

model, that is, a learning model with high generalization error rate. Assume that you are dealing with a training sample of fixed size. How do boosting by filtering and AdaBoost cope with this situation?

Mixture of Experts

7.4 Consider a piecewise-linear task described by

$$F(x_1, x_2, \dots, x_{10}) = \begin{cases} 3x_2 + 2x_3 + x_4 + 3 + \epsilon & \text{if } x_1 = 1 \\ 3x_5 + 2x_6 + x_7 - 3 + \epsilon & \text{if } x_1 = -1 \end{cases}$$

For comparison, the following network configurations are used:

1. Multilayer perceptron: "10→10→1" network
2. Mixture of experts: Gating networks: 10→2;
Expert networks: 10→1

Compare the computational complexities of these two networks.

7.5 The ME model described by the conditional probability density function of Eq. (7.30) is based on a scalar regression model, in which the error is Gaussian distributed with zero mean and unit variance.

- (a) Reformulate this equation for the more general case of an ME model corresponding to a multiple regression model, in which the desired response is a vector with dimension q and the error is a multivariate Gaussian distribution with zero mean and covariance matrix Σ .
- (b) How is the ME model for this reformulation different from the ME model shown in Fig. 7.8?

7.6 Derive the stochastic gradient algorithm for the training of the mixture of experts model.

Hierarchical Mixture of Experts

- 7.7 (a) Construct the block diagram of an HME model with three levels of hierarchy. Assume the use of a binary decision tree for the model.
- (b) Write the *a posteriori* probabilities for the nonterminal nodes of the HME described in part (a). Demonstrate the recursiveness of the computations involved in evaluating these probabilities.
- (c) Formulate the conditional probability density function for the HME model described in part (a).
- 7.8 Discuss the similarities and differences between HME models and radial-basis function (RBF) networks.
- 7.9 Derive the equations that describe the stochastic gradient algorithm for the training of an HME model with two levels of hierarchy. Assume a binary decision tree for the model.

EM Algorithm and its Application to the HME Model

- 7.10 Prove the monotonic-increasing property of the EM algorithm described in Eq. (7.62). For this derivation, do the following:
 - (a) Let

$$k(\mathbf{r}|d, \boldsymbol{\theta}) = \frac{f_c(\mathbf{r}|\boldsymbol{\theta})}{f_d(d|\boldsymbol{\theta})}$$

denote the conditional probability density function of the augmented data vector $\mathbf{r}$, given the observation d and parameter vector $\boldsymbol{\theta}$. Hence, the incomplete data log-likelihood function may be expressed as

$$L(\boldsymbol{\theta}) = L_c(\boldsymbol{\theta}) - \log k(\mathbf{r}|d, \boldsymbol{\theta})$$

where $L_c(\boldsymbol{\theta}) = \log f_c(\mathbf{r}|\boldsymbol{\theta})$ is the complete data log-likelihood function. By taking the expectation of $L(\boldsymbol{\theta})$ with respect to the conditional distribution of $\mathbf{r}$, given d , show that

$$L(\boldsymbol{\theta}) = Q(\boldsymbol{\theta}, \hat{\boldsymbol{\theta}}(n)) - K(\boldsymbol{\theta}, \hat{\boldsymbol{\theta}}(n))$$

where

$$K(\boldsymbol{\theta}, \hat{\boldsymbol{\theta}}(n)) = E[\log k(\mathbf{r}|d, \hat{\boldsymbol{\theta}})]$$

Hence, show that

$$\begin{aligned} L(\hat{\boldsymbol{\theta}}(n+1)) - L(\hat{\boldsymbol{\theta}}(n)) &= [Q(\hat{\boldsymbol{\theta}}(n+1), \hat{\boldsymbol{\theta}}(n)) - Q(\hat{\boldsymbol{\theta}}(n), \hat{\boldsymbol{\theta}}(n))] \\ &\quad - [K(\hat{\boldsymbol{\theta}}(n+1), \hat{\boldsymbol{\theta}}(n)) - K(\hat{\boldsymbol{\theta}}(n), \hat{\boldsymbol{\theta}}(n))] \end{aligned}$$

- (b) *Jensen's inequality* states that if $f(\cdot)$ is a convex function and u is a random variable, then

$$E[g(u)] \geq g(E[u])$$

where E is the expectation operator; moreover, if $g(\cdot)$ is strictly convex, then equality in this relation implies that $u = E[u]$ with probability 1 (Cover and Thomas, 1991). Using Jensen's inequality, show that

$$K(\hat{\boldsymbol{\theta}}(n+1), \hat{\boldsymbol{\theta}}(n)) - K(\hat{\boldsymbol{\theta}}(n), \hat{\boldsymbol{\theta}}(n)) \leq 0$$

Hence, show that Eq. (7.62) holds for $n = 0, 1, 2, \dots$

- 7.11 The EM algorithm is easily modified to accommodate the maximum *a posteriori* (MAP) estimate of a parameter vector $\boldsymbol{\theta}$. Using Bayes' rule, modify the E-step and M-step of the EM algorithm to provide for this estimation.
- 7.12 For an HME trained with the EM algorithm and an MLP trained with the back-propagation algorithm to provide a similar level of performance for a given task, we would intuitively expect the computational complexity of the HME to exceed that of the MLP. Argue in favor of or against the plausibility of this statement.
- 7.13 Justify the relations between the indicator variables and corresponding *a posteriori* probabilities described in Eqs. (7.66) to (7.68).
- 7.14 Equation (7.75) describes the weighted least-squares for the optimization of the expert networks in the HME model of Fig. 7.11, assuming that the desired response d is scalar. How is this relation modified for the case of a multidimensional desired response?

CHAPTER 8

Principal Components Analysis

I INTRODUCTION

An important feature of neural networks is the ability they have to *learn* from their environment, and through learning to *improve* performance in some sense. In the previous four chapters, the focus was on algorithms for supervised learning, for which a set of targets of interest is provided by an external teacher. The targets take the form of a desired input–output mapping, which the network is required to approximate. In this chapter and the next three chapters, we study algorithms for *self-organized learning* or *unsupervised learning*. The purpose of an algorithm for self-organized learning is to *discover* significant patterns or features in the input data, and to do the discovery *without* a teacher. To do so, the algorithm is provided with a set of rules of a *local* nature, which enables it to learn to compute an input–output mapping with specific desirable properties; the term “local” means that the change applied to the synaptic weight of a neuron is confined to the immediate neighborhood of that neuron. The modeling of network structures used for self-organized learning tends to follow neurobiological structures to a much greater extent than for supervised learning. This may not be surprising, because the process of network organization is fundamental to the organization of the brain.

The structure of a self-organizing system may take on a variety of different forms. It may, for example, consist of an *input (source) layer* and an *output (representation) layer*, with feedforward connections from input to output and lateral connections between neurons in the output layer. Another example is a feedforward network with multiple layers, in which the self-organization proceeds on a layer-by-layer basis. In both examples, the learning process consists of repeatedly modifying the synaptic weights of all the connections in the system in response to input (activation) patterns and in accordance with prescribed rules, until a final configuration develops.

This chapter on self-organizing systems is restricted to Hebbian learning. The primary focus of the chapter is *principal components analysis*, which is a standard technique commonly used for data reduction in statistical pattern recognition and signal processing.

Organization of the Chapter

The material in this chapter is organized as follows. In Section 8.2 we use qualitative arguments to describe the basic principles of self-organization. This is followed by introductory material on principal components analysis in Section 8.3, which is also basic to the self-organizing systems discussed in the rest of the chapter.

With this basic background material on hand, we then proceed to study some specific self-organizing systems. In Section 8.4 we describe a simple model consisting of a single neuron, which extracts the first principal component in a self-organized manner. In Section 8.5 we describe a more elaborate self-organizing system in the form of a feedforward network with a single layer of neurons, which extracts all the principal components by building on the previous simple model. This procedure is illustrated by a computer experiment on image coding presented in Section 8.6. In Section 8.7 we describe another self-organizing system for a similar function; this system is even more elaborate because it also includes lateral connections.

In Section 8.8 we present a classification of algorithms for principal components analysis using neural networks. This is followed by Section 8.9 on the classification of data reduction algorithms into adaptive and batch methods.

In Section 8.10 we describe a nonlinear form of principal components analysis that builds on the idea of an inner-product kernel defined in accordance with Mercer’s theorem, which is discussed in Chapter 6 on support vector machines.

The chapter concludes in Section 8.11 with some final thoughts on principal components analysis.

8.2 SOME INTUITIVE PRINCIPLES OF SELF-ORGANIZATION

As mentioned previously, self-organized (unsupervised) learning consists of repeatedly modifying the synaptic weights of a neural network in response to activation patterns and in accordance with prescribed rules, until a final configuration develops. The key question, of course, is how a useful configuration can finally develop from self-organization. The answer lies essentially in the following observation (Turing, 1952):

Global order can arise from local interactions.

This observation is of fundamental importance; it applies to the brain and to artificial neural networks. In particular, many originally random local interactions between neighboring neurons of a network can coalesce into states of global order and ultimately lead to coherent behavior in the form of spatial patterns or temporal rhythms; these are the essence of self-organization.

Network organization takes place at two different levels that interact with each other in the form of a *feedback* loop. The two levels are:

- *Activity*. Certain activity patterns are produced by a given network in response to input signals.
- *Connectivity*. Connection strengths (synaptic weights) of the network are modified in response to neuronal signals in the activity patterns, due to synaptic plasticity.

The feedback between changes in synaptic weights and changes in activity patterns must be *positive* in order to achieve self-organization (instead of stabilization) of the network. Accordingly, we may abstract the first principle of self-organization (von der Malsburg, 1990a):

PRINCIPLE 1. Modifications in synaptic weights tend to self-amplify.

The process of self-amplification is constrained by the requirement that modifications in synaptic weights must be based on locally available signals, namely presynaptic signals and postsynaptic signals. The requirements of self-reinforcement and locality specify the mechanism whereby a strong synapse leads to coincidence of presynaptic and postsynaptic signals. In turn, the synapse is increased in strength by such a coincidence. The mechanism described here is in fact a restatement of Hebb's postulate of learning!

In order to stabilize the system there must be some form of competition for "limited" resources (e.g., number of inputs, energy resources). Specifically, an increase in the strength of some synapses in the network must be compensated for by a decrease in others. Accordingly, only the "successful" synapses can grow, while the less successful ones tend to weaken and may eventually disappear. This observation leads us to abstract the second principle of self-organization (von der Malsburg, 1990a):

PRINCIPLE 2. Limitation of resources leads to competition among synapses and therefore the selection of the most vigorously growing synapses (i.e., the fittest) at the expense of the others.

This principle is also made possible by synaptic plasticity.

For our next observation we note that a single synapse on its own cannot efficiently produce favorable events. To do so, we need cooperation among a set of synapses converging onto a particular neuron and carrying coincident signals strong enough to activate that neuron. We may therefore abstract the third principle of self-organization (von der Malsburg, 1990a):

PRINCIPLE 3. Modifications in synaptic weights tend to cooperate.

The presence of a vigorous synapse can enhance the fitness of other synapses, in spite of the overall competition in the network. This form of cooperation may arise due to synaptic plasticity, or due to simultaneous stimulation of presynaptic neurons brought on by the existence of the right conditions in the external environment.

All three principles of self-organization described thus far relate only to the neural network itself. However, for self-organized learning to perform a useful information-processing function, there must be *redundancy* in the activation patterns supplied to the network by the environment. The issue of redundancy is discussed in Shannon's framework of information theory in Chapter 10. For now it suffices to postulate the last principle of self-organized learning as follows (Barlow, 1989):

PRINCIPLE 4. Order and structure in the activation patterns represent redundant information that is acquired by the neural network in the form of knowledge, which is a necessary prerequisite to self-organized learning.

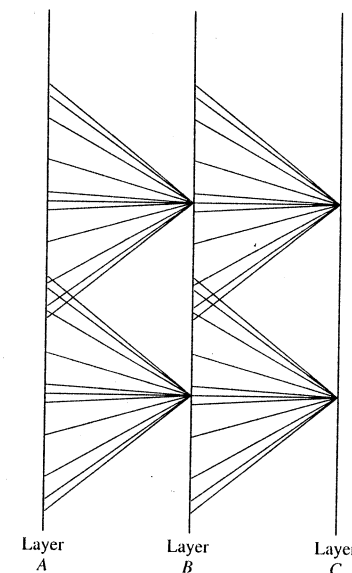


FIGURE 8.1 Layout of modular self-adaptive network.

Some of this knowledge may be obtained by observations of statistical parameters such as the mean, variance, and correlation matrix of the input data.

Principles 1 through 4 on self-organized learning provide the neurobiological basis for the adaptive algorithms for principal components analysis described in this chapter and for Kohonen's self-organizing map presented in the next chapter. These principles are also incorporated in many other self-organized models that are motivated by neurobiological considerations. One such model that deserves to be mentioned is *Linsker's model* of the mammalian visual system (Linsker, 1986).

Self-Organized Feature Analysis

Processing information in the visual system is performed in stages. In particular, simple features such as contrast and edge orientation are analyzed in the early stages of the system, whereas more elaborate complex features are analyzed in later stages. Figure 8.1 shows the gross structure of a modular network that resembles the visual system. In Linsker's model, the neurons of the network in Fig. 8.1 are organized into two-dimensional layers, with local forward connections from one layer to the next. Each neuron receives information from a limited number of neurons located in an overlying region of the previous layer, which constitutes the *receptive field* of that neuron. The receptive fields of the network play a crucial role in the synaptic development process because they make it possible for neurons in one layer to respond to *spatial correlations* of the neuronal activities in the previous layer. Two assumptions of a structural nature are made:

1. The positions of the synaptic connections are fixed for the entire neuronal development process once they have been chosen.
2. Each neuron acts as a linear combiner.

The model combines aspects of Hebb-like synaptic modification with cooperative and competitive learning in such a way that the network's outputs optimally discriminate among an ensemble of inputs, with the self-organized learning proceeding on a *layer-by-layer basis*. That is, the learning process permits the self-organized feature-analyzing properties of each layer to develop fully before proceeding to the next layer. In Linsker (1986), simulation results are presented that are qualitatively similar to properties found in the early stages of visual processing in cats and monkeys. Recognizing the highly complex nature of the visual system, it is indeed remarkable that the simple model considered by Linsker is capable of developing similar feature-analyzing neurons. The point is not to imply that feature-analyzing neurons in the mammalian visual system develop in exactly the manner described in Linsker's model. Rather, such structures may be produced by a relatively simple layered network whose synaptic connections develop in accordance with a Hebbian form of learning.

Our primary interest in this chapter, however, is in principal components analysis and how it can be performed using self-organizing systems based on Hebbian learning.

3.3 PRINCIPAL COMPONENTS ANALYSIS

A common problem in statistical pattern recognition is that of feature selection or feature extraction. *Feature selection* refers to a process whereby a *data space* is transformed into a *feature space* that, in theory, has exactly the same dimension as the original data space. However, the transformation is designed in such a way that the data set may be represented by a reduced number of "effective" features and yet retain most of the intrinsic information content of the data; in other words, the data set undergoes a *dimensionality reduction*. To be specific, suppose we have an m -dimensional vector $\mathbf{x}$ and wish to transmit it using l numbers, where $l < m$. If we simply truncate the vector $\mathbf{x}$, we will cause a mean-square error equal to the sum of the variances of the elements eliminated from $\mathbf{x}$. So we ask the following question: Does there exist an invertible *linear* transformation $\mathbf{T}$ such that the truncation of $\mathbf{T}\mathbf{x}$ is optimum in the mean-squared error sense? Clearly, the transformation $\mathbf{T}$ should have the property that some of its components have low variance. *Principal components analysis* (also known as the *Karhunen-Loève transformation* in communication theory) maximizes the rate of decrease of variance and is therefore the right choice. In this chapter we derive Hebbian-based learning algorithms that can perform principal components analysis¹ on a data vector of interest.

Let $\mathbf{X}$ denote an m -dimensional *random vector* representing the environment of interest. We assume that the random vector $\mathbf{X}$ has zero mean:

$$E[\mathbf{X}] = \mathbf{0}$$

where E is the statistical expectation operator. If $\mathbf{X}$ has a nonzero mean, we subtract the mean from it before proceeding with the analysis. Let $\mathbf{q}$ denote a *unit vector*, also of

dimension m , onto which the vector $\mathbf{X}$ is to be *projected*. This projection is defined by the inner product of the vectors $\mathbf{X}$ and $\mathbf{q}$, as shown by

$$A = \mathbf{X}^T \mathbf{q} = \mathbf{q}^T \mathbf{X} \quad (8.1)$$

subject to the constraint

$$\|\mathbf{q}\| = (\mathbf{q}^T \mathbf{q})^{1/2} = 1 \quad (8.2)$$

The projection A is a *random variable* with a mean and variance related to the statistics of the random vector $\mathbf{X}$. Under the assumption that the random vector $\mathbf{X}$ has zero mean, it follows that the mean value of the projection A is zero too:

$$E[A] = \mathbf{q}^T E[\mathbf{X}] = 0$$

The variance of A is therefore the same as its mean-square value, and so we may write

$$\begin{aligned} \sigma^2 &= E[A^2] \\ &= E[(\mathbf{q}^T \mathbf{X})(\mathbf{X}^T \mathbf{q})] \\ &= \mathbf{q}^T E[\mathbf{X}\mathbf{X}^T] \mathbf{q} \\ &= \mathbf{q}^T \mathbf{R} \mathbf{q} \end{aligned} \quad (8.3)$$

The m -by- m matrix $\mathbf{R}$ is the *correlation matrix* of the random vector $\mathbf{X}$, formally defined as the expectation of the outer product of the vector $\mathbf{X}$ with itself, as shown by

$$\mathbf{R} = E[\mathbf{X}\mathbf{X}^T] \quad (8.4)$$

We observe that the correlation matrix $\mathbf{R}$ is *symmetric*, which means that

$$\mathbf{R}^T = \mathbf{R} \quad (8.5)$$

From this property it follows that if $\mathbf{a}$ and $\mathbf{b}$ are any m -by-1 vectors, then

$$\mathbf{a}^T \mathbf{R} \mathbf{b} = \mathbf{b}^T \mathbf{R} \mathbf{a} \quad (8.6)$$

From Eq. (8.3) we see that the variance σ^2 of the projection A is a function of the unit vector $\mathbf{q}$; we may thus write

$$\begin{aligned} \psi(\mathbf{q}) &= \sigma^2 \\ &= \mathbf{q}^T \mathbf{R} \mathbf{q} \end{aligned} \quad (8.7)$$

on the basis of which we may think of $\psi(\mathbf{q})$ as a *variance probe*.

Eigenstructure of Principal Components Analysis

The next issue to be considered is that of finding those unit vectors $\mathbf{q}$ along which $\psi(\mathbf{q})$ has *extremal* or *stationary* values (local maxima or minima), subject to a constraint on the Euclidean norm of $\mathbf{q}$. The solution to this problem lies in the eigenstructure of the correlation matrix $\mathbf{R}$. If $\mathbf{q}$ is a unit vector such that the variance probe $\psi(\mathbf{q})$ has an extremal value, then for any small perturbation $\delta\mathbf{q}$ of the unit vector $\mathbf{q}$, we find that, to a first order in $\delta\mathbf{q}$,

$$\psi(\mathbf{q} + \delta\mathbf{q}) = \psi(\mathbf{q}) \quad (8.8)$$

Now, from the definition of the variance probe given in Eq. (8.7), we have

$$\begin{aligned}\psi(\mathbf{q} + \delta\mathbf{q}) &= (\mathbf{q} + \delta\mathbf{q})^T \mathbf{R} (\mathbf{q} + \delta\mathbf{q}) \\ &= \mathbf{q}^T \mathbf{R} \mathbf{q} + 2(\delta\mathbf{q})^T \mathbf{R} \mathbf{q} + (\delta\mathbf{q})^T \mathbf{R} \delta\mathbf{q}\end{aligned}$$

where in the second line, we have made use of Eq. (8.6). Ignoring the second-order term $(\delta\mathbf{q})^T \mathbf{R} \delta\mathbf{q}$ and invoking the definition of Eq. (8.7), we may therefore write

$$\begin{aligned}\psi(\mathbf{q} + \delta\mathbf{q}) &= \mathbf{q}^T \mathbf{R} \mathbf{q} + 2(\delta\mathbf{q})^T \mathbf{R} \mathbf{q} \\ &= \psi(\mathbf{q}) + 2(\delta\mathbf{q})^T \mathbf{R} \mathbf{q}\end{aligned}\quad (8.9)$$

Hence, the use of Eq. (8.8) in (8.9) implies that

$$(\delta\mathbf{q})^T \mathbf{R} \mathbf{q} = 0 \quad (8.10)$$

Just any perturbations $\delta\mathbf{q}$ of $\mathbf{q}$ are not admissible; rather, we are restricted to use only those perturbations for which the Euclidean norm of the perturbed vector $\mathbf{q} + \delta\mathbf{q}$ remains equal to unity; that is

$$\|\mathbf{q} + \delta\mathbf{q}\| = 1$$

or equivalently,

$$(\mathbf{q} + \delta\mathbf{q})^T (\mathbf{q} + \delta\mathbf{q}) = 1$$

Hence, in light of Eq. (8.2), we require that to a first order in $\delta\mathbf{q}$,

$$(\delta\mathbf{q})^T \mathbf{q} = 0 \quad (8.11)$$

This means that the perturbations $\delta\mathbf{q}$ must be orthogonal to $\mathbf{q}$, and therefore only a change in the direction of $\mathbf{q}$ is permitted.

By convention, the elements of the unit vector $\mathbf{q}$ are dimensionless in a physical sense. If, therefore, we are to combine Eqs. (8.10) and (8.11), we must introduce a scaling factor λ into the latter equation with the same dimensions as the entries in the correlation matrix $\mathbf{R}$. We may then write

$$(\delta\mathbf{q})^T \mathbf{R} \mathbf{q} - \lambda (\delta\mathbf{q})^T \mathbf{q} = 0$$

or equivalently,

$$(\delta\mathbf{q})^T (\mathbf{R} \mathbf{q} - \lambda \mathbf{q}) = 0 \quad (8.12)$$

For the condition of Eq. (8.12) to hold, it is necessary and sufficient to have

$$\mathbf{R} \mathbf{q} = \lambda \mathbf{q} \quad (8.13)$$

This is the equation that governs the unit vectors $\mathbf{q}$ for which the variance probe $\psi(\mathbf{q})$ has extremal values.

Equation (8.13) is recognized as the *eigenvalue problem*, commonly encountered in linear algebra (Strang, 1980). The problem has nontrivial solutions (i.e., $\mathbf{q} \neq \mathbf{0}$) only for special values of λ that are called the *eigenvalues* of the correlation matrix $\mathbf{R}$. The associated values of $\mathbf{q}$ are called *eigenvectors*. A correlation matrix is characterized by real, nonnegative eigenvalues. The associated eigenvectors are unique, assuming that the eigenvalues are distinct. Let the eigenvalues of the m -by- m matrix $\mathbf{R}$ be denoted by

$\lambda_1, \lambda_2, \dots, \lambda_m$, and the associated eigenvectors be denoted by $\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_m$, respectively. We may then write

$$\mathbf{R} \mathbf{q}_j = \lambda_j \mathbf{q}_j, \quad j = 1, 2, \dots, m \quad (8.14)$$

Let the corresponding eigenvalues be arranged in decreasing order:

$$\lambda_1 > \lambda_2 > \dots > \lambda_j > \dots > \lambda_m \quad (8.15)$$

so that $\lambda_1 = \lambda_{\max}$. Let the associated eigenvectors be used to construct an m -by- m matrix:

$$\mathbf{Q} = [\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_j, \dots, \mathbf{q}_m] \quad (8.16)$$

We may then combine the set of m equations represented in (8.14) into a single equation:

$$\mathbf{R} \mathbf{Q} = \mathbf{Q} \mathbf{\Lambda} \quad (8.17)$$

where $\mathbf{\Lambda}$ is a diagonal matrix defined by the eigenvalues of matrix $\mathbf{R}$:

$$\mathbf{\Lambda} = \text{diag}[\lambda_1, \lambda_2, \dots, \lambda_j, \dots, \lambda_m] \quad (8.18)$$

The matrix $\mathbf{Q}$ is an *orthogonal (unitary) matrix* in the sense that its column vectors (i.e., the eigenvectors of $\mathbf{R}$) satisfy the *conditions of orthonormality*:

$$\mathbf{q}_i^T \mathbf{q}_j = \begin{cases} 1, & j = i \\ 0, & j \neq i \end{cases} \quad (8.19)$$

Equation (8.19) requires distinct eigenvalues. Equivalently, we may write

$$\mathbf{Q}^T \mathbf{Q} = \mathbf{I}$$

from which we deduce that the inverse of matrix $\mathbf{Q}$ is the same as its transpose, as shown by

$$\mathbf{Q}^T = \mathbf{Q}^{-1} \quad (8.20)$$

This means that we may rewrite Eq. (8.17) in a form known as the *orthogonal similarity transformation*:

$$\mathbf{Q}^T \mathbf{R} \mathbf{Q} = \mathbf{\Lambda} \quad (8.21)$$

or in expanded form,

$$\mathbf{q}_j^T \mathbf{R} \mathbf{q}_k = \begin{cases} \lambda_j, & k = j \\ 0, & k \neq j \end{cases} \quad (8.22)$$

The orthogonal similarity (unitary) transformation of Eq. (8.21) transforms the correlation matrix $\mathbf{R}$ into a diagonal matrix of eigenvalues. The correlation matrix $\mathbf{R}$ may itself be expressed in terms of its eigenvalues and eigenvectors as:

$$\mathbf{R} = \sum_{i=1}^m \lambda_i \mathbf{q}_i \mathbf{q}_i^T \quad (8.23)$$

which is referred to as the *spectral theorem*. The outer product $\mathbf{q}_i \mathbf{q}_i^T$ is of *rank 1* for all i .

Equations (8.21) and (8.23) are two equivalent representations of the *eigendecomposition* of the correlation matrix $\mathbf{R}$.

Principal components analysis and eigendecomposition of matrix $\mathbf{R}$ are basically one and the same, just viewing the problem in different ways. This equivalence follows from Eqs. (8.7) and (8.23) where we see that the variance probes and eigenvalues are indeed equal, as shown by

$$\psi(\mathbf{q}_j) = \lambda_j, \quad j = 1, 2, \dots, m \quad (8.24)$$

We may now summarize the two important findings we have made from the eigenstructure of principal components analysis:

- The eigenvectors of the correlation matrix $\mathbf{R}$ pertaining to the zero-mean random vector $\mathbf{X}$ define the unit vectors $\mathbf{q}_j$, representing the principal directions along which the variance probes $\psi(\mathbf{q}_j)$ have their extremal values.
- The associated eigenvalues define the extremal values of the variance probes $\psi(\mathbf{u}_j)$.

Basic Data Representations

Let the *data vector* $\mathbf{x}$ denote a realization of the random vector $\mathbf{X}$.

With m possible solutions for the unit vector $\mathbf{q}$, we find that there are m possible projections of the data vector $\mathbf{x}$ to be considered. Specifically, from Eq. (8.1) we note that

$$a_j = \mathbf{q}_j^T \mathbf{x} = \mathbf{x}^T \mathbf{q}_j, \quad j = 1, 2, \dots, m \quad (8.25)$$

where the a_j are the projections of $\mathbf{x}$ onto the principal directions represented by the unit vectors $\mathbf{u}_j$. The a_j are called the *principal components*; they have the same physical dimensions as the data vector $\mathbf{x}$. The formula in Eq. (8.25) may be viewed as one of *analysis*.

To reconstruct the original data vector $\mathbf{x}$ exactly from the projections a_j , we proceed as follows. First, we combine the set of projections $\{a_j | j = 1, 2, \dots, m\}$ into a single vector, as shown by

$$\begin{aligned} \mathbf{a} &= [a_1, a_2, \dots, a_m]^T \\ &= [\mathbf{x}^T \mathbf{q}_1, \mathbf{x}^T \mathbf{q}_2, \dots, \mathbf{x}^T \mathbf{q}_m]^T \\ &= \mathbf{Q}^T \mathbf{x} \end{aligned} \quad (8.26)$$

Next, we premultiply both sides of Eq. (8.26) by the matrix $\mathbf{Q}$, and then use the relation of Eq. (8.20). Accordingly, the original data vector $\mathbf{x}$ may be reconstructed as follows:

$$\begin{aligned} \mathbf{x} &= \mathbf{Q} \mathbf{a} \\ &= \sum_{j=1}^m a_j \mathbf{q}_j \end{aligned} \quad (8.27)$$

which may be viewed as the formula for *synthesis*. In this sense, the unit vectors $\mathbf{q}_j$ represent a *basis* of the data space. Indeed, Eq. (8.27) is nothing but a coordinate transformation, according to which a point $\mathbf{x}$ in the data space is transformed into a corresponding point $\mathbf{a}$ in the feature space.

Dimensionality Reduction

From the perspective of statistical pattern recognition, the practical value of principal components analysis is that it provides an effective technique for *dimensionality reduction*. In particular, we may reduce the number of features needed for effective data representation by discarding those linear combinations in Eq. (8.27) that have small variances and retain only those terms that have large variances. Let $\lambda_1, \lambda_2, \dots, \lambda_l$ denote the largest l eigenvalues of the correlation matrix $\mathbf{R}$. We may then approximate the data vector $\mathbf{x}$ by *truncating* the expansion of Eq. (8.27) after l terms as follows:

$$\begin{aligned} \hat{\mathbf{x}} &= \sum_{j=1}^l a_j \mathbf{q}_j \\ &= [\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_l] \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_l \end{bmatrix}, \quad l \leq m \end{aligned} \quad (8.28)$$

Given the original data vector $\mathbf{x}$, we may use Eq. (8.25) to compute the set of principal components retained in Eq. (8.28) as follows:

$$\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_l \end{bmatrix} = \begin{bmatrix} \mathbf{q}_1^T \\ \mathbf{q}_2^T \\ \vdots \\ \mathbf{q}_l^T \end{bmatrix} \mathbf{x}, \quad l \leq m \quad (8.29)$$

The linear projection of Eq. (8.29) from $\mathbb{R}^m$ to $\mathbb{R}^l$ (i.e., the mapping from the data space to the feature space) represents an *encoder* for the approximate representation of the data vector $\mathbf{x}$ as illustrated in Fig. 8.2a. Correspondingly, the linear projection of Eq. (8.28)

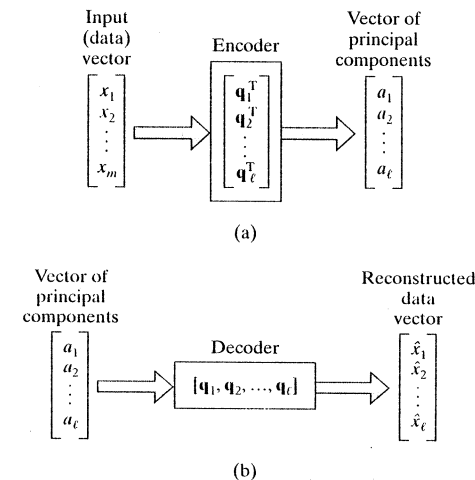


FIGURE 8.2 Illustration of two phases of principal components analysis: (a) Encoding. (b) Decoding.

from $\mathbb{R}^l$ to $\mathbb{R}^m$ (i.e., the mapping from the feature space back to the data space) represents a *decoder* for the approximate reconstruction of the original data vector $\mathbf{x}$, as illustrated in Fig. 8.2b. Note that the *dominant* (i.e., largest) eigenvalues $\lambda_1, \lambda_2, \dots, \lambda_l$ do not enter the computations described in Eqs. (8.28) and (8.29); they merely determine the number of principal components used for encoding and decoding, respectively.

The *approximation error vector* $\mathbf{e}$ equals the difference between the original data vector $\mathbf{x}$ and the approximating data vector $\hat{\mathbf{x}}$, as shown by

$$\mathbf{e} = \mathbf{x} - \hat{\mathbf{x}} \quad (8.30)$$

Substituting Eqs. (8.27) and (8.28) in (8.30) yields

$$\mathbf{e} = \sum_{j=l+1}^m a_j \mathbf{q}_j \quad (8.31)$$

The error vector $\mathbf{e}$ is orthogonal to the approximating data vector $\hat{\mathbf{x}}$, as illustrated in Fig. 8.3. In other words, the inner product of the vectors $\hat{\mathbf{x}}$ and $\mathbf{e}$ is zero. This property is shown by using Eqs. (8.28) and (8.31) as follows:

$$\begin{aligned} \mathbf{e}^T \hat{\mathbf{x}} &= \sum_{i=l+1}^m a_i \mathbf{q}_i^T \sum_{j=1}^l a_j \mathbf{q}_j \\ &= \sum_{i=l+1}^m \sum_{j=1}^l a_i a_j \mathbf{q}_i^T \mathbf{q}_j \\ &= 0 \end{aligned} \quad (8.32)$$

where we have made use of the second condition in Eq. (8.19). Equation (8.32) is known as the *principle of orthogonality*.

The total variance of the m components of the data vector $\mathbf{x}$ is, via Eq. (8.7) and the first line of Eq. (8.22),

$$\sum_{j=1}^m \sigma_j^2 = \sum_{j=1}^m \lambda_j \quad (8.33)$$

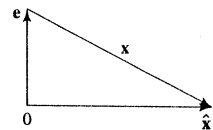
where σ_j^2 is the variance of the j th principal component a_j . The total variance of the l elements of the approximating vector $\hat{\mathbf{x}}$ is

$$\sum_{j=1}^l \sigma_j^2 = \sum_{j=1}^l \lambda_j \quad (8.34)$$

The total variance of the $(l - m)$ elements in the approximation error vector $\mathbf{x} - \hat{\mathbf{x}}$ is therefore

$$\sum_{j=l+1}^m \sigma_j^2 = \sum_{j=l+1}^m \lambda_j \quad (8.35)$$

FIGURE 8.3 Illustration of the relationship between vector $\mathbf{x}$, its reconstructed version $\hat{\mathbf{x}}$, and error vector $\mathbf{e}$.



The eigenvalues $\lambda_{l+1}, \dots, \lambda_m$ are the *smallest* $(m - l)$ eigenvalues of the correlation matrix $\mathbf{R}$; they correspond to the terms discarded from the expansion of Eq. (8.28) used to construct the approximating vector $\hat{\mathbf{x}}$. The closer all these eigenvalues are to zero, the more effective the dimensionality reduction (resulting from the application of the principal components analysis to the data vector $\mathbf{x}$) will be in preserving the information content of the original input data. Thus, to perform dimensionality reduction on some input data, we *compute the eigenvalues and eigenvectors of the correlation matrix of the input data vector, and then project the data orthogonally onto the subspace spanned by the eigenvectors belonging to the dominant eigenvalues*. This method of data representation is commonly referred to as *subspace decomposition* (Oja, 1983).

Example 8.1 Bivariate Data Set

To illustrate the application of principal components analysis, consider the example of a bivariate (two-dimensional) data set depicted in Fig. 8.4, where it is assumed that both feature axes are approximately of the same scale. The horizontal and vertical axes of the diagram represent the natural coordinates of the data set. The rotated axes labeled 1 and 2 result from the application of principal components analysis to this data set. From Fig. 8.4 we see that projecting the data set onto axis 1 captures the salient feature of the data, namely the fact that the data set is bimodal (i.e., there are two clusters in its structure). Indeed, the variance of the projections of the data points onto axis 1 is greater than that for any other projection axis in the figure. By contrast, the

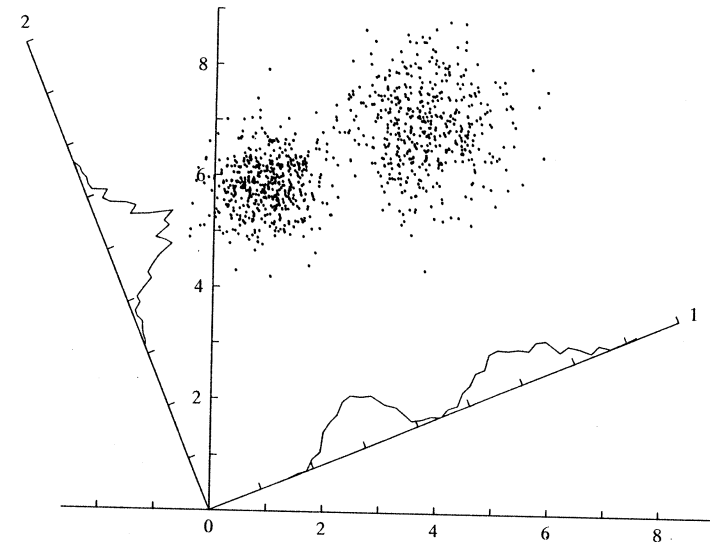


FIGURE 8.4 A cloud of data points is shown in two dimensions, and the density plots formed by projecting this cloud onto each of two axes, 1 and 2, are indicated. The projection onto axis 1 has maximum variance, and clearly shows the bimodal, or clustered character of the data.

inherent bimodal nature of the data set is completely obscured when it is projected onto the orthogonal axis 2.

The important point to note from this simple example is that although the cluster structure of the data set is evident from the two-dimensional plot of the raw data displayed in the framework of the horizontal and vertical axes, this is not always the case in practice. In the more general case of high-dimensional data sets, it is quite conceivable to have the intrinsic cluster structure of the data concealed, and to see it we must perform a statistical analysis similar to principal components analysis (Linsker, 1988a).

HEBBIAN-BASED MAXIMUM EIGENFILTER

There is a close correspondence between the behavior of self-organized neural networks and the statistical method of principal components analysis. In this section we demonstrate this correspondence by establishing a remarkable result: A single linear neuron with a Hebbian-type adaptation rule for its synaptic weights can evolve into a filter for the first principal component of the input distribution (Oja, 1982).

To proceed with the demonstration, consider the simple neuronal model depicted in Fig. 8.5a. The model is *linear* in the sense that the model output is a linear combination of its inputs. The neuron receives a set of m input signals $x_1, x_2, \dots, x_m$ through a corresponding set of m synapses with weights $w_1, w_2, \dots, w_m$, respectively. The resulting model output y is thus defined by

$$y = \sum_{i=1}^m w_i x_i \quad (8.36)$$

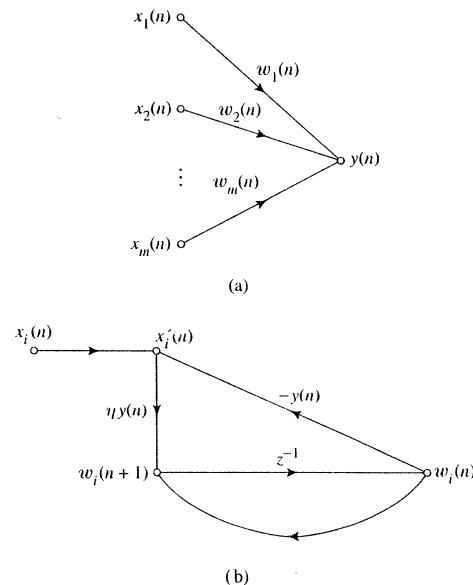


FIGURE 8.5 Signal-flow graph representation of maximum eigenfilter. (a) Graph of Eq. (8.36). (b) Graph of Eqs. (8.41) and (8.42).

Note that in the situation described here we have a single neuron to deal with, so there is no need to use double subscripts to identify the synaptic weights of the network.

In accordance with Hebb's postulate of learning, a synaptic weight w_i varies with time, growing strong when the presynaptic signal x_i and postsynaptic signal y coincide with each other. Specifically, we write

$$w_i(n+1) = w_i(n) + \eta y(n)x_i(n), \quad i = 1, 2, \dots, m \quad (8.37)$$

where n denotes discrete time and η is the *learning-rate parameter*. However, this learning rule in its basic form leads to unlimited growth of the synaptic weight w_i , which is unacceptable on physical grounds. We may overcome this problem by incorporating some form of *saturation or normalization* in the learning rule for the adaptation of synaptic weights. The use of normalization has the effect of introducing competition among the synapses of the neuron over limited resources, which, from Principle 2 of self-organization, is essential for stabilization. From a mathematical point of view, a convenient form of normalization is described by (Oja, 1982):

$$w_i(n+1) = \frac{w_i(n) + \eta y(n)x_i(n)}{(\sum_{i=1}^m [w_i(n) + \eta y(n)x_i(n)]^2)^{1/2}} \quad (8.38)$$

where the summation in the denominator extends over the complete set of synapses associated with the neuron. Assuming that the learning-rate parameter η is small, we may expand Eq. (8.38) as a power series in η , and so write

$$w_i(n+1) = w_i(n) + \eta y(n)[x_i(n) - y(n)w_i(n)] + O(\eta^2) \quad (8.39)$$

where the term $O(\eta^2)$ represents second- and higher-order effects in η . For small η , we may justifiably ignore this term, and therefore approximate Eq. (8.38) to first order in η as follows:

$$w_i(n+1) = w_i(n) + \eta y(n)[x_i(n) - y(n)w_i(n)] \quad (8.40)$$

The term $y(n)x_i(n)$ on the right-hand side of Eq. (8.40) represents the usual Hebbian modifications to synaptic weight w_i , and therefore accounts for the self-amplification effect dictated by Principle 1 of self-organization. The inclusion of the negative term $-y(n)w_i(n)$ is responsible for stabilization in accordance with Principle 2; it modifies the input $x_i(n)$ into a form that is dependent on the associated synaptic weight $w_i(n)$ and the output $y(n)$, as shown by

$$x'_i(n) = x_i(n) - y(n)w_i(n) \quad (8.41)$$

which may be viewed as the *effective input* of the i th synapse. We may now use the definition given in Eq. (8.41) to rewrite the learning rule of Eq. (8.40) as follows:

$$w_i(n+1) = w_i(n) + \eta y(n)x'_i(n) \quad (8.42)$$

The overall operation of the neuron is represented by a combination of two signal-flow graphs, as shown in Fig. 8.5. The signal-flow graph of Fig. 8.5a shows the dependence of the output $y(n)$ on the weights $w_1(n), w_2(n), \dots, w_m(n)$, in accordance with Eq. (8.36). The signal-flow graph of Fig. 8.5b provides a portrayal of Eqs. (8.41) and (8.42); the transmittance z^{-1} in the middle portion of the graph represents a unit-delay operator. The output signal $y(n)$ produced in Fig. 8.5a acts as a transmittance in Fig. 8.5b. The graph of

Fig. 8.5b clearly exhibits the following two forms of internal feedback acting on the neuron:

- Positive feedback for self-amplification and therefore growth of the synaptic weight $w_i(n)$, according to its external input $x_i(n)$.
- Negative feedback due to $-y(n)$ for controlling the growth, thereby resulting in stabilization of the synaptic weight $w_i(n)$.

The product term $-y(n)w_i(n)$ is related to a *forgetting* or *leakage factor* that is frequently used in learning rules, but with a difference: The forgetting factor becomes more pronounced with a stronger response $y(n)$. This kind of control appears to have neurobiological support (Stent, 1973).

Matrix Formulation of the Algorithm

For convenience of presentation, let

$$\mathbf{x}(n) = [x_1(n), x_2(n), \dots, x_m(n)]^T \quad (8.43)$$

and

$$\mathbf{w}(n) = [w_1(n), w_2(n), \dots, w_m(n)]^T \quad (8.44)$$

The input vector $\mathbf{x}(n)$ and the synaptic weight vector $\mathbf{w}(n)$ are typically both realizations of random vectors. Using this vector notation we may rewrite Eq. (8.36) in the form of an inner product as follows:

$$y(n) = \mathbf{x}^T(n)\mathbf{w}(n) = \mathbf{w}^T(n)\mathbf{x}(n) \quad (8.45)$$

Similarly, we may rewrite Eq. (8.40) as

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta y(n)[\mathbf{x}(n) - y(n)\mathbf{w}(n)] \quad (8.46)$$

Hence, substituting Eq. (8.45) in (8.46) yields

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta[\mathbf{x}(n)\mathbf{x}^T(n)\mathbf{w}(n) - \mathbf{w}^T(n)\mathbf{x}(n)\mathbf{x}^T(n)\mathbf{w}(n)] \quad (8.47)$$

The learning algorithm of Eq. (8.47) represents a *nonlinear stochastic difference equation*, which makes convergence analysis of the algorithm mathematically difficult. To pave the way for this convergence analysis, we will digress briefly to introduce a general tool for convergence analysis of stochastic approximation algorithms.

Asymptotic Stability Theorem

The self-organized learning algorithm of Eq. (8.47) is a special case of the generic stochastic approximation algorithm

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta(n)h(\mathbf{w}(n), \mathbf{x}(n)), \quad n = 0, 1, 2, \dots, \quad (8.48)$$

The sequence $\eta(\cdot)$ is assumed to be a sequence of positive scalars.

The *update function* $h(\cdot, \cdot)$ is a deterministic function with some regularity conditions imposed on it. This function, together with the scalar sequence $\eta(\cdot)$, specify the complete structure of the algorithm.

The goal of the procedure described here is to associate a *deterministic ordinary differential equation (ODE)* with the stochastic nonlinear difference equation (8.48). The stability properties of the differential equation are then tied to the convergence properties of the algorithm. This procedure is a fairly general tool and has wide applicability. It was developed independently by Ljung (1977) and by Kushner and Clark (1978), who used different approaches.²

To begin with, the procedure assumes that the stochastic approximation algorithm described by Eq. (8.48) satisfies the following set of conditions, using our terminology:

1. The $\eta(n)$ is a decreasing sequence of positive real numbers, such that we have:

$$(a) \quad \sum_{n=1}^{\infty} \eta(n) = \infty \quad (8.49)$$

$$(b) \quad \sum_{n=1}^{\infty} \eta^p(n) < \infty \quad \text{for } p > 1 \quad (8.50)$$

$$(c) \quad \eta(n) \rightarrow 0 \quad \text{as } n \rightarrow \infty \quad (8.51)$$

2. The sequence of parameter (synaptic weight) vectors $\mathbf{w}(\cdot)$ is bounded with probability 1.
3. The update function $h(\mathbf{w}, \mathbf{x})$ is continuously differentiable with respect to $\mathbf{w}$ and $\mathbf{x}$, and its derivatives are bounded in time.
4. The limit

$$\bar{h}(\mathbf{w}) = \lim_{n \rightarrow \infty} E[h(\mathbf{w}, \mathbf{X})] \quad (8.52)$$

exists for each $\mathbf{w}$; the statistical expectation operator E is over the random vector $\mathbf{X}$ with a realization denoted by $\mathbf{x}$.

5. There is a locally asymptotically stable (in the sense of Lyapunov) solution to the ordinary differential equation

$$\frac{d}{dt} \mathbf{w}(t) = \bar{h}(\mathbf{w}(t)) \quad (8.53)$$

where t denotes continuous time; stability in the sense of Lyapunov is discussed in Chapter 14.

6. Let $\mathbf{q}_1$ denote the solution to Eq. (8.53) with a basin of attraction $\mathcal{B}(\mathbf{q}_1)$; basin of attraction is defined in Chapter 14. Then the parameter vector $\mathbf{w}(n)$ enters a compact subset $\mathcal{A}$ of the basin of attraction $\mathcal{B}(\mathbf{q}_1)$ infinitely often, with probability 1.

The six conditions described here are all reasonable. In particular, condition 1(a) is a necessary condition that makes it possible for the algorithm to move the estimate to a desired limit, regardless of the initial conditions. Condition 1(b) gives a condition on how fast $\eta(n)$ must tend to zero; it is considerably less restrictive than the usual condition

$$\sum_{n=1}^{\infty} \eta^2(n) < \infty$$

Condition 4 is the basic assumption that makes it possible to associate a differential equation with the algorithm of Eq. (8.48).

Consider then, a stochastic approximation algorithm described by the recursive equation (8.48), subject to assumptions 1 through 6. We may then state the *asymptotic stability theorem* for this class of stochastic approximation algorithms as follows (Ljung, 1977; Kushner and Clark, 1978):

$$\lim_{n \rightarrow \infty} \mathbf{w}(n) = \mathbf{q}_1, \quad \text{infinitely often with probability 1} \quad (8.54)$$

We emphasize, however, that although the procedure described here can provide us with information about asymptotic properties of the algorithm (8.48), it usually does not tell us how large the number of iterations n has to be for the results of the analysis to be applicable. Moreover, in tracking problems where a time-varying parameter vector is to be tracked using algorithm (8.48), it is not feasible to require

$$\eta(n) \rightarrow 0 \quad \text{as } n \rightarrow \infty$$

as stipulated by condition 1(c). We may overcome this latter difficulty by assigning some small, positive value to η , the size of which usually depends on the application of interest. This is usually done in the practical use of stochastic approximation algorithms in neural networks.

Stability Analysis of the Maximum Eigenfilter

In the ODE approach to stability, we have the tool we need to investigate the convergence behavior of the recursive algorithm of Eq. (8.46) pertaining to a maximum eigenfilter, as described here.

To satisfy condition 1 of the asymptotic stability theorem, we let

$$\eta(n) = \frac{1}{n}$$

Next, we note from Eq. (8.47) that the update function $h(\mathbf{w}, \mathbf{x})$ is defined by

$$\begin{aligned} h(\mathbf{w}, \mathbf{x}) &= \mathbf{x}(n)y(n) - y^2(n)\mathbf{w}(n) \\ &= \mathbf{x}(n)\mathbf{x}^T(n)\mathbf{w}(n) - [\mathbf{w}^T(n)\mathbf{x}(n)\mathbf{x}^T(n)\mathbf{w}(n)]\mathbf{w}(n) \end{aligned} \quad (8.55)$$

which clearly satisfies condition 3 of the theorem. Equation (8.55) results from the use of a realization $\mathbf{x}$ of the random vector $\mathbf{X}$ in the update function $h(\mathbf{w}, \mathbf{X})$. For condition 4 we take the expectation of $h(\mathbf{w}, \mathbf{X})$ over $\mathbf{X}$, and thus write

$$\begin{aligned} \bar{h} &= \lim_{n \rightarrow \infty} E[\mathbf{X}(n)\mathbf{X}^T(n)\mathbf{w}(n) - (\mathbf{w}^T(n)\mathbf{X}(n)\mathbf{X}^T(n)\mathbf{w}(n))\mathbf{w}(n)] \\ &= \mathbf{R}\mathbf{w}(\infty) - [\mathbf{w}^T(\infty)\mathbf{R}\mathbf{w}(\infty)]\mathbf{w}(\infty) \end{aligned} \quad (8.56)$$

where $\mathbf{R}$ is the correlation matrix of the stochastic process represented by the random vector $\mathbf{X}(n)$, and $\mathbf{w}(\infty)$ is the limiting value of the synaptic weight vector.

In accordance with condition 5 and in light of Eqs. (8.53) and (8.56), we seek stable points of the nonlinear differential equation

$$\begin{aligned} \frac{d}{dt} \mathbf{w}(t) &= \bar{h}(\mathbf{w}(t)) \\ &= \mathbf{R}\mathbf{w}(t) - [\mathbf{w}^T(t)\mathbf{R}\mathbf{w}(t)]\mathbf{w}(t) \end{aligned} \quad (8.57)$$

Let $\mathbf{w}(t)$ be expanded in terms of the complete orthonormal set of eigenvectors of the correlation matrix $\mathbf{R}$ as follows:

$$\mathbf{w}(t) = \sum_{k=1}^m \theta_k(t) \mathbf{q}_k \quad (8.58)$$

where $\mathbf{q}_k$ is the k th normalized eigenvector of the matrix $\mathbf{R}$, and the coefficient $\theta_k(t)$ is the time-varying projection of the vector $\mathbf{w}(t)$ onto $\mathbf{q}_k$. Substituting Eq. (8.58) in (8.57), and using the basic definitions

$$\mathbf{R}\mathbf{q}_k = \lambda_k \mathbf{q}_k$$

and

$$\mathbf{q}_k^T \mathbf{R} \mathbf{q}_k = \lambda_k$$

where λ_k is the eigenvalue associated with $\mathbf{q}_k$, we finally get

$$\sum_{k=1}^m \frac{d\theta_k(t)}{dt} \mathbf{q}_k = \sum_{k=1}^m \lambda_k \theta_k(t) \mathbf{q}_k - \left[\sum_{l=1}^m \lambda_l \theta_l^2(t) \right] \sum_{k=1}^m \theta_k(t) \mathbf{q}_k \quad (8.59)$$

Equivalently, we may write

$$\frac{d\theta_k(t)}{dt} = \lambda_k \theta_k(t) - \theta_k(t) \sum_{l=1}^m \lambda_l \theta_l^2(t), \quad k = 1, 2, \dots, m \quad (8.60)$$

We have thus reduced the convergence analysis of the stochastic approximation algorithm of (8.48) to the stability analysis of a system of ordinary differential equations (8.60) involving the *principal modes* $\theta_k(t)$.

There are two cases to be considered here, depending on the value assigned to the index k . Case I corresponds to $1 < k \leq m$, and case II corresponds to $k = 1$; m is the dimension of both $\mathbf{x}(n)$ and $\mathbf{w}(n)$. These two cases are considered in turn.

Case I. $1 < k \leq m$. For the treatment of this case we define

$$\alpha_k(t) = \frac{\theta_k(t)}{\theta_1(t)}, \quad 1 < k \leq m \quad (8.61)$$

Hence it is assumed that $\theta_1(t) \neq 0$, which is true with probability 1 provided that the initial values $\mathbf{w}(0)$ are chosen at random. Then, differentiating both sides of Eq. (8.61) with respect to time t , we get

$$\begin{aligned} \frac{d\alpha_k(t)}{dt} &= \frac{1}{\theta_1(t)} \frac{d\theta_k(t)}{dt} - \frac{\theta_k(t)}{\theta_1^2(t)} \frac{d\theta_1(t)}{dt} \\ &= \frac{1}{\theta_1(t)} \frac{d\theta_k(t)}{dt} - \frac{\alpha_k(t)}{\theta_1(t)} \frac{d\theta_1(t)}{dt}, \quad 1 < k \leq m \end{aligned} \quad (8.62)$$

Next, using Eq. (8.60) in (8.62), applying the definition of Eq. (8.61), and then simplifying the result, we get

$$\frac{d\alpha_k(t)}{dt} = -(\lambda_1 - \lambda_k)\alpha_k(t), \quad 1 < k \leq m \quad (8.63)$$

With the eigenvalues of the correlation matrix $\mathbf{R}$ assumed to be distinct and arranged in decreasing order, we have

$$\lambda_1 > \lambda_2 > \cdots > \lambda_k > \cdots > \lambda_m > 0 \quad (8.64)$$

It follows therefore that the eigenvalue difference $\lambda_1 - \lambda_k$, representing the reciprocal of a time constant in Eq. (8.63), is positive, so we find that for case I:

$$\alpha_k(t) \rightarrow 0 \quad \text{as } t \rightarrow \infty \quad \text{for } 1 < k \leq m \quad (8.65)$$

Case II. $k = 1$. From Eq. (8.60), this second case is described by the differential equation

$$\begin{aligned} \frac{d\theta_1(t)}{dt} &= \lambda_1 \theta_1(t) - \theta_1(t) \sum_{l=1}^m \lambda_l \theta_l^2(t) \\ &= \lambda_1 \theta_1(t) - \lambda_1 \theta_1^3(t) - \theta_1(t) \sum_{l=2}^m \lambda_l \theta_l^2(t) \\ &= \lambda_1 \theta_1(t) - \lambda_1 \theta_1^3(t) - \theta_1^3(t) \sum_{l=2}^m \lambda_l \alpha_l^2(t) \end{aligned} \quad (8.66)$$

However, from case I we know that $\alpha_l \rightarrow 0$ for $l \neq 1$ as $t \rightarrow \infty$. Hence the last term on the right-hand side of Eq. (8.66) approaches zero as time t approaches infinity. Ignoring this term, Eq. (8.66) simplifies to

$$\frac{d\theta_1(t)}{dt} = \lambda_1 \theta_1(t) [1 - \theta_1^2(t)] \quad \text{for } t \rightarrow \infty \quad (8.67)$$

It must be emphasized, however, that Eq. (8.67) holds only in an asymptotic sense.

Equation (8.67) represents an *autonomous system* (i.e., a system with no explicit time dependence). The stability of such a system is best handled using a positive-definite function called the *Lyapunov function*, a detailed treatment of which is deferred to Chapter 14. Let $\mathbf{s}$ denote the state vector of an autonomous system, and $V(t)$ denote a Lyapunov function of the system. An equilibrium state $\bar{\mathbf{s}}$ of the system is asymptotically stable if

$$\frac{d}{dt} V(t) < 0 \quad \text{for } \mathbf{s} \in \mathcal{U} - \bar{\mathbf{s}}$$

where $\mathcal{U}$ is a small neighborhood around $\bar{\mathbf{s}}$.

For the problem at hand, we assert that the differential equation (8.67) has a Lyapunov function defined by

$$V(t) = [\theta_1^2(t) - 1]^2 \quad (8.68)$$

To validate this assertion, we must show that $V(t)$ satisfies two conditions:

$$1. \quad \frac{dV(t)}{dt} < 0 \quad \text{for all } t \quad (8.69)$$

$$2. \quad V(t) \text{ has a minimum} \quad (8.70)$$

Differentiating Eq. (8.68) with respect to time, we get

$$\begin{aligned} \frac{dV(t)}{dt} &= 4\theta_1(t)[\theta_1(t) - 1] \frac{d\theta_1}{dt} \\ &= -4\lambda_1 \theta_1^2(t) [\theta_1^2(t) - 1]^2 \quad \text{for } t \rightarrow \infty \end{aligned} \quad (8.71)$$

where in the second line we have made use of Eq. (8.67). Since the eigenvalue λ_1 is positive, we find from Eq. (8.71) that the condition of Eq. (8.69) is true for t approaching infinity. Furthermore, from Eq. (8.71) we note that $V(t)$ has a minimum [i.e., $dV(t)/dt$ is zero] at $\theta_1(t) = \pm 1$, and so the condition of Eq. (8.70) is also satisfied. We may therefore conclude the analysis of case II by stating that

$$\theta_1(t) \rightarrow \pm 1 \quad \text{as } t \rightarrow \infty \quad (8.72)$$

In light of the result described in Eq. (8.72) and the definition of Eq. (8.71), we may restate the result of case I given in Eq. (8.65) in its final form:

$$\theta_k(t) \rightarrow 0 \quad \text{as } t \rightarrow \infty \quad \text{for } 1 < k \leq m \quad (8.73)$$

The overall conclusion drawn from the analysis of cases I and II is twofold:

- The only principal mode of the stochastic approximation algorithm described in Eq. (8.47) that will converge is $\theta_1(t)$; all the other modes of the algorithm will decay to zero.
- The mode $\theta_1(t)$ will converge to ± 1 .

Hence, condition 5 of the asymptotic stability theorem is satisfied. Specifically, in light of the expansion described in Eq. (8.58), we may formally state that

$$\mathbf{w}(t) \rightarrow \mathbf{q}_1 \quad \text{as } t \rightarrow \infty$$

where $\mathbf{q}_1$ is the normalized eigenvector associated with the largest eigenvalue λ_1 of the correlation matrix $\mathbf{R}$.

We must next show that, in accordance with condition 6 of the asymptotic stability theorem, there exists a subset $\mathcal{A}$ of the set of all vectors, such that

$$\lim_{n \rightarrow \infty} \mathbf{w}(n) = \mathbf{q}_1 \quad \text{infinitely often with probability 1}$$

To do so, we must first satisfy condition 2, which we do by *hard-limiting* the entries of $\mathbf{w}(n)$ so that their magnitudes remain below some threshold a . We may then define the norm of $\mathbf{w}(n)$ by writing

$$\|\mathbf{w}(n)\| = \max_j |w_j(n)| \leq a \quad (8.74)$$

Let $\mathcal{A}$ be the compact subset of $\mathbb{R}^m$ defined by the set of vectors with norm less than or equal to a . It is straightforward to show that (Sanger, 1989b)

If $\|\mathbf{w}(n)\| \leq a$, and the constant a is sufficiently large, then $\|\mathbf{w}(n+1)\| < \|\mathbf{w}(n)\|$ with probability 1.

Thus, as the number of iterations n increases, $\mathbf{w}(n)$ will eventually be within $\mathcal{A}$, and it will remain inside $\mathcal{A}$ (infinitely often) with probability 1. Since the basin of attraction

$\mathcal{B}(\mathbf{q}_1)$ includes all vectors with bounded norm, we have $\mathcal{A} \in \mathcal{B}(\mathbf{q}_1)$. In other words, condition 6 is satisfied.

We have now satisfied all six conditions of the asymptotic stability theorem, and thereby shown that (subject to the aforementioned assumptions) the stochastic approximation algorithm of (8.47) will cause $\mathbf{w}(n)$ to converge with probability 1 to the eigenvector $\mathbf{q}_1$ associated with the largest eigenvalue λ_1 of the correlation matrix $\mathbf{R}$. This is not the only fixed point of the algorithm, but it is the only one that is asymptotically stable.

Summarizing Properties of the Hebbian-Based Maximum Eigenfilter

The convergence analysis just presented shows that a single linear neuron governed by the self-organized learning rule of Eq. (8.39), or equivalently that of Eq. (8.46), adaptively extracts the first principal component of a stationary input. This first principal component corresponds to the largest eigenvalue λ_1 of the correlation matrix of the random vector $\mathbf{X}(n)$; in fact, λ_1 is related to the variance of the model output $y(n)$, as shown here.

Let $\sigma^2(n)$ denote the variance of random variable $Y(n)$ with a realization of it denoted by $y(n)$, that is,

$$\sigma^2(n) = E[Y^2(n)] \quad (8.75)$$

where $Y(n)$ has zero mean for a zero-mean input. Letting $n \rightarrow \infty$ in Eq. (8.46) and using the fact that, in a corresponding way, $\mathbf{w}(n)$ approaches $\mathbf{q}_1$, we obtain

$$\mathbf{x}(n) = y(n)\mathbf{q}_1 \quad \text{for } n \rightarrow \infty$$

Using this relation, we can show that the variance $\sigma^2(n)$ approaches λ_1 as the number of iterations n approaches infinity; see Problem 8.2.

In summary, a Hebbian-based linear neuron whose operation is described by Eq. (8.46) converges with probability 1 to a fixed point, which is characterized as follows (Oja, 1982):

1. The variance of the model output approaches the largest eigenvalue of the correlation matrix $\mathbf{R}$, as shown by

$$\lim_{n \rightarrow \infty} \sigma^2(n) = \lambda_1 \quad (8.76)$$

2. The synaptic weight vector of the model approaches the associated eigenvector, as shown by

$$\lim_{n \rightarrow \infty} \mathbf{w}(n) = \mathbf{q}_1 \quad (8.77)$$

with

$$\lim_{n \rightarrow \infty} \|\mathbf{w}(n)\| = 1 \quad (8.78)$$

These results assume that the correlation matrix $\mathbf{R}$ is positive definite with the largest eigenvalue λ_1 having multiplicity 1. They also hold for a nonnegative definite correlation matrix $\mathbf{R}$ provided that $\lambda_1 > 0$ with multiplicity 1.

Example 8.2 Matched Filter

Consider a random vector $\mathbf{X}(n)$ composed as follows:

$$\mathbf{X}(n) = \mathbf{s} + \mathbf{V}(n)$$

where $\mathbf{s}$ is a fixed unit vector representing the *signal component*, and $\mathbf{V}(n)$ is a zero-mean *white-noise component*. The correlation matrix of the input vector is

$$\begin{aligned} \mathbf{R} &= E[\mathbf{X}(n)\mathbf{X}^T(n)] \\ &= \mathbf{s}\mathbf{s}^T + \sigma^2\mathbf{I} \end{aligned}$$

where σ^2 is the variance of the elements of the noise vector $\mathbf{V}(n)$, and $\mathbf{I}$ is the identity matrix. The largest eigenvalue of the correlation matrix $\mathbf{R}$ is therefore

$$\lambda_1 = 1 + \sigma^2$$

The associated eigenvector $\mathbf{q}_1$ is

$$\mathbf{q}_1 = \mathbf{s}$$

It is readily shown that this solution satisfies the eigenvalue problem

$$\mathbf{R}\mathbf{q}_1 = \lambda_1\mathbf{q}_1$$

Hence, for the situation described in this example, the self-organized linear neuron (upon convergence to its stable condition) acts as a *matched filter* in the sense that its impulse response (represented by the synaptic weights) is matched to the signal component $\mathbf{s}$ of the input vector $\mathbf{X}(n)$. ■

8.5 HEBBIAN-BASED PRINCIPAL COMPONENTS ANALYSIS

The Hebbian-based maximum eigenfilter of the previous section extracts the first principal component of the input. This single linear neuronal model may be expanded into a feedforward network with a single layer of linear neurons for the purpose of principal components analysis of arbitrary size on the input (Sanger, 1989b).

To be specific, consider the feedforward network shown in Fig. 8.6. The following two assumptions of a structural nature are made:

1. Each neuron in the output layer of the network is *linear*.
2. The network has m inputs and l outputs, both of which are specified. Moreover, the network has fewer outputs than inputs (i.e., $l < m$).

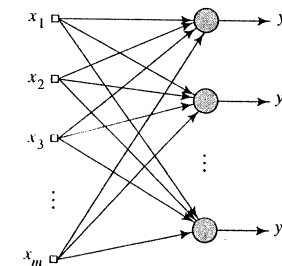


FIGURE 8.6 Feedforward network with a single layer of computation nodes.

The only aspect of the network that is subject to training is the set of synaptic weights $\{w_{ji}\}$ connecting source nodes i in the input layer to computation nodes j in the output layer, where $i = 1, 2, \dots, m$, and $j = 1, 2, \dots, l$.

The output $y_j(n)$ of neuron j at time n , produced in response to the set of inputs $\{x_i(n)|i = 1, 2, \dots, m\}$ is given by (see Fig 8.7a)

$$y_j(n) = \sum_{i=1}^m w_{ji}(n)x_i(n), \quad j = 1, 2, \dots, l \quad (8.79)$$

The synaptic weight $w_{ji}(n)$ is adapted in accordance with a generalized form of Hebbian learning, as shown by (Sanger, 1989b):

$$\Delta w_{ji}(n) = \eta \left[y_j(n)x_i(n) - y_j(n) \sum_{k=1}^l w_{ki}(n)y_k(n) \right], \quad \begin{matrix} i = 1, 2, \dots, m \\ j = 1, 2, \dots, l \end{matrix} \quad (8.80)$$

where $\Delta w_{ji}(n)$ is the change applied to the synaptic weight $w_{ji}(n)$ at time n , and η is the learning-rate parameter. The *generalized Hebbian algorithm* (GHA) of Eq. (8.80) for a layer of l neurons includes the algorithm of Eq. (8.39) for a single neuron as a special case, that is, $j = 1$.

To develop insight into the behavior of the generalized Hebbian algorithm, we rewrite Eq. (8.80) in the form

$$\Delta w_{ji}(n) = \eta y_j(n)[x'_i(n) - w_{ji}(n)y_j(n)], \quad \begin{matrix} i = 1, 2, \dots, m \\ j = 1, 2, \dots, l \end{matrix} \quad (8.81)$$

where $x'_i(n)$ is a modified version of the i th element of the input vector $\mathbf{x}(n)$; it is a function of the index j , as shown by

$$x'_i(n) = x_i(n) - \sum_{k=1}^{j-1} w_{ki}(n)y_k(n) \quad (8.82)$$

For a specified neuron j , the algorithm described in Eq. (8.81) has exactly the same mathematical form as that of Eq. (8.39), except for the fact that the input signal $x_i(n)$ is replaced by its modified value $x'_i(n)$ in Eq. (8.82). We may go one step further and rewrite Eq. (8.81) in a form that corresponds to Hebb's postulate of learning, as shown by

$$\Delta w_{ji}(n) = \eta y_j(n)x''_i(n) \quad (8.83)$$

where

$$x''_i(n) = x'_i(n) - w_{ji}(n)y_j(n) \quad (8.84)$$

Thus noting that

$$w_{ji}(n+1) = w_{ji}(n) + \Delta w_{ji}(n) \quad (8.85)$$

and

$$w_{ji}(n) = z^{-1}[w_{ji}(n+1)] \quad (8.86)$$

where z^{-1} is the unit-delay operator, we may construct the signal-flow graph of Fig. 8.7b for the generalized Hebbian algorithm. From this graph we see that the algorithm lends itself to a *local* form of implementation, provided that it is formulated as in

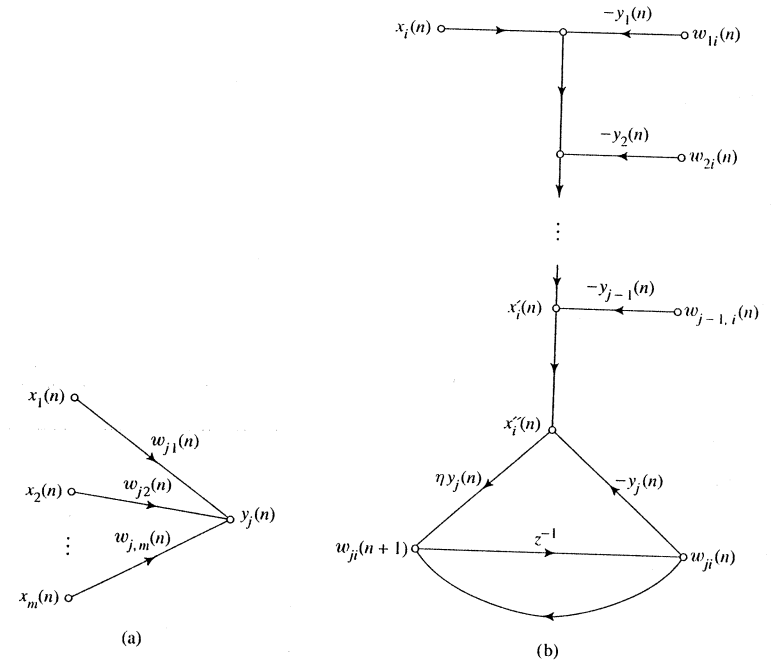


FIGURE 8.7 The signal-flow graph representation of generalized Hebbian algorithm. (a) Graph of Eq. (8.79). (b) Graph of Eqs. (8.80) through (8.81).

Eq. (8.85). Note also that $y_j(n)$, responsible for feedback in the signal-flow graph of Fig. 8.7b, is itself determined by Eq. (8.79); signal-flow graph representation of this latter equation is shown in Fig. 8.7a.

For a heuristic understanding of how the generalized Hebbian algorithm actually operates, we first use matrix notation to rewrite the version of the algorithm defined in Eq. (8.81) as follows:

$$\Delta \mathbf{w}_j(n) = \eta y_j(n) \mathbf{x}'(n) - \eta y_j^2(n) \mathbf{w}_j(n), \quad j = 1, 2, \dots, l \quad (8.87)$$

where

$$\mathbf{x}'(n) = \mathbf{x}(n) - \sum_{k=1}^{j-1} \mathbf{w}_k(n) y_k(n) \quad (8.88)$$

The vector $\mathbf{x}'(n)$ represents a modified form of the input vector. Based on the representation given in Eq. (8.87), we make the following observations (Sanger, 1989b):

1. For the first neuron of the feedforward network shown in Fig. 9.6, we have

$$j = 1: \quad \mathbf{x}'(n) = \mathbf{x}(n)$$

In this case, the generalized Hebbian algorithm reduces to that of Eq. (8.46) for a single neuron. From the material presented in Section 8.5 we already know that this neuron will discover the first principal component of the input vector $\mathbf{x}(n)$.

2. For the second neuron of the network in Fig. 8.6, we write

$$j = 2: \quad \mathbf{x}'(n) = \mathbf{x}(n) - \mathbf{w}_1(n)y_1(n)$$

Provided that the first neuron has already converged to the first principal component, the second neuron sees an input vector $\mathbf{x}'(n)$ from which the first eigenvector of the correlation matrix $\mathbf{R}$ has been removed. The second neuron therefore extracts the first principal component of $\mathbf{x}'(n)$, which is equivalent to the second principal component of the original input vector $\mathbf{x}(n)$.

3. For the third neuron we write

$$j = 3: \quad \mathbf{x}'(n) = \mathbf{x}(n) - \mathbf{w}_1(n)y_1(n) - \mathbf{w}_2(n)y_2(n)$$

Suppose that the first two neurons have already converged to the first and second principal components, as explained in steps 1 and 2. The third neuron now sees an input vector $\mathbf{x}'(n)$ from which the first two eigenvectors have been removed. Therefore, it extracts the first principal component of the vector $\mathbf{x}'(n)$, which is equivalent to the third principal component of the original input vector $\mathbf{x}(n)$.

4. Proceeding in this fashion for the remaining neurons of the feedforward network in Fig. 8.6, it is now apparent that each output of the network trained in accordance with the generalized Hebbian algorithm of Eq. (8.81) represents the response to a particular eigenvector of the correlation matrix of the input vector, and that the individual outputs are ordered by decreasing eigenvalue.

This method of computing eigenvectors is similar to a technique known as *Hotelling's deflation technique* (Kreyszig, 1988); it follows a procedure similar to Gram-Schmidt orthogonalization (Strang, 1980).

The neuron-by-neuron description presented here is intended merely to simplify the explanation. In practice, all the neurons in the generalized Hebbian algorithm tend to converge together.

Convergence Considerations

Let $\mathbf{W}(n) = \{\mathbf{w}_j(n)\}$ denote the l -by- m synaptic weight matrix of the feedforward network shown in Fig. 8.6; that is,

$$\mathbf{W}(n) = [\mathbf{w}_1(n), \mathbf{w}_2(n), \dots, \mathbf{w}_l(n)]^T \quad (8.89)$$

Let the learning-rate parameter of the generalized Hebbian algorithm of Eq. (8.81) take a time-varying form $\eta(n)$, such that in the limit we have

$$\lim_{n \rightarrow \infty} \eta(n) = 0 \quad \text{and} \quad \sum_{n=0}^{\infty} \eta(n) = \infty \quad (8.90)$$

We may then rewrite this algorithm in the matrix form

$$\Delta \mathbf{W}(n) = \eta(n) [\mathbf{y}(n) \mathbf{x}^T(n) - \text{LT}[\mathbf{y}(n) \mathbf{y}^T(n) \mathbf{W}(n)]] \quad (8.91)$$

where the operator $\text{LT}[\cdot]$ sets all the elements above the diagonal of its matrix argument to zero, thereby making that matrix *lower triangular*. Under these conditions, and invoking the assumptions made in Section 8.4, convergence of the GHA algorithm is proved by following a procedure similar to that presented in the previous section for the maximum eigenfilter. Thus we may state the following theorem (Sanger, 1989b):

If the synaptic weight matrix $\mathbf{W}(n)$ is assigned random values at time step $n = 0$, then with probability 1, the generalized Hebbian algorithm of Eq. (8.91) will converge to a fixed point with $\mathbf{W}^T(n)$ approaching a matrix whose columns are the first l eigenvectors of the m -by- m correlation matrix $\mathbf{R}$ of the m -by-1 input vector, ordered by decreasing eigenvalue.

The practical significance of this theorem is that it guarantees the generalized Hebbian algorithm to find the first l eigenvectors of the correlation matrix $\mathbf{R}$, assuming that the associated eigenvalues are distinct. Equally important is the fact that we do not need to compute the correlation matrix $\mathbf{R}$. Rather, the first l eigenvectors of $\mathbf{R}$ are computed by the algorithm directly from the input data. The resulting computational savings can be enormous especially if the dimensionality m of the input space is very large, and the required number of the eigenvectors associated with the l largest eigenvalues of the correlation matrix $\mathbf{R}$ is a small fraction of m .

The convergence theorem is formulated in terms of a time-varying learning-rate parameter $\eta(n)$. In practice, the learning-rate parameter is chosen to be a small constant η , in which case convergence is guaranteed with mean-squared error in synaptic weights of order η .

In Chatterjee et al. (1998), the convergence properties of the GHA algorithm described in Eq. (8.91) are investigated. The analysis presented therein shows that increasing η leads to faster convergence and larger asymptotic mean-square error, which is intuitively satisfying. In that paper, the tradeoff between the accuracy of computation and speed of learning is made explicit, among other things.

Optimality of the Generalized Hebbian Algorithm

Suppose that in the limit we write

$$\Delta \mathbf{w}_j(n) \rightarrow 0 \quad \text{and} \quad \mathbf{w}_j(n) \rightarrow \mathbf{q}_j \quad \text{as } n \rightarrow \infty \quad \text{for } j = 1, 2, \dots, l \quad (8.92)$$

and that we have

$$\|\mathbf{w}_j(n)\| = 1 \quad \text{for all } j \quad (8.93)$$

Then the limiting values $\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_l$ of the synaptic weight vectors of the neurons in the feedforward network of Fig. 8.5 represent the *normalized eigenvectors* associated with l dominant eigenvalues of the correlation matrix $\mathbf{R}$, and which are ordered in descending eigenvalue. At equilibrium we may therefore write

$$\mathbf{q}_j^T \mathbf{R} \mathbf{q}_k = \begin{cases} \lambda_j, & k = j \\ 0, & k \neq j \end{cases} \quad (8.94)$$

where $\lambda_1 > \lambda_2 > \dots > \lambda_l$.

For the output of neuron j , we have the limiting value

$$\lim_{n \rightarrow \infty} y_j(n) = \mathbf{x}^T(n) \mathbf{q}_j = \mathbf{q}_j^T \mathbf{x}(n) \quad (8.95)$$

Let $Y_j(n)$ denote a random variable with a realization denoted by the output $y_j(n)$. The cross-correlation between the random variables $Y_j(n)$ and $Y_k(n)$ at equilibrium is given by

$$\begin{aligned} \lim_{n \rightarrow \infty} E[Y_j(n)Y_k(n)] &= E[\mathbf{q}_j^T \mathbf{X}(n) \mathbf{X}^T(n) \mathbf{q}_k] \\ &= \mathbf{q}_j^T \mathbf{R} \mathbf{q}_k \\ &= \begin{cases} \lambda_j, & k = j \\ 0, & k \neq j \end{cases} \end{aligned} \quad (8.96)$$

Hence, we may state that at equilibrium the generalized Hebbian algorithm of Eq. (8.91) acts as an *eigen-analyzer* of the input data.

Let $\hat{\mathbf{x}}(n)$ denote the particular value of the input vector $\mathbf{x}(n)$ for which the limiting conditions of Eq. (8.92) are satisfied for $j = l - 1$. Hence, from the matrix form of Eq. (8.80), we find that in the limit

$$\hat{\mathbf{x}}(n) = \sum_{k=1}^l y_k(n) \mathbf{q}_k \quad (8.97)$$

This means that given two sets of quantities, the limiting values $\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_l$ of the synaptic weight vectors of the neurons in the feedforward network of Fig. 8.5 and the corresponding outputs $y_1(n), y_2(n), \dots, y_l(n)$, we may then construct a *linear least-squares estimate* $\hat{\mathbf{x}}(n)$ of the input vector $\mathbf{x}(n)$. In effect, the formula of Eq. (8.97) may be viewed as one of *data reconstruction*, as depicted in Fig. 8.8. Note that in light of the discussion presented in Section 8.3, this method of data reconstruction is subject to an approximation error vector that is orthogonal to the estimate $\hat{\mathbf{x}}(n)$.

Summary of the GHA

The computations involved in the generalized Hebbian algorithm (GHA) are simple; they may be summarized as follows:

1. Initialize the synaptic weights of the network, w_{ji} , to small random values at time $n = 1$. Assign a small positive value to the learning-rate parameter η .

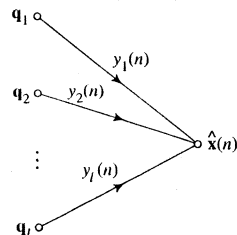


FIGURE 8.8 Signal-flow graph representation of how the reconstructed vector $\hat{\mathbf{x}}$ is computed.

2. For $n = 1, j = 1, 2, \dots, l$, and $i = 1, 2, \dots, m$, compute

$$\begin{aligned} y_j(n) &= \sum_{i=1}^m w_{ji}(n) x_i(n) \\ \Delta w_{ji}(n) &= \eta \left[y_j(n) x_i(n) - y_j(n) \sum_{k=1}^l w_{ki}(n) y_k(n) \right] \end{aligned}$$

where $x_i(n)$ is the i th component of the m -by-1 input vector $\mathbf{x}(n)$ and l is the desired number of principal components.

3. Increment n by 1, go to step 2, and continue until the synaptic weights w_{ji} reach their steady-state values. For large n , the synaptic weight w_{ji} of neuron j converges to the i th component of the eigenvector associated with the j th eigenvalue of the correlation matrix of the input vector $\mathbf{x}(n)$.

8.6 COMPUTER EXPERIMENT: IMAGE CODING

We complete discussion of the generalized Hebbian learning algorithm by examining its use for solving an *image coding* problem.

Figure 8.9a shows an image of parents used for training; this image emphasizes *edge* information. It was digitized to form a 256×256 image with 256 gray levels. The image was coded using a linear feedforward network with a single layer of 8 neurons, each with 64 inputs. To train the network, 8×8 nonoverlapping blocks of the image were used. The experiment was performed with 2000 scans of the picture and a small learning rate $\eta = 10^{-4}$.

Figure 8.9b shows the 8×8 masks representing the synaptic weights learned by the network. Each of the eight masks displays the set of synaptic weights associated with a particular neuron of the network. Specifically, excitatory synapses (positive weights) are shown white, whereas inhibitory synapses (negative weights) are shown black; gray indicates zero weights. In our notation, the masks represent the columns of the 64×8 synaptic weight matrix $\mathbf{W}^T$ after the generalized Hebbian algorithm has converged.

To code the image, the following procedure was used:

- Each 8×8 block of the image was multiplied by each of the 8 masks shown in Fig. 8.9b, thereby generating 8 coefficients for image coding; Fig. 8.9c shows the reconstructed image based on the dominant 8 principal components without quantization.
- Each coefficient was uniformly quantized with a number of bits approximately proportional to the logarithm of the variance of that coefficient over the image. Thus, the first three masks were assigned 6 bits each, the next two masks 4 bits each, the next two masks 3 bits each, and the last mask 2 bits. Based on this representation, a total of 34 bits were needed to code each 8×8 block of pixels, resulting in a data rate of 0.53 bits per pixel.

To reconstruct the image from the quantized coefficients, all the masks were weighted by their quantized coefficients, then added to reconstitute each block of the image. The reconstructed parents' image with 15 to 1 compression ratio is shown in Fig. 8.9d.

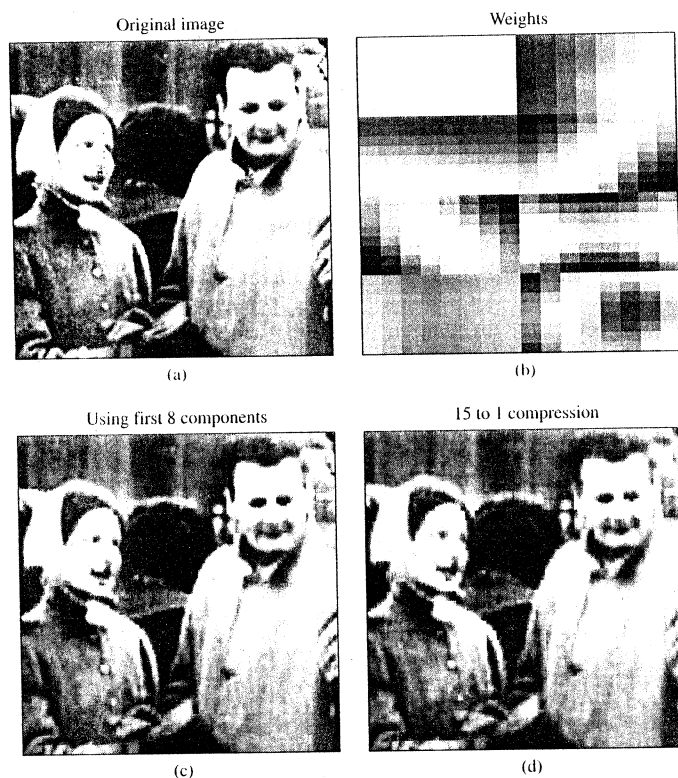


FIGURE 8.9 (a) An image of parents used in the image coding experiment. (b) 8×8 masks representing the synaptic weights learned by the GHA. (c) Reconstructed image of parents obtained using the dominant 8 principal components without quantization. (d) Reconstructed image of parents with 15 to 1 compression ratio using quantization.

For a variation on the first image, we next applied the generalized Hebbian algorithm to the image of an ocean scene shown in Fig. 8.10a. This second image emphasizes *textural* information. Figure 8.10b shows the 8×8 masks of synaptic weights learned by the network by proceeding in the same manner described; note the difference between these masks and those of Fig. 8.9b. Figure 8.10c shows the reconstructed image of the ocean scene based on the dominant 8 principal components without quantization. To study the effect of quantization, the outputs of the first 2 masks were quantized using 5 bits each, the third with 3 bits, and the remaining 5 masks with 2 bits each. Thus a total of 23 bits were needed to code each 8×8 block of pixels, resulting in a bit rate of 0.36 bits per pixel. Figure 8.10d shows the reconstructed image of the ocean scene, using its own masks quantized in the manner just described. The compression ratio of this image was 22 to 1.

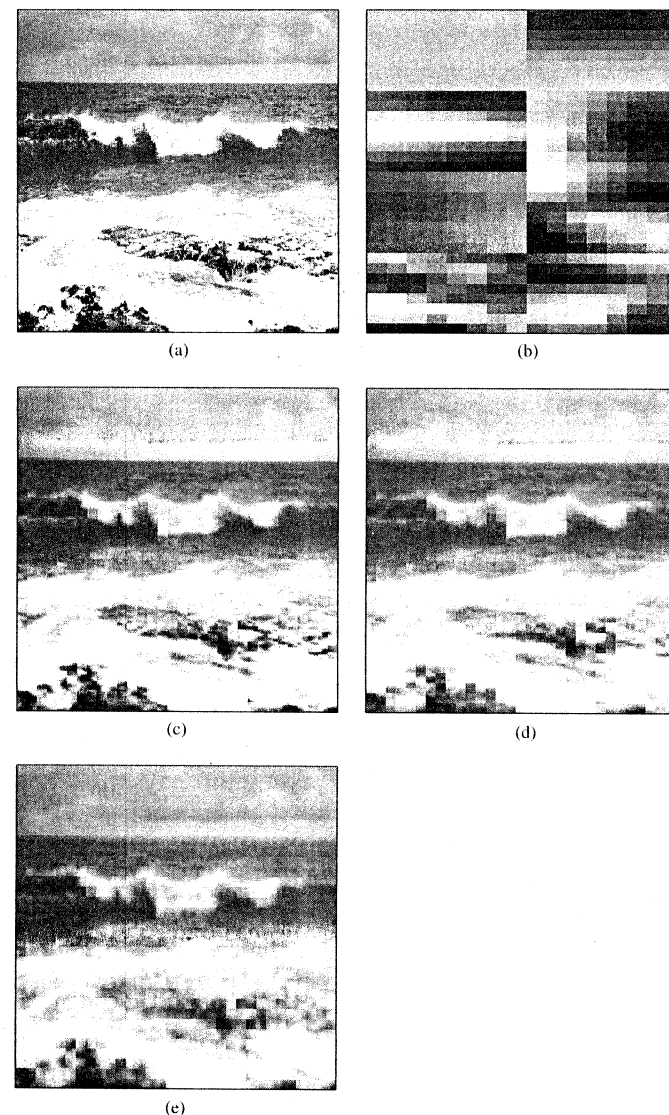


FIGURE 8.10 (a) Image of ocean scene. (b) 8×8 masks representing the synaptic weights learned by the GHA algorithm applied to the ocean scene. (c) Reconstructed image of ocean scene using 8 dominant principal components. (d) Reconstructed image of ocean scene with 22 to 1 compression ratio, using masks of part (b) with quantization. (e) Reconstructed image of ocean scene using the masks of Fig. 8.9(b) for encoding, with quantization for a compression of 22 to 1, same as that in part (d).

To test the “generalization” performance of the generalized Hebbian algorithm, we finally used the masks of Fig. 8.9b to decompose the ocean scene of Fig. 8.10a and then applied the same quantization procedure that was used to generate the reconstructed image of Fig. 8.10d. The result of this image reconstruction is shown in Fig. 8.10e with a compression ratio of 22 to 1, the same as that in Fig. 8.10d. While the reconstructed images in Figs. 8.10d and 8.10e do bear a striking accord with each other, it can be seen that Fig. 8.10d possesses a greater amount of “true” textural information and thus looks less “blocky” than Fig. 8.10e. The reason for this behavior lies in the network weights. For the training performed on the images of the parents and the ocean scene, the first four weights are very similar. However, for the parents image the final four weights encode edge information, but in the case of the ocean scene these weights encode the textural information. Thus when encoding of the ocean scene occurs with the edge-type weights, the reconstruction of textural data is crude, thereby resulting in a blocky appearance.

7 ADAPTIVE PRINCIPAL COMPONENTS ANALYSIS USING LATERAL INHIBITION

The generalized Hebbian algorithm described in the previous section relies on the exclusive use of feedforward connections for principal components analysis. In this section we describe another algorithm called the *adaptive principal components extraction* (APEX) algorithm (Kung and Diamantaras, 1990; Diamantaras and Kung, 1996). The APEX algorithm uses both feedforward and feedback connections.³ The algorithm is iterative in nature in that if we are given the first $(j - 1)$ principal components, the j th principal component is readily computed.

Figure 8.11 shows the network model used for the derivation of the APEX algorithm. As before, the input vector $\mathbf{x}$ has dimension m , with its components denoted by $x_1, x_2, \dots, x_m$. Each neuron in the network is assumed to be linear. As depicted in Fig. 8.11, there are two kinds of synaptic connections in the network:

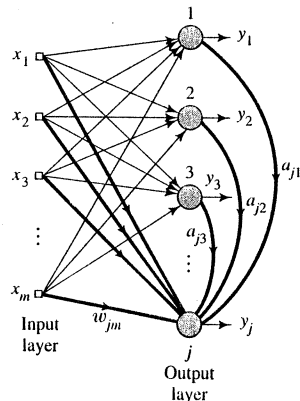


FIGURE 8.11 Network with feedforward and lateral connections for deriving the APEX algorithm.

- *Feedforward connections* from the input nodes to each of the neurons $1, 2, \dots, j$, with $j < m$. Of particular interest here are the feedforward connections to neuron j ; these connections are represented by the feedforward weight vector

$$\mathbf{w}_j = [w_{j1}(n), w_{j2}(n), \dots, w_{jm}(n)]^T$$

The feedforward connections operate in accordance with a *Hebbian learning rule*; they are *excitatory* and therefore provide for self-amplification.

- *Lateral connections* from the individual outputs of neurons $1, 2, \dots, j - 1$ to neuron j , thereby applying *feedback* to the network. These connections are represented by the feedback weight vector

$$\mathbf{a}_j(n) = [a_{j1}(n), a_{j2}(n), \dots, a_{j,j-1}(n)]^T$$

The lateral connections operate in accordance with an *anti-Hebbian learning rule*, which has the effect of making them *inhibitory*.

In Fig. 8.11 the feedforward and feedback connections of neuron j are boldfaced merely to emphasize that neuron j is the subject of study.

The output $y_j(n)$ of neuron j is given by

$$y_j(n) = \mathbf{w}_j^T(n)\mathbf{x}(n) + \mathbf{a}_j^T(n)\mathbf{y}_{j-1}(n) \quad (8.98)$$

where the contribution $\mathbf{w}_j^T(n)\mathbf{x}(n)$ is due to the feedforward connections, and the remaining contribution $\mathbf{a}_j^T(n)\mathbf{y}_{j-1}(n)$ is due to the lateral connections. The feedback signal vector $\mathbf{y}_{j-1}(n)$ is defined by the outputs of neurons $1, 2, \dots, j - 1$:

$$\mathbf{y}_{j-1}(n) = [y_1(n), y_2(n), \dots, y_{j-1}(n)]^T \quad (8.99)$$

It is assumed that the input vector $\mathbf{x}(n)$ is drawn from a stationary process whose correlation matrix $\mathbf{R}$ has *distinct eigenvalues arranged in decreasing order* as follows:

$$\lambda_1 > \lambda_2 > \dots > \lambda_{j-1} > \lambda_j > \dots > \lambda_m \quad (8.100)$$

It is further assumed that neurons $1, 2, \dots, j - 1$ of the network in Fig. 8.11 have *already converged to their respective stable conditions*, as shown by

$$\mathbf{w}_k(0) = \mathbf{q}_k, \quad k = 1, 2, \dots, j - 1 \quad (8.101)$$

$$\mathbf{a}_k(0) = \mathbf{0}, \quad k = 1, 2, \dots, j - 1 \quad (8.102)$$

where $\mathbf{q}_k$ is the eigenvector associated with the k th eigenvalue of the correlation matrix $\mathbf{R}$, and time step $n = 0$ refers to the start of computations by neuron j of the network. We may then use Eqs. (8.98), (8.99), (8.101), and (8.102) to write

$$\begin{aligned} \mathbf{y}_{j-1}(n) &= [\mathbf{q}_1^T \mathbf{x}(n), \mathbf{q}_2^T \mathbf{x}(n), \dots, \mathbf{q}_{j-1}^T \mathbf{x}(n)] \\ &= \mathbf{Q} \mathbf{x}(n) \end{aligned} \quad (8.103)$$

where $\mathbf{Q}$ is a $(j - 1)$ -by- m matrix defined in terms of the eigenvectors $\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_{j-1}$ associated with the $(j - 1)$ largest eigenvalues $\lambda_1, \lambda_2, \dots, \lambda_{j-1}$ of the correlation matrix $\mathbf{R}$; that is,

$$\mathbf{Q} = [\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_{j-1}]^T \quad (8.104)$$

The requirement is to use neuron j in the network of Fig. 8.11 to compute the next largest eigenvalue λ_j of the correlation matrix $\mathbf{R}$ of the input vector $\mathbf{x}(n)$ and the associated eigenvector $\mathbf{q}_j$.

The update equations for the feedforward weight vector $\mathbf{w}_j(n)$ and the feedback weight vector $\mathbf{a}_j(n)$ for neuron j are defined as, respectively,

$$\mathbf{w}_j(n+1) = \mathbf{w}_j(n) + \eta[y_j(n)\mathbf{x}(n) - y_j^2(n)\mathbf{w}_j(n)] \quad (8.105)$$

and

$$\mathbf{a}_j(n+1) = \mathbf{a}_j(n) - \eta[y_j(n)\mathbf{y}_{j-1}(n) + y_j^2(n)\mathbf{a}_j(n)] \quad (8.106)$$

where η is the *learning-rate parameter*, assumed to be the same for both update equations. The term $y_j(n)\mathbf{x}(n)$ on the right-hand side of Eq. (8.106) represents Hebbian learning, whereas the term $-y_j(n)\mathbf{y}_{j-1}(n)$ on the right-hand side of Eq. (8.106) represents anti-Hebbian learning. The remaining terms, $-y_j^2(n)\mathbf{w}_j(n)$ and $y_j^2(n)\mathbf{a}_j(n)$, are included in these two equations to assure the stability of the algorithm. Basically, Eq. (8.105) is the vector form of Oja's learning rule described in Eq. (8.40), whereas Eq. (8.106) is *new*, accounting for the use of lateral inhibition (Kung and Diamantaras, 1990; Diamantaras and Kung, 1996).

We prove absolute stability of the neural network of Fig. 8.11 by *induction*, as follows:

- First, we prove that if neurons $1, 2, \dots, j-1$ have converged to their stable conditions, then neuron j converges to its own stable condition by extracting the next largest eigenvalue λ_j of the correlation matrix $\mathbf{R}$ of the input vector $\mathbf{x}(n)$ and the associated eigenvector $\mathbf{q}_j$.
- Next, we complete the proof by induction by recognizing that neuron 1 has no feedback and therefore the feedback weight vector $\mathbf{a}_1$ is zero. Hence this particular neuron operates in exactly the same way as Oja's neuron, and from Section 8.4 we know that this neuron is absolutely stable under certain conditions.

The only matter that requires attention is therefore the first point.

To proceed then, we invoke the fundamental assumptions made in Section 8.4, and so state the following theorem in the context of neuron j in the neural network of Fig. 8.11 operating under the conditions described by Eqs. (8.105) and (8.106) (Kung and Diamantaras, 1990; Diamantaras and Kung, 1996):

Given that the learning-rate parameter η is assigned a sufficiently small value to ensure that the adjustments to the weight vectors proceed slowly, then in the limit, the feedforward weight vector and the average output power (variance) of neuron j approaches the normalized eigenvector $\mathbf{q}_j$ and corresponding eigenvalue λ_j of the correlation matrix $\mathbf{R}$, as shown by, respectively,

$$\lim_{n \rightarrow \infty} \mathbf{w}_j(n) = \mathbf{q}_j$$

and

$$\lim_{n \rightarrow \infty} \sigma_j^2(n) = \lambda_j$$

where $\sigma_j^2(n) = E[y_j^2(n)]$, and $\lambda_1 > \lambda_2 > \dots > \lambda_j > \dots > \lambda_m > 0$. In other words, given the eigenvectors $\mathbf{q}_1, \dots, \mathbf{q}_{j-1}$, neuron j in the network of Fig. 8.11 computes the next largest eigenvalue λ_j and associated eigenvector $\mathbf{q}_j$.

To prove this theorem, consider first Eq. (8.105). Using Eqs. (8.98) and (8.99), and recognizing that

$$\mathbf{a}_j^T(n)\mathbf{y}_{j-1}(n) = \mathbf{y}_{j-1}^T(n)\mathbf{a}_j(n)$$

we may recast Eq. (8.105) as follows:

$$\mathbf{w}_j(n+1) = \mathbf{w}_j(n) + \eta[\mathbf{x}(n)\mathbf{x}^T(n)\mathbf{w}_j(n) + \mathbf{x}(n)\mathbf{x}^T(n)\mathbf{Q}^T\mathbf{a}_j(n) - y_j^2(n)\mathbf{w}_j(n)] \quad (8.107)$$

where the matrix $\mathbf{Q}$ is defined by Eq. (8.104). The term $y_j^2(n)$ in Eq. (8.107) has not been touched for a reason that will become apparent. Invoking the fundamental assumptions described in Section 8.4, we find that applying the statistical expectation operator to both sides of Eq. (8.107) yields

$$\mathbf{w}_j(n+1) = \mathbf{w}_j(n) + \eta[\mathbf{R}\mathbf{w}_j(n) + \mathbf{R}\mathbf{Q}^T\mathbf{a}_j(n) - \sigma_j^2(n)\mathbf{w}_j(n)] \quad (8.108)$$

where $\mathbf{R}$ is the correlation matrix of the input vector $\mathbf{x}(n)$, and $\sigma_j^2(n)$ is the average output power of neuron j . Let the synaptic weight vector $\mathbf{w}_j(n)$ be expanded in terms of the entire orthonormal set of eigenvectors of the correlation matrix $\mathbf{R}$ as follows:

$$\mathbf{w}_j(n) = \sum_{k=1}^m \theta_{jk}(n)\mathbf{q}_k \quad (8.109)$$

where $\mathbf{q}_k$ is the eigenvector associated with the eigenvalue λ_k of matrix $\mathbf{R}$, and $\theta_{jk}(n)$ is a time-varying coefficient of the expansion. We may then use the basic relation (see Eq. (8.14))

$$\mathbf{R}\mathbf{q}_k = \lambda_k\mathbf{q}_k$$

to express the matrix product $\mathbf{R}\mathbf{w}_j(n)$ as follows:

$$\begin{aligned} \mathbf{R}\mathbf{w}_j(n) &= \sum_{k=1}^m \theta_{jk}(n)\mathbf{R}\mathbf{q}_k \\ &= \sum_{k=1}^m \lambda_k \theta_{jk}(n)\mathbf{q}_k \end{aligned} \quad (8.110)$$

Similarly, using Eq. (8.104), we may express the matrix product $\mathbf{R}\mathbf{Q}^T\mathbf{a}_j(n)$ as

$$\begin{aligned} \mathbf{R}\mathbf{Q}^T\mathbf{a}_j(n) &= \mathbf{R}[\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_{j-1}]\mathbf{a}_j(n) \\ &= [\lambda_1\mathbf{q}_1, \lambda_2\mathbf{q}_2, \dots, \lambda_{j-1}\mathbf{q}_{j-1}] \begin{bmatrix} a_{j1}(n) \\ a_{j2}(n) \\ \vdots \\ a_{j,j-1}(n) \end{bmatrix} \\ &= \sum_{k=1}^{j-1} \lambda_k a_{jk}(n)\mathbf{q}_k \end{aligned} \quad (8.111)$$

Hence, substituting Eqs. (8.109), (8.110), and (8.111) in (8.108), and simplifying, we get (Kung and Diamantaras, 1990)

$$\sum_{k=1}^m \theta_{jk}(n+1) \mathbf{q}_k = \sum_{k=1}^m \{1 + \eta[\lambda_k - \sigma_j^2(n)]\} \theta_{jk}(n) \mathbf{q}_k + \eta \sum_{k=1}^{j-1} \lambda_k a_{jk}(n) \mathbf{q}_k \quad (8.112)$$

Following a procedure similar to that described, it is possible to show that the update equation (8.106) for the feedback weight vector $\mathbf{a}_j(n)$ may be transformed as follows (see Problem 8.7):

$$\mathbf{a}_j(n+1) = -\eta \lambda_k \theta_{jk}(n) \mathbf{1}_k + \{1 - \eta[\lambda_k + \sigma_j^2(n)]\} \mathbf{a}_j(n) \quad (8.113)$$

where $\mathbf{1}_k$ is a vector all of whose j elements are zero, except for the k th element, which is equal to 1. The index k is restricted to lie in the range $1 \leq k \leq j-1$.

There are two cases to be considered, depending on the value assigned to index k in relation to $j-1$. Case I refers to $1 \leq k \leq j-1$, which pertains to the analysis of the "old" principal modes of the network. Case II refers to $j \leq k \leq m$, which pertains to the analysis of the remaining "new" principal modes. The total number of principal modes is m , the dimension of the input vector $\mathbf{x}(n)$.

CASE I: $1 \leq k \leq j-1$

In this case we deduce the following update equations for the coefficient $\theta_{jk}(n)$ associated with eigenvector $\mathbf{q}_k$ and the feedback weight $a_{jk}(n)$ from Eqs. (8.112) and (8.113), respectively:

$$\theta_{jk}(n+1) = \eta \lambda_k a_{jk}(n) + \{1 + \eta[\lambda_k - \sigma_j^2(n)]\} \theta_{jk}(n) \quad (8.114)$$

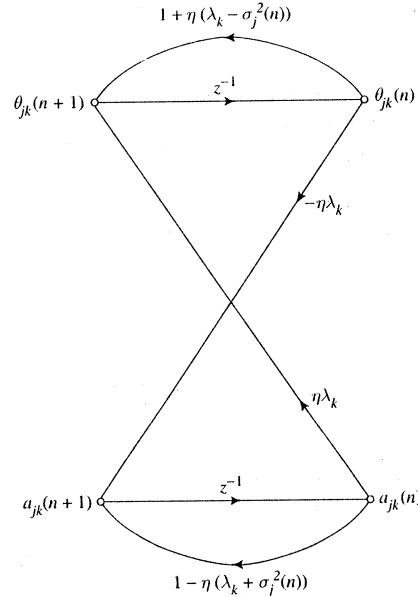


FIGURE 8.12 Signal-flow graph representation of Eqs. (8.114) and (8.115).

and

$$a_{jk}(n+1) = -\eta \lambda_k \theta_{jk}(n) + \{1 - \eta[\lambda_k + \sigma_j^2(n)]\} a_{jk}(n) \quad (8.115)$$

Figure 8.12 presents a signal-flow graph representation of Eqs. (8.114) and (8.115).

In matrix form we may rewrite Eqs. (8.114) and (8.115) as

$$\begin{bmatrix} \theta_{jk}(n+1) \\ a_{jk}(n+1) \end{bmatrix} = \begin{bmatrix} 1 + \eta[\lambda_k - \sigma_j^2(n)] & \eta \lambda_k \\ -\eta \lambda_k & 1 - \eta[\lambda_k + \sigma_j^2(n)] \end{bmatrix} \begin{bmatrix} \theta_{jk}(n) \\ a_{jk}(n) \end{bmatrix} \quad (8.116)$$

The system matrix described in Eq. (8.116) has a double eigenvalue at

$$\rho_{jk} = [1 - \eta \sigma_j^2(n)]^2 \quad (8.117)$$

From Eq. (8.117) we can make two important observations:

1. The double eigenvalue ρ_{jk} of the system matrix in Eq. (8.116) is independent of all the eigenvalues λ_k of the correlation matrix $\mathbf{R}$, corresponding to $k = 1, 2, \dots, j-1$.
2. For all k , the double eigenvalue ρ_{jk} depends solely on the learning-rate parameter η and the average output power σ_j^2 of neuron j . It is therefore less than unity, provided that η is a sufficiently small, positive number.

Given that $\rho_{jk} < 1$, the coefficients $\theta_{jk}(n)$ of the expansion in Eq. (8.109) and the feedback weights $a_{jk}(n)$ will, for all k , approach zero asymptotically with the same speed, since all the principal modes of the network have the same eigenvalue (Kung and Diamantaras, 1990; Diamantaras and Kung, 1996). This result is a consequence of the property that the orthogonality of eigenvectors of a correlation matrix does *not* depend on the eigenvalues. In other words, the expansion of $\mathbf{w}_j(n)$ in terms of the orthonormal set of eigenvectors of the correlation matrix $\mathbf{R}$ given in Eq. (8.109), which is basic to the result described in Eq. (8.117), is *invariant* to the choice of eigenvalues $\lambda_1, \lambda_2, \dots, \lambda_{j-1}$.

CASE II: $j \leq k \leq m$

In this second case, the feedback weights $a_{jk}(n)$ have *no* influence on the modes of the network, as shown by

$$a_{jk}(n) = 0 \quad \text{for } j \leq k \leq m \quad (8.118)$$

Hence, for every principal mode $k \geq j$ we have a very simple equation:

$$\theta_{jk}(n+1) = \{1 + \eta[\lambda_k - \sigma_j^2(n)]\} \theta_{jk}(n) \quad (8.119)$$

which follows directly from Eqs. (8.112) and (8.118). According to case I, both $\theta_{jk}(n)$ and $a_{jk}(n)$ will eventually converge to zero for $k = 1, 2, \dots, j-1$. With the random variable $Y_j(n)$ representing the output of neuron j , we may express its average output power as follows:

$$\begin{aligned} \sigma_j^2(n) &= E[Y_j^2(n)] \\ &= \sum_{k=j}^m \lambda_k \theta_{jk}^2(n) \end{aligned} \quad (8.120)$$

where in the last line we have made use of the following relation:

$$\mathbf{q}_k^T \mathbf{R} \mathbf{q}_l = \begin{cases} \lambda_k, & l = k \\ 0, & \text{otherwise} \end{cases}$$

It follows therefore that Eq. (8.119) cannot diverge, because whenever $\theta_{jk}(n)$ becomes large such that $\sigma_j^2(n) > \lambda_k$, then $1 + \eta[\lambda_k - \sigma_j^2(n)]$ becomes smaller than unity, in which case $\theta_{jk}(n)$ will decrease in magnitude. Let the algorithm be initialized, with $\theta_{ij}(0) \neq 0$. Also define

$$r_{jk}(n) = \frac{\theta_{jk}(n)}{\theta_{jj}(n)}, \quad k = j + 1, \dots, m \quad (8.121)$$

We may then use Eq. (8.119) to write

$$r_{jk}(n+1) = \frac{1 + \eta[\lambda_k - \sigma_j^2(n)]}{1 + \eta[\lambda_j - \sigma_j^2(n)]} r_{jk}(n) \quad (8.122)$$

With the eigenvalues of the correlation matrix arranged in the descending order

$$\lambda_1 > \lambda_2 > \dots > \lambda_k > \dots > \lambda_j \dots > \lambda_m$$

it follows that

$$\frac{\theta_{jk}(n)}{\theta_{jj}(n)} < 1 \quad \text{for all } n, \text{ and for } k = j + 1, \dots, m \quad (8.123)$$

Moreover, we note from Eqs. (8.119) and (8.120) that $\theta_{jj}(n+1)$ remains bounded; therefore,

$$r_{jk}(n) \rightarrow 0 \quad \text{as } n \rightarrow \infty \text{ for } k = j + 1, \dots, m \quad (8.124)$$

Equivalently, in light of the definition given in Eq. (8.121) we may state that

$$\theta_{jk}(n) \rightarrow 0 \quad \text{as } n \rightarrow \infty \text{ for } k = j + 1, \dots, m \quad (8.125)$$

Under this condition, Eq. (8.120) simplifies to

$$\sigma_j^2(n) = \lambda_j \theta_{jj}^2(n) \quad (8.126)$$

and so Eq. (8.119) for $k = j$ becomes

$$\theta_{jj}(n+1) = \{1 + \eta\lambda_j [1 - \theta_{jj}(n)]\} \theta_{jj}(n) \quad (8.127)$$

From this equation we immediately deduce that

$$\theta_{jj}(n) \rightarrow 1 \quad \text{as } n \rightarrow \infty \quad (8.128)$$

The implications of this limiting condition and that of Eq. (8.125) are twofold:

1. From Eq. (8.126) we have

$$\sigma_j^2(n) \rightarrow \lambda_j \quad \text{as } n \rightarrow \infty \quad (8.129)$$

2. From Eq. (8.109) we have

$$\mathbf{w}_j(n) \rightarrow \mathbf{q}_j \quad \text{as } n \rightarrow \infty \quad (8.130)$$

In other words, the neural network model of Fig. 8.11 extracts the j th eigenvalue and associated eigenvector of the correlation matrix $\mathbf{R}$ of the input vector $\mathbf{x}(n)$ as the number of iterations n approaches infinity. This of course assumes that neurons 1, 2, ..., $j-1$ of the network have already converged to the respective eigenvalues and associated eigenvectors of the correlation matrix $\mathbf{R}$.

The treatment of the APEX algorithm presented here rests on the premise that neurons 1, 2, ..., $j-1$ have converged before neuron j begins to act. This was done merely to explain the operation of the algorithm in a simple way. In practice, however, the neurons in the APEX algorithm tend to converge together.⁴

Learning Rate

In the APEX algorithm described in Eqs. (8.105) and (8.106), the same learning-rate parameter η is used for updating both the feedforward weight vector $\mathbf{w}_j(n)$ and feedback weight vector $\mathbf{a}_j(n)$. The relationship of Eq. (8.117) may be exploited to define an optimum value for the learning-rate parameter for each neuron j by setting the double eigenvalue ρ_{jk} equal to zero. In such a case, we have

$$\eta_{j,\text{opt}}(n) = \frac{1}{\sigma_j^2(n)} \quad (8.131)$$

where $\sigma_j^2(n)$ is the average output power of neuron j . A more practical proposition, however, is to set (Kung and Diamantaras, 1990; Diamantaras and Kung, 1996):

$$\eta_j = \frac{1}{\lambda_{j-1}} \quad (8.132)$$

which yields an underestimated value for the learning-rate parameter, since $\lambda_{j-1} > \lambda_j$ and $\sigma_j^2(n) \rightarrow \lambda_j$ as $n \rightarrow \infty$. Note that the eigenvalue λ_{j-1} is computed by neuron $j-1$ and therefore available for use in updating the feedforward and feedback weights of neuron j .

Summary of the APEX Algorithm

1. Initialize the feedforward weight vector $\mathbf{w}_j$ and the feedback weight vector $\mathbf{a}_j$ to small random values at time $n = 1$, where $j = 1, 2, \dots, m$. Assign a small positive value to the learning-rate parameter η .
2. Set $j = 1$, and for $n = 1, 2, \dots$, compute

$$y_1(n) = \mathbf{w}_1^T(n) \mathbf{x}(n)$$

$$\mathbf{w}_1(n+1) = \mathbf{w}_1(n) + \eta[y_1(n) \mathbf{x}(n) - y_1^2(n) \mathbf{w}_1(n)]$$

where $\mathbf{x}(n)$ is the input vector. For large n , we have $\mathbf{w}_1(n) \rightarrow \mathbf{q}_1$, where $\mathbf{q}_1$ is the eigenvector associated with the largest eigenvalue λ_1 of the correlation matrix of $\mathbf{x}(n)$.

3. Set $j = 2$, and for $n = 1, 2, \dots$, compute

$$\mathbf{y}_{j-1}(n) = [y_1(n), y_2(n), \dots, y_{j-1}(n)]^T$$

$$y_j(n) = \mathbf{w}_j^T(n) \mathbf{x}(n) + \mathbf{a}_j^T(n) \mathbf{y}_{j-1}(n)$$

$$\mathbf{w}_j(n+1) = \mathbf{w}_j(n) + \eta[y_j(n) \mathbf{x}(n) - y_j^2(n) \mathbf{w}_j(n)]$$

$$\mathbf{a}_j(n+1) = \mathbf{a}_j(n) - \eta[y_j(n) \mathbf{y}_{j-1}(n) + y_j^2(n) \mathbf{a}_j(n)]$$

4. Increment j by 1, go to step 3, and continue until $j = m$, where m is the desired number of principal components. (Note that $j = 1$ corresponds to the eigenvector associated with the largest eigenvalue, which is taken care of in step 2.) For large

data matrix $\mathbf{A}$ is related to the eigendecomposition of the estimate $\hat{\mathbf{R}}(N)$ of the correlation matrix in the following ways:

- Except for the scaling factor $1/\sqrt{N}$, the singular values of the data matrix $\mathbf{A}$ are the square roots of the eigenvalues of the estimate $\hat{\mathbf{R}}(N)$.
- The left singular vectors of $\mathbf{A}$ are the eigenvectors of $\hat{\mathbf{R}}(N)$.

Now we can see the numerical advantage of singular value decomposition over eigendecomposition. For a prescribed accuracy of computation, the singular value decomposition procedure requires half the numerical precision of the eigendecomposition procedure. Moreover, several algorithms and highly accurate canned routines are available for implementing the singular value decomposition procedure on a computer (Golub and Van Loan, 1996; Haykin, 1996). However, in practice, storage requirements may restrict the use of these routines to values of sample size that are not too excessive.

Turning next to the category of *adaptive methods*, these methods work with an arbitrarily large sample size N . For all practical purposes there is no restriction on N . Adaptive methods are exemplified by Hebbian-based neural networks, whose operation is inspired by ideas from neurobiology. The storage requirement of such methods is relatively modest, since intermediate values of the eigenvalues and associated eigenvectors do not have to be stored. Another attractive feature of adaptive algorithms is that, in a *nonstationary environment*, they have an inherent ability to *track* gradual changes in the optimal solution in an inexpensive way compared to batch methods. However, the main drawback of adaptive algorithms of the stochastic approximation type is their relatively slow rate of convergence, which puts them at a disadvantage compared to classical batch techniques; this is especially so on large stationary problems, even when the adaptive methods are implemented on parallel neural hardware (Kotilainen, 1993).

8.10 KERNEL PRINCIPAL COMPONENTS ANALYSIS

The form of PCA discussed up to this point in the chapter involves computations in the input (data) space. We now consider another form of PCA where the computations are performed in a feature space that is *nonlinearly* related to the input space. The feature space we have in mind is that defined by an *inner-product kernel* in accordance with Mercer's theorem; the notion of inner-product kernels is discussed in Chapter 6 on support vector machines. The idea of *kernel-based principal components analysis* is due to Schölkopf et al. (1998).

Due to the nonlinear relationship between the input space and feature space, kernel PCA is nonlinear. However, unlike other forms of nonlinear PCA,⁶ the implementation of kernel PCA relies on linear algebra. We may therefore think of kernel PCA as a natural extension of ordinary PCA.

Let vector $\varphi(\mathbf{x}_i)$ denote the image of an input vector $\mathbf{x}_i$ induced in a feature space defined by the nonlinear map: $\varphi: \mathbb{R}^{m_0} \rightarrow \mathbb{R}^{m_1}$, where m_0 is the dimensionality of the input space and m_1 is the dimensionality of the feature space. Given the set of examples $\{\mathbf{x}_i\}_{i=1}^N$, we have a corresponding set of feature vectors $\{\varphi(\mathbf{x}_i)\}_{i=1}^N$. Accordingly, we may define an m_1 -by- m_1 correlation matrix in the feature space, denoted by $\tilde{\mathbf{R}}$, as follows:

$$\tilde{\mathbf{R}} = \frac{1}{N} \sum_{i=1}^N \varphi(\mathbf{x}_i) \varphi^T(\mathbf{x}_i) \quad (8.142)$$

As with ordinary PCA, the first thing we have to do is to ensure that the set of feature vectors $\{\varphi(\mathbf{x}_i)\}_{i=1}^N$ has zero mean:

$$\frac{1}{N} \sum_{i=1}^N \varphi(\mathbf{x}_i) = \mathbf{0}$$

To satisfy this condition in the feature space is a more difficult proposition than it is in the input space; in Problem 8.10 we describe a procedure for catering to this requirement. Proceeding then on the assumption that the feature vectors have been centered, we may adapt the use of Eq. (8.14) to our present situation by writing

$$\tilde{\mathbf{R}}\tilde{\mathbf{q}} = \tilde{\lambda}\tilde{\mathbf{q}} \quad (8.143)$$

where $\tilde{\lambda}$ is an eigenvalue of the correlation matrix $\tilde{\mathbf{R}}$ and $\tilde{\mathbf{q}}$ is the associated eigenvector. Now we note that all eigenvectors that satisfy Eq. (8.143) for $\tilde{\lambda} \neq 0$ lie in the span of the set of feature vectors $\{\varphi(\mathbf{x}_i)\}_{i=1}^N$. Consequently, there does exist a corresponding set of coefficients $\{\alpha_j\}_{j=1}^N$ for which we can write

$$\tilde{\mathbf{q}} = \sum_{j=1}^N \alpha_j \varphi(\mathbf{x}_j) \quad (8.144)$$

Thus substituting Eqs. (8.142) and (8.144) into (8.143), we obtain

$$\sum_{i=1}^N \sum_{j=1}^N \alpha_j \varphi(\mathbf{x}_i) K(\mathbf{x}_i, \mathbf{x}_j) = N \tilde{\lambda} \sum_{j=1}^N \alpha_j \varphi(\mathbf{x}_j) \quad (8.145)$$

where $K(\mathbf{x}_i, \mathbf{x}_j)$ is an *inner-product kernel* defined in terms of the feature vectors by

$$K(\mathbf{x}_i, \mathbf{x}_j) = \varphi^T(\mathbf{x}_i) \varphi(\mathbf{x}_j) \quad (8.146)$$

We need to go one step further with Eq. (8.145) so that the relationship is expressed entirely in terms of the inner-product kernel. To do so, we premultiply both sides of Eq. (8.145) by the transposed vector $\varphi^T(\mathbf{x}_k)$, thereby obtaining

$$\sum_{i=1}^N \sum_{j=1}^N \alpha_j K(\mathbf{x}_k, \mathbf{x}_i) K(\mathbf{x}_i, \mathbf{x}_j) = N \tilde{\lambda} \sum_{j=1}^N \alpha_j K(\mathbf{x}_k, \mathbf{x}_j), \quad k = 1, 2, \dots, N \quad (8.147)$$

where the definitions of $K(\mathbf{x}_k, \mathbf{x}_i)$ and $K(\mathbf{x}_i, \mathbf{x}_j)$ follow Eq. (8.146).

We now introduce two matrix definitions:

- The N -by- N matrix $\mathbf{K}$, called the *kernel matrix*, whose ij -th element is the inner-product kernel $K(\mathbf{x}_i, \mathbf{x}_j)$.
- The N -by-1 vector α , whose j th element is the coefficient α_j .

Accordingly, we may recast Eq. (8.147) in the compact matrix form:

$$\mathbf{K}^2 \alpha = N \tilde{\lambda} \mathbf{K} \alpha \quad (8.148)$$

where the squared matrix $\mathbf{K}^2$ denotes the product of $\mathbf{K}$ with itself. Since $\mathbf{K}$ is common to both sides of Eq. (8.148), all the solutions of this eigenvalue problem that are of interest are equally well represented in the simpler eigenvalue problem:

$$\mathbf{K} \alpha = N \tilde{\lambda} \alpha \quad (8.149)$$

Let $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_N$ denote the eigenvalues of the kernel matrix $\mathbf{K}$; that is

$$\lambda_j = N\tilde{\lambda}_j, \quad j = 1, 2, \dots, N \quad (8.150)$$

where $\tilde{\lambda}_j$ is the j th eigenvalue of the correlation matrix $\tilde{\mathbf{R}}$. Then Eq. (8.149) takes the standard form

$$\mathbf{K}\boldsymbol{\alpha} = \lambda\boldsymbol{\alpha} \quad (8.151)$$

where the coefficient vector $\boldsymbol{\alpha}$ plays the role of the eigenvector associated with the eigenvalue λ of the kernel matrix $\mathbf{K}$. The vector $\boldsymbol{\alpha}$ is normalized by requiring that the eigenvector $\tilde{\mathbf{q}}$ of the correlation matrix $\tilde{\mathbf{R}}$ is normalized to unit length; that is

$$\tilde{\mathbf{q}}_k^T \tilde{\mathbf{q}}_k = 1 \quad \text{for } k = 1, 2, \dots, p \quad (8.152)$$

where it is assumed that the eigenvalues are arranged in decreasing order, with λ_p being the smallest nonzero eigenvalue of the kernel matrix $\mathbf{K}$. Using Eq. (8.144) and then invoking Eq. (8.151), we may show that the normalization condition of Eq. (8.152) is equivalent to

$$\boldsymbol{\alpha}_k^T \boldsymbol{\alpha}_k = \frac{1}{\lambda_k}, \quad k = 1, 2, \dots, p \quad (8.153)$$

For the extraction of principal components, we need to compute the projections onto the eigenvectors $\tilde{\mathbf{q}}_k$ in feature space, as shown by

$$\begin{aligned} \tilde{\mathbf{q}}_k^T \boldsymbol{\varphi}(\mathbf{x}) &= \sum_{j=1}^N \alpha_{k,j} \boldsymbol{\varphi}^T(\mathbf{x}_j) \boldsymbol{\varphi}(\mathbf{x}) \\ &= \sum_{j=1}^N \alpha_{k,j} K(\mathbf{x}_j, \mathbf{x}), \quad k = 1, 2, \dots, p \end{aligned} \quad (8.154)$$

where the vector $\mathbf{x}$ is a "test" point, and $\alpha_{k,j}$ is the j th coefficient of eigenvector $\boldsymbol{\alpha}_k$ associated with the k th eigenvalue of matrix $\mathbf{K}$. The projections of Eq. (8.154) define the *nonlinear principal components* in the m_1 -dimensional feature space.

Figure 8.13 illustrates the basic idea of kernel PCA, where the feature space is nonlinearly related to the input space via the transformation $\boldsymbol{\varphi}(\mathbf{x})$. Parts a and b of the figure refer to the input space and feature space, respectively. The contour lines shown in Fig. 8.13b represent constant projections onto a principal eigenvector, which is shown as a dashed arrow. In this figure it is assumed that the transformation $\boldsymbol{\varphi}(\mathbf{x})$ has been chosen in such a way that the images of the data points induced in the feature space congregate themselves essentially along the eigenvector. Figure 8.13a shows the *nonlinear* contour lines in the input space that correspond to those in the feature space. Note that we purposely have not shown a pre-image of the eigenvector in the input space, as it may not even exist (Schölkopf et al., 1998).

For inner-product kernels defined in accordance with Mercer's theorem, we are basically performing ordinary PCA in an m_1 -dimensional feature space, where the dimension m_1 is a design parameter. All the properties of ordinary PCA that are described in Section 8.3 carry over to kernel PCA. In particular, kernel PCA is linear in

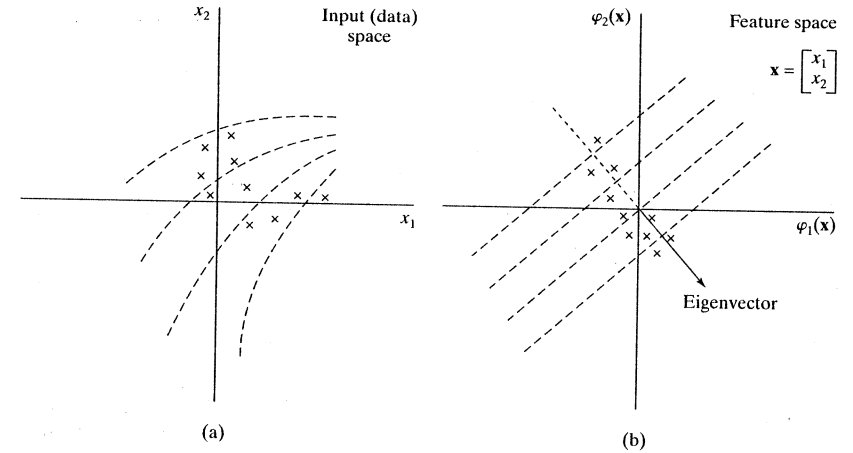


FIGURE 8.13 Illustration of the kernel PCA. (a) Two-dimensional input space, displaying a set of data points. (b) Two-dimensional feature space, displaying the induced images of the data points congregating around a principal eigenvector. The uniformly spaced dashed lines in part (b) represent contours of constant projections onto the eigenvector; the corresponding contours are nonlinear in the input space.

the feature space, but nonlinear in the input space. As such, it can be applied to all those domains where ordinary PCA has been used for feature extraction or data reduction, for which nonlinear extension would make sense.

In Chapter 6, we presented three methods for constructing inner-product kernels that were based on the use of polynomials, radial-basis functions, and hyperbolic functions; see Table 6.1. The question of how to select the kernel best suited for a given task (i.e., the appropriate feature space) is an open problem (Schölkopf, 1997).

Summary of the Kernel PCA

1. Given the training examples $\{\mathbf{x}_i\}_{i=1}^N$, compute the N -by- N kernel matrix $\mathbf{K} = \{K(\mathbf{x}_i, \mathbf{x}_j)\}$, where

$$K(\mathbf{x}_i, \mathbf{x}_j) = \boldsymbol{\varphi}^T(\mathbf{x}_i) \boldsymbol{\varphi}(\mathbf{x}_j)$$

2. Solve the eigenvalue problem:

$$\mathbf{K}\boldsymbol{\alpha} = \lambda\boldsymbol{\alpha}$$

where λ is an eigenvalue of the kernel matrix $\mathbf{K}$ and $\boldsymbol{\alpha}$ is the associated eigenvector.

3. Normalize the eigenvectors so computed by requiring that

$$\boldsymbol{\alpha}_k^T \boldsymbol{\alpha}_k = \frac{1}{\lambda_k}, \quad k = 1, 2, \dots, p$$

where λ_p is the smallest nonzero eigenvalue of matrix $\mathbf{K}$, assuming that the eigenvalues are arranged in decreasing order.

4. For the extraction of principal components of a test point $\mathbf{x}$, compute the projections

$$\begin{aligned} a_k &= \mathbf{q}_k^T \boldsymbol{\varphi}(\mathbf{x}) \\ &= \sum_{j=1}^N \alpha_{k,j} K(\mathbf{x}_j, \mathbf{x}), \quad k = 1, 2, \dots, p \end{aligned}$$

where $\alpha_{k,j}$ is the j th element of eigenvector $\boldsymbol{\alpha}_k$.

Example 8.3

To provide some intuitive understanding for the operation of kernel PCA, we show in Fig. 8.14 the results of a simple experiment described in Schölkopf et al. (1998). The two-dimensional data, consisting of components x_1 and x_2 , used in this experiment were generated as follows: The x_1 -values have a uniform distribution in the interval $[-1, 1]$. The x_2 -values are nonlinearly related to the x_1 -values by the formula:

$$x_2 = x_1^2 + v$$

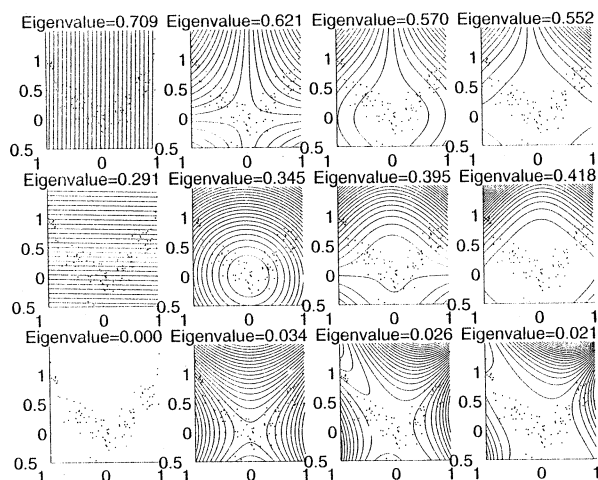


FIGURE 8.14 Two-dimensional example illustrating the kernel PCA. From left to right, the polynomial degree of the kernel is $d = 1, 2, 3, 4$. From top to bottom, the first three eigenvectors in the feature space are shown. The first column corresponds to ordinary PCA, and the other three columns correspond to kernel PCA with polynomial degree $d = 2, 3, 4$. (Reproduced with permission from Dr. Klaus-Robert Müller.)

where v is an additive Gaussian noise of zero mean and variance 0.04.

The results of PCA shown in Fig. 8.14 were obtained using kernel polynomials:

$$K(\mathbf{x}, \mathbf{x}_i) = (\mathbf{x}^T \mathbf{x}_i)^d, \quad d = 1, 2, 3, 4$$

where $d = 1$ corresponds to linear PCA, and $d = 2, 3, 4$ corresponds to kernel PCA. Linear PCA, shown on the left of Fig. 8.14, results in only two eigenvalues since the dimensionality of the input space is two. In contrast, kernel PCA permits the extraction of higher-order components, as shown by the results depicted in columns 2, 3, and 4 of Fig. 8.14, corresponding to polynomial degree $d = 2, 3, 4$, respectively. The contour lines shown in each part of the figure (except for the zero eigenvalue in the case of linear PCA) represent constant principal values (i.e., constant projections onto the eigenvector associated with the eigenvalue in question).

Based on the results shown in Fig. 8.14, we make the following observations:

- As expected, linear PCA fails to provide an adequate representation of the nonlinear input data.
- In all cases, the first principal component varies monotonically along a parabola that underlies the input data.
- In the kernel PCA, the second and third principal components exhibit a behavior that appears somewhat similar for different values of polynomial degree d .
- In the case of polynomial degree $d = 2$, the third principal component of kernel PCA appears to pick up the variance due to the additive Gaussian noise v . By removing the contribution due to this component, we would in effect be performing some form of noise reduction.

8.11 SUMMARY AND DISCUSSION

In this chapter we present material dealing with the theory of principal components analysis and the use of neural networks for its implementation. It is appropriate that we now reflect over that material and ask: How useful is principal components analysis? The answer to this question, of course, depends on the application of interest.

If the main objective is to achieve good *data compression* while preserving as much information about the inputs as possible, the use of principal components analysis offers a useful self-organized learning procedure. Here we note from the material presented in Section 8.3 that the use of a subspace decomposition method based on the “first l principal components” of the input data provides a linear mapping, which is optimum in the sense that it permits reconstruction of the original input data in the mean-square error sense. Moreover, a representation based on the first l principal components is preferable to an arbitrary subspace representation, because the principal components of the input data are naturally ordered in decreasing eigenvalue or, equivalently, decreasing variance. Accordingly, we may optimize the use of principal components analysis for data compression by employing the greatest numerical precision to encode the first principal component of the input, and progressively less precision to encode the remaining $l - 1$ components.

A related issue is the representation of a data set made up of an aggregate of several *clusters*. For the clusters to be individually visible, the separation between them has to be larger than the internal scatter of the clusters. If it so happens that there are only a few clusters in the data set, then the leading principal axes found by using the

principal components analysis will tend to pick projections of clusters with good separations, thereby providing an effective basis for *feature extraction*.

In this latter context we mention a useful application of a principal components analyzer as the *preprocessor* for a supervised neural network (e.g., multilayer perceptron trained with the back-propagation algorithm). The motivation here is to speed up convergence of the learning process by decorrelating the input data. A supervised learning procedure such as back-propagation relies on steepest descent. The convergence process in this form of learning is typically slow due to interacting effects of a multilayer perceptron's synaptic weights on the error signal, even with the use of simple local accelerating procedures such as momentum and adaptive learning rates for individual weights. If, however, the inputs to the multilayer perceptron consist of uncorrelated components, then from the discussion presented in Chapter 4 we note that the Hessian matrix of the cost function $\mathcal{E}(n)$ with respect to the free parameters of the network is more nearly diagonal than would be the case otherwise. With this form of diagonalization in place, the use of simple local accelerating procedures permits a considerable acceleration in the convergence process, which is made possible by appropriate scaling of the learning rates along each weight axis independently (Becker, 1991).

With the Hebbian-based algorithms of this chapter motivated by ideas taken from neurobiology, it is fitting to conclude our discussion by commenting on the role of principal components analysis in biological perceptual systems. Linsker (1990a) questions the "sufficiency" of principal components analysis as a principle for determining the response property developed by a neuron to analyze an ensemble of input "scenes." In particular, the optimality of principal components analysis with respect to the accurate reconstruction of an input signal from a neuron's response is considered to be of questionable relevance. In general, it appears that a brain does much more than simply try to reproduce the input scenes received by its sensory units. Rather, some underlying "meaningful cues" or features are extracted so as to permit high-level interpretations of the inputs. We may therefore sharpen the question we raised at the beginning of this discussion and ask: How useful is principal components analysis for perceptual processing?

Ambros-Ingerson et al. (1990) point out the significance of the algorithms set forth by Oja (1982) and Sanger (1989a) for principal components analysis (i.e., the Hebbian-inspired algorithms discussed in Sections 8.4 and 8.5) in a *hierarchical clustering algorithm*. They put forward the hypothesis that hierarchical clustering may emerge as a fundamental property (at least in part) of memories based on long-term potentiation (LTP)—like synaptic modifications of the kind found in cortico-bulbar networks and circuitry of similar design in other regions of the brain, and which property may be used for recognizing environmental cues. The point is that self-organized principal components analysis may be significant in hierarchical clustering of learned cues in the cerebral cortex not because of its optimal reconstruction property, but rather by virtue of its intrinsic property of picking projections of clusters with good separations.

Another interesting role for principal components analysis in perceptual processing appears in an approach to the *shape-from-shading* problem, proposed in Atick et al. (1996). The problem may be stated as follows: How is the brain able to perceive

a three-dimensional shape from the shading patterns cast in a two-dimensional image? Atick et al. propose a hierarchical solution to the shape-from-shading problem, consisting of two notions:

1. The brain, through evolution or prior experience, has discovered that objects can be classified into lower-dimensional object classes with regard to their shape. This notion in effect builds on the fact that the *cues* the brain uses to extract a three-dimensional interpretation are well understood.
2. In light of notion 1, the extraction of a shape from shading patterns is reduced to the much simpler problem of *parameter estimation* in a low-dimensional space.

For example, the gross structure of a human head shape is invariably the same, in the sense that all people have noses representing protrusions, eye sockets representing depressions, and foreheads and cheeks representing flatter regions. This invariance suggests that any given face, expressed as $r(\theta, l)$ in cylindrical coordinates, may be described as the sum of two components:

$$r(\theta, l) = r_0(\theta, l) + \rho(\theta, l)$$

where $r_0(\theta, l)$ denotes a *mean-head* for a particular category of people (e.g., male adults or female adults), and $\rho(\theta, l)$ denotes *perturbations* that capture the identity of a particular person. Typically, $\rho(\theta, l)$ is small compared to $r_0(\theta, l)$. To represent $\rho(\theta, l)$, Atick et al. use principal components analysis, whereby the fluctuations are represented in terms of a set of eigenfunctions (i.e., the two-dimensional counterparts of eigenvectors). Results are presented in Atick et al. (1996), demonstrating the ability of the two-stage hierarchical approach to recover the three-dimensional surface for a given person from a single two-dimensional image of that person.

NOTES AND REFERENCES

1. Principal components analysis (PCA) is perhaps the oldest and best known technique in multivariate analysis (Jolliffe, 1986; Preisendorfer, 1988). It was first introduced by Pearson (1901), who used it in a biological context to recast linear regression analysis into a new form. It was then developed by Hotelling (1933) in work done on psychometry. It appeared once again and quite independently in the setting of probability theory, as considered by Karhunen (1947); and was subsequently generalized by Loève (1963).
2. The approaches taken by Ljung (1977) and Kushner and Clark (1978) for studying the dynamical behavior of a stochastic approximation algorithm reduce the problem to that of studying the dynamics of an associated differential equation. However, these two approaches are fundamentally different. Ljung's approach involves the use of a Lyapunov function, whereas the approach taken by Kushner and Clark involves a linear interpolation process and invokes the so-called Arzelà–Ascoli theorem (Dunford and Schwartz, 1966). The approach by Kushner and Clark is followed in Diamantaras and Kung (1996) for studying the convergence of the Hebbian-based maximum eigenfilter. The conclusions reached therein are the same as those obtained using Ljung's approach.
3. Földiák (1989) expanded the neural network configuration for principal components analysis by including anti-Hebbian feedback connections. The motivation for this modification was derived from some earlier work by Barlow and Földiák (1989) on adaptation and decorrelation in the visual cortex; there it was argued that if the neurons interact according

to an anti-Hebbian rule, then the outputs of the neurons define a coordinate system in which there are no correlations even when the incoming signals have strong correlations.

The use of lateral inhibitions among the output neurons was also proposed by Rubner and Tavan (1989) and Rubner and Schulten (1990). However, unlike the model proposed by Földiák, the lateral network considered by Rubner et al. is not symmetrically connected. Rather, the lateral network is hierarchical, in that neuron i (say) inhibits all other neurons in the model except for $1, 2, \dots, i-1$, where $i = 1, 2, \dots$

The APEX model studied in Kung and Diamantaras (1990) has the same network topology as that of the model due to Rubner et al., but it uses Oja's single-neuron learning rule (described in Section 8.4) for adjusting the synaptic weights in both the feedforward and lateral connections in the model.

4. A rigorous proof of convergence of the APEX algorithm, with all the neurons tending to converge together, is given in Chen and Liu (1992).
5. For a discussion of several neural models for principal components analysis and their comparison, see the book by Diamantaras and Kung (1996).
6. Nonlinear PCA methods, excluding kernel PCA, may be grouped into three classes (Diamantaras and Kung, 1996):
 - *Hebbian networks*, which are obtained by replacing the linear neurons in Hebbian-based PCA algorithms with nonlinear neurons (Karhunen and Joutsensalo, 1995).
 - *Replicator networks* or *autoencoders*, which are built around multilayer perceptrons: replicator networks are discussed in Chapter 4.
 - *Principal curves*, which are based on an iterative estimation of a curve or surface capturing the structure of the data (Hastie and Stuetzle, 1989). In Ritter et al. (1992) and Cherkassky and Mulier (1995), it is pointed out that Kohonen's self-organizing map can be viewed as a computational procedure for finding a discrete approximation of principal curves; self-organizing maps are discussed in the next chapter.

PROBLEMS

Hebbian-based maximum eigenfilter

- 8.1 For the matched filter considered in Example 8.2, the eigenvalue λ_1 and associated eigenvector $\mathbf{q}_1$ are defined by

$$\lambda_1 = 1 + \sigma^2$$

$$\mathbf{q}_1 = \mathbf{s}$$

Show that these parameters satisfy the basic relation

$$\mathbf{R}\mathbf{q}_1 = \lambda_1\mathbf{q}_1$$

where $\mathbf{R}$ is the correlation matrix of the input vector $\mathbf{X}$.

- 8.2 Consider the maximum eigenfilter where the weight vector $\mathbf{w}(n)$ evolves in accordance with Eq. (8.46). Show that the variance of the filter output approaches $\lambda_{\max}$ as n approaches infinity, where $\lambda_{\max}$ is the largest eigenvalue of the correlation matrix of the input vector.
- 8.3 *Minor components analysis* (MCA) is the opposite of principal components analysis. In MCA, we seek to find those directions that *minimize* the projection variance. The directions so found are the eigenvectors corresponding to the smallest (minimum) eigenvalues of the correlation matrix $\mathbf{R}$ of the input vector $\mathbf{X}(n)$.

In this problem we explore how to modify the single neuron of Section 8.4 so as to find the minor component of $\mathbf{R}$. In particular, we make a change of sign in the learning rule of Eq. (8.40), obtaining (Xu et al., 1992)

$$w_i(n+1) = w_i(n) - \eta y(n)[x_i(n) - y(n)w_i(n)]$$

Show that if the smallest eigenvalue of the correlation matrix $\mathbf{R}$ is λ_m with multiplicity 1, then

$$\lim_{n \rightarrow \infty} \mathbf{w}(n) = \eta \mathbf{q}_m$$

where $\mathbf{q}_m$ is the eigenvector associated with λ_m .

Hebbian-based principal components analysis

- 8.4 Construct a signal-flow graph to represent the vector-valued Eqs. (8.87) and (8.88).
- 8.5 The ordinary differential equation approach to convergence analysis described in Section 8.4 does not apply directly to the generalized Hebbian-learning algorithm (GHA). However, by expressing the synaptic weight matrix $\mathbf{W}(n)$ in Eq. (8.91) as a vector made up of the individual columns of $\mathbf{W}(n)$, we may interpret the update function $h(\cdot, \cdot)$ in the usual way, then proceed to apply the asymptotic stability theorem. Hence, in light of what has been said here, explore the convergence theorem for the generalized Hebbian-learning algorithm.
- 8.6 In this problem we explore the use of the generalized Hebbian algorithm to study two-dimensional receptive fields produced by a random input (Sanger, 1990). The random input consists of a two-dimensional field of independent Gaussian noise with zero mean and unit variance, which is convolved with a Gaussian mask (filter) and then multiplied by a Gaussian window. The Gaussian mask has a standard deviation of 2 pixels, and the Gaussian window has a standard deviation of 8 pixels. The resulting random input $x(r, s)$ at position (r, s) may thus be written as follows:

$$x(r, s) = m(r, s)[g(r, s) * w(r, s)]$$

where $w(r, s)$ is the field of independent and identically distributed Gaussian noise, $g(r, s)$ is the Gaussian mask, and $m(r, s)$ is the Gaussian window function. The circular convolution of $g(r, s)$ and $w(r, s)$ is defined by

$$g(r, s) * w(r, s) = \sum_{p=0}^{N-1} \sum_{q=0}^{N-1} g(p, q)w(r-p, s-q)$$

where $g(r, s)$ and $w(r, s)$ are both assumed to be periodic.

Use 2000 samples of the random input $x(r, s)$ to train a single-layer feedforward network by means of the generalized Hebbian algorithm. The network has 4096 inputs arranged as a 64×64 grid of pixels, and 16 outputs. The resulting synaptic weights of the trained network are represented as 64×64 arrays of numbers. Perform the computations described herein and display the 16 arrays of synaptic weights as two-dimensional masks. Comment on your results.

- 8.7 Equation (8.113) defines the transformed version of the update equation (8.106) for computing the feedback weight vector $\mathbf{a}_j(n)$. The transformation is based on the definition of the synaptic weight vector $\mathbf{w}_j(n)$ in terms of the m principal modes of the network given in Eq. (8.109). Derive Eq. (8.113).

- 8.8 Consider the system matrix of Eq. (8.116), represented by the signal-flow graph of Fig. 8.12 which corresponds to $1 \leq k \leq j - 1$.
- Formulate the characteristic equation of this 2×2 matrix.
 - Show that the matrix has a double eigenvalue.
 - Justify the statement that all the principal modes of the network have the same eigenvalue.
- 8.9 The GHA uses forward connections only, whereas the APEX algorithm uses both forward and lateral connections. Yet despite these differences, the long-term convergence behavior of the APEX algorithm is in theory exactly the same as that of the GHA. Justify the validity of this statement.

Kernel PCA

- 8.10 Let $\bar{K}_{ij}$ denote the centered counterpart of the ij -th element K_{ij} of kernel matrix $\mathbf{K}$. Show that (Schölkopf, 1997)

$$\bar{K}_{ij} = K_{ij} - \frac{1}{N} \sum_{m=1}^N \varphi^T(\mathbf{x}_m) \varphi(\mathbf{x}_j) - \frac{1}{N} \sum_{n=1}^N \varphi^T(\mathbf{x}_i) \varphi(\mathbf{x}_n) + \frac{1}{N^2} \sum_{m=1}^N \sum_{n=1}^N \varphi^T(\mathbf{x}_m) \varphi(\mathbf{x}_n)$$

Suggest a compact representation of this relation in matrix form.

- 8.11 Show that the normalization of eigenvector α of the kernel matrix $\mathbf{K}$ is equivalent to the requirement that Eq. (8.153) be satisfied.
- 8.12 Summarize the properties of kernel PCA.

CHAPTER 9

Self-Organizing Maps

9.1 INTRODUCTION

In this chapter we continue our study of self-organizing systems by considering a special class of artificial neural networks known as self-organizing maps. These networks are based on *competitive learning*; the output neurons of the network compete among themselves to be activated or fired, with the result that only *one* output neuron, or one neuron per group, is on at any one time. An output neuron that wins the competition is called a *winner-takes-all neuron* or simply a *winning neuron*. One way of inducing a winner-takes-all competition among the output neurons is to use lateral inhibitory connections (i.e., negative feedback paths) between them; such an idea was originally proposed by Rosenblatt (1958).

In a *self-organizing map*, the neurons are placed at the nodes of a *lattice* that is usually one- or two-dimensional. Higher-dimensional maps are also possible but not as common. The neurons become *selectively tuned* to various input patterns (stimuli) or classes of input patterns in the course of a competitive learning process. The locations of the neurons so tuned (i.e., the winning neurons) become ordered with respect to each other in such a way that a meaningful coordinate system for different input *features* is created over the lattice (Kohonen, 1990a). A self-organizing map is therefore characterized by the formation of a *topographic map* of the input patterns in which the *spatial locations* (i.e., *coordinates*) of the neurons in the lattice are indicative of *intrinsic statistical features contained in the input patterns*, hence the name “self-organizing map.”

As a neural model, the self-organizing map provides a bridge between two levels of adaptation:

- Adaptation rules formulated at the microscopic level of a single neuron.
- Formation of experimentally better and physically accessible patterns of feature selectivity at the microscopic level of neural layers.

Because a self-organizing map is inherently nonlinear, it may thus be viewed as a nonlinear generalization of principal components analysis (Ritter, 1995).

The development of self-organizing maps as a neural model is motivated by a distinct feature of the human brain: The brain is organized in many places in such a way that different sensory inputs are represented by *topologically ordered computational*

maps. In particular, sensory inputs such as tactile (Kaas et al., 1983), visual (Hubel and Wiesel, 1962, 1977), and acoustic (Suga, 1985) are mapped onto different areas of the cerebral cortex in a topologically ordered manner. Thus the computational map constitutes a basic building block in the information-processing infrastructure of the nervous system. A computational map is defined by an array of neurons representing slightly differently tuned processors or filters, which operate on the sensory information-bearing signals in parallel. Consequently, the neurons transform input signals into a *place-coded probability distribution* that represents the computed values of parameters by sites of maximum relative activity within the map (Knudsen et al., 1987). The information so derived is of such a form that it can be readily accessed by higher-order processors using relatively simple connection schemes.

Organization of the Chapter

The material presented in this chapter on computational maps is organized as follows. In Section 9.2 we describe two feature-mapping models, which in their own individual ways are able to explain or capture the essential features of computational maps in the brain. The two models differ from each other in the form of the inputs used.

The rest of the chapter is devoted to detailed considerations of one of these models, commonly referred to as a “self-organizing map” due to Kohonen (1982). In Section 9.3 we use neurobiological considerations to develop a mathematical formalism of Kohonen’s model. A summary of the model is presented in Section 9.4. Important properties of the model are described in Section 9.5, which is followed by computer simulations in Section 9.6. The performance of the feature map may finally be fine-tuned through a supervised technique known as learning vector quantization; this technique is described in Section 9.7. Section 9.8 describes a computer experiment on adaptive pattern classification that combines the use of learning vector quantization and the self-organizing map. In Section 9.9 we describe hierarchical vector quantization built around the self-organizing map for data compression. Section 9.10 describes another application of the self-organizing map for building contextual maps that find applications in unsupervised categorization of phonemic classes from text, remote sensing, and data exploration. The chapter concludes with some final remarks in Section 9.12.

9.2 TWO BASIC FEATURE-MAPPING MODELS

Anyone who examines a human brain cannot help but be impressed by the extent to which the brain is dominated by the cerebral cortex. The brain is almost completely enveloped by the cerebral cortex, which obscures the other parts. For sheer complexity, the cerebral cortex probably exceeds any other known structure in the universe (Hubel and Wiesel, 1977). What is equally impressive is the way in which different sensory inputs (motor, somatosensory, visual, auditory, etc.) are *mapped* onto corresponding areas of the cerebral cortex in an *orderly* fashion; to appreciate this point, see the cytoarchitectural maps of the cerebral cortex in Fig. 2.4. The use of computational maps offers the following properties (Knudsen et al., 1987):

- At each stage of representation, each incoming piece of information is kept in its proper context.
- Neurons, dealing with closely related pieces of information, are close together so that they can interact via short synaptic connections.

Our interest lies in building artificial topographic maps that learn through self-organization in a neurobiologically inspired manner. In this context, the one important point that emerges from the very brief discussion of computational maps in the brain is the principle of topographic map formation, which may be stated as (Kohonen, 1990a):

The spatial location of an output neuron in a topographic map corresponds to a particular domain or feature of data drawn from the input space.

This principle has provided the neurobiological motivation for two basically different *feature-mapping models*¹ described herein.

Figure 9.1 displays the layout of the two models. In both cases the output neurons are arranged in a two-dimensional lattice. This kind of topology ensures that each neuron has a set of neighbors. The models differ from each other in the manner in which the input patterns are specified.

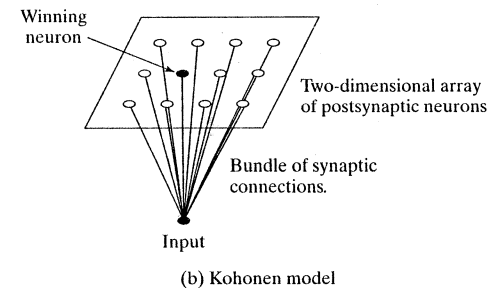
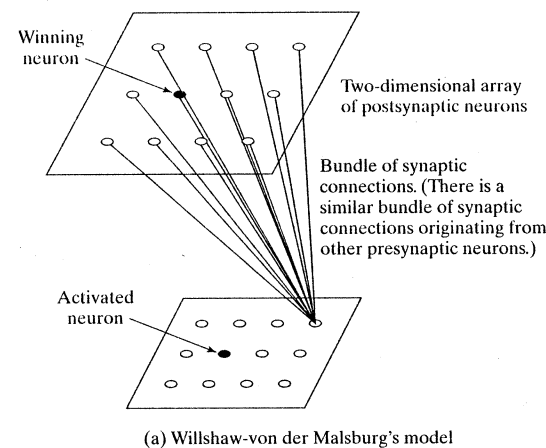


FIGURE 9.1 Two self-organized feature maps.