8.8 Consider the system matrix of Eq. (8.116), represented by the signal-flow graph of Fig. 8.12 which corresponds to $1 \leq k \leq j - 1$.

(a) Formulate the characteristic equation of this 2×2 matrix.

(b) Show that the matrix has a double eigenvalue.

(c) 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 $\boldsymbol{\alpha}$ 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 i proper context.
- Neurons, dealing with closely related pieces of information, are close togeth that they can interact via short synaptic connections.

Our interest lies in building artificial topographic maps that learn through s organization in a neurobiologically inspired manner. In this context, the one impor point that emerges from the very brief discussion of computational maps in the bra the *principle of topographic map formation*, which may be stated as (Kohonen, 199

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

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

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

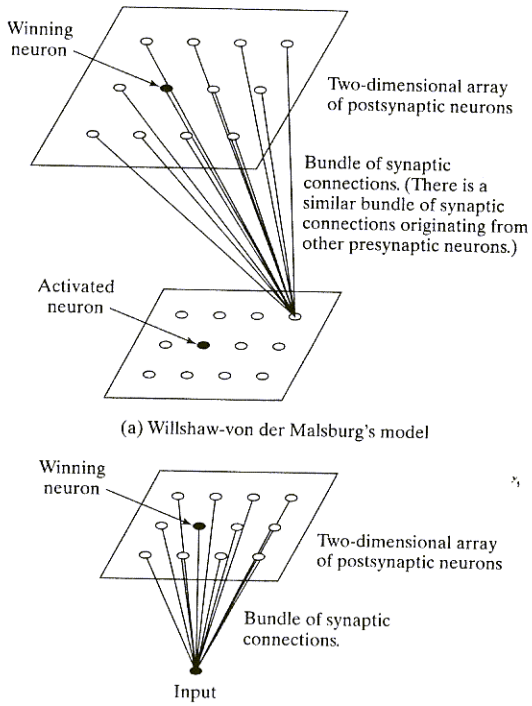

FIGURE 9.1 Two self-organized feature maps.

The model of Fig. 9.1a was originally proposed by Willshaw and von der Malsburg (1976) on biological grounds to explain the problem of retinotopic mapping from the retina to the visual cortex (in higher vertebrates). Specifically, there are two separate two-dimensional lattices of neurons connected together, with one projecting onto the other. One lattice represents presynaptic (input) neurons, and the other lattice represents postsynaptic (output) neurons. The postsynaptic lattice uses a *short-range excitatory mechanism* as well as a *long-range inhibitory mechanism*. These two mechanisms are local in nature and critically important for self-organization. The two lattices are interconnected by modifiable synapses of a Hebbian type. Strictly speaking, therefore, the postsynaptic neurons are not winner takes all; rather, a threshold is used to ensure that only a few postsynaptic neurons will fire at any one time. Moreover, to prevent a steady buildup in the synaptic weights that may lead to network instability, the total weight associated with each postsynaptic neuron is limited by an upper boundary condition.[2] Thus, for each neuron some synaptic weights increase while others decrease. The basic idea of the Willshaw–von der Malsburg model is for the geometric proximity of presynaptic neurons to be coded in the form of correlations in their electrical activity, and to use these correlations in the postsynaptic lattice so as to connect neighboring presynaptic neurons to neighboring postsynaptic neurons. A topologically ordered mapping is thereby produced by self-organization. Note, however, that the Willshaw–von der Malsburg model is specialized to mapping where the input dimension is the same as the output dimension.

The second model of Fig. 9.1b, introduced by Kohonen (1982), is not meant to explain neurobiological details. The model captures the essential features of computational maps in the brain and yet remains computationally tractable.[3] It appears that the Kohonen model is more general than the Willshaw–von der Malsburg model in the sense that it is capable of performing data compression (i.e., dimensionality reduction on the input).

In reality, the Kohonen model belongs to the class of *vector-coding* algorithms. The model provides a topological mapping that optimally places a fixed number of vectors (i.e., code words) into a higher-dimensional input space, and thereby facilitates data compression. The Kohonen model may therefore be derived in two ways. We may use basic ideas of self-organization, motivated by neurobiological considerations, to derive the model, which is the traditional approach (Kohonen, 1982, 1990a, 1997a). Alternatively, we may use a vector quantization approach that uses a model involving an encoder and a decoder, which is motivated by communication–theoretic considerations (Luttrell, 1989b, 1991a). In this chapter we consider both approaches.

The Kohonen model has received much more attention in the literature than the Willshaw–von der Malsburg model. It possesses certain properties discussed later in the chapter, which make it particularly interesting for understanding and modeling cortical maps in the brain. The remainder of the chapter is devoted to the derivation of the *self-organizing map*, its basic properties and ramifications.

9.3 SELF-ORGANIZING MAP

The principal goal of the self-organizing map (SOM) is to transform an incoming signal pattern of arbitrary dimension into a one- or two-dimensional discrete map, and to perform this transformation adaptively in a topologically ordered fashion. Figure 9.2 shows the schematic diagram of a two-dimensional lattice of neurons commonly used

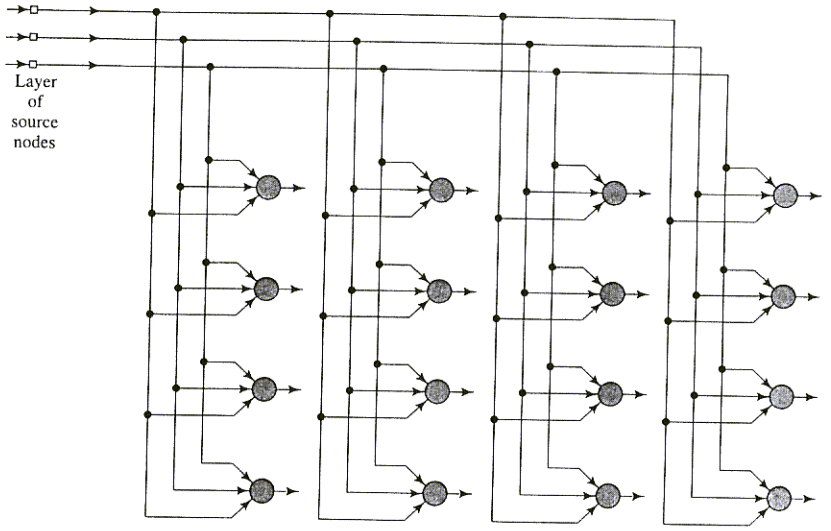

FIGURE 9.2 Two-dimensional lattice of neurons.

as the discrete map. Each neuron in the lattice is fully connected to all the source nodes in the input layer. This network represents a feedforward structure with a single computational layer consisting of neurons arranged in rows and columns. A one-dimensional lattice is a special case of the configuration depicted in Fig. 9.2: in this special case the computational layer consists simply of a single column or row of neurons.

Each input pattern presented to the network typically consists of a localized region or "spot" of activity against a quiet background. The location and nature of such a spot usually varies from one realization of the input pattern to another. All the neurons in the network should therefore be exposed to a sufficient number of different realizations of the input pattern to ensure that the self-organization process has a chance to mature properly.

The algorithm responsible for the formation of the self-organizing map proceeds first by *initializing* the synaptic weights in the network. This can be done by assigning them *small values picked from a random number generator*; in so doing, no prior order is imposed on the feature map. Once the network has been properly initialized, there are three essential processes involved in the formation of the self-organizing map, as summarized here:

1. *Competition.* For each input pattern, the neurons in the network compute their respective values of a discriminant function. This discriminant function provides the basis for competition among the neurons. The particular neuron with the largest value of discriminant function is declared winner of the competition.
2. *Cooperation.* The winning neuron determines the spatial location of a topological neighborhood of excited neurons, thereby providing the basis for cooperation among such neighboring neurons.

3. *Synaptic Adaptation.* This last mechanism enables the excited neurons to increase their individual values of the discriminant function in relation to the input pattern through suitable adjustments applied to their synaptic weights. The adjustments made are such that the response of the winning neuron to the subsequent application of a similar input pattern is enhanced.

The processes of competition and cooperation are in accordance with two of the four principles of self-organization described in Chapter 8. As for the principle of self-amplification, it comes in a modified form of Hebbian learning in the adaptive process. As explained in Chapter 8, the presence of redundancy in the input data (though not mentioned explicitly in describing the SOM algorithm) is needed for learning since it provides knowledge. Detailed descriptions of the processes of competition, cooperation, and synaptic adaptation are now presented.

Competitive Process

Let m denote the dimension of the input (data) space. Let an input pattern (vector) selected at random from the input space be denoted by

$$\mathbf{x} = [x_1, x_2, \ldots, x_m]^T \tag{9.1}$$

The synaptic weight vector of each neuron in the network has the same dimension as the input space. Let the synaptic weight vector of neuron j be denoted by

$$\mathbf{w}_j = [w_{j1}, w_{j2}, \ldots, w_{jm}]^T, \qquad j = 1, 2, \ldots, l \tag{9.2}$$

where l is the total number of neurons in the network. To find the best match of the input vector $\mathbf{x}$ with the synaptic weight vectors $\mathbf{w}_j$, compare the inner products $\mathbf{w}_j^T\mathbf{x}$ for $j = 1, 2, \ldots, l$ and select the largest. This assumes that the same threshold is applied to all the neurons; the threshold is the negative of bias. Thus, by selecting the neuron with the largest inner product $\mathbf{w}_j^T\mathbf{x}$, we will have in effect determined the location where the topological neighborhood of excited neurons is to be centered.

From Chapter 1 we recall that the best matching criterion, based on maximizing the inner product $\mathbf{w}_j^T\mathbf{x}$, is mathematically equivalent to minimizing the Euclidean distance between the vectors $\mathbf{x}$ and $\mathbf{w}_j$. If we use the index $i(\mathbf{x})$ to identify the neuron that best matches the input vector $\mathbf{x}$, we may then determine $i(\mathbf{x})$ by applying the condition[4]

$$i(\mathbf{x}) = \arg\min_j \|\mathbf{x} - \mathbf{w}_j\|, \qquad j = 1, 2, \ldots, l \tag{9.3}$$

which sums up the essence of the competition process among the neurons. According to Eq. (9.3), $i(\mathbf{x})$ is the subject of attention because we want the identity of neuron i. The particular neuron i that satisfies this condition is called the *best-matching* or *winning neuron* for the input vector $\mathbf{x}$. Equation (9.3) leads to this observation:

> *A continuous input space of activation patterns is mapped onto a discrete output space of neurons by a process of competition among the neurons in the network.*

Depending on the application of interest, the response of the network could be either the index of the winning neuron (i.e., its position in the lattice), or the synaptic weight vector that is closest to the input vector in a Euclidean sense.

Cooperative Process

The winning neuron locates the center of a topological neighborhood of cooperating neurons. The key question is: How do we define a topological neighborhood that is neurobiologically correct? To answer this question, remember that there is neurobiological evidence for *lateral interaction* among a set of excited neurons. In particular, a neuron that is firing tends to excite the neurons in its immediate neighborhood more than those farther away from it, which is intuitively satisfying. This observation leads us to make the topological neighborhood around the winning neuron i decay smoothly with lateral distance[5] (Lo et al., 1991, 1993; Ritter et al., 1992). To be specific, let $h_{j,i}$ denote the *topological neighborhood* centered on winning neuron i, and encompassing a set of excited (cooperating) neurons, a typical one of which is denoted by j. Let $d_{j,i}$ denote the *lateral* distance between winning neuron i and excited neuron j. Then we may assume that the topological neighborhood $h_{j,i}$ is a unimodal function of the lateral distance $d_{j,i}$, such that it satisfies two distinct requirements:

- The topological neighborhood $h_{j,i}$ is symmetric about the maximum point defined by $d_{i,j} = 0$; in other words, it attains its maximum value at the winning neuron i for which the distance $d_{j,i}$ is zero.
- The amplitude of the topological neighborhood $h_{j,i}$ decreases monotonically with increasing lateral distance $d_{j,i}$, decaying to zero for $d_{j,i} \to \infty$; this is a necessary condition for convergence.

A typical choice of $h_{j,i}$ that satisfies these requirements is the *Gaussian* function[6]

$$h_{j,i(\mathbf{x})} = \exp\left(-\frac{d_{j,i}^2}{2\sigma^2}\right) \tag{9.4}$$

which is *translation invariant* (i.e., independent of the location of the winning neuron). The parameter σ is the "effective width" of the topological neighborhood as illustrated in Fig. 9.3; it measures the degree to which excited neurons in the vicinity of the winning neuron participate in the learning process. In a qualitative sense, the Gaussian topological neighborhood of Eq. (9.4) is more biologically appropriate than a rectangular one

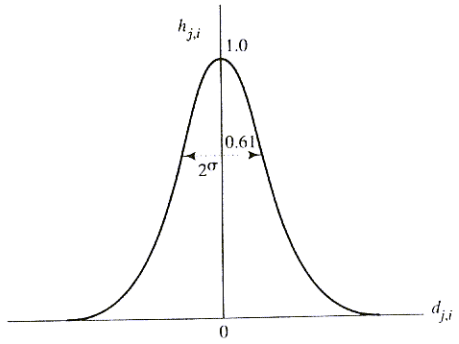

FIGURE 9.3 Gaussian neighborhood function.

Its use also makes the SOM algorithm converge more quickly than a rectangular topological neighborhood would (Lo et al., 1991, 1993; Erwin et al., 1992a).

For cooperation among neighboring neurons to hold, it is necessary that topological neighborhood $h_{j,i}$ be dependent on lateral distance $d_{j,i}$ between winning neuron i and excited neuron j in the output space rather than on some distance measure in the original input space. This is precisely what we have in Eq. (9.4). In the case of a one-dimensional lattice, $d_{j,i}$ is an integer equal to $|j-i|$. On the other hand, in the case of a two-dimensional lattice it is defined by

$$d_{j,i}^2 = \|\mathbf{r}_j - \mathbf{r}_i\|^2 \tag{9.5}$$

where the discrete vector $\mathbf{r}_j$ defines the position of excited neuron j and $\mathbf{r}_i$ defines the discrete position of winning neuron i, both of which are measured in the discrete output space.

Another unique feature of the SOM algorithm is that the size of the topological neighborhood shrinks with time. This requirement is satisfied by making the width σ of the topological neighborhood function $h_{j,i}$ decrease with time. A popular choice for the dependence of σ on discrete time n is the exponential decay described by (Ritter et al., 1992; Obermayer et al., 1991)

$$\sigma(n) = \sigma_0 \exp\left(-\frac{n}{\tau_1}\right) \qquad n = 0, 1, 2, \ldots, \tag{9.6}$$

where σ_0 is the value of σ at the initiation of the SOM algorithm, and τ_1 is a *time constant*. Correspondingly, the topological neighborhood assumes a time-varying form of its own, as shown by

$$h_{j,i(\mathbf{x})}(n) = \exp\left(-\frac{d_{j,i}^2}{2\sigma^2(n)}\right), \qquad n = 0, 1, 2, \ldots, \tag{9.7}$$

where $\sigma(n)$ is defined by Eq. (9.6). Thus, as time n (i.e., the number of iterations) increases, the width $\sigma(n)$ decreases at an exponential rate, and the topological neighborhood shrinks in a corresponding manner. Henceforth we will refer to $h_{j,i(\mathbf{x})}(n)$ as the *neighborhood function*.

Another useful way of viewing the variation of the neighborhood function $h_{j,i(\mathbf{x})}(n)$ around a winning neuron $i(\mathbf{x})$ is as follows (Luttrell, 1989a). The purpose of a wide $h_{j,i(\mathbf{x})}(n)$ is essentially to *correlate* the directions of the weight updates of a large number of excited neurons in the lattice. As the width of $h_{j,i(\mathbf{x})}(n)$ is decreased, so is the number of neurons whose update directions are correlated. This phenomenon becomes particularly obvious when the training of a self-organizing map is played on a computer graphics screen. It is rather wasteful of computer resources to move a large number of degrees of freedom around a winning neuron in a correlated fashion, as in the standard SOM algorithm. Instead it is much better to use a *renormalized* SOM form of training, according to which we work with a much smaller number of *normalized degrees of freedom*. This operation is easily performed in discrete form by having a neighborhood function $h_{j,i(\mathbf{x})}(n)$ of *constant* width, but gradually *increasing* the total number of neurons. The new neurons are inserted halfway between the old ones, and the smoothness properties of the SOM algorithm guarantee that the new ones join the

synaptic adaptation in a graceful manner (Luttrell, 1989a). A summary of the re
malized SOM algorithm is presented in Problem 9.13.

Adaptive Process

Now we come to the last process, the synaptic adaptive process, in the self-organi
formation of a feature map. For the network to be self-organizing, the synaptic wei
vector $\mathbf{w}_j$ of neuron j in the network is required to change in relation to the input v
tor $\mathbf{x}$. The question is how to make the change. In Hebb's postulate of learning, a syn
tic weight is increased with a simultaneous occurrence of presynaptic and postsynaj
activities. The use of such a rule is well suited for associative learning. For the type
unsupervised learning being considered here, however, the Hebbian hypothesis in
basic form is unsatisfactory for the following reason: Changes in connectivities occu
one direction only, which finally drive all the synaptic weights into saturation. To ov
come this problem we modify the Hebbian hypothesis by including a *forgetting term*
$g(y_j)\mathbf{w}_j$, where $\mathbf{w}_j$ is the synaptic weight vector of neuron j and $g(y_j)$ is some positi
scalar function of the response y_j. The only requirement imposed on the function $g($
is that the constant term in the Taylor series expansion of $g(y_j)$ be zero, so that we m
write

$$g(y_j) = 0 \qquad \text{for } y_j = 0 \tag{9.}$$

The significance of this requirement will become apparent momentarily. Given such
function, we may then express the change to the weight vector of neuron j in the latti
as follows:

$$\Delta\mathbf{w}_j = \eta y_j \mathbf{x} - g(y_j)\mathbf{w}_j \tag{9.}$$

where η is the *learning-rate parameter* of the algorithm. The first term on the right-han
side of Eq. (9.9) is the Hebbian term and the second term is the forgetting term. To sa
isfy the requirement of Eq. (9.8), we choose a linear function for $g(y_j)$, as shown by

$$g(y_j) = \eta y_j \tag{9.10}$$

We may further simplify Eq. (9.9) by setting

$$y_j = h_{j,i(\mathbf{x})} \tag{9.11}$$

Using Eqs. (9.10) and (9.11) in (9.9), we obtain

$$\Delta\mathbf{w}_j = \eta h_{j,i(\mathbf{x})}(\mathbf{x} - \mathbf{w}_j) \tag{9.12}$$

Finally, using discrete-time formalism, given the synaptic weight vector $\mathbf{w}_j(n)$ o
neuron j at time n, the updated weight vector $\mathbf{w}_j(n+1)$ at time $n+1$ is defined by
(Kohonen, 1982; Ritter et al., 1992; Kohonen, 1997a):

$$\mathbf{w}_j(n+1) = \mathbf{w}_j(n) + \eta(n)h_{j,i(\mathbf{x})}(n)(\mathbf{x} - \mathbf{w}_j(n)) \tag{9.13}$$

which is applied to all the neurons in the lattice that lie inside the topological neigh-
borhood of winning neuron i. Equation (9.13) has the effect of moving the synaptic

weight vector $\mathbf{w}_i$ of winning neuron i toward the input vector $\mathbf{x}$. Upon repeated presentations of the training data, the synaptic weight vectors tend to follow the distribution of the input vectors due to the neighborhood updating. The algorithm therefore leads to a *topological ordering* of the feature map in the input space in the sense that neurons that are adjacent in the lattice will tend to have similar synaptic weight vectors. We have more to say on this issue in Section 9.5.

Equation (9.13) is the desired formula for computing the synaptic weights of the feature map. In addition to this equation, however, we need the heuristic of Eq. (9.7) for selecting the neighborhood function $h_{j,i(\mathbf{x})}(n)$ and another heuristic for selecting the learning-rate parameter $\eta(n)$.

The learning-rate parameter $\eta(n)$ should be time varying as indicated in Eq. (9.13), which is how it should be for stochastic approximation. In particular, it should start at an initial value η_0, and then decrease gradually with increasing time n. This requirement can be satisfied by choosing an exponential decay for $\eta(n)$, as shown by

$$\eta(n) = \eta_0 \exp\left(-\frac{n}{\tau_2}\right), \qquad n = 0, 1, 2, \ldots, \tag{9.14}$$

where τ_2 is another time constant of the SOM algorithm. Even though the exponential decay formulas described in Eqs. (9.6) and (9.14) for the width of the neighborhood function and the learning-rate parameter, respectively, may not be optimal, they are usually adequate for the formation of the feature map in a self-organized manner.

Two Phases of the Adaptive Process: Ordering and Convergence

Starting from an initial state of complete disorder, it is amazing how the SOM algorithm gradually leads to an organized representation of activation patterns drawn from the input space, provided that the parameters of the algorithm are selected properly. We may decompose the adaptation of the synaptic weights in the network, computed in accordance with Eq. (9.13), into two phases: an ordering or self-organizing phase followed by a convergence phase. These two phases of the adaptive process are described as follows (Kohonen, 1982, 1997a):

1. *Self-organizing* or *ordering phase*. It is during this first phase of the adaptive process that the topological ordering of the weight vectors takes place. The ordering phase may take as many as 1000 iterations of the SOM algorithm, and possibly more. Careful considerations must be given to the choice of the learning-rate parameter and neighborhood function:
   - The learning-rate parameter $\eta(n)$ should begin with a value close to 0.1; thereafter it should decrease gradually, but remain above 0.01. These desirable values are satisfied by the following choices in the formula of Eq. (9.14):

     $$\eta_0 = 0.1$$

     $$\tau_2 = 1000$$

   - The neighborhood function $h_{j,i}(n)$ should initially include almost all neurons in the network centered on the winning neuron i, and then shrink slowly with time.

Specifically, during the ordering phase that may occupy 1000 iterations or more, $h_{j,i}(n)$ is permitted to reduce to a small value of only a couple of neighboring neurons around a winning neuron or to the winning neuron itself. Assuming the use of a two-dimensional lattice of neurons for the discrete output space, we may thus set the initial size σ_0 of the neighborhood function equal to the "radius" of the lattice. Correspondingly, we may set the time constant τ_1 in the formula of Eq. (9.6) as follows:

$$\tau_1 = \frac{1000}{\log \sigma_0}$$

2. *Convergence phase*. This second phase of the adaptive process is needed to fine tune the feature map and therefore provide an accurate statistical quantification of the input space. As a general rule, the number of iterations constituting the convergence phase must be at least 500 times the number of neurons in the network. Thus, the convergence phase may have to go on for thousands and possibly tens of thousands of iterations:
   - For good statistical accuracy, the learning parameter $\eta(n)$ should be maintained during the convergence phase at a small value, on the order of 0.01. In any event, it must not be allowed to decrease to zero; otherwise, it is possible for the network to get stuck in a metastable state. A *metastable state* refers to a configuration of the feature map with a topological defect. The exponential decay of Eq. (9.14) guarantees against the possibility of metastable states.
   - The neighborhood function $h_{j,i(\mathbf{x})}$ should contain only the nearest neighbors of a winning neuron, which may eventually reduce to one or zero neighboring neurons.

9.4 SUMMARY OF THE SOM ALGORITHM

The essence of Kohonen's SOM algorithm is that it substitutes a simple geometric computation for the more detailed properties of the Hebb-like rule and lateral interactions. The essential ingredients/parameters of the algorithm are:

- A continuous input space of activation patterns that are generated in accordance with a certain probability distribution.
- A topology of the network in the form of a lattice of neurons, which defines a discrete output space.
- A time-varying neighborhood function $h_{j,i(\mathbf{x})}(n)$ that is defined around a winning neuron $i(\mathbf{x})$.
- A learning-rate parameter $\eta(n)$ that starts at an initial value η_0 and then decreases gradually with time, n, but never goes to zero.

For the neighborhood function and learning-rate parameter, we may use Eqs. (9.7) and (9.14), respectively, for the ordering phase (i.e., the first thousand iterations or so). For good statistical accuracy, $\eta(n)$ should be maintained at a small value (0.01 or less) during the convergence for a fairly long period of time, which is typically thousands of iterations. As for the neighborhood function, it should contain only the nearest neighl

of the winning neuron at the start of the convergence phase, and may eventually shrink to one or zero neighboring neurons.

There are three basic steps involved in the application of the algorithm after initialization: sampling, similarity matching, and updating. These three steps are repeated until formation of the feature map has completed. The algorithm is summarized as follows:

1. *Initialization.* Choose random values for the initial weight vectors $\mathbf{w}_j(0)$. The only restriction here is that the $\mathbf{w}_j(0)$ be different for $j = 1, 2, \ldots, l$, where l is the number of neurons in the lattice. It may be desirable to keep the magnitude of the weights small.

   Another way of initalizing the algorithm is to select the weight vectors $\{\mathbf{w}_j(0)\}_{j=1}^{l}$ from the available set of input vectors $\{\mathbf{x}_i\}_{i=1}^{N}$ in a random manner.
2. *Sampling.* Draw a sample $\mathbf{x}$ from the input space with a certain probability; the vector $\mathbf{x}$ represents the activation pattern that is applied to the lattice. The dimension of vector $\mathbf{x}$ is equal to m.
3. *Similarity Matching.* Find the best-matching (winning) neuron $i(\mathbf{x})$ at time step n by using the minimum-distance Euclidean criterion:

$$i(\mathbf{x}) = \arg\min_j \|\mathbf{x}(n) - \mathbf{w}_j\|, \quad j = 1, 2, \ldots, l$$

4. *Updating.* Adjust the synaptic weight vectors of all neurons by using the update formula

$$\mathbf{w}_j(n+1) = \mathbf{w}_j(n) + \eta(n) h_{j,i(\mathbf{x})}(n)(\mathbf{x}(n) - \mathbf{w}_j(n))$$

   where $\eta(n)$ is the learning-rate parameter, and $h_{j,i(\mathbf{x})}(n)$ is the neighborhood function centered around the winning neuron $i(\mathbf{x})$; both $\eta(n)$ and $h_{j,i(\mathbf{x})}(n)$ are varied dynamically during learning for best results.
5. *Continuation.* Continue with step 2 until no noticeable changes in the feature map are observed.

9.5 PROPERTIES OF THE FEATURE MAP

Once the SOM algorithm has converged, the *feature map* computed by the algorithm displays important statistical characteristics of the input space.

To begin with, let $\mathcal{X}$ denote a *spatially continuous input (data) space*, the topology of which is defined by the metric relationship of the vectors $\mathbf{x} \in \mathcal{X}$. Let $\mathcal{A}$ denote a *spatially discrete output space*, the topology of which is endowed by arranging a set of neurons as the computation nodes of a lattice. Let Φ denote a nonlinear transformation called a *feature map*, which maps the input space $\mathcal{X}$ onto the output space $\mathcal{A}$, as shown by

$$\Phi: \mathcal{X} \rightarrow \mathcal{A} \tag{9.15}$$

Equation (9.15) may be viewed as an abstraction of Eq. (9.3) that defines the location of a winning neuron $i(\mathbf{x})$ developed in response to an input vector $\mathbf{x}$. For example, in a neurobiological context, the input space $\mathcal{X}$ may represent the coordinate set of somatosensory receptors distributed densely over the entire body surface. Correspondingly, the output space $\mathcal{A}$ represents the set of neurons located in that layer of the cerebral cortex to which the somatosensory receptors are confined.

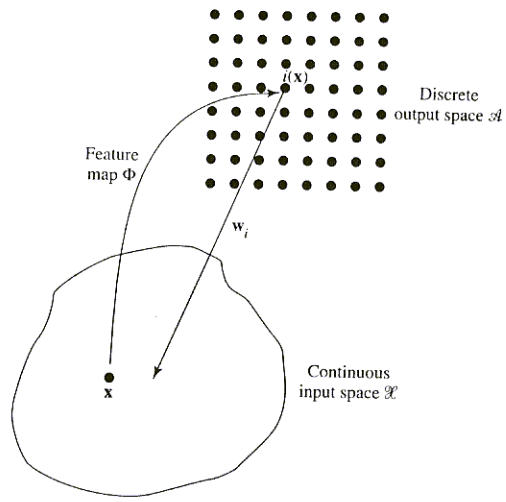

FIGURE 9.4 Illustration of the relationship between feature map Φ and weight vector $\mathbf{w}_i$ of winning neuron i.

Given an input vector $\mathbf{x}$, the SOM algorithm proceeds by first identifying a best-matching or winning neuron $i(\mathbf{x})$ in the output space $\mathcal{A}$, in accordance with the feature map Φ. The synaptic weight vector $\mathbf{w}_i$ of neuron $i(\mathbf{x})$ may then be viewed as a *pointer* for that neuron into the input space $\mathcal{X}$; that is, the synaptic elements of vector $\mathbf{w}_i$ may be viewed as the coordinates of the *image* of neuron i projected in the input space. These two operations are depicted in Fig. 9.4.

The feature map Φ has some important properties:

Property 1. Approximation of the Input Space. *The feature map Φ, represented by the set of synaptic weight vectors $\{\mathbf{w}_j\}$ in the output space $\mathcal{A}$, provides a good approximation to the input space $\mathcal{X}$.*

The basic aim of the SOM algorithm is to store a large set of input vectors $\mathbf{x} \in \mathcal{X}$ by finding a smaller set of prototypes $\mathbf{w}_j \in \mathcal{A}$, so as to provide a good approximation to the original input space $\mathcal{X}$. The theoretical basis of the idea just described is rooted in *vector quantization theory*, the motivation for which is dimensionality reduction or data compression (Gersho and Gray, 1992). It is therefore appropriate to present a brief discussion of this theory.

Consider Fig. 9.5, where $\mathbf{c}(\mathbf{x})$ acts as an *encoder* of the input vector $\mathbf{x}$ and $\mathbf{x}'(\mathbf{c})$ acts as a *decoder* of $\mathbf{c}(\mathbf{x})$. The vector $\mathbf{x}$ is selected at random from a training sample (i.e., input space $\mathcal{X}$), subject to an underlying probability density function $f_{\mathbf{X}}(\mathbf{x})$. The optimum encoding-decoding scheme is determined by varying the functions $\mathbf{c}(\mathbf{x})$ and $\mathbf{x}'(\mathbf{c})$, so as to minimize the *expected distortion* defined by

$$D = \frac{1}{2}\int_{-\infty}^{\infty} d\mathbf{x} f_{\mathbf{X}}(\mathbf{x}) d(\mathbf{x}, \mathbf{x}') \tag{9.16}$$

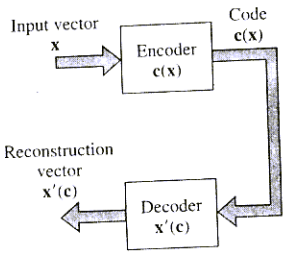

FIGURE 9.5 Encoder–decoder model.

where the factor 1/2 has been introduced for convenience of presentation, and $d(\mathbf{x},\mathbf{x}')$ is a *distortion* measure. The integration is performed over the entire input space $\mathscr{X}$ assumed to be of dimensionality m. A popular choice for the distortion measure $d(\mathbf{x},\mathbf{x}')$ is the square of the Euclidean distance between the input vector $\mathbf{x}$ and the reconstruction vector $\mathbf{x}'$; that is,

$$\begin{aligned} d(\mathbf{x},\mathbf{x}') &= \|\mathbf{x}-\mathbf{x}'\|^2 \\ &= (\mathbf{x}-\mathbf{x}')^T(\mathbf{x}-\mathbf{x}') \end{aligned} \tag{9.17}$$

Thus we may rewrite Eq. (9.16) as

$$D = \frac{1}{2}\int_{-\infty}^{\infty} d\mathbf{x} f_{\mathbf{X}}(\mathbf{x})\|\mathbf{x}-\mathbf{x}'\|^2 \tag{9.18}$$

The necessary conditions for the minimization of the expected distortion D are embodied in the *generalized Lloyd algorithm*[7] (Gersho and Gray, 1992). The conditions are twofold:

Condition 1. Given the input vector $\mathbf{x}$, choose the code $\mathbf{c} = \mathbf{c}(\mathbf{x})$ to minimize the squared error distortion $\|\mathbf{x}-\mathbf{x}'(\mathbf{c})\|^2$.

Condition 2. Given the code $\mathbf{c}$, compute the reconstruction vector $\mathbf{x}' = \mathbf{x}'(\mathbf{c})$ as the centroid of those input vectors $\mathbf{x}$ that satisfy condition 1.

Condition 1 is recognized as a *nearest-neighbor* encoding rule. Conditions 1 and 2 imply that the average distortion D is stationary (i.e., at a local minimum) with respect to variations in the encoder $\mathbf{c}(\mathbf{x})$ and decoder $\mathbf{x}'(\mathbf{c})$, respectively. To implement vector quantization, the generalized Lloyd algorithm operates in a *batch* training mode. Basically, the algorithm consists of alternately optimizing the encoder $\mathbf{c}(\mathbf{x})$ in accordance with condition 1, and then optimizing the decoder $\mathbf{x}'(\mathbf{c})$ in accordance with condition 2 until the expected distortion D reaches a minimum. In order to overcome the local-minimum problem, it may be necessary to run the generalized Lloyd algorithm several times with different initial code vectors.

The generalized Lloyd algorithm is closely related to the SOM algorithm, as shown in Luttrell (1989b). We may delineate the form of this relationship by considering the scheme shown in Fig. 9.6, where we have introduced a signal-independent *noise*

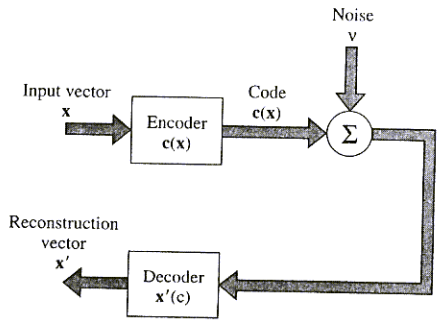

FIGURE 9.6 Noisy encoder–decoder model.

process $\boldsymbol{\nu}$ following the encoder $\mathbf{c}(\mathbf{x})$. The noise $\boldsymbol{\nu}$ is associated with a fictitious "communication channel" between the encoder and the decoder, the purpose of which is to account for the possibility that the output code $\mathbf{c}(\mathbf{x})$ may be distorted. On the basis of the model shown in Fig. 9.6, we may consider a *modified* form of expected distortion as follows:

$$D_1 = \frac{1}{2}\int_{-\infty}^{\infty} d\mathbf{x} f_{\mathbf{X}}(\mathbf{x}) \int_{-\infty}^{\infty} d\boldsymbol{\nu}\pi(\boldsymbol{\nu})\|\mathbf{x}-\mathbf{x}'(\mathbf{c}(\mathbf{x})+\boldsymbol{\nu})\|^2 \tag{9.19}$$

where $\pi(\boldsymbol{\nu})$ is the probability density function (pdf) of the additive noise $\boldsymbol{\nu}$, and the second integration is over all possible realizations of this noise.

In accordance with the strategy described for the generalized Lloyd algorithm, there are two separate optimizations to be considered for the model of Fig. 9.6, one pertaining to the encoder and the other pertaining to the decoder. To find the optimum encoder for a given $\mathbf{x}$, we need the partial derivative of the expected distortion measure D_1 with respect to the encoded vector $\mathbf{c}$. Using Eq. (9.19), we thus obtain

$$\frac{\partial D_1}{\partial \mathbf{c}} = \frac{1}{2} f_{\mathbf{X}}(\mathbf{x}) \int_{-\infty}^{\infty} d\boldsymbol{\nu}\pi(\boldsymbol{\nu}) \frac{\partial}{\partial \mathbf{c}} \|\mathbf{x}-\mathbf{x}'(\mathbf{c})\|^2 \,|_{\mathbf{c}=\mathbf{c}(\mathbf{x})+\boldsymbol{\nu}} \tag{9.20}$$

To find the optimum decoder for a given $\mathbf{c}$, we need the partial derivative of the expected distortion measure D_1 with respect to the decoded vector $\mathbf{x}'(\mathbf{c})$. Using Eq. (9.19), we thus obtain

$$\frac{\partial D_1}{\partial \mathbf{x}'(\mathbf{c})} = -\int_{-\infty}^{\infty} d\mathbf{x} f_{\mathbf{X}}(\mathbf{x})\pi(\mathbf{c}-\mathbf{c}(\mathbf{x}))\,(\mathbf{x}-\mathbf{x}'(\mathbf{c})) \tag{9.21}$$

Hence, in light of Eqs. (9.20) and (9.21), conditions 1 and 2 stated earlier for the generalized Lloyd algorithm must be modified as follows (Luttrell, 1989b):

Condition I. Given the input vector $\mathbf{x}$, choose the code $\mathbf{c} = \mathbf{c}(\mathbf{x})$ to minimize the distortion measure

$$D_2 = \int_{-\infty}^{\infty} d\boldsymbol{\nu}\pi(\boldsymbol{\nu})\|\mathbf{x}-\mathbf{x}'(\mathbf{c}(\mathbf{x})+\boldsymbol{\nu})\|^2 \tag{9.22}$$

Condition II. Given the code $\mathbf{c}$, compute the reconstruction vector $\mathbf{x}'(\mathbf{c})$ to satisfy the condition

$$\mathbf{x}'(\mathbf{c}) = \frac{\int_{-\infty}^{\infty} d\mathbf{x} f_{\mathbf{X}}(\mathbf{x})\pi(\mathbf{c} - \mathbf{c}(\mathbf{x}))\mathbf{x}}{\int_{-\infty}^{\infty} d\mathbf{x} f_{\mathbf{X}}(\mathbf{x})\pi(\mathbf{c} - \mathbf{c}(\mathbf{x}))} \tag{9.23}$$

Equation (9.23) is obtained by setting the partial derivative $\partial D_1/\partial \mathbf{x}'(\mathbf{c})$ in Eq. (9.21) equal to zero and then solving for $\mathbf{x}'(\mathbf{c})$.

The model described in Fig. 9.5 may be viewed as a special case of that shown in Fig. 9.6. In particular, if we set the probability density function $\pi(\boldsymbol{\nu})$ of the noise $\boldsymbol{\nu}$ equal to a Dirac delta function $\delta(\boldsymbol{\nu})$, conditions I and II reduce to conditions 1 and 2 for the generalized Lloyd algorithm, respectively.

To simplify condition I, we assume that $\pi(\boldsymbol{\nu})$ is a smooth function of $\boldsymbol{\nu}$. It may then be shown that, to a second-order of approximation, the distortion measure D_2 defined in Eq. (9.22) consists of two components (Luttrell, 1989b):

- The *conventional* distortion term, defined by the squared error distortion $\|\mathbf{x} - \mathbf{x}'(\mathbf{c})\|^2$
- A *curvature* term that arises from the noise model $\pi(\boldsymbol{\nu})$

Assuming that the curvature term is small, condition I for the model of Fig. 9.6 may be approximated by condition 1 for the noiseless model of Fig. 9.5. This in turn reduces condition I to a nearest-neighbor encoding rule as before.

As for condition II, we may realize it by using stochastic descent learning. In particular, we choose input vectors $\mathbf{x}$ at random from the input space $\mathscr{X}$ using the factor $\int d\mathbf{x} f_{\mathbf{X}}(\mathbf{x})$, and update the reconstruction vector $\mathbf{x}'(\mathbf{c})$ as follows (Luttrell, 1989b):

$$\mathbf{x}'_{\text{new}}(\mathbf{c}) \leftarrow \mathbf{x}'_{\text{old}}(\mathbf{c}) + \eta\pi(\mathbf{c} - \mathbf{c}(\mathbf{x}))[\mathbf{x} - \mathbf{x}'_{\text{old}}(\mathbf{c})] \tag{9.24}$$

where η is the learning-rate parameter, and $\mathbf{c}(\mathbf{x})$ is the nearest-neighbor encoding approximation to condition 1. The update equation (9.24) is obtained by inspection of the partial derivative in Eq. (9.21). This update is applied to all $\mathbf{c}$, for which we have

$$\pi(\mathbf{c} - \mathbf{c}(\mathbf{x})) > 0 \tag{9.25}$$

We may think of the gradient descent procedure described in Eq. (9.24) as a way of minimizing the distortion measure D_1 of Eq. (9.19). That is, Eqs. (9.23) and (9.24) are essentially of the same type, except for the fact that (9.23) is batch and (9.24) is continuous (i.e., in flowthrough form).

The update equation (9.24) is identical to the (continuous) SOM algorithm of Eq. (9.13), bearing in mind the correspondences listed in Table 9.1. Accordingly, we may state that the generalized Lloyd algorithm for vector quantization is the batch training version of the SOM algorithm with zero neighborhood size; for zero neighborhood, $\pi(0) = 1$. Note that in order to obtain the generalized Lloyd algorithm from the batch version of the SOM algorithm we do *not* need to make any approximations because the curvature terms (and all higher-order terms) make no contribution when the neighborhood has *zero* width.

TABLE 9.1 Correspondence between the SOM Algorithm and the Model of Fig. 9.6

Encoding-Decoding Model of Fig. 9.6	SOM Algorithm
Encoder $\mathbf{c}(\mathbf{x})$	Best-matching neuron $i(\mathbf{x})$
Reconstruction vector $\mathbf{x}'(\mathbf{c})$	Synaptic weight vector $\mathbf{w}_j$
Probability density function $\pi(\mathbf{c} - \mathbf{c}(\mathbf{x}))$	Neighborhood function $h_{j,i(\mathbf{x})}$

The important points to note from the discussion presented here are:

- The SOM algorithm is a vector quantization algorithm, which provides a goo approximation to the input space $\mathscr{X}$. This viewpoint provides another approac for deriving the SOM algorithm, as exemplified by Eq. (9.24).
- According to this viewpoint, the neighborhood function $h_{j,i(\mathbf{x})}$ in the SOM algo rithm has the form of a probability density function. In Luttrell (1991a), a zero mean Gaussian model is considered appropriate for the noise $\boldsymbol{\nu}$ in the model o Fig. 9.6. We thus also have theoretical justification for adopting the Gaussia neighborhood function of Eq. (9.4).

The *batch SOM*[8] is merely a rewrite of Eq. (9.23), with summations used to approximate the integrals in the numerator and denominator of the right-hand side of the equation. Note that in this version of the SOM algorithm the order in which the input patterns are presented to the network has no effect on the final form of the feature map, and there is no need for a learning-rate schedule. But the algorithm still requires the use of a neighborhood function.

Property 2. Topological Ordering. *The feature map Φ computed by the SOM algorithm is topologically ordered in the sense that the spatial location of a neuron in the lattice corresponds to a particular domain or feature of input patterns.*

The topological ordering property[9] is a direct consequence of the update equation (9.13) that forces the synaptic weight vector $\mathbf{w}_i$ of the winning neuron $i(\mathbf{x})$ to move toward the input vector $\mathbf{x}$. It also has the effect of moving the synaptic weight vectors $\mathbf{w}_j$ of the closest neurons j along with the winning neuron $i(\mathbf{x})$. We may therefore visualize the feature map Φ as an *elastic* or *virtual net* with the topology of a one- or two-dimensional lattice as prescribed in the output space $\mathscr{A}$, and whose nodes have weights as coordinates in the input space $\mathscr{X}$ (Ritter, 1995). The overall aim of the algorithm may thus be stated as:

> *Approximate the input space $\mathscr{X}$ by pointers or prototypes in the form of synaptic weight vectors w_j, in such a way that the feature map Φ provides a faithful representation of the important features that characterize the input vectors $\mathbf{x} \in \mathscr{X}$ in terms of a certain criterion.*

The feature map Φ is usually displayed in the input space $\mathscr{X}$. Specifically, all the pointers (i.e., synaptic weight vectors) are shown as dots, and the pointers of neighboring neurons are connected with lines in accordance with the topology of the lattice. Thus,

by using a line to connect two pointers $\mathbf{w}_i$ and $\mathbf{w}_j$, we are indicating that the corresponding neurons i and j are neighboring neurons in the lattice.

Property 3. Density Matching. *The feature map Φ reflects variations in the statistics of the input distribution: regions in the input space $\mathcal{X}$ from which sample vectors $\mathbf{x}$ are drawn with a high probability of occurrence are mapped onto larger domains of the output space $\mathcal{A}$, and therefore with better resolution than regions in $\mathcal{X}$ from which sample vectors $\mathbf{x}$ are drawn with a low probability of occurrence.*

Let $f_{\mathbf{X}}(\mathbf{x})$ denote the multidimensional pdf of the random input vector $\mathbf{X}$. This pdf, integrated over the entire input space $\mathcal{X}$, must equal unity, by definition:

$$\int_{-\infty}^{\infty} f_{\mathbf{X}}(\mathbf{x})d\mathbf{x} = 1$$

Let $m(\mathbf{x})$ denote the map *magnification factor*, defined as the number of neurons in a small volume $d\mathbf{x}$ of the input space $\mathcal{X}$. The magnification factor, integrated over the input space $\mathcal{X}$, must contain the total number l of neurons in the network, as shown by

$$\int_{-\infty}^{\infty} m(\mathbf{x})d\mathbf{x} = l \tag{9.26}$$

For the SOM algorithm to *match the input density* exactly, we require that (Amari, 1980)

$$m(\mathbf{x}) \propto f_{\mathbf{X}}(\mathbf{x}) \tag{9.27}$$

This property implies that if a particular region of the input space contains frequently occurring stimuli, it will be represented by a larger area in the feature map than a region of the input space where the stimuli occur less frequently.

Generally in two-dimensional feature maps the magnification factor $m(\mathbf{x})$ is not expressible as a simple function of the probability density function $f_{\mathbf{X}}(\mathbf{x})$ of the input vector $\mathbf{x}$. It is only in the case of a one-dimensional feature map that it is possible to derive such a relationship. For this special case we find that, contrary to earlier supposition (Kohonen, 1982), the magnification factor $m(\mathbf{x})$ is *not* proportional to $f_{\mathbf{X}}(\mathbf{x})$. Two different results are reported in the literature, depending on the encoding method advocated:

1. *Minimum-distortion encoding*, according to which the curvature terms and all higher-order terms in the distortion measure of Eq. (9.22) due to the noise model $\pi(\boldsymbol{\nu})$ are retained. This encoding method yields the result

$$m(\mathbf{x}) \propto f_{\mathbf{X}}^{1/3}(\mathbf{x}) \tag{9.28}$$

   which is the same as the result obtained for the standard vector quantizer (Luttrell, 1991a).
2. *Nearest-neighbor encoding*, which emerges if the curvature terms are ignored, as in the standard form of the SOM algorithm. This encoding method yields the result (Ritter, 1991)

$$m(\mathbf{x}) \propto f_{\mathbf{X}}^{2/3}(\mathbf{x}) \tag{9.29}$$

Our earlier statement that a cluster of frequently occurring input stimuli is represented by a larger area in the feature map still holds, albeit in a distorted version of the ideal condition described in Eq. (9.27).

As a general rule (confirmed by computer simulations), the feature map computed by the SOM algorithm tends to overrepresent regions of low input density and to underrepresent regions of high input density. In other words, the SOM algorithm fails to provide a faithful representation of the probability distribution that underlies the input data.[10]

Property 4. Feature selection. *Given data from an input space with a nonlinear distribution, the self-organizing map is able to select a set of best features for approximating the underlying distribution.*

This property is a natural culmination of Properties 1 through 3. It brings to mind the idea of principal components analysis that is discussed in the previous chapter, but with an important difference as illustrated in Fig. 9.7. In Fig. 9.7a we show a two-dimensional distribution of zero-mean data points resulting from a linear input–output mapping corrupted by additive noise. In such a situation, principal components analysis works perfectly fine: It tells us that the best description of the "linear" distribution in Fig. 9.7a is defined by a straight line (i.e., one-dimensional "hyperplane") that passes through the origin and runs parallel to the eigenvector associated with the largest eigenvalue of the correlation matrix of the data. Consider next the situation described in Fig. 9.7b, which is the result of a nonlinear input–output mapping corrupted by additive noise of zero mean. In this second situation, it is impossible for a straight-line approximation computed from principal components analysis to provide an acceptable description of the data. On the other hand, the use of a self-organizing map built on a one-dimensional lattice of neurons is able to overcome this approximation problem by virtue of its topological-ordering property. This latter approximation is illustrated in Fig. 9.7b.

In precise terms we may state that self-organizing feature maps provide a *discrete* approximation of the so-called *principal curves*[11] or *principal surfaces* (Hastie and Stuetzle, 1989), and may therefore be viewed as a nonlinear generalization of principal components analysis.

9.6 COMPUTER SIMULATIONS

Two-Dimensional Lattice Driven by a Two-Dimensional Distribution

We illustrate the behavior of the SOM algorithm by using computer simulations to study a network with 100 neurons, arranged in the form of a two-dimensional lattice with 10 rows and 10 columns. The network is trained with a two-dimensional input vector $\mathbf{x}$, whose elements x_1 and x_2 are uniformly distributed in the region $\{(-1 < x_1 < +1); (-1 < x_2 < +1)\}$. To initialize the network the synaptic weights are chosen from a random set.

Figure 9.8 shows three stages of training as the network learns to represent the input distribution. Figure 9.8a shows the uniform distribution of data used to train the

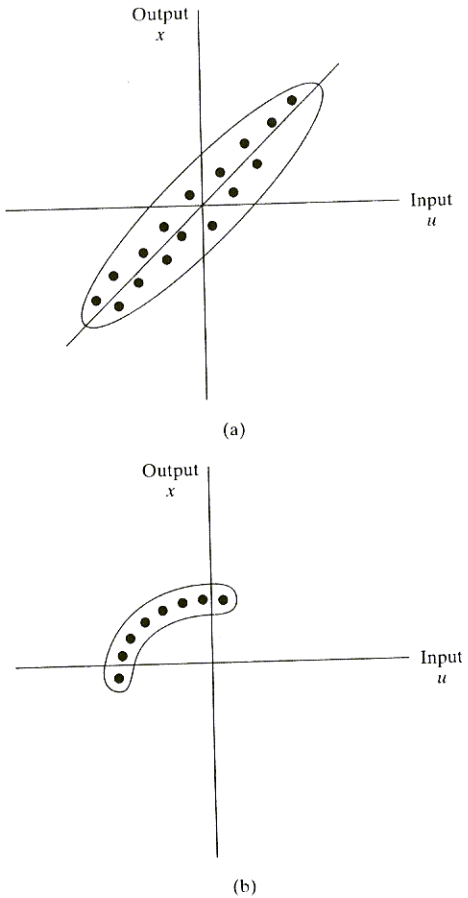

FIGURE 9.7 (a) Two-dimensional distribution produced by a linear input–output mapping. (b) Two-dimensional distribution produced by a nonlinear input–output mapping.

feature map. Figure 9.8b shows the initial values of the synaptic weights, randomly chosen. Figures 9.8c and 9.8d present the values of the synaptic weight vectors, plotted as dots in the input space, after completion of the ordering and convergence phases, respectively. The lines drawn in Fig. 9.8 connect neighboring neurons (across rows and columns) in the network.

The results shown in Fig. 9.8 demonstrate the ordering phase and the convergence phase that characterize the learning process of the SOM algorithm. During the ordering phase the map *unfolds* to form a mesh, as shown in Fig. 9.8c. The neurons are mapped in the correct order at the end of this phase. During the convergence phase the map spreads out to fill the input space. At the end of this second phase, shown in Fig. 9.8d,

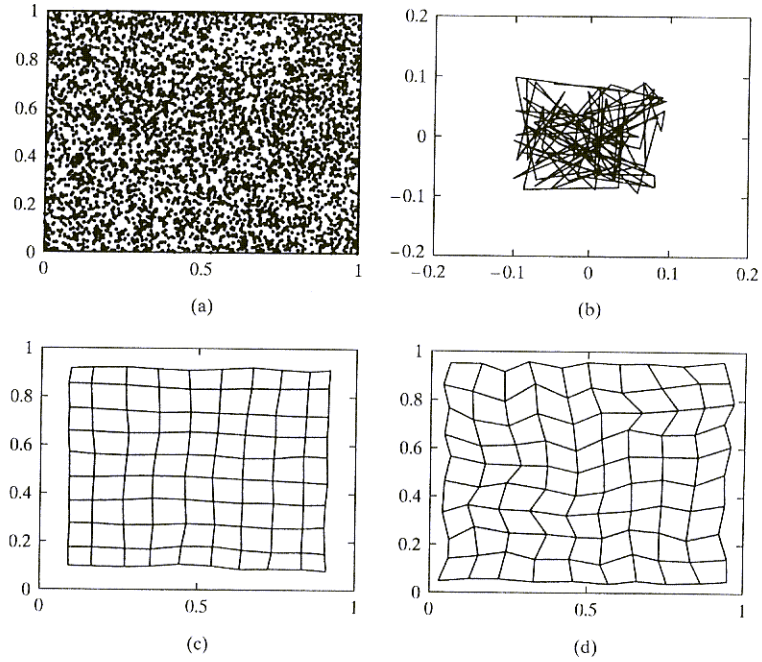

FIGURE 9.8 (a) Input data distribution. (b) Initial condition of the two-dimensional lattice. (c) Condition of the lattice at the end of the ordering phase. (d) Condition of the lattice at the end of the convergence phase.

the statistical distribution of the neurons in the map approaches that of the input vectors, except for some edge effects. Comparing the final state of the feature map in Fig. 9.8d with the uniform distribution of the input in Fig. 9.8a, we see that the tuning of the map during the convergence phase has captured the local irregularities that can be seen in the input distribution.

The topological ordering property of the SOM algorithm is well illustrated in Fig. 9.8d. In particular we observe that the algorithm (after convergence) captures the underlying topology of the uniform distribution at the input. In the computer simulations presented in Fig. 9.8, the input space $\mathcal{X}$ and output space $\mathcal{A}$ are both two-dimensional.

One-Dimensional Lattice Driven by a Two-Dimensional Distribution

We now examine the case when the dimension of the input space $\mathcal{X}$ is greater than the dimension of the output space $\mathcal{A}$. In spite of this mismatch, the feature map Φ is often able to form a topological representation of the input distribution. Figure 9.9 shows

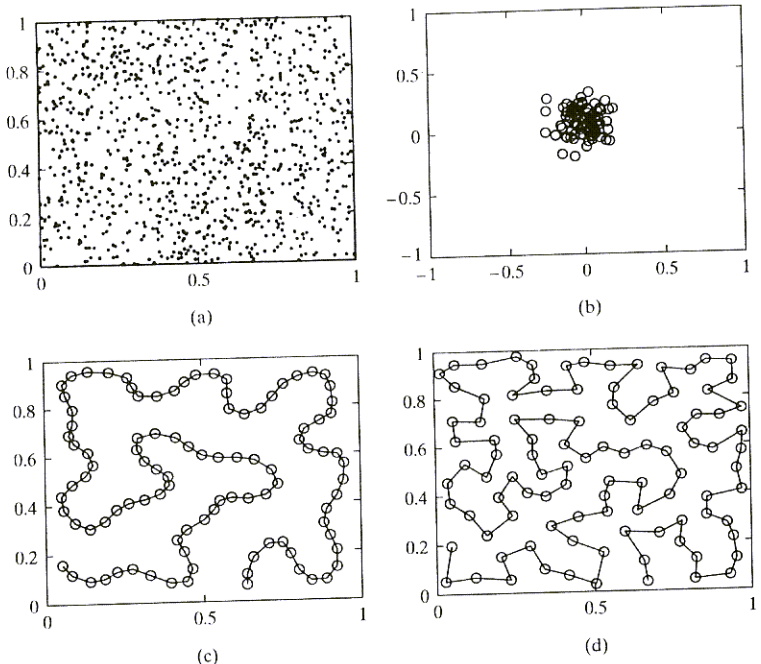

FIGURE 9.9 (a) Two-dimensional input data distribution. (b) Initial condition of the one-dimensional lattice. (c) Condition of the lattice at the end of the ordering phase. (d) Condition of the lattice at the end of the convergence phase.

three different stages in the evolution of a feature map initialized as in Fig. 9.9b and trained with input data drawn from a uniform distribution inside a square as in Fig. 9.9a, but this time the computation is performed with a one-dimensional lattice of 100 neurons. Figures 9.9c and 9.9d show the feature map after the completion of the ordering and convergence phases, respectively. Here we see that the feature map computed by the algorithm is very distorted in order to fill the square as densely as possible and thereby provide a good approximation to the underlying topology of the two-dimensional input space $\mathcal{X}$. The approximating curve shown in Fig. 9.9d resembles a *Peano curve* (Kohonen, 1990a). An operation of the kind exemplified by the feature map of Fig. 9.9, where an input space $\mathcal{X}$ is represented by projecting it onto a lower-dimensional output space $\mathcal{A}$, is referred to as *dimensionality reduction*.

Parameter Specifications for the Simulations

Figure 9.10 presents details of the variations of the neighborhood function $h_{j,i}(n)$ and learning-rate parameter $\eta(n)$ with time (i.e., number of epochs) for the experiments involving a one-dimensional lattice. The neighborhood-function parameter $\sigma(n)$, shown

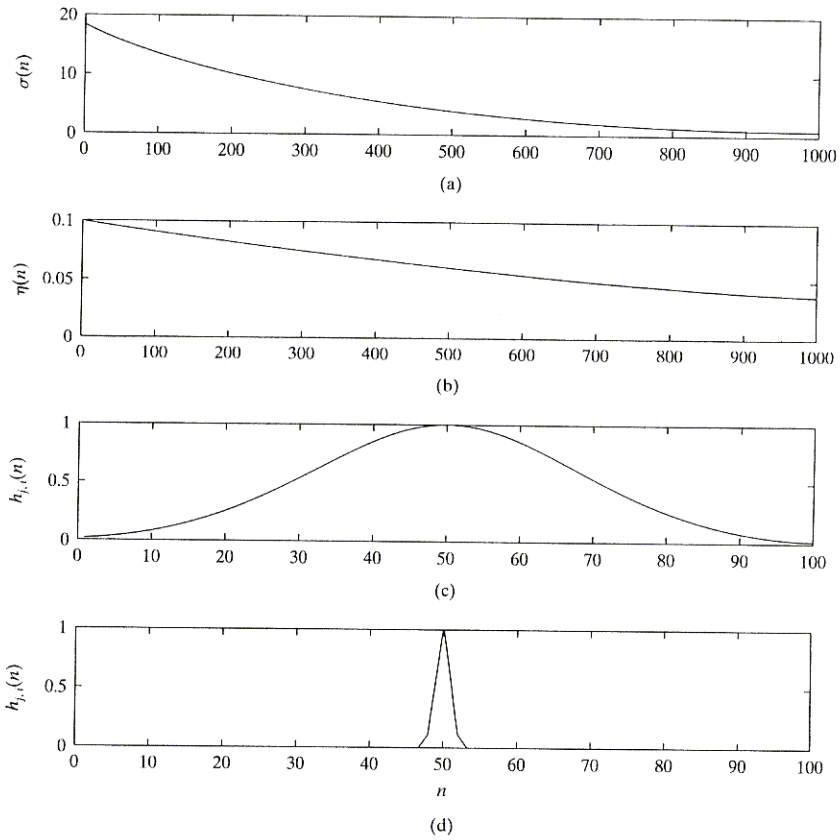

FIGURE 9.10 (a) Exponential decay of neighborhood function parameter $\sigma(n)$. (b) Exponential decay of learning-rate parameter $\eta(n)$. (c) Initial shape of the Gaussian neighborhood function. (d) Shape of the neighborhood function at the end of the ordering phase (i.e., beginning of the convergence phase).

in Fig. 9.10a, starts with an initial value $\sigma_0 = 18$ and then shrinks to about 1 in 1000 iterations during the ordering phase. During that same phase, the learning-rate parameter $\eta(n)$ starts with an initial value $\eta_0 = 0.1$ and then decreases to 0.037. Figure 9.10c shows the initial Gaussian distribution of neurons around a winning neuron located at the midpoint of the one-dimensional lattice. Figure 9.10d shows the shape of the neighborhood function at the end of the ordering phase. During the convergence phase the learning-rate parameter decreases linearly from 0.037 to 0.001 in 5000 iterations. During the same phase the neighborhood function decreases essentially to zero.

The specifications of the ordering phase and convergence phase for the computer simulations in Fig. 9.8 involving the two-dimensional lattice are similar to those used

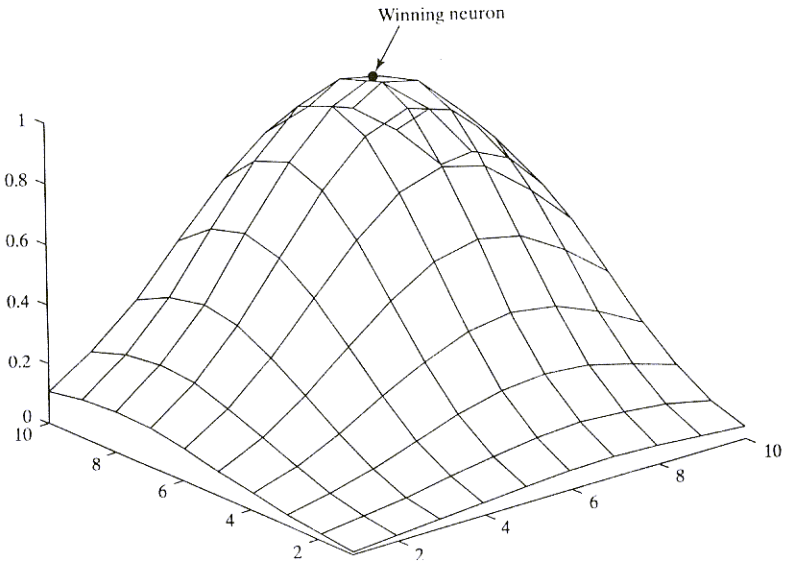

FIGURE 9.11 Initial condition of two-dimensional Gaussian neighborhood function centered on a winning neuron located at the point (7, 8) in a two-dimesional lattice of 10 × 10 neurons.

for the one-dimensional lattice, except for the fact that the neighborhood function is now two-dimensional. The parameter $\sigma(n)$ starts at the initial value $\sigma_0 = 3$ and then decreases to 0.75 in 1000 iterations. Figure 9.11 displays the initial value of the two-dimensional Gaussian neighborhood function $h_{j,i}$ for $\sigma_0 = 3$ and a winning neuron centered on the point (7,8) inside the two-dimensional lattice of 10 × 10 neurons.

9.7 LEARNING VECTOR QUANTIZATION

Vector quantization, discussed previously in Section 9.6, is a technique that exploits the underlying structure of input vectors for the purpose of data compression (Gersho and Gray, 1992). Specifically, an input space is divided into a number of distinct regions, and for each region a reconstruction vector is defined. When the quantizer is presented a new input vector, the region in which the vector lies is first determined, and is then represented by the reproduction vector for that region. Thus, by using an encoded version of this reproduction vector for storage or transmission in place of the original input vector, considerable savings in storage or transmission bandwidth can be realized, at the expense of some distortion. The collection of possible reproduction vectors is called the *code book* of the quantizer, and its members are called *code words*.

A vector quantizer with minimum encoding distortion is called a *Voronoi* or *nearest-neighbor quantizer*, since the *Voronoi cells* about a set of points in an input space correspond to a partition of that space according to the *nearest-neighbor rule* based on the Euclidean metric (Gersho and Gray, 1992). Figure 9.12 shows an example of an input space divided into four Voronoi cells with their associated Voronoi vectors (i.e., reconstruction vectors). Each Voronoi cell contains those points of the input space that are the closest to the Voronoi vector among the totality of such points.

The SOM algorithm provides an approximate method for computing the Voronoi vectors in an unsupervised manner, with the approximation being specified by the synaptic weight vectors of the neurons in the feature map; this is merely restating property 1 of the SOM algorithm discussed in Section 9.6. Computation of the feature map may therefore be viewed as the first of two stages for adaptively solving a pattern classification problem, as depicted in Fig. 9.13. The second stage is provided by learning vector quantization, which provides a mechanism for the final fine tuning of a feature map.

Learning vector quantization[12] (LVQ) is a supervised learning technique that uses class information to move the Voronoi vectors slightly, so as to improve the quality of the classifier decision regions. An input vector $\mathbf{x}$ is picked at random from the input space. If the class labels of the input vector $\mathbf{x}$ and a Voronoi vector $\mathbf{w}$ agree, the Voronoi vector $\mathbf{w}$ is moved in the direction of the input vector $\mathbf{x}$. If, on the other hand, the class labels of the input vector $\mathbf{x}$ and the Voronoi vector $\mathbf{w}$ disagree, the Voronoi vector $\mathbf{w}$ is moved away from the input vector $\mathbf{x}$.

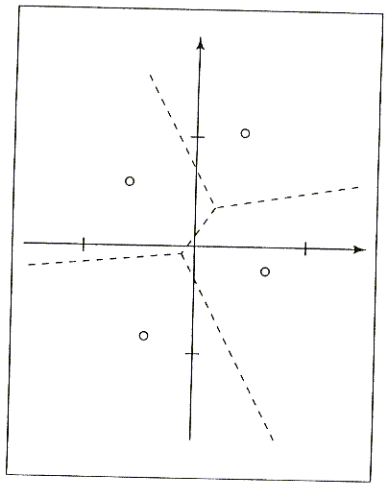

FIGURE 9.12 Voronoi diagram involving four cells. (Adapted from R.M Gray, 1984, with permission of IEEE.)

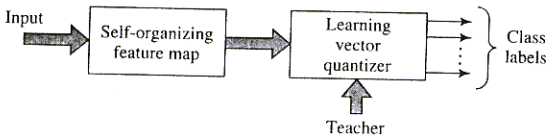

FIGURE 9.13 Block diagram of adaptive pattern classification, using a self-organizing feature map and learning vector quantizer.

Let $\{\mathbf{w}_j\}_{j=1}^{l}$ denote the set of Voronoi vectors, and let $\{\mathbf{x}_i\}_{i=1}^{N}$ denote the set of input (observation) vectors. We assume that there are many more input vectors than Voronoi vectors, which is typically the case in practice. The learning vector quantization (LVQ) algorithm proceeds as follows:

(i) Suppose that the Voronoi vector $\mathbf{w}_c$ is the closest to the input vector $\mathbf{x}_i$. Let $\mathscr{C}_{\mathbf{w}_c}$ denote the class associated with the Voronoi vector $\mathbf{w}_c$ and $\mathscr{C}_{\mathbf{x}_i}$ denote the class label of the input vector $\mathbf{x}_i$. The Voronoi vector $\mathbf{w}_c$ is adjusted as follows:

- If $\mathscr{C}_{\mathbf{w}_c} = \mathscr{C}_{\mathbf{x}_i}$, then

$$\mathbf{w}_c(n+1) = \mathbf{w}_c(n) + \alpha_n[\mathbf{x}_i - \mathbf{w}_c(n)] \tag{9.30}$$

where $0 < \alpha_n < 1$.

- If, on the other hand, $\mathscr{C}_{\mathbf{w}_c} \neq \mathscr{C}_{\mathbf{x}_i}$, then

$$\mathbf{w}_c(n+1) = \mathbf{w}_c(n) - \alpha_n[\mathbf{x}_i - \mathbf{w}_c(n)] \tag{9.31}$$

(ii) The other Voronoi vectors are not modified.

It is desirable for the learning constant α_n to decrease monotonically with the number of iterations n. For example, α_n may initially be about 0.1 or smaller, and then decrease linearly with n. After several passes through the input data, the Voronoi vectors typically converge, and the training is complete. However, difficulties may be experienced if the method is applied without proper care.

9.8 COMPUTER EXPERIMENT: ADAPTIVE PATTERN CLASSIFICATION

In pattern classification, the first and most important step is *feature selection* (extraction), which is ordinarily performed in an unsupervised manner. The objective of this first step is to select a reasonably small set of features, in which the essential information content of the input data (to be classified) is concentrated. The self-organizing map, by virtue of property 4 discussed in Section 9.5, is well suited for the task of feature selection, particularly if the input data are generated by a nonlinear process.

The second step in pattern classification is the actual *classification*, where the features selected from the input data are assigned to individual classes. Although a self-organizing map is equipped to perform the role of classification too, the recommended procedure for best performance is to accompany it with a supervised learning scheme for the second stage of classification. The combination of a self-organizing map and a supervised learning scheme forms the basis of an *adaptive pattern classification* that is *hybrid* in nature.

Such a hybrid approach to pattern classification may take different forms, depending on how the supervised learning scheme is implemented. One simple scheme is to use a learning vector quantizer, which is described in the previous section. We thus have the two-stage adaptive pattern classifier shown in Fig. 9.13.

In this experiment we revisit the classification of overlapping two-dimensional, Gaussian-distributed patterns labeled 1 (class $\mathscr{C}_1$) and labeled 2 (class $\mathscr{C}_2$), which was first described in Chapter 4 involving the use of a multilayer perceptron trained with the back-propagation algorithm. The scatter plots for the data used in the experiment are shown in Fig. 4.13.

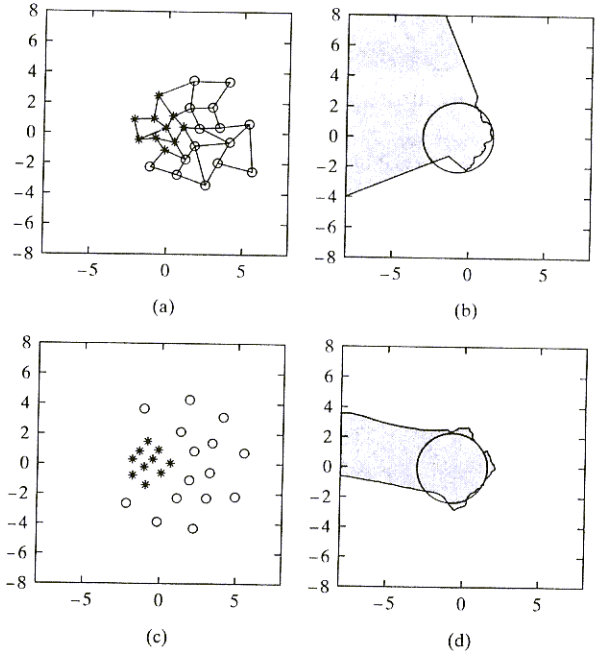

FIGURE 9.14 (a) Self-organizing map after labeling. (b) Decision boundary constructed by the feature map of part a. (c) Labeled map after learning-vector quantization. (d) Decision boundary constructed by the feature map of part c.

Figure 9.14a shows the two-dimensional feature map of 5×5 neurons after training with the SOM algorithm is complete. The feature map has been labeled, with each neuron assigned to one class or the other depending on how it responds to test data drawn from the input distribution. Figure 9.14b shows the decision boundary realized by the feature map operating on its own.

Figure 9.14c shows the modified feature map after it was tuned in a supervised manner using LVQ. Figure 9.14d shows the decision boundary produced by the combined action of the SOM and LVQ algorithms. Comparing these two figures with their counterparts shown in Figs. 9.14a and 9.14b, we see, in a qualitative manner, the beneficial effect obtained by using the LVQ.

Table 9.2 presents a summary of the classification performances of the feature map on its own and the feature map working together with the learning vector quantizer. The results presented here were obtained on 10 independent trials of the experiment, with each experiment involving the use of 30,000 patterns as test data. On each trial of the experiment there was an improvement in classification performance due to the use of LVQ. The average classification performance for the feature map on its own

TABLE 9.2 Summary of Classification Performances (Percentage) for the Computer Experiment on Overlapping Two-Dimensional Gaussian Distribution Using 5 × 5 Lattice

Trial	Feature map on its own	Cascade combination of feature map and learning vector quantizer
1	79.05	80.18
2	79.79	80.56
3	79.41	81.17
4	79.38	79.84
5	80.30	80.43
6	79.55	80.36
7	79.79	80.86
8	78.48	80.21
9	80.00	80.51
10	80.32	81.06
Average	79.61%	80.52%

is 79.61 percent, and for the combination of the feature map and the learning vector quantizer is 80.52 percent, which represents an improvement of 0.91 percent over the feature map on its own. For a frame of reference, we recall that the performance of the optimum Bayes classifier for this experiment is 81.51 percent.

9.9 HIERARCHICAL VECTOR QUANTIZATION

In discussing property 1 of the self-organizing feature map in Section 9.6, we pointed out that it is closely related to the generalized Lloyd algorithm for vector quantization. Vector quantization is a form of *lossy* data compression, lossy in the sense that some information contained in the input data is lost as a result of the compression. Data compression is rooted in a branch of Shannon's information theory known as *rate distortion theory* (Cover and Thomas, 1991). For the purpose of our present discussion dealing with hierarchical vector quantization, it is appropriate to begin by stating the following fundamental result of rate distortion theory (Gray, 1984):

> Better data compression performance can always be achieved by coding vectors instead of scalars, even if the source of data is memoryless (e.g., it provides a sequence of independent random variables), or if the data compression system has memory (i.e., the action of an encoder depends on past encoder inputs or outputs).

This fundamental result underlies the extensive research effort that has been devoted to vector quantization (Gersho and Gray, 1992).

However, conventional vector quantization algorithms require a prohibitive amount of computation, which has hindered their practical use. The most time consuming part of vector quantization is the encoding operation. For encoding, the input vector must be compared with each code vector in the code book in order to determine

which particular code yields the minimum distortion. For a code book containing N code vectors, for example, the time taken for encoding is on the order of N, which can therefore be large for large N. In Luttrell (1989a), a *multistage hierarchical vector quantizer* is described that trades off accuracy for speed of encoding. This scheme is not simply the standard tree search of a code book; it is genuinely new. The multistage hierarchical vector quantizer attempts to factorize the overall vector quantization into a number of suboperations, each of which requires very little computation. Desirably, the factorization is reduced to a single table look-up per suboperation. By clever use of the SOM algorithm to train each stage of the quantizer, the loss in accuracy can be small (as low as a fraction of a decibel), while the gain in speed of computation is large.

Consider two vector quantizers VQ_1 and VQ_2, with VQ_1 feeding its output into VQ_2. The output from VQ_2 is the final encoded version of the original input signal applied to VQ_1. In performing its quantization, it is inevitable for VQ_2 to discard some information. As far as VQ_1 is concerned, the sole effect of VQ_2 is therefore to distort the information output by VQ_1. It thus appears that the appropriate training method for VQ_1 is the SOM algorithm, which accounts for the signal distortion induced by VQ_2 (Luttrell, 1989a). In order to use the generalized Lloyd algorithm to train VQ_2 we need only assume that the output of VQ_2 is not corrupted before we do the reconstruction. Then we do not need to introduce any noise model (at the output of VQ_2) with its associated finite width neighborhood function.

We can generalize this heuristic argument to a multistage vector quantizer. Each stage must be designed to account for the distortion induced by all *subsequent* stages, and model it as noise. To do so, the SOM algorithm is used to train all the stages of the quantizer, except for the last stage for which the generalized Lloyd algorithm is adequate.

Hierarchical vector quantization is a special case of multistage vector quantization (Luttrell, 1989a). As an illustration, consider the quantization of 4 × 1 input vector

$$\mathbf{x} = [x_1, x_2, x_3, x_4]^T$$

In Fig. 9.15a we show a single-stage vector quantizer for $\mathbf{x}$. Alternatively, we may use a two-stage hierarchical vector quantizer as depicted in Fig. 9.15b. The significant difference between these two schemes is that the input dimension of the quantizer in Fig. 9.15a is four, whereas for the quantizer in Fig. 9.15b it is two. Accordingly, the quantizer of Fig. 9.15b requires a look-up table of smaller size, and is therefore simpler to implement than that of Fig. 9.15a. This is the advantage of a hierarchical quantizer over a conventional quantizer.

Luttrell (1989a) has demonstrated the performance of a multistage hierarchical vector quantizer applied to various stochastic time series, with little loss in encoding accuracy. In Fig. 9.16 we have reproduced Luttrell's results for the case of a correlated Gaussian noise process generated using a *first-order autoregressive (AR) model*:

$$x(n+1) = \rho x(n) + \nu(n) \tag{9.32}$$

where ρ is the AR coefficient and the $\nu(n)$ are independent and identically distributed (iid) Gaussian random variables of zero mean and unit variance. Hence we may show that $x(n)$ is characterized as follows:

$$E[x(n)] = 0 \tag{9.33}$$

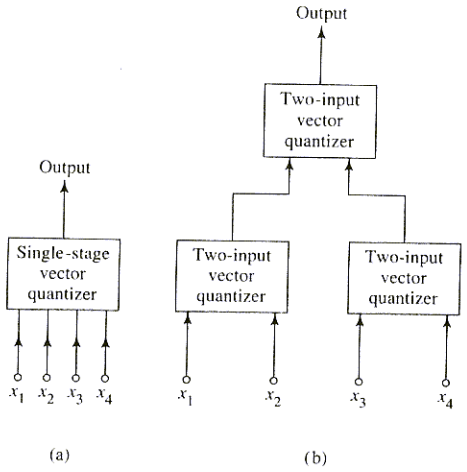

FIGURE 9.15 (a) Single-stage vector quantizer with four-dimensional input. (b) Two-stage hierarchical vector quantizer using two-input vector quantizers. (From S.P. Luttrell, 1989a, British Crown copyright.)

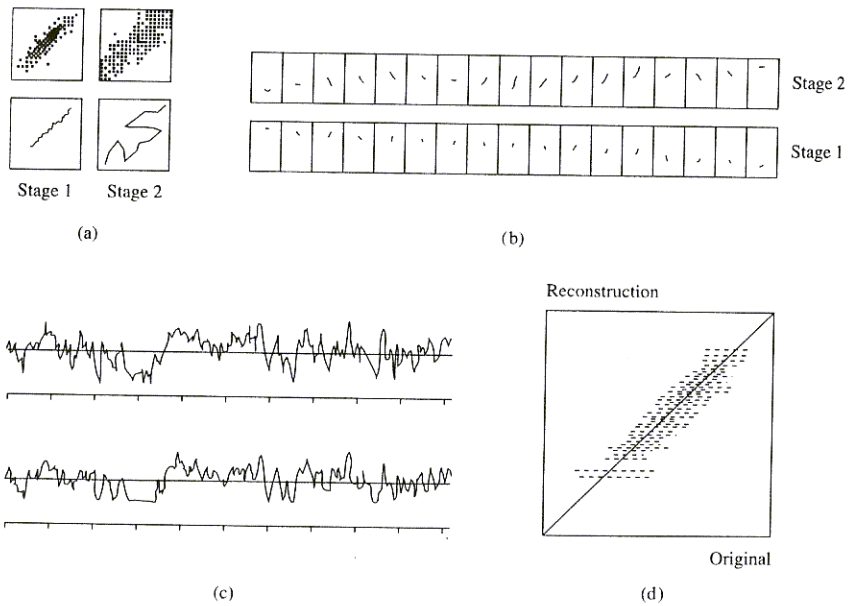

FIGURE 9.16 Two-stage encoding/decoding results for correlated Gaussian noise input. Correlation coefficient $\rho = 0.85$. (From S.P.Luttrell, 1989a, British Crown copyright.)

$$E[x^2(n)] = \frac{1}{1 - \rho^2} \tag{9.34}$$

$$\frac{E[x(n+1)x(n)]}{E[x^2(n)]} = \rho \tag{9.35}$$

Thus ρ may also be viewed as the *correlation coefficient* of the time series $\{x(n)\}$. To initiate the generation of the time series according to Eq. (9.32), a Gaussian random variable of zero mean and variance $1/(1 - \rho^2)$ was used for $x(0)$, and the value $\rho = 0.85$ was used for the correlation coefficient.

For the vector quantization a hierarchical encoder with a four-dimensional input space, like the binary tree of Fig. 9.15b, was used. For the AR time series $\{x(n)\}$, translational symmetry implies that only *two* distinct look-up tables are needed. The size of each table depends exponentially on the number of input bits, and linearly on the number of output bits. During training, a large number of bits is needed to represent numbers for a correct computation of the updates described in Eq. (9.24); so the look-up tables are not used during training. Once training is complete, however, the number of bits may be reduced to their normal level, and the table entries filled in as required. For the encoder shown in Fig. 9.15b, the input samples were approximated by using four bits per sample. For all stages of the encoder, N (= 17) code vectors were used, so the number of output bits from each lookup table was approximately four, too. Thus the address space size of both the first stage and second stage look-up tables is 256 ($= 2^{4+4}$), which means that the overall memory requirements for representing the tables are modest.

Figure 9.16 shows the encoding-decoding results obtained with $x(n)$ as the input. The lower half of Fig. 9.16a shows the code vectors for each of the two stages as a curve embedded in a two-dimensional input space; the upper half of Fig. 9.16a presents estimates of the corresponding co-occurrence matrices using 16×16 bins. Figure 9.16b presents, as fragments of the time series, the following:

- The code vector computed by the first encoder stage
- The reconstruction vector computed by the second stage that minimizes the mean-squares distortion, while keeping all other variables fixed

Figure 9.16c presents 512 samples of both the original time series (top curve) and its reconstruction (bottom curve) from the output of the last encoder stage; the horizontal scale in Fig. 9.16c is half that in Fig. 9.16b. Finally, Fig. 9.16d presents a co-occurrence matrix created from a pair of samples: an original time series sample and its corresponding reconstruction. The width of the band in Fig. 9.16d indicates the extent of the distortion produced by the hierarchical vector quantization.

Examining the waveforms in Fig. 9.16c, we see that the reconstruction is a good representation of the original time series, except for some positive and negative peaks that were clipped. According to Luttrell (1989a), the normalized mean-squared distortion was computed as 0.15, which is almost as good (0.05 dB loss) as the 8.8 dB obtained with a single-stage four-sample block encoder using one bit per sample (Jayant and Noll, 1984).

CONTEXTUAL MAPS

There are two fundamentally different ways of visualizing a self-organizing feature map. In one method of visualization the feature map is viewed as an elastic net with the synaptic weight vectors treated as pointers for the respective neurons, which are directed into the input space. This method of visualization is particularly useful for displaying the topological ordering property of the SOM algorithm, as illustrated by the results of computer simulation experiments presented in Section 9.6.

In the second method of visualization, class labels are assigned to neurons in a two-dimensional lattice (representing the output layer of the network), depending on how each *test pattern* (not seen before) excites a particular neuron in the self-organized network. As a result of this second stage of stimulation, the neurons in the two-dimensional lattice are partitioned into a number of *coherent regions*, coherent in the sense that each grouping of neurons represents a distinct set of contiguous symbols or labels (Ritter and Kohonen, 1989). This assumes that the right conditions have been followed for the development of a well-ordered feature map in the first place.

Consider, for example, the set of data given in Table 9.3, which pertains to a number of different animals. Each column of the table is a schematic description of an animal, based on the presence (= 1) or absence (= 0) of some of the 13 different attributes given on the left. Some attributes such as "feathers" and "two legs" are correlated, while many of the other attributes are uncorrelated. For each animal given at the top of the table we have an *attribute code* $\mathbf{x}_a$ made up of 13 elements. The animal is itself specified by a *symbol code* $\mathbf{x}_s$, the composition of which must *not* convey any information or known similarities between the animals. For the example at hand, $\mathbf{x}_s$ consists of a column vector whose kth element, representing animal $k = 1, 2, \ldots, 16$, is given a fixed value of a; the remaining elements are all set equal to zero. The parameter a determines the relative influence of the symbol code compared to the attribute code. To make sure that the attribute code is the dominant one, a is chosen equal to 0.2. The input vector $\mathbf{x}$ for each animal is a vector of 29 elements, representing a concatenation of the attribute code $\mathbf{x}_a$ and the symbol code $\mathbf{x}_s$, as shown by

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_s \\ \mathbf{x}_a \end{bmatrix} = \begin{bmatrix} \mathbf{x}_s \\ \mathbf{0} \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ \mathbf{x}_a \end{bmatrix}$$

Finally, each data vector is normalized to unit length. The patterns of the data set thus generated are presented to a two-dimensional lattice of 10×10 neurons, and the synaptic weights of the neurons are adjusted in accordance with the SOM algorithm summarized in Section 9.4. The training is continued for 2000 iterations, whereafter the feature map should have reached a steady state. Next, a test pattern defined by $\mathbf{x} = [\mathbf{x}_s, \mathbf{0}]^T$ containing the symbol code of only one of the animals, is presented to the self-organized network and the neuron with the strongest response is identified. This is repeated for all 16 animals.

Proceeding in the manner just described, we obtain the map shown in Fig. 9.17, where the labeled neurons represent those with the strongest responses to their respective test patterns; the dots represent neurons with weaker responses.

Figure 9.18 shows the result of "simulated electrode penetration mapping" for the same self-organized network. This time, however, each neuron in the network has been marked by the particular animal for which it produces the best response. Figure 9.18 clearly shows that the feature map has essentially captured the "family relationships" among the 16 different animals. There are three distinct clusters, one representing "birds," a second representing "peaceful species," and the third representing animals that are "hunters."

A feature map of the type illustrated in Fig. 9.18 is referred to as a *contextual map* or *semantic map* (Ritter and Kohonen, 1989; Kohonen, 1997a). Such a map resembles cortical maps (i.e., the computational maps formed in the cerebral cortex) that are discussed briefly in Section 9.2. Contextual maps, resulting from the use of the SOM algorithm, find applications in such diverse fields as unsupervised categorization of phonemic classes from text, remote sensing (Kohonen, 1997a), and data exploration or data mining (Kohonen, 1997b).

TABLE 9.3 Animal Names and Their Attributes

Animal		Dove	Hen	Duck	Goose	Owl	Hawk	Eagle	Fox	Dog	Wolf	Cat	Tiger	Lion	Horse	Zebra	Cow
is	small	1	1	1	1	1	1	0	0	0	0	1	0	0	0	0	0
	medium	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
	big	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
has	2 legs	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
	4 legs	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
	hair	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
	hooves	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
	mane	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0
	feathers	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
likes to	hunt	0	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0
	run	0	0	0	0	0	0	0	0	1	1	0	1	1	1	1	0
	fly	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0
	swim	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

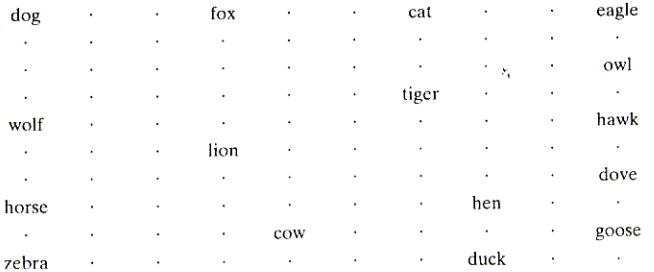

FIGURE 9.17 Feature map containing labeled neurons with strongest responses to their respective inputs.

dog	dog	fox	fox	fox	cat	cat	cat	eagle	eagle
dog	dog	fox	fox	fox	cat	cat	cat	eagle	eagle
wolf	wolf	wolf	fox	cat	tiger	tiger	tiger	owl	owl
wolf	wolf	lion	lion	lion	tiger	tiger	tiger	hawk	hawk
wolf	wolf	lion	lion	lion	tiger	tiger	tiger	hawk	hawk
wolf	wolf	lion	lion	lion	owl	dove	hawk	dove	dove
horse	horse	lion	lion	lion	dove	hen	hen	dove	dove
horse	horse	zebra	cow	cow	cow	hen	hen	dove	dove
zebra	zebra	zebra	cow	cow	cow	hen	hen	duck	goose
zebra	zebra	zebra	cow	cow	cow	duck	duck	duck	goose

FIGURE 9.18 Semantic map obtained through the use of simulated electrode penetration mapping. The map is divided into three regions representing: birds, peaceful species, and hunters.

1 SUMMARY AND DISCUSSION

The self-organizing map due to Kohonen (1982) is an ingenious neural network built around a one- or two-dimensional lattice of neurons for capturing the important features contained in an input (data) space of interest. In so doing, it provides a structural representation of the input data by the neurons' weight vectors as prototypes. The SOM algorithm is neurobiologically inspired, incorporating all the mechanisms that are basic to self-organization: competition, cooperation, and self-amplification that are discussed in Chapter 8. It may therefore serve as a generic though degenerate model for describing the emergence of collective ordering phenomena in complex systems after starting from total disorder.

The self-organizing map may also be viewed as a vector quantizer, thereby providing a principled approach for deriving the update rule used to adjust the weight vectors (Luttrell, 1989b). This latter approach clearly emphasizes the role of the neighborhood function as a probability density function.

It should, however, be emphasized that this latter approach, based on the use of average distribution D_1 in Eq. (9.19) as the cost function to be minimized, can be justified only when the feature map is already well ordered. In Erwin et al. (1992b), it is shown that the learning dynamics of a self-organizing map during the ordering phase of the adaptive process (i.e., during the topological ordering of a feature map that is initially highly disordered) *cannot* be described by a stochastic gradient descent on a *single* cost function. But in the case of a one-dimensional lattice, it may be described using a set of cost functions, one for each neuron in the network, which are independently minimized following a stochastic gradient descent.

What is astonishing about Kohonen's SOM algorithm is that it is so simple to implement, yet mathematically so difficult to analyze its properties in a general setting. Some fairly powerful methods have been used to analyze it by several investigators, but they have only produced results of limited applicability. In Cottrell et al. (1997), a survey of results on theoretical aspects of the SOM algorithm is given. In particular, a recent result due to Forte and Pagés (1995, 1997) is highlighted, and states that in the case of a one-dimensional lattice we have a rigorous proof of the "almost sure" convergence of the SOM algorithm to a unique state after completion of the self-organization phase. This important result has been shown to hold for a general class of neighborhood functions. However, the same cannot be said in a multidimensional setting.

One final point of enquiry is in order. With the self-organizing feature map being inspired by ideas derived from cortical maps in the brain, it seems natural to enquire whether such a model could actually explain the formation of cortical maps. Erwin et al. (1995) have performed such an investigation. They have shown that the self-organizing feature map is able to explain the formation of computational maps in the primary visual cortex of the macaque monkey. The input space used in this study has five dimensions: two dimensions for representing the position of a receptive field in retinotopic space, and the remaining three dimensions for representing orientation preference, orientation selectivity, and ocular dominance. The cortical surface is divided into small patches that are considered as computational units (i.e., artificial neurons) of a two-dimensional square lattice. Under certain assumptions, it is shown that Hebbian learning leads to spatial patterns of orientation and ocular dominance that are remarkably similar to those found in the macaque monkey.

NOTES AND REFERENCES

1. The two feature-mapping models of Fig. 9.1 were inspired by the pioneering self-organizing studies of von der Malsburg (1973), who noted that a model of the visual cortex could not be entirely genetically predetermined; rather, a self-organizing process involving synaptic learning may be responsible for the *local* ordering of feature-sensitive cortical cells. However, global topographic ordering was *not* achieved in von der Malsburg's model because the model used a fixed (small) neighborhood. The computer simulation by von der Malsburg was perhaps the first to demonstrate self-organization.
2. Amari (1980) relaxes this restriction on the synaptic weights of the postsynaptic neurons somewhat. The mathematical analysis presented by Amari elucidates the dynamical stability of a cortical map formed by self-organization.
3. Neurobiological feasibility of the self-organizing map (SOM) is discussed in Kohonen (1993, 1997a).
4. The competitive learning rule described in Eq. (9.3) was first introduced into the neural network literature in Grossberg (1969b).
5. In the original form of the SOM algorithm derived by Kohonen (1982), the topological neighborhood is assumed to have a constant amplitude. Let $d_{j,i}$ denote the *lateral distance* between winning neuron i and excited neuron j inside the neighborhood function. The topological neighborhood for the case of a one-dimensional lattice is thus defined by

$$h_{j,i} = \begin{cases} 1, & -K \le d_{j,i} \le K \\ 0, & \text{otherwise} \end{cases} \qquad (1$$

where $2K$ is the overall size of the one-dimensional neighborhood of excited neurons. Contrary to neurobiological considerations, the implication of the model described in Eq. (1) is that all the neurons located inside the topological neighborhood fire at the same rate, and the interaction among those neurons is independent of their lateral distance from the winning neuron i.
6. In Erwin et al. (1992b), it is shown that metastable states, representing topological defects in the configuration of a feature map, arise when the SOM algorithm uses

NEURAL NETWORKS

A Comprehensive Foundation

Second Edition

Simon Haykin
McMaster University
Hamilton, Ontario, Canada

Prentice Hall
Upper Saddle River, New Jersey 07458