

By adding and subtracting $f(\mathbf{x})$ to the argument $(d - F(\mathbf{x}, \mathcal{T}))$ and then using Eq. (2.54), we may write

$$\begin{aligned} d - F(\mathbf{x}, \mathcal{T}) &= (d - f(\mathbf{x})) + (f(\mathbf{x}) - F(\mathbf{x}, \mathcal{T})) \\ &= \epsilon + (f(\mathbf{x}) - F(\mathbf{x}, \mathcal{T})) \end{aligned}$$

By substituting this expression in Eq. (2.61) and then expanding terms, we may recast the cost function $\mathcal{E}(\mathbf{w})$ in the equivalent form

$$\mathcal{E}(\mathbf{w}) = \frac{1}{2}E_{\mathcal{T}}[\epsilon^2] + \frac{1}{2}E_{\mathcal{T}}[f(\mathbf{x}) - F(\mathbf{x}, \mathcal{T})]^2 + E_{\mathcal{T}}[\epsilon(f(\mathbf{x}) - F(\mathbf{x}, \mathcal{T}))] \quad (2.62)$$

However, the last expectation term on the right-hand side of Eq. (2.62) is zero for two reasons:

- The expectational error ϵ is uncorrelated with the regression function $f(\mathbf{x})$ by virtue of Eq. (2.57), interpreted in terms of the operator $E_{\mathcal{T}}$.
- The expectational error ϵ pertains to the regressive model of Fig. 2.20a, whereas the approximating function $F(\mathbf{x}, \mathbf{w})$ pertains to the neural network model of Fig. 2.20b.

Accordingly, Eq. (2.62) reduces to

$$\mathcal{E}(\mathbf{w}) = \frac{1}{2}E_{\mathcal{T}}[\epsilon^2] + \frac{1}{2}E_{\mathcal{T}}[(f(\mathbf{x}) - F(\mathbf{x}, \mathcal{T}))^2] \quad (2.63)$$

The first term on the right-hand side of Eq. (2.63) is the variance of the expectational (regressive modeling) error ϵ , evaluated over the training sample \mathcal{T} . This term represents the *intrinsic error* because it is independent of the weight vector \mathbf{w} . It may be ignored as far as the minimization of the cost function $\mathcal{E}(\mathbf{w})$ with respect to \mathbf{w} is concerned. Hence, the particular value of the weight vector \mathbf{w}^* that minimizes the cost function $\mathcal{E}(\mathbf{w})$ will also minimize the ensemble average of the squared distance between the regression function $f(\mathbf{x})$ and the approximating function $F(\mathbf{x}, \mathbf{w})$. In other words, the *natural measure* of the effectiveness of $F(\mathbf{x}, \mathbf{w})$ as a predictor of the desired response d is defined by

$$L_{av}(f(\mathbf{x}), F(\mathbf{x}, \mathbf{w})) = E_{\mathcal{T}}[f(\mathbf{x}) - F(\mathbf{x}, \mathcal{T})]^2 \quad (2.64)$$

This result is fundamentally important because it provides the mathematical basis for the tradeoff between the bias and variance resulting from the use of $F(\mathbf{x}, \mathbf{w})$ as the approximation to $f(\mathbf{x})$ (Geman et al., 1992).

Bias/Variance Dilemma

Invoking the use of Eq. (2.56), we may redefine the squared distance between $f(\mathbf{x})$ and $F(\mathbf{x}, \mathbf{w})$ as:

$$L_{av}(f(\mathbf{x}), F(\mathbf{x}, \mathbf{w})) = E_{\mathcal{T}}[(E[D|\mathbf{X} = \mathbf{x}] - F(\mathbf{x}, \mathcal{T}))^2] \quad (2.65)$$

This expression may also be viewed as the average value of the *estimation error* between the regression function $f(\mathbf{x}) = E[D|\mathbf{X} = \mathbf{x}]$ and the approximating function $F(\mathbf{x}, \mathbf{w})$, evaluated over the entire training sample \mathcal{T} . Notice that the conditional mean

$E[D|\mathbf{X} = \mathbf{x}]$ has a constant expectation with respect to the training data sample \mathcal{T} . Next we find that

$$E[D|\mathbf{X} = \mathbf{x}] - F(\mathbf{x}, \mathcal{T}) = (E[D|\mathbf{X} = \mathbf{x}] - E_{\mathcal{T}}[F(\mathbf{x}, \mathcal{T})]) + (E_{\mathcal{T}}[F(\mathbf{x}, \mathcal{T})] - F(\mathbf{x}, \mathcal{T}))$$

where we have simply added and subtracted the average $E_{\mathcal{T}}[F(\mathbf{x}, \mathcal{T})]$. By proceeding in a manner similar to that described for deriving Eq. (2.62) from (2.61), we may reformulate Eq. (2.65) as the sum of two terms (see Problem 2.22):

$$L_{av}(f(\mathbf{x}), F(\mathbf{x}, \mathcal{T})) = B^2(\mathbf{w}) + V(\mathbf{w}) \quad (2.66)$$

where $B(\mathbf{w})$ and $V(\mathbf{w})$ are themselves defined by

$$B(\mathbf{w}) = E_{\mathcal{T}}[F(\mathbf{x}, \mathcal{T})] - E[D|\mathbf{X} = \mathbf{x}] \quad (2.67)$$

and

$$V(\mathbf{w}) = E_{\mathcal{T}}[(F(\mathbf{x}, \mathcal{T}) - E_{\mathcal{T}}[F(\mathbf{x}, \mathcal{T})])^2] \quad (2.68)$$

We now make two important observations:

1. The term $B(\mathbf{w})$ is the *bias* of the average value of the approximating function $F(\mathbf{x}, \mathcal{T})$, measured with respect to the regression function $f(\mathbf{x}) = E[D|\mathbf{X} = \mathbf{x}]$. This term represents the inability of the neural network defined by the function $F(\mathbf{x}, \mathbf{w})$ to accurately approximate the regression function $f(\mathbf{x}) = E[D|\mathbf{X} = \mathbf{x}]$. We may therefore view the bias $B(\mathbf{w})$ as an *approximation error*.
2. The term $V(\mathbf{w})$ is the *variance* of the approximating function $F(\mathbf{x}, \mathbf{w})$, measured over the entire training sample \mathcal{T} . This second term represents the inadequacy of the information contained in the training sample \mathcal{T} about the regression function $f(\mathbf{x})$. We may therefore view the variance $V(\mathbf{w})$ as the manifestation of an *estimation error*.

Figure 2.21 illustrates the relations between the target and approximating functions, and shows how the estimation errors, namely bias and variance, accumulate. To

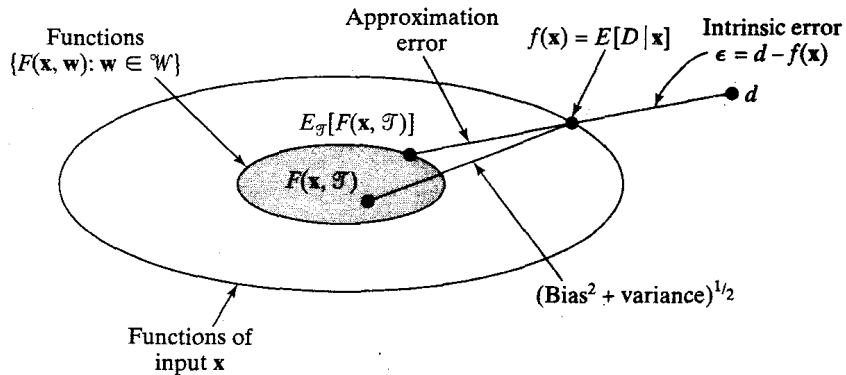


FIGURE 2.21 Illustration of the various sources of error in solving the regression problem.

achieve good overall performance, the bias $B(\mathbf{w})$ and the variance $V(\mathbf{w})$ of the approximating function $F(\mathbf{x}, \mathbf{w}) = F(\mathbf{x}, \mathcal{T})$ would both have to be small.

Unfortunately, we find that in a neural network that learns by example and does so with a training sample of fixed size, the price for achieving a small bias is a large variance. For a single neural network, it is only when the size of the training sample becomes infinitely large that we can hope to eliminate both bias and variance at the same time. We then have a *bias/variance dilemma*, and the consequence is prohibitively slow convergence (Geman et al., 1992). The bias/variance dilemma may be circumvented if we are willing to *purposely* introduce bias, which then makes it possible to eliminate the variance or to reduce it significantly. Needless to say, we must be sure that the bias built into the network design is harmless. In the context of pattern classification, for example, the bias is said to be “harmless” in the sense that it will contribute significantly to mean-square error only if we try to infer regressions that are not in the anticipated class. In general, bias must be *designed* for each specific application of interest. A practical way of achieving such an objective is to use a *constrained* network architecture, which usually performs better than a general-purpose architecture. For example, the constraints and therefore the bias may take the form of prior knowledge built into the network design using (1) *weight-sharing* where several synapses of the network are controlled by a single weight, and/or (2) *local receptive fields* assigned to individual neurons in the network, as demonstrated in the application of a multilayer perceptron to the optical character recognition problem (LeCun et al., 1990a). These network design issues were briefly discussed in Section 1.7.

2.14 STATISTICAL LEARNING THEORY

In this section we continue the statistical characterization of neural networks by describing a *learning theory* that addresses the fundamental issue of how to control the generalization ability of a neural network in mathematical terms. The discussion is presented in the context of supervised learning.

A model of supervised learning consists of three interrelated components, illustrated in Fig. 2.22 and abstracted in mathematical terms as follows (Vapnik, 1992, 1998):

1. *Environment*. The environment is stationary, supplying a vector \mathbf{x} with a fixed but unknown cumulative (probability) distribution function $F_{\mathbf{x}}(\mathbf{x})$.
2. *Teacher*. The teacher provides a desired response d for every input vector \mathbf{x} received from the environment, in accordance with a conditional cumulative distribution function $F_{\mathbf{x}}(\mathbf{x}|d)$ that is also fixed but unknown. The desired response d and input vector \mathbf{x} are related by

$$d = f(\mathbf{x}, v) \quad (2.69)$$

where v is a noise term, permitting the teacher to be “noisy.”

3. *Learning machine (algorithm)*. The learning machine (neural network) is capable of implementing a set of input–output mapping functions described by

$$y = F(\mathbf{x}, \mathbf{w}) \quad (2.70)$$

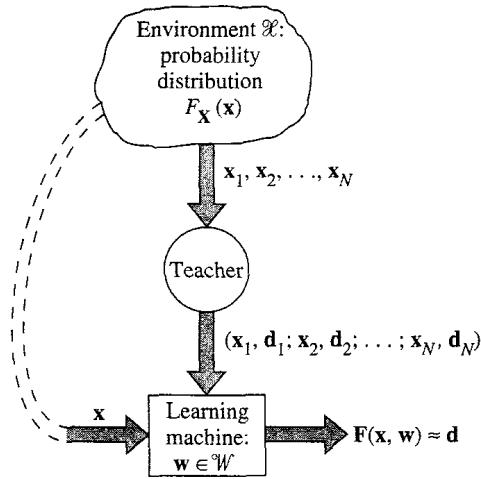


FIGURE 2.22 Model of the supervised learning process.

where y is the actual response produced by the learning machine in response to the input \mathbf{x} , and \mathbf{w} is a set of free parameters (synaptic weights) selected from the parameter (weight) space \mathcal{W} .

Equations (2.69) and (2.70) are written in terms of the examples used to perform the training.

The supervised learning problem is that of selecting the particular function $F(\mathbf{x}, \mathbf{w})$ that approximates the desired response d in an optimum fashion, with “optimum” being defined in some statistical sense. The selection itself is based on the set of N independent, identically distributed (*iid*) training examples described in Eq. (2.53) and reproduced here for convenience of presentation:

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

Each example pair is drawn by the learning machine from \mathcal{T} with a joint cumulative (probability) distribution function $F_{\mathbf{x},d}(\mathbf{x}, d)$, which, like the other distribution functions, is also fixed but unknown. The feasibility of supervised learning depends on this question: Do the training examples $\{(\mathbf{x}_i, d_i)\}$ contain sufficient information to construct a learning machine capable of good generalization performance? An answer to this fundamental question lies in the use of tools pioneered by Vapnik and Chervonenkis (1971). Specifically, we proceed by viewing the supervised learning problem as an *approximation problem*, which involves finding the function $F(\mathbf{x}, \mathbf{w})$ that is the best possible approximation to the desired function $f(\mathbf{x})$.

Let $L(d, F(\mathbf{x}, \mathbf{w}))$ denote a measure of the *loss* or *discrepancy* between the desired response d corresponding to an input vector \mathbf{x} and the actual response $F(\mathbf{x}, \mathbf{w})$ produced by the learning machine. A popular definition for the loss $L(d, F(\mathbf{x}, \mathbf{w}))$ is the *quadratic loss function* defined as the squared distance between $d = f(\mathbf{x})$ and the approximation $F(\mathbf{x}, \mathbf{w})$ as shown by¹²

$$L(d, F(\mathbf{x}, \mathbf{w})) = (d - F(\mathbf{x}, \mathbf{w}))^2 \quad (2.71)$$

The squared distance of Eq. (2.64) is the ensemble-averaged extension of $L(d, F(\mathbf{x}, \mathbf{w}))$, with the averaging being performed over all the example pairs (\mathbf{x}, d) .

Most of the literature on statistical learning theory deals with a specific loss. The strong point of the statistical learning theory presented here is that it does *not* depend critically on the form of the loss function $L(d, F(\mathbf{x}, \mathbf{w}))$. Later in the section we do restrict the discussion to a specific loss function.

The expected value of the loss is defined by the *risk functional*

$$R(\mathbf{w}) = \int L(d, F(\mathbf{x}, \mathbf{w})) dF_{\mathbf{x},D}(\mathbf{x}, d) \quad (2.72)$$

where the integral is a multi-fold integral taken over all possible values of the example pair (\mathbf{x}, d) . The goal of supervised learning is to minimize the risk functional $R(\mathbf{w})$ over the class of approximating functions $\{F(\mathbf{x}, \mathbf{w}), \mathbf{w} \in \mathcal{W}\}$. However, evaluation of the risk functional $R(\mathbf{w})$ is complicated because the joint cumulative distribution function $F_{\mathbf{x},D}(\mathbf{x}, d)$ is usually unknown. In supervised learning, the only information available is contained in the training data set \mathcal{T} . To overcome this mathematical difficulty, we use the inductive principle of empirical risk minimization (Vapnik, 1982). This principle relies entirely on availability of the training data set \mathcal{T} , which makes it perfectly suited to the design philosophy of neural networks.

Some Basic Definitions

Before proceeding further, we digress briefly to introduce some basic definitions that we use in the material to follow.

Convergence in probability. Consider a sequence of random variables a_1, a_2, \dots, a_N . This sequence of random variables is said to *converge in probability* to a random variable a_0 if for any $\delta > 0$, the probabilistic relation

$$P(|a_N - a_0| > \delta) \xrightarrow{P} 0 \quad \text{as } N \rightarrow \infty \quad (2.73)$$

holds.

Supremum and infimum. The supremum of a nonempty set \mathcal{A} of scalars, denoted by $\sup \mathcal{A}$, is defined as the smallest scalar x such that $x \geq y$ for all $y \in \mathcal{A}$. If no such scalar exists, we say that the supremum of the nonempty set \mathcal{A} is ∞ . Similarly, the infimum of set \mathcal{A} , denoted by $\inf \mathcal{A}$, is defined as the largest scalar x such that $x \leq y$ for all $y \in \mathcal{A}$. If no such scalar exists, we say that the infimum of the nonempty set \mathcal{A} is ∞ .

Empirical risk functional. Given the training sample $\mathcal{T} = \{(\mathbf{x}_i, d_i)\}_{i=1}^N$, the empirical risk functional is defined in terms of the loss function $L(d_i, F(\mathbf{x}_i, \mathbf{w}))$ as

$$R_{\text{emp}}(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^N L(d_i, F(\mathbf{x}_i, \mathbf{w})) \quad (2.74)$$

Strict Consistency. Consider the set \mathcal{W} of functions $L(d, F(\mathbf{x}, \mathbf{w}))$ whose underlying distribution is defined by the joint cumulative distribution function $F_{\mathbf{x}, D}(\mathbf{x}, d)$. Let $\mathcal{W}(c)$ be any nonempty subset of this set of functions, such that

$$\mathcal{W}(c) = \left\{ \mathbf{w}: \int L(d, F(\mathbf{x}, \mathbf{w})) \geq c \right\} \quad (2.75)$$

where $c \in (-\infty, \infty)$. The empirical risk functional is said to be *strictly (nontrivially) consistent* if for any subset $\mathcal{W}(c)$ the following convergence in probability

$$\inf_{\mathbf{w} \in \mathcal{W}(c)} R_{\text{emp}}(\mathbf{w}) \xrightarrow{P} \inf_{\mathbf{w} \in \mathcal{W}(c)} R(\mathbf{w}) \quad \text{as } N \rightarrow \infty \quad (2.76)$$

holds.

With these definitions we may resume the discussion of Vapnik's statistical learning theory.

Principle of Empirical Risk Minimization

The basic idea of the principle of *empirical risk minimization* is to work with the empirical risk functional $R_{\text{emp}}(\mathbf{w})$ defined in Eq. (2.74). This new functional differs from the risk functional $R(\mathbf{w})$ of Eq. (2.72) in two desirable ways:

1. It does *not* depend on the unknown distribution function $F_{\mathbf{x}, D}(\mathbf{x}, d)$ in an explicit sense.
2. In theory it can be minimized with respect to the weight vector \mathbf{w} .

Let \mathbf{w}_{emp} and $F(\mathbf{x}, \mathbf{w}_{\text{emp}})$ denote the weight vector and the corresponding mapping that minimize the empirical risk functional $R_{\text{emp}}(\mathbf{w})$ in Eq. (2.74). Similarly, let \mathbf{w}_o and $F(\mathbf{x}, \mathbf{w}_o)$ denote the weight vector and the corresponding mapping that minimize the actual risk functional $R(\mathbf{w})$ in Eq. (2.72). Both \mathbf{w}_{emp} and \mathbf{w}_o belong to the weight space \mathcal{W} . The problem we must now consider is the conditions under which the approximate mapping $F(\mathbf{x}, \mathbf{w}_{\text{emp}})$ is "close" to the desired mapping $F(\mathbf{x}, \mathbf{w}_o)$ as measured by the mismatch between $R(\mathbf{w}_{\text{emp}})$ and $R(\mathbf{w}_o)$.

For some fixed $\mathbf{w} = \mathbf{w}^*$, the risk functional $R(\mathbf{w}^*)$ determines the *mathematical expectation* of a random variable defined by

$$Z_{\mathbf{w}^*} = L(d, F(\mathbf{x}, \mathbf{w}^*)) \quad (2.77)$$

In contrast, the empirical risk functional $R_{\text{emp}}(\mathbf{w}^*)$ is the *empirical (arithmetic) mean* of the random variable $Z_{\mathbf{w}^*}$. According to the *law of large numbers*, which constitutes one of the main theorems of probability theory, in general cases we find that as the size N of the training sample \mathcal{T} is made infinitely large, the empirical mean of the random variable $Z_{\mathbf{w}^*}$ converges to its expected value. This observation provides theoretical justification for the use of the empirical risk functional $R_{\text{emp}}(\mathbf{w})$ in place of the risk functional $R(\mathbf{w})$. However, just because the empirical mean of $Z_{\mathbf{w}^*}$ converges to its expected value, there is no reason to expect that the weight vector \mathbf{w}_{emp} that minimizes the empirical risk functional $R_{\text{emp}}(\mathbf{w})$ will also minimize the risk functional $R(\mathbf{w})$.

We may satisfy this requirement in an approximate fashion by proceeding as follows. If the empirical risk functional $R_{\text{emp}}(\mathbf{w})$ approximates the original risk functional

$R(\mathbf{w})$ uniformly in \mathbf{w} with some precision ϵ , then the minimum of $R_{\text{emp}}(\mathbf{w})$ deviates from the minimum of $R(\mathbf{w})$ by an amount not exceeding 2ϵ . Formally, this means that we must impose a stringent condition, such that for any $\mathbf{w} \in \mathcal{W}$ and $\epsilon > 0$, the probabilistic relation

$$P\left(\sup_{\mathbf{w}} |R(\mathbf{w}) - R_{\text{emp}}(\mathbf{w})| > \epsilon\right) \rightarrow 0 \quad \text{as } N \rightarrow \infty \quad (2.78)$$

holds (Vapnik, 1982). When Eq. (2.78) is satisfied, we say that a *uniform convergence in the weight vector \mathbf{w} of the empirical mean risk to its expected value occurs*. Equivalently, provided that for any prescribed precision ϵ we can assert the inequality

$$P\left(\sup_{\mathbf{w}} |R(\mathbf{w}) - R_{\text{emp}}(\mathbf{w})| > \epsilon\right) < \alpha \quad (2.79)$$

for some $\alpha > 0$, then as a consequence the following inequality also holds:

$$P(R(\mathbf{w}_{\text{emp}}) - R(\mathbf{w}_o) > 2\epsilon) < \alpha \quad (2.80)$$

In other words, if the condition (2.79) holds, then with probability at least $(1 - \alpha)$, the solution $F(\mathbf{x}, \mathbf{w}_{\text{emp}})$ that minimizes the empirical risk functional $R_{\text{emp}}(\mathbf{w})$ will give an actual risk $R(\mathbf{w}_{\text{emp}})$ that deviates from the true minimum possible actual risk $R(\mathbf{w}_o)$ by an amount not exceeding 2ϵ . Indeed, the condition (2.79) implies that with probability $(1 - \alpha)$ the following two inequalities are satisfied simultaneously (Vapnik, 1982):

$$R(\mathbf{w}_{\text{emp}}) - R_{\text{emp}}(\mathbf{w}_{\text{emp}}) < \epsilon \quad (2.81)$$

$$R_{\text{emp}}(\mathbf{w}_o) - R(\mathbf{w}_o) < \epsilon \quad (2.82)$$

These two equations define the differences between the true risk and empirical risk functionals at $\mathbf{w} = \mathbf{w}_{\text{emp}}$ and $\mathbf{w} = \mathbf{w}_o$, respectively. Furthermore, since \mathbf{w}_{emp} and \mathbf{w}_o are the minimum points of $R_{\text{emp}}(\mathbf{w})$ and $R(\mathbf{w})$, respectively, it follows that

$$R_{\text{emp}}(\mathbf{w}_{\text{emp}}) \leq R_{\text{emp}}(\mathbf{w}_o) \quad (2.83)$$

By adding the inequalities (2.81) and (2.82), and then using (2.83), we may write the following inequality

$$R(\mathbf{w}_{\text{emp}}) - R(\mathbf{w}_o) < 2\epsilon \quad (2.84)$$

Also, since the inequalities (2.81) and (2.82) are both satisfied simultaneously with probability $(1 - \alpha)$, so is the inequality (2.84). We may also state that with probability α the inequality

$$R(\mathbf{w}_{\text{emp}}) - R(\mathbf{w}_o) > 2\epsilon$$

holds, which is a restatement of (2.80).

We are now ready to make a formal statement of the *principle of empirical risk minimization* in three interrelated parts (Vapnik, 1982, 1998):

1. In place of the risk functional $R(\mathbf{w})$, construct the empirical risk functional

$$R_{\text{emp}}(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^N L(d_i, F(\mathbf{x}_i, \mathbf{w}))$$

on the basis of the training set of i.i.d. examples

$$(\mathbf{x}_i, d_i), \quad i = 1, 2, \dots, N$$

2. Let \mathbf{w}_{emp} denote the weight vector that minimizes the empirical risk functional $R_{\text{emp}}(\mathbf{w})$ over the weight space \mathcal{W} . Then $R(\mathbf{w}_{\text{emp}})$ converges in probability to the minimum possible values of the actual risk $R(\mathbf{w})$, $\mathbf{w} \in \mathcal{W}$, as the size N of the training sample is made infinitely large, provided that the empirical risk functional $R_{\text{emp}}(\mathbf{w})$ converges uniformly to the actual risk functional $R(\mathbf{w})$.
3. Uniform convergence as defined by

$$P\left(\sup_{\mathbf{w} \in \mathcal{W}} |R(\mathbf{w}) - R_{\text{emp}}(\mathbf{w})| > \epsilon\right) \rightarrow 0 \quad \text{as } N \rightarrow \infty$$

is a necessary and sufficient condition for the consistency of the principle of empirical risk minimization.

For a physical interpretation of this important principle, we offer the following observation. Prior to the training of a learning machine, all approximating functions are equally likely. As the training of the learning machine progresses, the likelihood of those approximating functions $F(\mathbf{x}, \mathbf{w})$ that are consistent with the training data set $\{\mathbf{x}_i, d_i\}_{i=1}^N$ is increased. As the size N of the training data set grows, and the input space is thereby “densely” populated, the minimum point of the empirical risk functional $R_{\text{emp}}(\mathbf{w})$ converges in probability to the minimum point of the true risk functional $R(\mathbf{w})$.

VC Dimension

The theory of uniform convergence of the empirical risk functional $R_{\text{emp}}(\mathbf{w})$ to the actual risk functional $R(\mathbf{w})$ includes bounds on the rate of convergence, which are based on an important parameter called the *Vapnik–Chervonenkis dimension*, or simply the *VC dimension*, named in honor of its originators, Vapnik and Chervonenkis (1971). The VC dimension is a measure of the *capacity* or *expressive power* of the family of classification functions realized by the learning machine.

To describe the concept of VC dimension in a manner suitable for our purposes, consider a binary pattern classification problem, for which the desired response is written as $d \in \{0, 1\}$. We use the term *dichotomy* to refer to a binary classification function or decision rule. Let \mathcal{F} denote the ensemble of dichotomies implemented by a learning machine, that is,

$$\mathcal{F} = \{F(\mathbf{x}, \mathbf{w}): \mathbf{w} \in \mathcal{W}, F: \mathbb{R}^m \mathcal{W} \rightarrow \{0, 1\}\} \quad (2.85)$$

Let \mathcal{L} denote the set of N points in the m -dimensional space \mathcal{X} of input vectors, that is,

$$\mathcal{L} = \{\mathbf{x}_i \in \mathcal{X}; i = 1, 2, \dots, N\} \quad (2.86)$$

A dichotomy implemented by the learning machine partitions \mathcal{L} into two disjoint subsets \mathcal{L}_0 and \mathcal{L}_1 , such that we may write

$$F(\mathbf{x}, \mathbf{w}) = \begin{cases} 0 & \text{for } \mathbf{x} \in \mathcal{L}_0 \\ 1 & \text{for } \mathbf{x} \in \mathcal{L}_1 \end{cases} \quad (2.87)$$

Let $\Delta_{\mathcal{F}}(\mathcal{L})$ denote the number of distinct dichotomies implemented by the learning machine, and $\Delta_{\mathcal{F}}(l)$ denote the maximum of $\Delta_{\mathcal{F}}(\mathcal{L})$ over all \mathcal{L} with $|\mathcal{L}| = l$, where $|\mathcal{L}|$ is the number of elements of \mathcal{L} . We say that \mathcal{L} is *shattered* by \mathcal{F} if $\Delta_{\mathcal{F}}(\mathcal{L}) = 2^{|\mathcal{L}|}$, that is, if all possible dichotomies of \mathcal{L} can be induced by functions in \mathcal{F} . We refer to $\Delta_{\mathcal{F}}(l)$ as the *growth function*.

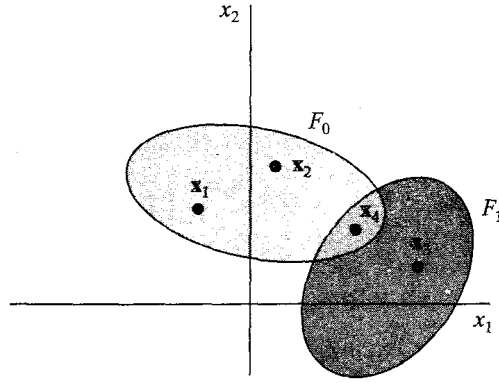


FIGURE 2.23 Diagram for Example 2.1

Example 2.1

Figure 2.23 illustrates a two-dimensional input space \mathcal{X} consisting of four points $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$, and \mathbf{x}_4 . The decision boundaries of functions F_0 and F_1 , indicated in the figure, correspond to the classes (hypotheses) 0 and 1 being true, respectively. From Fig. 2.23 we see that the function F_0 induces the dichotomy

$$\mathcal{D}_0 = \{\mathcal{S}_0 = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_4\}, \mathcal{S}_1 = \{\mathbf{x}_3\}\}$$

On the other hand, the function F_1 induces the dichotomy

$$\mathcal{D}_1 = \{\mathcal{S}_0 = \{\mathbf{x}_1, \mathbf{x}_2\}, \mathcal{S}_1 = \{\mathbf{x}_3, \mathbf{x}_4\}\}$$

With the set \mathcal{S} consisting of four points, the cardinality $|\mathcal{S}| = 4$. Hence,

$$\Delta_{\mathcal{F}}(\mathcal{S}) = 2^4 = 16$$

■

Returning to the general discussion delineated by the ensemble \mathcal{F} of dichotomies in Eq. (2.85) and the corresponding set of points \mathcal{L} in Eq. (2.86), we may now formally define the VC dimension as (Vapnik and Chervonenkis, 1971; Kearns and Vazirani, 1994; Vidyasagar, 1997; Vapnik, 1998):

The VC dimension of an ensemble of dichotomies \mathcal{F} is the cardinality of the largest set \mathcal{L} that is shattered by \mathcal{F} .

In other words, the VC dimension of \mathcal{F} , written as $\text{VCdim}(\mathcal{F})$, is the largest N such that $\Delta_{\mathcal{F}}(N) = 2^N$. Stated in more familiar terms, the VC dimension of the set of classification functions $\{F(\mathbf{x}, \mathbf{w}): \mathbf{w} \in \mathcal{W}\}$ is the maximum number of training examples that can be learned by the machine without error for all possible binary labelings of the classification functions.

Example 2.2

Consider a simple decision rule in an m -dimensional space \mathcal{X} of input vectors, which is described by

$$\mathcal{F}: y = \varphi(\mathbf{w}^T \mathbf{x} + b) \quad (2.88)$$

where \mathbf{x} is an m -dimensional weight vector and b is a bias. The activation function φ is a threshold function; that is,

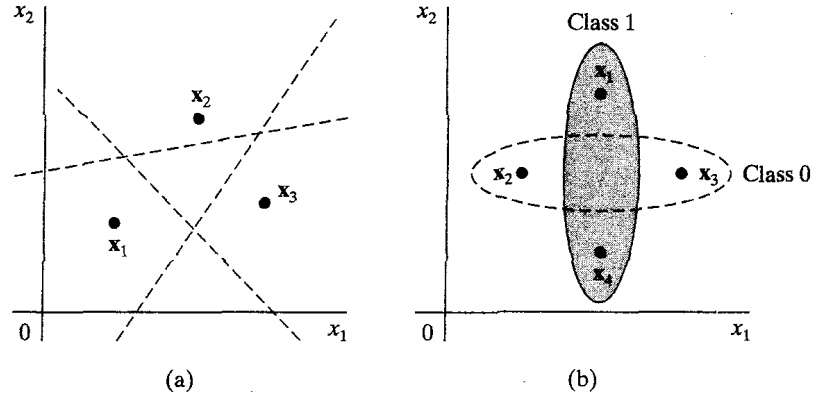


FIGURE 2.24 A pair of two-dimensional data distributions for Example 2.2.

$$\varphi(v) = \begin{cases} 1, & v \geq 0 \\ 0, & v < 0 \end{cases}$$

The VC dimension of the decision rule in Eq. (2.88) is given by

$$\text{VC dim}(\mathcal{F}) = m + 1 \quad (2.89)$$

To demonstrate this result, consider the situations described in Fig. 2.24 pertaining to a two-dimensional input space (i.e., $m = 2$). In Fig. 2.24a, we have three points \mathbf{x}_1 , \mathbf{x}_2 , and \mathbf{x}_3 . Three different possible labelings of these points are included in Fig. 2.24a, from which we readily see that a maximum of three lines can shatter these points. In Fig. 2.24b we have points \mathbf{x}_1 , \mathbf{x}_2 , \mathbf{x}_3 , and \mathbf{x}_4 , with points \mathbf{x}_2 and \mathbf{x}_3 labeled as 0 and points \mathbf{x}_1 and \mathbf{x}_4 labeled as 1. This time, however, we see that points \mathbf{x}_1 and \mathbf{x}_4 cannot be shattered from \mathbf{x}_2 and \mathbf{x}_3 by a line. The VC dimension of the decision rule described in Eq. (2.88) with $m = 2$ is therefore 3, which is in accord with the formula of Eq. (2.89). ■

Example 2.3

With the VC dimension providing a measure of the capacity of a set of classification (indicator) functions, we may be led to expect that a learning machine with many free parameters would have a high VC dimension, whereas a learning machine with few free parameters would have a low VC dimension. We now present a counterexample¹³ to this statement.

Consider the one parameter family of indicator functions, defined by

$$f(x, a) = \text{sgn}(\sin(ax)), \quad a \in \mathbb{R}$$

where $\text{sgn}(\cdot)$ is the signum function. Suppose we choose any number N and the requirement is to find N points that can be shattered. This requirement is satisfied by the set of functions $f(x, a)$ by choosing

$$x_i = 10^{-i}, \quad i = 1, 2, \dots, N$$

To separate these data points into two classes determined by the sequence

$$d_1, d_2, \dots, d_N, \quad d_i \in \{-1, 1\}$$

it is sufficient that we choose the parameter a according to the formula:

$$a = \pi \left(1 + \sum_{i=1}^N \frac{(1 - d_i)10^i}{2} \right)$$

We thus conclude that the VC dimension of the family of indicator functions $f(x, a)$ with a single free parameter a is infinite. ■

Importance of the VC dimension and its Estimation

The VC dimension is a purely *combinatorial concept* that has no connection with the geometric notion of dimension. It plays a central role in statistical learning theory as will be shown in the material presented in the next two subsections. The VC dimension is also important from a design point of view. Roughly speaking, the number of examples needed to learn a class of interest reliably is proportional to the VC dimension of that class. Therefore, an estimate of the VC dimension is of primary concern.

In some cases the VC dimension is determined by the free parameters of a neural network. In most practical cases, however, it is difficult to evaluate the VC dimension by analytic means. Nevertheless, *bounds* on the VC dimension of neural networks are often *tractable*. In this context, the following two results are of special interest:¹⁴

1. Let \mathcal{N} denote an arbitrary feedforward network built up from neurons with a threshold (Heaviside) activation function:

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

The VC dimension of \mathcal{N} is $O(W \log W)$ where W is the total number of free parameters in the network.

This first result is due to Cover (1968) and Baum and Haussler (1989).

2. Let \mathcal{N} denote a multilayer feedforward network whose neurons use a sigmoid activation function

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

The VC dimension of \mathcal{N} is $O(W^2)$, where W is the total number of free parameters in the network.

This second result is due to Koiran and Sontag (1996). They arrived at this result by first showing that networks consisting of two types of neurons, one linear and the other using a threshold activation function, already have a VC dimension proportional to W^2 . This is a rather surprising result, since a purely linear network has a VC dimension proportional to W as shown in Example 2.2, while a purely threshold neural network has a VC dimension proportional to $W \log W$ by virtue of result 1. The desired result pertaining to a sigmoid neural network is then obtained by invoking two approximations. First, neurons with threshold activation functions are approximated by sigmoidal ones with large synaptic weights. Second, linear neurons are approximated by sigmoidal neurons with small synaptic weights.

The important point to note here is that multilayer feedforward networks have a *finite* VC dimension.

Constructive Distribution-free Bounds on the Generalization Ability of Learning Machines

At this point in the discussion we find it instructive to consider the specific case of binary pattern classification, for which the desired response is defined by $d \in \{0, 1\}$. In a corresponding way the loss function has only two possible values as shown by

$$L(d, F(\mathbf{x}, \mathbf{w})) = \begin{cases} 0 & \text{if } F(\mathbf{x}, \mathbf{w}) = d \\ 1 & \text{otherwise} \end{cases} \quad (2.90)$$

Under these conditions the risk functional $R(\mathbf{w})$ and the empirical risk functional $R_{\text{emp}}(\mathbf{w})$ defined in Eqs. (2.72) and (2.74) respectively, assume the following interpretations:

- The risk functional $R(\mathbf{w})$ is the *probability of classification error* (i.e., error rate), denoted by $P(\mathbf{w})$.
- The empirical risk functional $R_{\text{emp}}(\mathbf{w})$ is the *training error* (i.e., frequency of errors made during the training session), denoted by $\nu(\mathbf{w})$.

Now, according to the *law of large numbers* (Gray and Davisson, 1986), the empirical frequency of occurrence of an event converges almost surely to the actual probability of that event as the number of trials (assumed to be independent and identically distributed) is made infinitely large. In the context of the discussion presented here, this result means that for any weight vector \mathbf{w} , which does not depend on the training set, and for any precision $\epsilon > 0$, the following condition holds (Vapnik, 1982):

$$P(|P(\mathbf{w}) - \nu(\mathbf{w})| > \epsilon) \rightarrow 0 \quad \text{as } N \rightarrow \infty \quad (2.91)$$

where N is the size of the training set. Note, however, that the condition (2.91) does not imply that the classification rule (i.e., a particular weight vector \mathbf{w}) that minimizes the training error $\nu(\mathbf{w})$ will also minimize the probability of classification error $P(\mathbf{w})$. For a training set of sufficiently large size N , the proximity between $\nu(\mathbf{w})$ and $P(\mathbf{w})$ follows from a stronger condition, which stipulates that the following condition holds for any $\epsilon > 0$ (Vapnik, 1982):

$$P(\sup_{\mathbf{w}} |P(\mathbf{w}) - \nu(\mathbf{w})| > \epsilon) \rightarrow 0 \quad \text{as } N \rightarrow \infty \quad (2.92)$$

In such a case, we speak of the *uniform convergence of the frequency of training errors to the probability* that $\nu(\mathbf{w}) = P(\mathbf{w})$.

The notion of VC dimension provides a bound on the rate of uniform convergence. Specifically, for the set of classification functions with VC dimension h , the following inequality holds (Vapnik, 1982, 1998):

$$P(\sup_{\mathbf{w}} |P(\mathbf{w}) - \nu(\mathbf{w})| > \epsilon) < \left(\frac{2eN}{h} \right)^h \exp(-\epsilon^2 N) \quad (2.93)$$

where N is the size of the training sample and e is the base of the natural logarithm. We want to make the right-hand side of the inequality (2.93) small for large N in order to achieve uniform convergence. The factor $\exp(-\epsilon^2 N)$ is helpful in this regard, since it decays exponentially with increasing N . The remaining factor $(2eN/h)^h$ represents a *bound* on the growth function $\Delta_{\mathcal{F}}(l)$ for the family of functions $\mathcal{F} = \{F(\mathbf{x}, \mathbf{w}); \mathbf{w} \in \mathcal{W}\}$ for $l \geq h \geq 1$ as obtained by *Sauer's lemma*.¹⁵ Provided that this function does *not* grow too fast, the right-hand side will go to zero as N goes to infinity; this requirement is satisfied if the VC dimension h is finite. In other words, a finite VC dimension is a necessary and sufficient condition for uniform convergence of the principle of empirical risk minimization. If the input space \mathcal{X} has finite cardinality, any family of dichotomies \mathcal{F} will have finite VC dimension with respect to \mathcal{X} , although the reverse is not necessarily true.

Let α denote the probability of occurrence of the event

$$\sup_{\mathbf{w}} |P(\mathbf{w}) - \nu(\mathbf{w})| \geq \epsilon$$

Then, with probability $1 - \alpha$, we may state that for all weight vectors $\mathbf{w} \in \mathcal{W}$ the following inequality holds:

$$P(\mathbf{w}) < \nu(\mathbf{w}) + \epsilon \quad (2.94)$$

Using the bound described in Eq. (2.93) and the definition for the probability α , we may thus set

$$\alpha = \left(\frac{2eN}{h} \right)^h \exp(-\epsilon^2 N) \quad (2.95)$$

Let $\epsilon_0(N, h, \alpha)$ denote the special value of ϵ that satisfies Eq. (2.95). Hence, we readily obtain the following important result (Vapnik, 1992):

$$\epsilon_0(N, h, \alpha) = \sqrt{\frac{h}{N} \left[\log \left(\frac{2N}{h} \right) + 1 \right] - \frac{1}{N} \log \alpha} \quad (2.96)$$

We refer to $\epsilon_0(N, h, \alpha)$ as a *confidence interval*, the value of which depends on the size N of the training sample, the VC dimension h , and the probability α .

The bound described in (2.93) with $\epsilon = \epsilon_0(N, h, \alpha)$ is achieved for the worst case $P(\mathbf{w}) = \frac{1}{2}$, but not, unfortunately, for small $P(\mathbf{w})$, which is the case of interest in practice. For small $P(\mathbf{w})$, a more useful bound is obtained by considering a modification of the inequality (2.93) as follows (Vapnik, 1982, 1998):

$$P \left(\sup_{\mathbf{w}} \frac{|P(\mathbf{w}) - \nu(\mathbf{w})|}{\sqrt{P(\mathbf{w})}} > \epsilon \right) < \left(\frac{2eN}{h} \right)^h \exp \left(-\frac{\epsilon^2 N}{4} \right) \quad (2.97)$$

In the literature, different results are reported for the bound in (2.97), depending on which particular form of inequality is used for its derivation. Nevertheless, they all have a similar form. From (2.97) it follows that with probability $1 - \alpha$, and simultaneously for all $\mathbf{w} \in \mathcal{W}$ (Vapnik, 1992, 1998),

$$P(\mathbf{w}) \leq \nu(\mathbf{w}) + \epsilon_1(N, h, \alpha, \nu) \quad (2.98)$$

where $\epsilon_1(N, h, \alpha, \nu)$ is a new confidence interval defined in terms of the former confidence interval $\epsilon_0(N, h, \alpha)$ as follows (see Problem 2.25):

$$\epsilon_1(N, h, \alpha, \nu) = 2\epsilon_0^2(N, h, \alpha) \left(1 + \sqrt{1 + \frac{\nu(\mathbf{w})}{\epsilon_0^2(N, h, \alpha)}} \right) \quad (2.99)$$

This second confidence interval depends on the training error $\nu(\mathbf{w})$. For $\nu(\mathbf{w}) = 0$ it reduces to the special form

$$\epsilon_1(N, h, \alpha, 0) = 4\epsilon_0^2(N, h, \alpha) \quad (2.100)$$

We may now summarize the two bounds we have derived for the rate of uniform convergence:

1. In general, we have the following bound on the rate of uniform convergence:

$$P(\mathbf{w}) \leq \nu(\mathbf{w}) + \epsilon_1(N, h, \alpha, \nu)$$

where $\epsilon_1(N, h, \alpha, \nu)$ is as defined in Eq. (2.99).

2. For a small training error $\nu(\mathbf{w})$ close to zero, we have

$$P(\mathbf{w}) \leq \nu(\mathbf{w}) + 4\epsilon_0^2(N, h, \alpha)$$

which provides a fairly precise bound for real-case learning.

3. For a large training error $\nu(\mathbf{w})$ close to unity, we have the bound

$$P(\mathbf{w}) \leq \nu(\mathbf{w}) + \epsilon_0(N, h, \alpha)$$

Structural Risk Minimization

The *training error* is the frequency of errors made by a learning machine of some weight vector \mathbf{w} during the training session. Similarly, the *generalization error* is defined as the frequency of errors made by the machine when it is tested with examples not seen before. Here it is assumed that the test data are drawn from the same population as the training data. Let these two errors be denoted by $\nu_{\text{train}}(\mathbf{w})$ and $\nu_{\text{gene}}(\mathbf{w})$, respectively. Note that $\nu_{\text{train}}(\mathbf{w})$ is the *same* as the $\nu(\mathbf{w})$ used in the previous subsection; we used $\nu(\mathbf{w})$ there to simplify the notation. Let h be the VC dimension of a family of classification functions $\{F(\mathbf{x}, \mathbf{w}); \mathbf{w} \in \mathcal{W}\}$ with respect to the input space \mathcal{X} . Then, in light of the theory on the rate of uniform convergence, we may state that with probability $1 - \alpha$, for a number of training examples $N > h$, and simultaneously for all classification functions $F(\mathbf{x}, \mathbf{w})$, the generalization error $\nu_{\text{gene}}(\mathbf{w})$ is lower than a *guaranteed risk* defined by the sum of a pair of competing terms (Vapnik, 1992, 1998)

$$\nu_{\text{guarant}}(\mathbf{w}) = \nu_{\text{train}}(\mathbf{w}) + \epsilon_1(N, h, \alpha, \nu_{\text{train}}) \quad (2.101)$$

where the confidence interval $\epsilon_1(N, h, \alpha, \nu_{\text{train}})$ is itself defined by Eq. (2.99). For a fixed number of training examples N , the training error decreases monotonically as the capacity or VC dimension h is increased, whereas the confidence interval increases monotonically. Accordingly, both the guaranteed risk and the generalization error go through a minimum. These trends are illustrated in a generic way in Fig. 2.25. Before the minimum point is reached, the learning problem is *overdetermined* in the sense that the machine capacity h is too small for the amount of training detail. Beyond the mini-

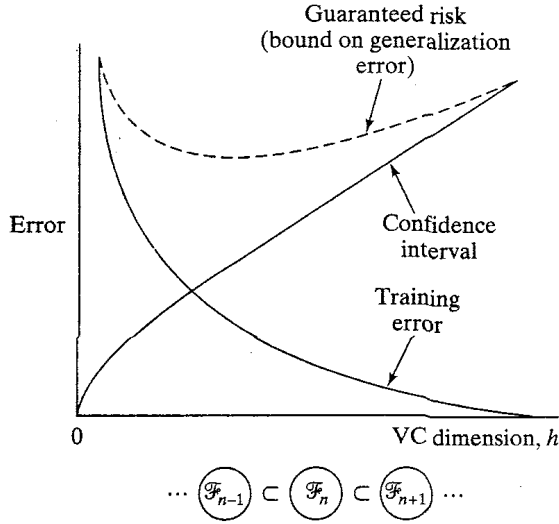


FIGURE 2.25 Illustration of the relationship between training error, confidence interval, and guaranteed risk.

mum point, the learning problem is *underdetermined* because the machine capacity is too large for the amount of training data.

The challenge in solving a supervised learning problem is therefore to realize the best generalization performance by matching the machine capacity to the available amount of training data for the problem at hand. The *method of structural risk minimization* provides an inductive procedure for achieving this goal by making the VC dimension of the learning machine a *control variable* (Vapnik, 1992, 1998). To be specific, consider an ensemble of pattern classifiers $\{F(\mathbf{x}, \mathbf{w}); \mathbf{w} \in \mathcal{W}\}$, and define a nested structure of n such machines

$$\mathcal{F}_k = \{F(\mathbf{x}, \mathbf{w}); \mathbf{w} \in \mathcal{W}_k\}, \quad k = 1, 2, \dots, n \quad (2.102)$$

such that we have (see Fig. 2.25)

$$\mathcal{F}_1 \subset \mathcal{F}_2 \subset \dots \subset \mathcal{F}_n \quad (2.103)$$

where the symbol \subset signifies “is contained in.” Correspondingly, the VC dimensions of the individual pattern classifiers satisfy the condition

$$h_1 \leq h_2 \leq \dots \leq h_n \quad (2.104)$$

which implies that the VC dimension of each pattern classifier is finite. Then, the method of structural risk minimization may proceed as follows:

- The empirical risk (i.e., training error) for each pattern classifier is minimized.
- The pattern classifier \mathcal{F}^* with the smallest guaranteed risk is identified; this particular machine provides the best compromise between the training error (i.e., quality of approximation of the training data) and the confidence interval, (i.e., complexity of the approximating function) which compete with each other.

Our goal is to find a network structure such that decreasing the VC dimension occurs at the expense of the smallest possible increase in training error.

The principle of structural risk minimization may be implemented in a variety of ways. For example, we may vary the VC dimension h by varying the number of hidden

neurons. Specifically, we evaluate an ensemble of fully connected multilayer feedforward networks, in which the number of neurons in one of the hidden layers is increased in a monotonic fashion. The principle of structural risk minimization states that the best network in this ensemble is the one for which the guaranteed risk is the minimum.

The VC dimension is not only central to the principle of structural risk minimization but also to an equally powerful learning model called probably approximately correct (PAC). This latter model, discussed in the next section, completes the last part of the chapter dealing with probabilistic and statistical aspects of learning.

2.15 PROBABLY APPROXIMATELY CORRECT MODEL OF LEARNING

The *probably approximately correct* (PAC) learning model is credited to Valiant (1984). As the name implies, the PAC model is a probabilistic framework for the study of learning and generalization in binary classification systems. It is closely related to supervised learning.

We begin with an environment \mathcal{X} . A set of \mathcal{X} is called a *concept*, and a set of subsets of \mathcal{X} is called a *concept class*. An *example* of a concept is an object in the domain of interest, together with a class label. If the example is a member of the concept, we refer to it as a *positive example*; if the object is *not* a member of the concept, we refer to it as a *negative example*. A concept for which examples are provided is called a *target concept*. We may acquire a sequence of training data of length N for a target concept c as shown by

$$\mathcal{T} = \{(\mathbf{x}_i, c(\mathbf{x}_i))\}_{i=1}^N \quad (2.105)$$

which may contain repeated examples. The examples $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$ are drawn from the environment \mathcal{X} at random, according to some fixed but unknown probability distribution. The following points are also noteworthy in Eq. (2.105):

- The target concept $c(\mathbf{x}_i)$ is treated as a function from \mathcal{X} to $\{0, 1\}$. Moreover, $c(\mathbf{x}_i)$ is assumed to be unknown.
- The examples are usually assumed to be statistically independent, which means that the joint probability density function of any two examples, \mathbf{x}_i and \mathbf{x}_j , say, is equal to the product of their individual probability density functions.

In the context of our previous terminology, the environment \mathcal{X} may be identified with the input space of a neural network, and the target concept may be identified with the desired response for the network.

The set of concepts derived from the environment \mathcal{X} is referred to as a *concept space* \mathcal{C} . For example, the concept space may contain “the letter A,” “the letter B,” and so on. Each of these concepts may be coded differently to generate a set of positive examples and a set of negative examples. In the framework of supervised learning, however, we have another set of concepts. A learning machine typically represents a set of functions, with each function corresponding to a specific state. For example, the machine may be designed to recognize “the letter A,” “the letter B,” and so on. The set of all functions (i.e., concepts) determined by the learning machine is referred to as a *hypothesis space* \mathcal{H} . The hypothesis space may or may not be equal to the concept

space. In a way the notions of concept space and hypothesis space are analogous to the function $f(\mathbf{x})$ and approximating function $F(\mathbf{x}, \mathbf{w})$, respectively, that were used in the previous section.

Suppose then we are given a target concept $c(\mathbf{x}) \in \mathcal{C}$, which takes only the value 0 or 1. We wish to learn this concept by means of a neural network by training it on the data set \mathcal{T} defined in Eq. (2.105). Let $g(\mathbf{x}) \in \mathcal{G}$ denote the hypothesis corresponding to the input–output mapping that results from this training. One way of assessing the success of the learning process is to measure how close the hypothesis $g(\mathbf{x})$ is to the target concept $c(\mathbf{x})$. There will naturally be errors incurred, making $g(\mathbf{x}) \neq c(\mathbf{x})$. The reason errors are incurred is that we are trying to learn a function on the basis of limited information available about that function. The probability of training error is defined by

$$\nu_{\text{train}} = P(\mathbf{x} \in \mathcal{X} : g(\mathbf{x}) \neq c(\mathbf{x})) \quad (2.106)$$

The probability distribution in this equation must be the same as the one responsible for generating the examples. The goal of PAC learning is to ensure that ν_{train} is *usually small*. The domain that is available to the learning algorithm is controlled by the size N of the training sample \mathcal{T} . In addition, the learning algorithm is supplied with two control parameters:

- *Error parameter* $\epsilon \in (0, 1]$. This parameter specifies the error allowed in a good approximation of the target concept $c(\mathbf{x})$ by the hypothesis $g(\mathbf{x})$.
- *Confidence parameter* $\delta \in (0, 1]$. This second parameter controls the likelihood of constructing a good approximation.

We may thus visualize the PAC learning model as depicted in Fig. 2.26.

With this background we may now formally state the PAC learning model (Valiant, 1984; Kearns and Vazirani, 1994; Vidyasagar, 1997):

Let \mathcal{C} be a concept class over the environment \mathcal{X} . The concept class \mathcal{C} is said to be PAC learnable if there exists an algorithm \mathcal{L} with the following property: For every target concept $c \in \mathcal{C}$, for every probability distribution on \mathcal{X} , and for all $0 < \epsilon < 1/2$ and $0 < \delta < 1/2$, if the learning algorithm \mathcal{L} is supplied the set of training examples $\mathcal{T} = \{(\mathbf{x}_i, c(\mathbf{x}_i))\}_{i=1}^N$ and the parameters ϵ and δ , then with probability at least $1 - \delta$, the learning algorithm \mathcal{L} outputs a hypothesis g with error $\nu_{\text{train}} \leq \epsilon$. This probability is taken over the random examples drawn from the set \mathcal{T} and any internal randomization that may exist in the learning algorithm \mathcal{L} . The sample size N must be greater than a function of δ and ϵ .

In other words, provided that the size N of the training sample \mathcal{T} is large enough, after the neural network has been trained on that data set it is “probably” the case that the

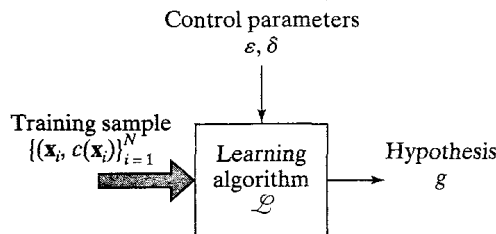


FIGURE 2.26 Block diagram illustrating the PAC learning model.

input–output mapping computed by the network is “approximately correct.” Note that although there is a dependence on δ and ϵ , the number of examples, N , does not have to be dependent on the target concept c or the underlying probability distribution of \mathcal{X} .

Sample Complexity

In PAC learning theory, an issue of particular interest with practical implications is the issue of *sample complexity*. The focus in this issue is on how many random examples should be presented to the learning algorithm for it to acquire sufficient information to learn an unknown target concept c chosen from the concept class \mathcal{C} . Or, how large should the size N of the training set \mathcal{T} be?

The issue of sample complexity is closely linked with the VC dimension. However, before proceeding further on this issue, we need to define the notion of a consistent concept. Let $\mathcal{T} = \{(\mathbf{x}_i, d_i)\}_{i=1}^N$ be any set of labeled examples, where each $\mathbf{x}_i \in \mathcal{X}$ and each $d_i \in (0,1)$. Let c be a target concept over the environment \mathcal{X} . Then, concept c is said to be *consistent* with the training set \mathcal{T} (or, equivalently, \mathcal{T} is consistent with c) if for all $1 \leq i \leq N$ we have $c(\mathbf{x}_i) = d_i$ (Kearns and Vazirani, 1994). Now, as far as PAC learning is concerned, it is *not* the size of the set of input–output functions computable by a neural network that is crucial, but rather it is the VC dimension of the network. More precisely, we have a key result presented in two parts (Blumer et al., 1989; Anthony and Biggs, 1992; Vidyasagar, 1997):

Consider a neural network with a finite VC dimension $h \geq 1$.

1. Any consistent learning algorithm for that neural network is a PAC learning algorithm.
2. There is a constant K such that a sufficient size of training set \mathcal{T} for any such algorithm is

$$N = \frac{K}{\epsilon} \left(h \log \left(\frac{1}{\epsilon} \right) + \log \left(\frac{1}{\delta} \right) \right) \quad (2.107)$$

where ϵ is the error parameter and δ is the confidence parameter.

The generality of this result is impressive: it is applicable to a supervised learning process regardless of the type of learning algorithm used, and the underlying probability distribution for generating the labeled examples. It is the broad generality of this result that has made it a subject of intensive research interest in neural network literature. Comparison of results predicted from bounds on measures based on the VC dimension with experimental results reveals a wide numerical discrepancy.¹⁶ In a sense this should not be surprising because the discrepancy is merely a reflection of the *distribution-free, worst-case* nature of the theoretical measures, and on *average* we can always do better.

Computational Complexity

Another issue of primary concern in PAC learning is that of computational complexity. This issue concerns the computational effectiveness of a learning algorithm. More precisely, *computational complexity* deals with the worst-case “running time” needed to

train a neural network (learning machine), given a set of labeled examples of some finite size N .

In a practical situation, the running time of an algorithm naturally depends on the speed with which the underlying computations are performed. From a theoretical perspective, however, the intention is to have a definition of running time that is independent of the device used to do the computations. With this intention in mind, running time, and therefore computational complexity, is usually measured in terms of the number of operations (additions, multiplications, and storage) needed to perform the computation.

In assessing the computational complexity of a learning algorithm, we like to know how it varies with the example size m (i.e., size of the input layer of the neural network being trained). For the algorithm to be computationally *efficient* in this context, the running time should be $O(m^r)$ for some fixed integer $r \geq 1$. In such a case, the running time is said to increase polynomially with m , and the algorithm itself is said to be a *polynomial time algorithm*. Learning tasks performed by a polynomial time algorithm are usually regarded as “easy” (Anthony and Biggs, 1992).

The other parameter that requires attention is the error parameter ϵ . Whereas in the case of sample complexity the parameter ϵ is fixed but arbitrary, in assessing the computational complexity of a learning algorithm we like to know how it varies with ϵ . Intuitively, we expect that as ϵ is decreased, the learning task under study would become more difficult. It follows then that some condition would have to be imposed on the time taken for the algorithm to produce a probably approximately correct output. For efficient computation, the appropriate condition is to have the running time polynomial in $1/\epsilon$.

By putting these considerations together, we may make the following formal statement on computational complexity (Anthony and Biggs, 1992):

A learning algorithm is computationally efficient with respect to error parameter ϵ , example size m , and size N of the training set if its running time is polynomial in N and if there is a value of $N_0(\delta, \epsilon)$ sufficient for PAC learning that is polynomial in both m and ϵ^{-1} .

2.16 SUMMARY AND DISCUSSION

In this chapter we discussed some important issues relating to the many facets of the learning process in the context of neural networks. In so doing we have laid down the foundations for much of the material in the rest of this book. The five learning rules, *error-correction learning*, *memory-based learning*, *Hebbian learning*, *competitive learning*, and *Boltzmann learning*, are basic to the design of neural networks. Some of these algorithms require the use of a teacher and some do not. The important point is that these rules enable us to go far beyond the reach of linear adaptive filters in both capability and universality.

In the study of supervised learning, a key provision is a “teacher” capable of supplying exact corrections to the network outputs when an error occurs as in error-correction learning; or “clamping” the free-running input and output units of the network to the environment as in Boltzmann learning. Neither of these models is possible in biological organisms, which have neither the exact reciprocal nervous connections needed for the back propagation of error corrections (in a multilayer feedforward

network) nor the nervous means for the imposition of behavior from outside. Nevertheless, supervised learning has established itself as a powerful paradigm for the design of artificial neural networks, as is demonstrated in Chapters 3 through 7.

In contrast, self-organized (unsupervised) learning rules such as Hebbian learning and competitive learning are motivated by neurobiological considerations. However, to improve our understanding of self-organized learning, we also need to look at Shannon's *information theory* for relevant ideas. Here we should mention the *maximum mutual information (Infomax) principle* due to Linsker (1988a, b), which provides the mathematical formalism for the processing of information in a self-organized neural network in a manner somewhat analogous to the transmission of information in a communication channel. The Infomax principle and its variants are discussed in Chapter 10.

A discussion of learning methods would be incomplete without mentioning the *Darwinian selective learning model* (Edelman, 1987; Reeke et al., 1990). *Selection* is a powerful biological principle with applications in both evolution and development. It is at the heart of the immune system (Edelman, 1973), the best understood biological recognition system. The Darwinian selective learning model is based on the theory of neural group selection. It presupposes that the nervous system operates by a form of selection akin to natural selection in evolution but takes place within the brain during the lifetime of each animal. According to this theory, the basic operational units of the nervous system are not single neurons but rather local groups of strongly interconnected cells. The membership of neurons in a group is changed by alterations in the neurons' synaptic weights. Local competition and cooperation among cells are clearly necessary to produce local order in the network. A collection of neuronal groups is referred to as a *repertoire*. Groups in a repertoire respond best to overlapping but similar input patterns due to the random nature of neural growth. One or more neuronal groups in a repertoire respond to every input pattern, thereby ensuring some response to unexpected input patterns that may be important. Darwinian selective learning is different from the learning algorithms commonly used in neural network design in that it assumes that there are many subnetworks by design, and that only those with the desired response are selected during the training process.

We complete this discussion with some concluding remarks on statistical and probabilistic aspects of learning. The VC dimension has established itself as a central parameter in statistical learning theory. It is basic to structural risk minimization and the probably approximately correct (PAC) model of learning. The VC dimension is an integral part of the underlying theory of so-called support vector machines, discussed in Chapter 6. In Chapter 7 we discuss a class of committee machines based on boosting, the theory of which is rooted in PAC learning.

As we progress through the rest of the book there will be many occasions and good reasons for revisiting the material presented in this chapter on the fundamentals of learning processes.

NOTES AND REFERENCES

1. The word "algorithm" is derived from the name of the Persian mathematician Mohammed al-Kowârisimi, who lived during the ninth century and who is credited with developing the step-by-step rules for the addition, subtraction, multiplication, and divi-