1.2 WHAT IS A NEURAL NET?

1.2.1 Artificial Neural Networks

An artificial neural network is an information-processing system that has certain performance characteristics in common with biological neural networks. Artificial neural networks have been developed as generalizations of mathematical models of human cognition or neural biology, based on the assumptions that:

1. Information processing occurs at many simple elements called neurons.
2. Signals are passed between neurons over connection links.
3. Each connection link has an associated weight, which, in a typical neural net, multiplies the signal transmitted.
4. Each neuron applies an activation function (usually nonlinear) to its net input (sum of weighted input signals) to determine its output signal.

A neural network is characterized by (1) its pattern of connections between the neurons (called its architecture), (2) its method of determining the weights on the connections (called its training, or learning, algorithm), and (3) its activation function.

Since what distinguishes (artificial) neural networks from other approaches to information processing provides an introduction to both how and when to use neural networks, let us consider the defining characteristics of neural networks further.

A neural net consists of a large number of simple processing elements called neurons, units, cells, or nodes. Each neuron is connected to other neurons by means of directed communication links, each with an associated weight. The weights represent information being used by the net to solve a problem. Neural nets can be applied to a wide variety of problems, such as storing and recalling data or patterns, classifying patterns, performing general mappings from input patterns to output patterns, grouping similar patterns, or finding solutions to constrained optimization problems.

Each neuron has an internal state, called its activation or activity level, which is a function of the inputs it has received. Typically, a neuron sends its activation as a signal to several other neurons. It is important to note that a neuron can send only one signal at a time, although that signal is broadcast to several other neurons.

For example, consider a neuron \( Y \), illustrated in Figure 1.1, that receives inputs from neurons \( X_1, X_2, \) and \( X_3 \). The activations (output signals) of these neurons are \( x_1, x_2, \) and \( x_3 \), respectively. The weights on the connections from \( X_1, X_2, \) and \( X_3 \) to neuron \( Y \) are \( w_1, w_2, \) and \( w_3 \), respectively. The net input, \( y_{\text{in}} \), to neuron \( Y \) is the sum of the weighted signals from neurons \( X_1, X_2, \) and \( X_3 \), i.e.,

\[
y_{\text{in}} = w_1 x_1 + w_2 x_2 + w_3 x_3.
\]
The activation \( y \) of neuron \( Y \) is given by some function of its net input, 
\[ y = f(x, w, b), \]
e.g., the logistic sigmoid function (an \( S \)-shaped curve) 
\[ f(x) = \frac{1}{1 + \exp(-x)}, \]
or any of a number of other activation functions. Several common activation 
functions are illustrated in Section 1.4.3.

Now suppose further that neuron \( Y \) is connected to neurons \( Z_1 \) and \( Z_2 \), with 
weights \( v_1 \) and \( v_2 \), respectively, as shown in Figure 1.2. Neuron \( Y \) sends its signal 
\( y \) to each of these units. However, in general, the values received by neurons \( Z_1 \) 
and \( Z_2 \) will be different, because each signal is scaled by the appropriate weight, 
\( v_1 \) or \( v_2 \). In a typical net, the activations \( z_1 \) and \( z_2 \) of neurons \( Z_1 \) and \( Z_2 \) would 
depend on inputs from several or even many neurons, not just one, as shown in 
this simple example.

Although the neural network in Figure 1.2 is very simple, the presence of 
a hidden unit, together with a nonlinear activation function, gives it the ability to 
solve many more problems than can be solved by a net with only input and output 
units. On the other hand, it is more difficult to train (i.e., find optimal values for 
the weights) a net with hidden units. The arrangement of the units (the architecture

\[ \text{Input Units} \quad \text{Hidden Units} \quad \text{Output Units} \]
Sec. 1.2 What is a Neural Net?

of the net) and the method of training the net are discussed further in Section 1.4. A detailed consideration of these ideas for specific nets, together with simple examples of an application of each net, is the focus of the following chapters.

1.2.2 Biological Neural Networks

The extent to which a neural network models a particular biological neural system varies. For some researchers, this is a primary concern; for others, the ability of the net to perform useful tasks (such as approximate or represent a function) is more important than the biological plausibility of the net. Although our interest lies almost exclusively in the computational capabilities of neural networks, we shall present a brief discussion of some features of biological neurons that may help to clarify the most important characteristics of artificial neural networks. In addition to being the original inspiration for artificial nets, biological neural systems suggest features that have distinct computational advantages.

There is a close analogy between the structure of a biological neuron (i.e., a brain or nerve cell) and the processing element (or artificial neuron) presented in the rest of this book. In fact, the structure of an individual neuron varies much less from species to species than does the organization of the system of which the neuron is an element.

A biological neuron has three types of components that are of particular interest in understanding an artificial neuron: its dendrites, soma, and axon. The many dendrites receive signals from other neurons. The signals are electric impulses that are transmitted across a synaptic gap by means of a chemical process. The action of the chemical transmitter modifies the incoming signal (typically, by scaling the frequency of the signals that are received) in a manner similar to the action of the weights in an artificial neural network.

The soma, or cell body, sums the incoming signals. When sufficient input is received, the cell fires; that is, it transmits a signal over its axon to other cells. It is often supposed that a cell either fires or doesn’t at any instant of time, so that transmitted signals can be treated as binary. However, the frequency of firing varies and can be viewed as a signal of either greater or lesser magnitude. This corresponds to looking at discrete time steps and summing all activity (signals received or signals sent) at a particular point in time.

The transmission of the signal from a particular neuron is accomplished by an action potential resulting from differential concentrations of ions on either side of the neuron’s axon sheath (the brain’s “white matter”). The ions most directly involved are potassium, sodium, and chloride.

A generic biological neuron is illustrated in Figure 1.3, together with axons from two other neurons (from which the illustrated neuron could receive signals) and dendrites for two other neurons (to which the original neuron would send signals). Several key features of the processing elements of artificial neural networks are suggested by the properties of biological neurons, viz., that:
1. The processing element receives many signals.
2. Signals may be modified by a weight at the receiving synapse.
3. The processing element sums the weighted inputs.
4. Under appropriate circumstances (sufficient input), the neuron transmits a single output.
5. The output from a particular neuron may go to many other neurons (the axon branches).

Other features of artificial neural networks that are suggested by biological neurons are:

6. Information processing is local (although other means of transmission, such as the action of hormones, may suggest means of overall process control).
7. Memory is distributed:
   a. Long-term memory resides in the neurons’ synapses or weights.
   b. Short-term memory corresponds to the signals sent by the neurons.
8. A synapse’s strength may be modified by experience.
9. Neurotransmitters for synapses may be excitatory or inhibitory.

Yet another important characteristic that artificial neural networks share with biological neural systems is fault tolerance. Biological neural systems are fault tolerant in two respects. First, we are able to recognize many input signals that are somewhat different from any signal we have seen before. An example of this is our ability to recognize a person in a picture we have not seen before or to recognize a person after a long period of time.

Second, we are able to tolerate damage to the neural system itself. Humans are born with as many as 100 billion neurons. Most of these are in the brain, and most are not replaced when they die [Johnson & Brown, 1988]. In spite of our continuous loss of neurons, we continue to learn. Even in cases of traumatic neural
loss, other neurons can sometimes be trained to take over the functions of the damaged cells. In a similar manner, artificial neural networks can be designed to be insensitive to small damage to the network, and the network can be retrained in cases of significant damage (e.g., loss of data and some connections).

Even for uses of artificial neural networks that are not intended primarily to model biological neural systems, attempts to achieve biological plausibility may lead to improved computational features. One example is the use of a planar array of neurons, as is found in the neurons of the visual cortex, for Kohonen's self-organizing maps (see Chapter 4). The topological nature of these maps has computational advantages, even in applications where the structure of the output units is not itself significant.

Other researchers have found that computationally optimal groupings of artificial neurons correspond to biological bundles of neurons [Rogers & Kabrisky, 1989]. Separating the action of a backpropagation net into smaller pieces to make it more local (and therefore, perhaps more biologically plausible) also allows improvement in computational power (cf. Section 6.2.3) [D. Fausett, 1990].
4.2 KOHONEN SELF-ORGANIZING MAPS

The self-organizing neural networks described in this section, also called *topology-preserving maps*, assume a topological structure among the cluster units. This property is observed in the brain, but is not found in other artificial neural networks. There are \( m \) cluster units, arranged in a one- or two-dimensional array; the input signals are \( n \)-tuples [Kohonen, 1989a].

The weight vector for a cluster unit serves as an exemplar of the input patterns associated with that cluster. During the self-organization process, the cluster unit whose weight vector matches the input pattern most closely (typically, the square of the minimum Euclidean distance) is chosen as the winner. The winning unit and its neighboring units (in terms of the topology of the cluster units) update their weights. The weight vectors of neighboring units are not, in general, close to the input pattern. For example, for a linear array of cluster units, the neighborhood of radius \( R \) around cluster unit \( J \) consists of all units \( j \) such that \( \max(1, J - R) \leq j \leq \min(J + R, m) \).

The architecture and algorithm that follow for the net can be used to cluster a set of \( p \) continuous-valued vectors \( \mathbf{x} = (x_1, \ldots, x_i, \ldots, x_n) \) into \( m \) clusters. Note that the connection weights do not multiply the signal sent from the input units to the cluster units (unless the dot product measure of similarity is being used).

4.2.1 Architecture

The architecture of the Kohonen self-organizing map is shown in Figure 4.5. Neighborhoods of the unit designated by \# of radii \( R = 2, 1, \) and 0 in a one-dimensional topology (with 10 cluster units) are shown in Figure 4.6.

The neighborhoods of radii \( R = 2, 1 \) and 0 are shown in Figure 4.7 for a rectangular grid and in Figure 4.8 for a hexagonal grid (each with 49 units). In each illustration, the winning unit is indicated by the symbol \"\#\" and the other units are denoted by \"\*\".

Note that each unit has eight nearest neighbors in the rectangular grid, but only six in the hexagonal grid. Winning units that are close to the edge of the grid will have some neighborhoods that have fewer units than that shown in the respective figure. (Neighborhoods do not \"wrap around\" from one side of the grid to the other; \"missing\" units are simply ignored.)
4.2.2 Algorithm

**Step 0.** Initialize weights $w_{ij}$. (Possible choices are discussed below.)
Set topological neighborhood parameters.
Set learning rate parameters.

**Step 1.** While stopping condition is false, do Steps 2–8.

**Step 2.** For each input vector $x$, do Steps 3–5.

**Step 3.** For each $j$, compute:

$$D(j) = \sum_i (w_{ij} - x_i)^2.$$ 

**Step 4.** Find index $J$ such that $D(J)$ is a minimum.

**Step 5.** For all units $j$ within a specified neighborhood of $J$, and for all $i$:

$$w_{ij}(\text{new}) = w_{ij}(\text{old}) + \alpha [x_i - w_{ij}(\text{old})].$$

**Step 6.** Update learning rate.
Sec. 4.2 Kohonen Self-Organizing Maps

Figure 4.7 Neighborhoods for rectangular grid.

Figure 4.8 Neighborhoods for hexagonal grid.
Step 7. Reduce radius of topological neighborhood at specified times.

Step 8. Test stopping condition.

Alternative structures are possible for reducing $R$ and $\alpha$.

The learning rate $\alpha$ is a slowly decreasing function of time (or training epochs). Kohonen (1989a, p. 133) indicates that a linearly decreasing function is satisfactory for practical computations; a geometric decrease would produce similar results.

The radius of the neighborhood around a cluster unit also decreases as the clustering process progresses.

The formation of a map occurs in two phases: the initial formation of the correct order and the final convergence. The second phase takes much longer than the first and requires a small value for the learning rate. Many iterations through the training set may be necessary, at least in some applications [Kohonen, 1989a].

Random values may be assigned for the initial weights. If some information is available concerning the distribution of clusters that might be appropriate for a particular problem, the initial weights can be taken to reflect that prior knowledge. In Examples 4.4–4.9, the weights are initialized to random values (chosen from the same range of values as the components of the input vectors).

4.2.3 Application

Neural networks developed by Kohonen have been applied to an interesting variety of problems. One recent development of his is a neural network approach to computer-generated music [Kohonen, 1989b]. Angeniol, Vaubois, and Le Texier (1988) have applied Kohonen self-organizing maps to the solution of the well-known traveling salesman problem. These applications are discussed briefly later in this section. A more common neural network approach to the traveling salesman problem is discussed in Chapter 7.

Simple example

Example 4.4 A Kohonen self-organizing map (SOM) to cluster four vectors

Let the vectors to be clustered be

$$(1, 1, 0, 0); (0, 0, 0, 1); (1, 0, 0, 0); (0, 0, 1, 1).$$

The maximum number of clusters to be formed is

$$m = 2.$$

Suppose the learning rate (geometric decrease) is

$$\alpha(0) = .6,$$

$$\alpha(t + 1) = .5 \alpha(t).$$
Sec. 4.2 Kohonen Self-Organizing Maps

With only two clusters available, the neighborhood of node \( J \) (Step 4) is set so that only one cluster updates its weights at each step (i.e., \( R = 0 \)).

**Step 0.** Initial weight matrix:

\[
\begin{bmatrix}
.2 & .8 \\
.6 & .4 \\
.5 & .7 \\
.9 & .3
\end{bmatrix}
\]

Initial radius:

\( R = 0 \).

Initial learning rate:

\( \alpha(0) = 0.6 \).

**Step 1.** Begin training.

**Step 2.** For the first vector, (1, 1, 0, 0), do Steps 3–5.

**Step 3.**

\[
D(1) = (.2 - 1)^2 + (.6 - 1)^2
+ (.5 - 0)^2 + (.9 - 0)^2 = 1.86;
\]

\[
D(2) = (.8 - 1)^2 + (.4 - 1)^2
+ (.7 - 0)^2 + (.3 - 0)^2 = 0.98.
\]

**Step 4.** The input vector is closest to output node 2, so

\( J = 2 \).

**Step 5.** The weights on the winning unit are updated:

\[
w_{j2}(\text{new}) = w_{j2}(\text{old}) + .6 \{x_i - w_{j2}(\text{old})\}
= .4 w_{j2}(\text{old}) + .6 x_i.
\]

This gives the weight matrix

\[
\begin{bmatrix}
.2 & .92 \\
.6 & .76 \\
.5 & .28 \\
.9 & .12
\end{bmatrix}
\]

**Step 2.** For the second vector, (0, 0, 0, 1), do Steps 3–5.

**Step 3.**

\[
D(1) = (.2 - 0)^2 + (.6 - 0)^2
+ (.5 - 0)^2 + (.9 - 1)^2 = 0.66;
\]

\[
D(2) = (.92 - 0)^2 + (.76 - 0)^2
+ (.28 - 0)^2 + (.12 - 1)^2 = 2.2768.
\]

**Step 4.** The input vector is closest to output node 1, so

\( J = 1 \).
Step 5. Update the first column of the weight matrix:
\[
\begin{bmatrix}
.08 & .92 \\
.24 & .76 \\
.20 & .28 \\
.96 & .12
\end{bmatrix}
\]

Step 2. For the third vector, \((1, 0, 0, 0)\), do Steps 3–5.

Step 3.
\[
D(1) = (.08 - 1)^2 + (.24 - 0)^2
\]
\[
+ (.2 - 0)^2 + (.96 - 0)^2 = 1.8656;
\]
\[
D(2) = (.92 - 1)^2 + (.76 - 0)^2
\]
\[
+ (.28 - 0)^2 + (.12 - 0)^2 = 0.6768.
\]

Step 4. The input vector is closest to output node 2, so \(J = 2\).

Step 5. Update the second column of the weight matrix:
\[
\begin{bmatrix}
.08 & .968 \\
.24 & .304 \\
.20 & .112 \\
.96 & .048
\end{bmatrix}
\]

Step 2. For the fourth vector, \((0, 0, 1, 1)\), do Steps 3–5.

Step 3.
\[
D(1) = (.08 - 0)^2 + (.24 - 0)^2
\]
\[
+ (.2 - 1)^2 + (.96 - 1)^2 = 0.7056;
\]
\[
D(2) = (.968 - 0)^2 + (.304 - 0)^2
\]
\[
+ (.112 - 1)^2 + (.048 - 1)^2 = 2.724.
\]

Step 4. \(J = 1\).

Step 5. Update the first column of the weight matrix:
\[
\begin{bmatrix}
.032 & .968 \\
.096 & .304 \\
.680 & .112 \\
.984 & .048
\end{bmatrix}
\]

Step 6. Reduce the learning rate:
\[
\alpha = .5 (0.6) = .3
\]

The weight update equations are now
\[
w_{ij}(\text{new}) = w_{ij}(\text{old}) + .3 [x_i - w_{ij}(\text{old})]
\]
\[
= .7w_{ij}(\text{old}) + .3x_i,
\]
The weight matrix after the second epoch of training is

\[
\begin{bmatrix}
0.16 & 0.980 \\
0.047 & 0.360 \\
0.630 & 0.055 \\
0.999 & 0.024
\end{bmatrix}
\]

Modifying the adjustment procedure for the learning rate so that it decreases geometrically from .6 to .01 over 100 iterations (epochs) gives the following results:

Iteration 0: Weight matrix:
\[
\begin{bmatrix}
0.2 & 0.8 \\
0.6 & 0.4 \\
0.5 & 0.7 \\
0.9 & 0.3
\end{bmatrix}
\]

Iteration 1: Weight matrix:
\[
\begin{bmatrix}
0.032 & 0.970 \\
0.096 & 0.300 \\
0.680 & 0.110 \\
0.980 & 0.048
\end{bmatrix}
\]

Iteration 2: Weight matrix:
\[
\begin{bmatrix}
0.0053 & 0.9900 \\
-0.1700 & 0.3000 \\
0.7000 & 0.0200 \\
1.0000 & 0.0086
\end{bmatrix}
\]

Iteration 10: Weight matrix:
\[
\begin{bmatrix}
1.5e-7 & 1.0000 \\
4.6e-7 & 0.3700 \\
0.6300 & 5.4e-7 \\
1.0000 & 2.3e-7
\end{bmatrix}
\]

Iteration 50: Weight matrix:
\[
\begin{bmatrix}
1.9e-19 & 1.0000 \\
5.7e-15 & 0.4700 \\
0.5300 & 6.6e-15 \\
1.0000 & 2.8e-15
\end{bmatrix}
\]

Iteration 100: Weight matrix:
\[
\begin{bmatrix}
6.7e-17 & 1.0000 \\
2.0e-16 & 0.4900 \\
0.5100 & 2.3e-16 \\
1.0000 & 1.0e-16
\end{bmatrix}
\]

These weight matrices appear to be converging to the matrix
\[
\begin{bmatrix}
0.0 & 1.0 \\
0.0 & 0.5 \\
0.5 & 0.0 \\
1.0 & 0.0
\end{bmatrix}
\]

the first column of which is the average of the two vectors placed in cluster 1 and the second column of which is the average of the two vectors placed in cluster 2.
Character Recognition
Examples 4.5–4.7 show typical results from using a Kohonen self-organizing map to cluster input patterns representing letters in three different fonts. The input patterns for fonts 1, 2, and 3 are given in Figure 4.9. In each of the examples, 25 cluster units are available, which means that a maximum of 25 clusters may be formed. Results are shown only for the units that are actually the winning unit for some input pattern after training. The effect of the topological structure is seen in the contrast between Example 4.5 (in which there is no structure), Example 4.6 (in which there is a linear structure as described before), and Example 4.7 (in which a rectangular structure is used). In each example, the learning rate is reduced linearly from an initial value of .6 to a final value of .01.

Example 4.5 A SOM to cluster letters from different fonts: no topological structure
If no structure is assumed for the cluster units, i.e., if only the winning unit is allowed to learn the pattern presented, the 21 patterns form 5 clusters:

<table>
<thead>
<tr>
<th>UNIT</th>
<th>PATTERNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>C1, C2, C3</td>
</tr>
<tr>
<td>13</td>
<td>B1, B3, D1, D3, E1, K1, K3</td>
</tr>
<tr>
<td>16</td>
<td>A1, A2, A3</td>
</tr>
<tr>
<td>18</td>
<td>J1, J2, J3</td>
</tr>
<tr>
<td>24</td>
<td>B2, D2, E2, K2</td>
</tr>
</tbody>
</table>

Example 4.6 A SOM to cluster letters from different fonts: linear structure
A linear structure (with $R = 1$) gives a better distribution of the patterns onto the available cluster units. The winning node $J$ and its topological neighbors ($J + 1$ and $J - 1$) are allowed to learn on each iteration. Note that in general, the neighboring nodes that learn do not initially have weight vectors that are particularly close to the input pattern.

<table>
<thead>
<tr>
<th>UNIT</th>
<th>PATTERNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>K2</td>
</tr>
<tr>
<td>10</td>
<td>J1, J2, J3</td>
</tr>
<tr>
<td>14</td>
<td>E1, E3</td>
</tr>
<tr>
<td>16</td>
<td>K1, K3</td>
</tr>
<tr>
<td>18</td>
<td>B1, B3, D1, D3</td>
</tr>
</tbody>
</table>

Note also that in many cases there are unused units between a pair of units that have clusters of patterns associated with them. This suggests that units which are being pulled in opposite directions during training do not learn any pattern very well. (In other words, in most cases, these input patterns form very distinct classes.)

Example 4.7 A SOM to cluster letters from different fonts: diamond structure
In this example, a simple two-dimensional topology is assumed for the cluster units, so that each cluster unit is indexed by two subscripts. If unit $X_{j,j}$ is the winning unit, the units $X_{j+1,j}$, $X_{j-1,j}$, $X_{j,j+1}$, and $X_{j,j-1}$ also learn. This gives a diamond to-
Figure 4.9 Training input patterns for character recognition examples.
Spanning tree

Example 4.8 Using a SOM: Spanning Tree Data

The 32 vectors [Kohonen, 1989a] shown in Figure 4.11 were presented in random order to a Kohonen self-organizing map with a rectangular topology on its cluster units. There were 70 cluster units arranged in a 10 × 7 array. The pattern names are for ease of identification of the results. The relationships between the patterns can be displayed graphically, as in Figure 4.12 [Kohonen, 1989a]: patterns that are adjacent to each other in the diagram differ by exactly 1 bit.

The net was used with random initial weights. In this example, the initial radius, \( R = 3 \), was reduced by 1 after each set of 75 iterations. During these 75 iterations, the learning rate was reduced linearly from .6 to .01. If unit \( X_{i,j} \) is the winning unit, the units \( X_{i,j} \) for all \( i \) and \( j \) such that \( I - R \leq i \leq I + R \) and \( J - R \leq j \leq J + R \) also learn (unless the value of \( i \) or \( j \) falls outside the permissible range for the topology and number of cluster units chosen). Note that when \( R = 3 \), as many as 49 units will learn (see Figure 4.7). When the Kohonen net is used with \( R = 0 \), only the winning cluster node is allowed to learn.

Figures 4.13–4.16 show the evolution of the solution, as \( R \) is decreased, for the data in Figure 4.11, using a rectangular array for the cluster units. The structure of the data is shown in Figure 4.16 to indicate how the positioning of the patterns on the cluster units reflects the spanning tree relationships among the patterns.

A hexagonal grid can also be used for a two-dimensional topology. The final results obtained using such a grid are shown in Figure 4.17. As in Figure 4.16, the structure of the data is also indicated to show how the position of the patterns on the cluster units reflects the original spanning tree. The same iteration scheme was used as before, i.e., 75 iterations at each radius, starting with \( R = 3 \) and decreasing to \( R = 0 \).

Other examples

Example 4.9 Using a SOM: A Geometric Example

The cluster units in a Kohonen self-organizing map can be viewed as having a position (given by their weight vector). For input patterns with two components, this position is easy to represent graphically. The topological relationships between cluster units
### Sec. 4.2 Kohonen Self-Organizing Maps

<table>
<thead>
<tr>
<th>PATTERN</th>
<th>COMPONENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
</tr>
<tr>
<td>E</td>
<td>5</td>
</tr>
<tr>
<td>F</td>
<td>3</td>
</tr>
<tr>
<td>G</td>
<td>3</td>
</tr>
<tr>
<td>H</td>
<td>3</td>
</tr>
<tr>
<td>I</td>
<td>3</td>
</tr>
<tr>
<td>J</td>
<td>3</td>
</tr>
<tr>
<td>K</td>
<td>3</td>
</tr>
<tr>
<td>L</td>
<td>3</td>
</tr>
<tr>
<td>M</td>
<td>3</td>
</tr>
<tr>
<td>N</td>
<td>3</td>
</tr>
<tr>
<td>O</td>
<td>3</td>
</tr>
<tr>
<td>P</td>
<td>3</td>
</tr>
<tr>
<td>Q</td>
<td>3</td>
</tr>
<tr>
<td>R</td>
<td>3</td>
</tr>
<tr>
<td>S</td>
<td>3</td>
</tr>
<tr>
<td>T</td>
<td>3</td>
</tr>
<tr>
<td>U</td>
<td>3</td>
</tr>
<tr>
<td>V</td>
<td>3</td>
</tr>
<tr>
<td>W</td>
<td>3</td>
</tr>
<tr>
<td>X</td>
<td>3</td>
</tr>
<tr>
<td>Y</td>
<td>3</td>
</tr>
<tr>
<td>Z</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
</tr>
</tbody>
</table>

Figure 4.11 Spanning tree test data [Kohonen, 1989a].

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
</tr>
<tr>
<td>I</td>
<td>S</td>
<td>W</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>T</td>
<td>X</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>U</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>Z</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.12 Spanning tree test data structure [Kohonen, 1989a].
<table>
<thead>
<tr>
<th>L,J</th>
<th>H.K</th>
<th>G</th>
<th>F</th>
<th>A,B,C,D,E</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L,M</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>T.U</td>
<td></td>
<td>V</td>
<td>Y,Z</td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>W</td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Q,R</td>
<td></td>
<td></td>
<td></td>
<td>4,5,6</td>
</tr>
</tbody>
</table>

**Figure 4.13** Results after 75 iterations with $R = 3$.

<table>
<thead>
<tr>
<th>H,L,J</th>
<th>G</th>
<th>A,B</th>
<th>F</th>
<th>C</th>
<th>D,E</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L,M</td>
<td>S</td>
<td>T.U</td>
<td></td>
<td></td>
<td>Y,Z</td>
</tr>
<tr>
<td>N</td>
<td>V</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>W</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q,R</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>5,6</td>
</tr>
</tbody>
</table>

**Figure 4.14** Results after 75 additional iterations with $R = 2$.

<table>
<thead>
<tr>
<th>L,J</th>
<th>G</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D,E</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>H</td>
<td></td>
<td></td>
<td>F</td>
<td></td>
</tr>
<tr>
<td></td>
<td>K</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td></td>
<td>V</td>
<td></td>
<td>Z</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>W</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>6</td>
<td></td>
<td>4</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

**Figure 4.15** Results after 75 more iterations with $R = 1$. 
Figure 4.16 Results after another 75 iterations with $R = 0$.

Figure 4.17 Results of spanning tree example using hexagonal array.
in Kohonen self-organizing maps are often indicated by drawing lines connecting the units.

In this example, we assume a linear structure. The initial weights are chosen randomly, with each component having a value between $-1$ and $1$. There are $50$ cluster units. The $100$ input vectors are chosen randomly from within a circle of radius $0.5$ (centered at the origin). The initial learning rate is $0.5$; it is reduced linearly to $0.01$ over $100$ epochs. Throughout training, the winning unit and its nearest neighbor unit on either side (units $J$, $J + 1$, and $J - 1$) are allowed to learn.

Figure 4.18 shows the training patterns. Figures 4.19–4.23 show the cluster units initially and after $10$, $20$, $30$, and $100$ epochs, respectively. Not only have the cluster units moved to represent the training inputs (i.e., all of the weight vectors for the cluster units now fall within the unit circle), but the curve connecting the cluster units has smoothed out somewhat as training progresses. An even smoother curve can be obtained by starting with a larger radius and gradually reducing it to $0$. This would involve using more training epochs. (See Kohonen, 1989a for many other interesting examples of this geometric interpretation of self-organizing maps.)

Example 4.10 Using a SOM: The Traveling Salesman Problem

In this example, we illustrate the use of the linear topology for the cluster units in a Kohonen self-organizing map to solve a classic problem in constrained optimization, the so-called traveling salesman problem (TSP). Several nets that are designed for constrained optimization problems are discussed in Chapter 7. The aim of the TSP is to find a tour of a given set of cities that is of minimum length. A tour consists

---

**Figure 4.18** Input patterns.
Figure 4.19  Initial cluster units.

Figure 4.20  Cluster units after 10 epochs.
Figure 4.21  Cluster units after 20 epochs.

Figure 4.22  Cluster units after 30 epochs.
of visiting each city exactly once and returning to the starting city. Angeniol, Vaubois, and Le Texier (1988) have illustrated the use of a Kohonen net to solve the TSP. The net uses the city coordinates as input; there are as many cluster units as there are cities to be visited. The net has a linear topology, with the first and last cluster unit also connected. Figure 4.24 shows the initial random position of the cluster units; Figure 4.25 shows the results after 100 epochs of training with $R = 1$ (learning rate decreasing from 0.5 to 0.4). The final tour after 100 epochs of training with $R = 0$ is shown in Figure 4.26.

This tour is ambiguous in terms of the order in which city $B$ and city $C$ are visited, because one cluster unit is positioned midway between the cities (rather than being directly on one city). Another unit has been trapped between city $J$ and cities $B$ and $C$; it is not being chosen as the winner when any input is presented and is therefore “wasted.” However, the results can easily be interpreted as representing one of the tours

\[ A \ D \ E \ F \ G \ H \ I \ J \ B \ C \]

and

\[ A \ D \ E \ F \ G \ H \ I \ J \ C \ B. \]

The coordinates of and distances between the cities are given in Chapter 7.

The same tour (with the same ambiguity) was found, using a variety of initial weights. Choosing initial weights within a small region of the input space (the center or any of the four corners), as is often done, did not change the results.
Figure 4.24  Initial position of cluster units and location of cities

Figure 4.25  Position of cluster units and location of cities after 100 epochs with $R = 1$. 
4.3 LEARNING VECTOR QUANTIZATION

Learning vector quantization (LVQ) [Kohonen, 1989a, 1990a] is a pattern classification method in which each output unit represents a particular class or category. (Several output units should be used for each class.) The weight vector for an output unit is often referred to as a reference (or codebook) vector for the class that the unit represents. During training, the output units are positioned (by adjusting their weights through supervised training) to approximate the decision surfaces of the theoretical Bayes classifier. It is assumed that a set of training patterns with known classifications is provided, along with an initial distribution of reference vectors (each of which represents a known classification).

After training, an LVQ net classifies an input vector by assigning it to the same class as the output unit that has its weight vector (reference vector) closest to the input vector.