# 16 Competitive Networks



## Objectives

The Hamming network, introduced in Chapter 3, demonstrated one technique for using a neural network for pattern recognition. It required that the prototype patterns be known beforehand and incorporated into the network as rows of a weight matrix.

In this chapter we will discuss networks that are very similar in structure and operation to the Hamming network. Unlike the Hamming network, however, they use the associative learning rules of Chapter 15 to adaptively learn to classify patterns. Three such networks are introduced in this chapter: the competitive network, the feature map and the learning vector quantization (LVQ) network.

# Theory and Examples

The Hamming network is one of the simplest examples of a competitive network. The neurons in the output layer of the Hamming network compete with each other to determine a winner. The winner indicates which prototype pattern is most representative of the input pattern. The competition is implemented by lateral inhibition — a set of negative connections between the neurons in the output layer. In this chapter we will illustrate how this competition can be combined with the associative learning rules of Chapter 15 to produce powerful self-organizing (unsupervised) networks.

As early as 1959, Frank Rosenblatt created a simple “spontaneous” classifier, an unsupervised network based on the perceptron, which learned to classify input vectors into two classes with roughly equal members.

In the late 1960s and early 1970s, Stephen Grossberg introduced many competitive networks that used lateral inhibition to good effect. Some of the useful behaviors he obtained were noise suppression, contrast-enhancement and vector normalization. His networks will be examined in Chapters 18 and 19.

In 1973, Christoph von der Malsburg introduced a self-organizing learning rule that allowed a network to classify inputs in such a way that neighboring neurons responded to similar inputs. The topology of his network mimicked, in some ways, the structures previously found in the visual cortex of cats by David Hubel and Torten Wiesel. His learning rule generated a great deal of interest, but it used a nonlocal calculation to ensure that weights were normalized. This made it less biologically plausible.

Grossberg extended von der Malsburg's work by rediscovering the instar rule, examined in Chapter 15. (The instar rule had previously been introduced by Nils Nilsson in his 1965 book *Learning Machines*.) Grossberg showed that the instar rule removed the necessity of re-normalizing weights, since weight vectors that learn to recognize normalized input vectors will automatically be normalized themselves.

The work of Grossberg and von der Malsburg emphasizes the biological plausibility of their networks. Another influential researcher, Teuvo Kohonen, has also been a strong proponent of competitive networks. However, his emphasis has been on engineering applications and efficient mathematical descriptions of the networks. During the 1970s he developed a simplified version of the instar rule and also, inspired by the work of von der Malsburg and Grossberg, found an efficient way to incorporate topology into a competitive network.

In this chapter we will concentrate on the Kohonen framework for competitive networks. His models illustrate the major features of competitive net-

works, and yet they are mathematically more tractable than the Grossberg networks. They provide a good introduction to competitive learning.

We will begin with the simple competitive network. Next we will present the *self-organizing feature map*, which incorporates a network topology. Finally, we will discuss *learning vector quantization*, which incorporates competition within a supervised learning framework.

## Hamming Network

Since the competitive networks discussed in this chapter are closely related to the Hamming network (shown in Figure 16.1), it is worth reviewing the key concepts of that network first.

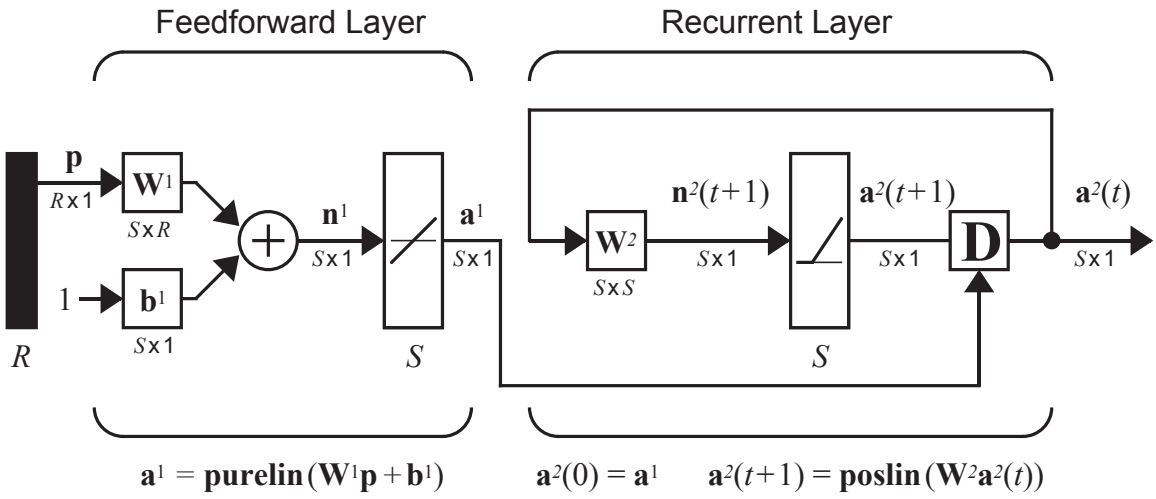



Figure 16.1 Hamming Network

The Hamming network consists of two layers. The first layer (which is a layer of instars) performs a correlation between the input vector and the prototype vectors. The second layer performs a competition to determine which of the prototype vectors is closest to the input vector.

### Layer 1

Recall from Chapter 15 (see page 15-9 and following) that a single instar is able to recognize only one pattern. In order to allow multiple patterns to be classified, we need to have multiple instars. This is accomplished in the Hamming network.

Suppose that we want the network to recognize the following prototype vectors:

$$\{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_Q\}. \tag{16.1}$$

Then the weight matrix, $\mathbf{W}^1$, and the bias vector, $\mathbf{b}^1$, for Layer 1 will be:

$$\mathbf{W}^1 = \begin{bmatrix} {}_1\mathbf{w}^T \\ {}_2\mathbf{w}^T \\ \vdots \\ {}_S\mathbf{w}^T \end{bmatrix} = \begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \\ \vdots \\ \mathbf{p}_Q^T \end{bmatrix}, \quad \mathbf{b}^1 = \begin{bmatrix} R \\ R \\ \vdots \\ R \end{bmatrix}, \tag{16.2}$$

where each row of $\mathbf{W}^1$ represents a prototype vector which we want to recognize, and each element of $\mathbf{b}^1$ is set equal to the number of elements in each input vector ($R$). (The number of neurons, $S$, is equal to the number of prototype vectors which are to be recognized, $Q$.)

Thus, the output of the first layer is

$$\mathbf{a}^1 = \mathbf{W}^1\mathbf{p} + \mathbf{b}^1 = \begin{bmatrix} \mathbf{p}_1^T\mathbf{p} + R \\ \mathbf{p}_2^T\mathbf{p} + R \\ \vdots \\ \mathbf{p}_Q^T\mathbf{p} + R \end{bmatrix}. \tag{16.3}$$

Note that the outputs of Layer 1 are equal to the inner products of the prototype vectors with the input, plus $R$. As we discussed in Chapter 3 (page 3-9), these inner products indicate how close each of the prototype patterns is to the input vector. (This was also discussed in our presentation of the instar on page 15-10.)

## Layer 2

In the instar of Chapter 15, a *hardlim* transfer function was used to decide if the input vector was close enough to the prototype vector. In Layer 2 of the Hamming network we have multiple instars, therefore we want to decide which prototype vector is closest to the input. Instead of the *hardlim* transfer function, we will use a competitive layer to choose the closest prototype.

Layer 2 is a competitive layer. The neurons in this layer are initialized with the outputs of the feedforward layer, which indicate the correlation between the prototype patterns and the input vector. Then the neurons compete with each other to determine a winner. After the competition, only one neuron will have a nonzero output. The winning neuron indicates which category of input was presented to the network (each prototype vector represents a category).

The first-layer output $\mathbf{a}^1$ is used to initialize the second layer.

$$\mathbf{a}^2(0) = \mathbf{a}^1 \tag{16.4}$$

Then the second-layer output is updated according to the following recurrence relation:

$$\mathbf{a}^2(t+1) = \mathbf{poslin}(\mathbf{W}^2\mathbf{a}^2(t)). \tag{16.5}$$

The second-layer weights $\mathbf{W}^2$ are set so that the diagonal elements are 1, and the off-diagonal elements have a small negative value.

$$w_{ij}^2 = \begin{cases} 1, & \text{if } i = j \\ -\varepsilon, & \text{otherwise} \end{cases}, \text{ where } 0 < \varepsilon < \frac{1}{S-1} \tag{16.6}$$

Lateral Inhibition

This matrix produces *lateral inhibition*, in which the output of each neuron has an inhibitory effect on all of the other neurons. To illustrate this effect, substitute weight values of 1 and $-\varepsilon$ for the appropriate elements of $\mathbf{W}^2$, and rewrite Eq. (16.5) for a single neuron.

$$a_i^2(t+1) = poslin\left(a_i^2(t) - \varepsilon\sum_{j\neq i} a_j^2(t)\right) \tag{16.7}$$

At each iteration, each neuron's output will decrease in proportion to the sum of the other neurons' outputs (with a minimum output of 0). The output of the neuron with the largest initial condition will decrease more slowly than the outputs of the other neurons. Eventually that neuron will be the only one with a positive output. At this point the network has reached steady state. The index of the second-layer neuron with a stable positive output is the index of the prototype vector that best matched the input.

Winner-Take-All

This is called a *winner-take-all competition*, since only one neuron will have a nonzero output. In Chapter 18 we will discuss other types of competition.

*You may wish to experiment with the Hamming network and the apple/orange classification problem. The Neural Network Design Demonstration Hamming Classification* (`nnd3hamc`) *was previously introduced in Chapter 3.*

## Competitive Layer

Competition

The second-layer neurons in the Hamming network are said to be in *competition* because each neuron excites itself and inhibits all the other neurons. To simplify our discussions in the remainder of this chapter, we will define a transfer function that does the job of a recurrent competitive layer:

$$\mathbf{a} = \mathbf{compet}(\mathbf{n}). \tag{16.8}$$

It works by finding the index $i^*$ of the neuron with the largest net input, and setting its output to 1 (with ties going to the neuron with the lowest index). All other outputs are set to 0.

$$a_i = \begin{cases} 1, i = i^* \\ 0, i \neq i^* \end{cases}, \text{ where } n_{i^*} \geq n_i, \forall i, \text{ and } i^* \leq i, \forall n_i = n_{i^*} \tag{16.9}$$

Replacing the recurrent layer of the Hamming network with a competitive transfer function on the first layer will simplify our presentations in this chapter. (We will study the competition process in more detail in Chapter 18.) A competitive layer is displayed in Figure 16.2.

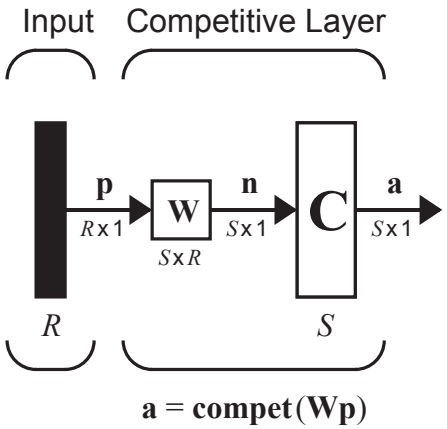



Figure 16.2 Competitive Layer

As with the Hamming network, the prototype vectors are stored in the rows of $\mathbf{W}$. The net input $\mathbf{n}$ calculates the distance between the input vector $\mathbf{p}$ and each prototype ${}_i\mathbf{w}$ (assuming vectors have normalized lengths of $L$). The net input $n_i$ of each neuron $i$ is proportional to the angle $\theta_i$ between $\mathbf{p}$ and the prototype vector ${}_i\mathbf{w}$:

$$\mathbf{n} = \mathbf{W}\mathbf{p} = \begin{bmatrix} {}_1\mathbf{w}^T \\ {}_2\mathbf{w}^T \\ \vdots \\ {}_S\mathbf{w}^T \end{bmatrix} \mathbf{p} = \begin{bmatrix} {}_1\mathbf{w}^T\mathbf{p} \\ {}_2\mathbf{w}^T\mathbf{p} \\ \vdots \\ {}_S\mathbf{w}^T\mathbf{p} \end{bmatrix} = \begin{bmatrix} L^2\cos\theta_1 \\ L^2\cos\theta_2 \\ \vdots \\ L^2\cos\theta_S \end{bmatrix}. \tag{16.10}$$

The competitive transfer function assigns an output of 1 to the neuron whose weight vector points in the direction closest to the input vector:

$$\mathbf{a} = \mathbf{compet}(\mathbf{W}\mathbf{p}). \tag{16.11}$$

*To experiment with the competitive network and the apple/orange classification problem, use the Neural Network Design Demonstration Competitive Classification* (**nnd14cc**).

## Competitive Learning

We can now design a competitive network classifier by setting the rows of $\mathbf{W}$ to the desired prototype vectors. However, we would like to have a learning rule that could be used to train the weights in a competitive network, without knowing the prototype vectors. One such learning rule is the instar rule from Chapter 15:

$$ {}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) + \alpha a_i(q)(\mathbf{p}(q) - {}_i\mathbf{w}(q-1)) . \tag{16.12}$$

For the competitive network, $\mathbf{a}$ is only nonzero for the winning neuron ($i = i^*$). Therefore, we can get the same results using the Kohonen rule.

$$\begin{aligned} {}_i\mathbf{w}(q) &= {}_i\mathbf{w}(q-1) + \alpha(\mathbf{p}(q) - {}_i\mathbf{w}(q-1)) \\ &= (1-\alpha){}_i\mathbf{w}(q-1) + \alpha\mathbf{p}(q) \end{aligned} \tag{16.13}$$

and

$$ {}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) \qquad i \neq i^* \tag{16.14}$$

Thus, the row of the weight matrix that is closest to the input vector (or has the largest inner product with the input vector) moves toward the input vector. It moves along a line between the old row of the weight matrix and the input vector, as shown in Figure 16.3.

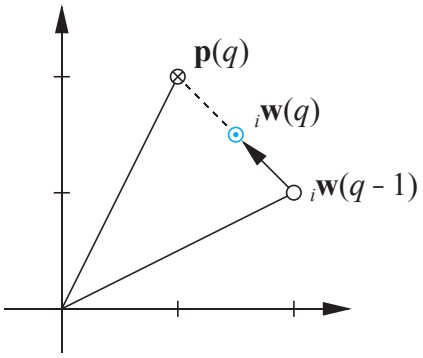



Figure 16.3 Graphical Representation of the Kohonen Rule

Let's use the six vectors in Figure 16.4 to demonstrate how a competitive layer learns to classify vectors. Here are the six vectors:

$$\begin{aligned} \mathbf{p}_1 &= \begin{bmatrix} -0.1961 \\ 0.9806 \end{bmatrix}, \mathbf{p}_2 = \begin{bmatrix} 0.1961 \\ 0.9806 \end{bmatrix}, \mathbf{p}_3 = \begin{bmatrix} 0.9806 \\ 0.1961 \end{bmatrix} \\ \mathbf{p}_4 &= \begin{bmatrix} 0.9806 \\ -0.1961 \end{bmatrix}, \mathbf{p}_5 = \begin{bmatrix} -0.5812 \\ -0.8137 \end{bmatrix}, \mathbf{p}_6 = \begin{bmatrix} -0.8137 \\ -0.5812 \end{bmatrix} . \end{aligned} \tag{16.15}$$

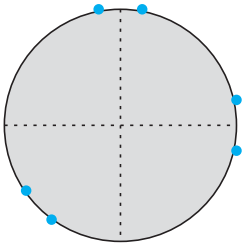
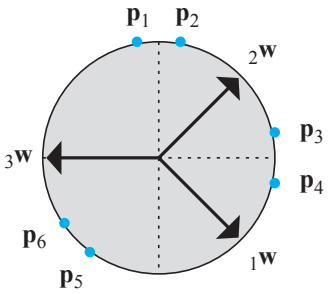


Figure 16.4 Sample Input Vectors

Our competitive network will have three neurons, and therefore it can classify vectors into three classes. Here are the "randomly" chosen normalized initial weights:

$$ {}_1\mathbf{w} = \begin{bmatrix} 0.7071 \\ -0.7071 \end{bmatrix}, {}_2\mathbf{w} = \begin{bmatrix} 0.7071 \\ 0.7071 \end{bmatrix}, {}_3\mathbf{w} = \begin{bmatrix} -1.0000 \\ 0.0000 \end{bmatrix}, \mathbf{W} = \begin{bmatrix} {}_1\mathbf{w}^T \\ {}_2\mathbf{w}^T \\ {}_3\mathbf{w}^T \end{bmatrix}. \quad (16.16) $$




The data vectors are shown at left, with the weight vectors displayed as arrows. Let's present the vector $\mathbf{p}_2$ to the network:

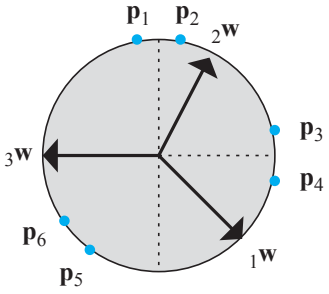
$$ \mathbf{a} = \mathbf{compet}(\mathbf{W}\mathbf{p}_2) = \mathbf{compet}\left( \begin{bmatrix} 0.7071 & -0.7071 \\ 0.7071 & 0.7071 \\ -1.0000 & 0.0000 \end{bmatrix} \begin{bmatrix} 0.1961 \\ 0.9806 \end{bmatrix} \right) \quad (16.17) $$

$$ = \mathbf{compet}\left( \begin{bmatrix} -0.5547 \\ 0.8321 \\ -0.1961 \end{bmatrix} \right) = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}. $$

The second neuron's weight vector was closest to $\mathbf{p}_2$, so it won the competition ($i^* = 2$) and output a 1. We now apply the Kohonen learning rule to the winning neuron with a learning rate of $\alpha = 0.5$.

$$ {}_2\mathbf{w}^{new} = {}_2\mathbf{w}^{old} + \alpha(\mathbf{p}_2 - {}_2\mathbf{w}^{old}) \quad (16.18) $$

$$ = \begin{bmatrix} 0.7071 \\ 0.7071 \end{bmatrix} + 0.5\left( \begin{bmatrix} 0.1961 \\ 0.9806 \end{bmatrix} - \begin{bmatrix} 0.7071 \\ 0.7071 \end{bmatrix} \right) = \begin{bmatrix} 0.4516 \\ 0.8438 \end{bmatrix} $$




The Kohonen rule moves ${}_2\mathbf{w}$ closer to $\mathbf{p}_2$, as can be seen in the diagram at left. If we continue choosing input vectors at random and presenting them to the network, then at each iteration the weight vector closest to the input vector will move toward that vector. Eventually, each weight vector will

point at a different cluster of input vectors. Each weight vector becomes a prototype for a different cluster.

This problem is simple enough that we can predict which weight vector will point at which cluster. The final weights will look something like those shown in Figure 16.5.

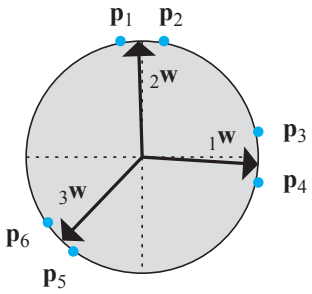
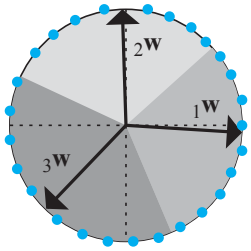
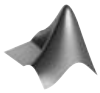



Figure 16.5 Final Weights

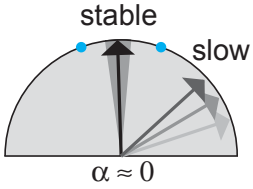



Once the network has learned to cluster the input vectors, it will classify new vectors accordingly.The diagram in the left margin uses shading to show which region each neuron will respond to. The competitive layer assigns each input vector **p** to one of these classes by producing an output of 1 for the neuron whose weight vector is closest to **p**.

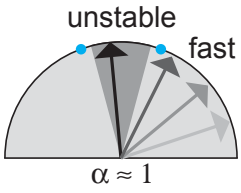


*To experiment with the competitive learning use the Neural Network Design Demonstration Competitive Learning* (`nnd14cl`).

## Problems with Competitive Layers




Competitive layers make efficient adaptive classifiers, but they do suffer from a few problems. The first problem is that the choice of learning rate forces a trade-off between the speed of learning and the stability of the final weight vectors. A learning rate near zero results in slow learning. However, once a weight vector reaches the center of a cluster it will tend to stay close to the center.




In contrast, a learning rate near 1.0 results in fast learning. However, once the weight vector has reached a cluster, it will continue to oscillate as different vectors in the cluster are presented.

Sometimes this trade-off between fast learning and stability can be used to advantage. Initial training can be done with a large learning rate for fast learning. Then the learning rate can be decreased as training progresses, to achieve stable prototype vectors. Unfortunately, this technique will not work if the network needs to continuously adapt to new arrangements of input vectors.

A more serious stability problem occurs when clusters are close together. In certain cases, a weight vector forming a prototype of one cluster may “in-

vade" the territory of another weight vector, and therefore upset the current classification scheme.

The series of four diagrams in Figure 16.6 illustrate this problem. Two input vectors (shown with blue circles in diagram (a)) are presented several times. The result is that the weight vectors representing the middle and right clusters shift to the right. Eventually one of the right cluster vectors is reclassified by the center weight vector. Further presentations move the middle vector over to the right until it "loses" some of its vectors, which then become part of the class associated with the left weight vector.

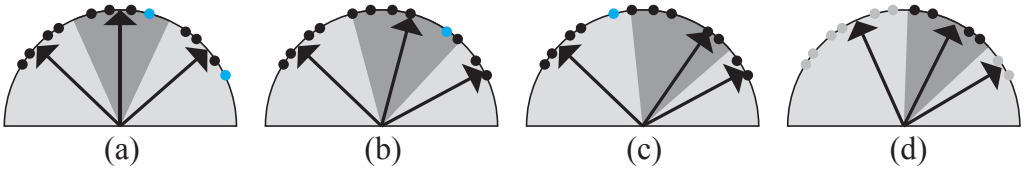
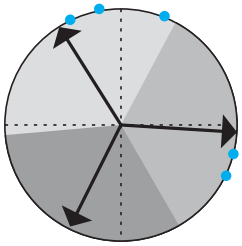



Figure 16.6 Example of Unstable Learning

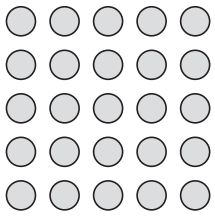


A third problem with competitive learning is that occasionally a neuron's initial weight vector is located so far from any input vectors that it never wins the competition, and therefore never learns. The result is a "dead" neuron, which does nothing useful. For example, the downward-pointing weight vector in the diagram to the left will never learn, regardless of the order in which vectors are presented. One solution to this problem consists of adding a negative bias to the net input of each neuron and then decreasing the bias each time the neuron wins. This will make it harder for a neuron to win the competition if it has won often. This mechanism is sometimes called a "conscience." (See Exercise E16.4.)

Finally, a competitive layer always has as many classes as it has neurons. This may not be acceptable for some applications, especially when the number of clusters is not known in advance. In addition, for competitive layers, each class consists of a convex region of the input space. Competitive layers cannot form classes with nonconvex regions or classes that are the union of unconnected regions.

Some of the problems discussed in this section are solved by the feature map and LVQ networks, which are introduced in later sections of this chapter, and the ART networks, which are presented in Chapter 19.

## Competitive Layers in Biology

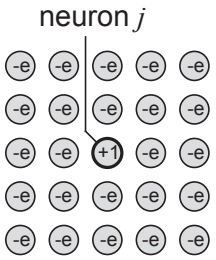


In previous chapters we have made no mention of how neurons are physically organized within a layer (the topology of the network). In biological neural networks, neurons are typically arranged in two-dimensional layers, in which they are densely interconnected through lateral feedback. The diagram to the left shows a layer of twenty-five neurons arranged in a two-dimensional grid.

Often weights vary as a function of the distance between the neurons they connect. For example, the weights for Layer 2 of the Hamming network are assigned as follows:

$$w_{ij} = \begin{cases} 1, & \text{if } i = j \\ -\varepsilon, & \text{if } i \neq j \end{cases} . \tag{16.19}$$

Eq. (16.20) assigns the same values as Eq. (16.19), but in terms of the distances $d_{ij}$ between neurons:




$$w_{ij} = \begin{cases} 1, & \text{if } d_{ij} = 0 \\ -\varepsilon, & \text{if } d_{ij} > 0 \end{cases} . \tag{16.20}$$

Either Eq. (16.19) or Eq. (16.20) will assign the weight values shown in the diagram at left. Each neuron $i$ is labeled with the value of the weight $w_{ij}$, which comes from it to the neuron marked $j$.

On-center/off-surround

The term *on-center/off-surround* is often used to describe such a connection pattern between neurons. Each neuron reinforces itself (center), while inhibiting all other neurons (surround).

It turns out that this is a crude approximation of biological competitive layers. In biology, a neuron reinforces not only itself, but also those neurons close to it. Typically, the transition from reinforcement to inhibition occurs smoothly as the distance between neurons increases.

This transition is illustrated on the left side of Figure 16.7. This is a function that relates the distance between neurons to the weight connecting them. Those neurons that are close provide excitatory (reinforcing) connections, and the magnitude of the excitation decreases as the distance increases. Beyond a certain distance, the neurons begin to have inhibitory connections, and the inhibition increases as the distance increases. Because of its shape, the function is referred to as the *Mexican-hat function*. On the right side of Figure 16.7 is a two-dimensional illustration of the Mexican-hat (on-center/off-surround) function. Each neuron $i$ is marked to show the sign and relative strength of its weight $w_{ij}$ going to neuron $j$.

Mexican-Hat Function

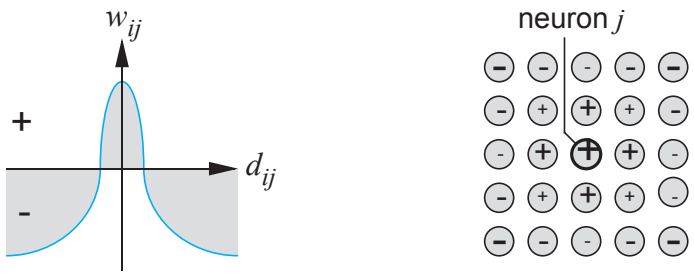



Figure 16.7 On-Center/Off-Surround Layer in Biology

Biological competitive systems, in addition to having a gradual transition between excitatory and inhibitory regions of the on-center/off-surround connection pattern, also have a weaker form of competition than the winner-take-all competition of the Hamming network. Instead of a single active neuron (winner), biological networks generally have "bubbles" of activity that are centered around the most active neuron. This is caused in part by the form of the on-center/off-surround connectivity pattern and also by nonlinear feedback connections. (See the discussion on contour enhancement in Chapter 18.)

## Self-Organizing Feature Maps

SOFM

In order to emulate the activity bubbles of biological systems, without having to implement the nonlinear on-center/off-surround feedback connections, Kohonen designed the following simplification. His self-organizing feature map (SOFM) network first determines the winning neuron $i^*$ using the same procedure as the competitive layer. Next, the weight vectors for all neurons within a certain neighborhood of the winning neuron are updated using the Kohonen rule,

$$\begin{aligned} {}_i\mathbf{w}(q) &= {}_i\mathbf{w}(q-1) + \alpha(\mathbf{p}(q) - {}_i\mathbf{w}(q-1)) \\ &= (1-\alpha){}_i\mathbf{w}(q-1) + \alpha\mathbf{p}(q) \end{aligned} \qquad i \in N_{i^*}(d), \tag{16.21}$$

Neighborhood

where the *neighborhood* $N_{i^*}(d)$ contains the indices for all of the neurons that lie within a radius $d$ of the winning neuron $i^*$:

$$N_i(d) = \{j, d_{ij} \le d\}. \tag{16.22}$$

When a vector $\mathbf{p}$ is presented, the weights of the winning neuron *and* its neighbors will move toward $\mathbf{p}$. The result is that, after many presentations, neighboring neurons will have learned vectors similar to each other.


To demonstrate the concept of a neighborhood, consider the two diagrams shown in Figure 16.8. The left diagram illustrates a two-dimensional neighborhood of radius $d = 1$ around neuron 13. The right diagram shows a neighborhood of radius $d = 2$.

The definition of these neighborhoods would be

$$N_{13}(1) = \{8, 12, 13, 14, 18\}, \tag{16.23}$$

$$N_{13}(2) = \{3, 7, 8, 9, 11, 12, 13, 14, 15, 17, 18, 19, 23\}. \tag{16.24}$$

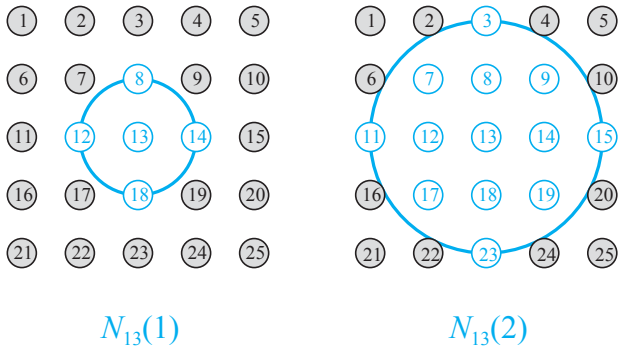



Figure 16.8 Neighborhoods

We should mention that the neurons in an SOFM do not have to be arranged in a two-dimensional pattern. It is possible to use a one-dimensional arrangement, or even three or more dimensions. For a one-dimensional SOFM, a neuron will only have two neighbors within a radius of 1 (or a single neighbor if the neuron is at the end of the line). It is also possible to define distance in different ways. For instance, Kohonen has suggested rectangular and hexagonal neighborhoods for efficient implementation. The performance of the network is not sensitive to the exact shape of the neighborhoods.

Now let's demonstrate the performance of an SOFM network. Figure 16.9 shows a feature map and the two-dimensional topology of its neurons.

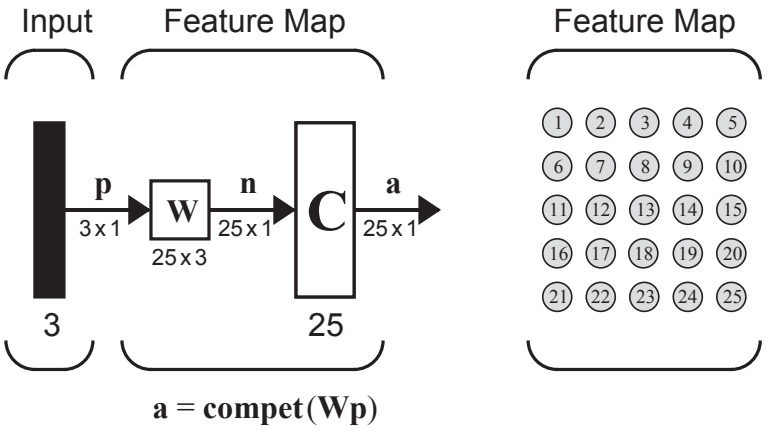
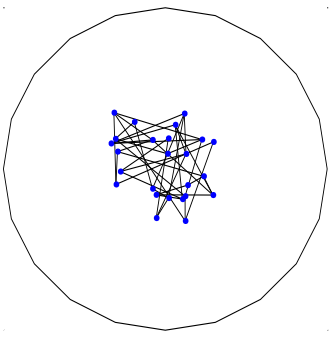
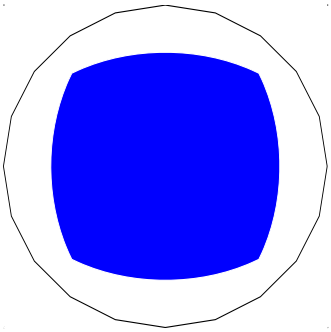



Figure 16.9 Self-Organizing Feature Map



The diagram in the left margin shows the initial weight vectors for the feature map. Each three-element weight vector is represented by a dot on the sphere. (The weights are normalized, therefore they will fall on the surface of a sphere.) Dots of neighboring neurons are connected by lines so you can see how the physical topology of the network is arranged in the input space.



The diagram to the left shows a square region on the surface of the sphere. We will randomly pick vectors in this region and present them to the feature map.

Each time a vector is presented, the neuron with the closest weight vector will win the competition. The winning neuron and its neighbors move their weight vectors closer to the input vector (and therefore to each other). For this example we are using a neighborhood with a radius of 1.

The weight vectors have two tendencies: first, they spread out over the input space as more vectors are presented; second, they move toward the weight vectors of neighboring neurons. These two tendencies work together to rearrange the neurons in the layer so that they evenly classify the input space.

The series of diagrams in Figure 16.10 shows how the weights of the twenty-five neurons spread out over the active input space and organize themselves to match its topology.

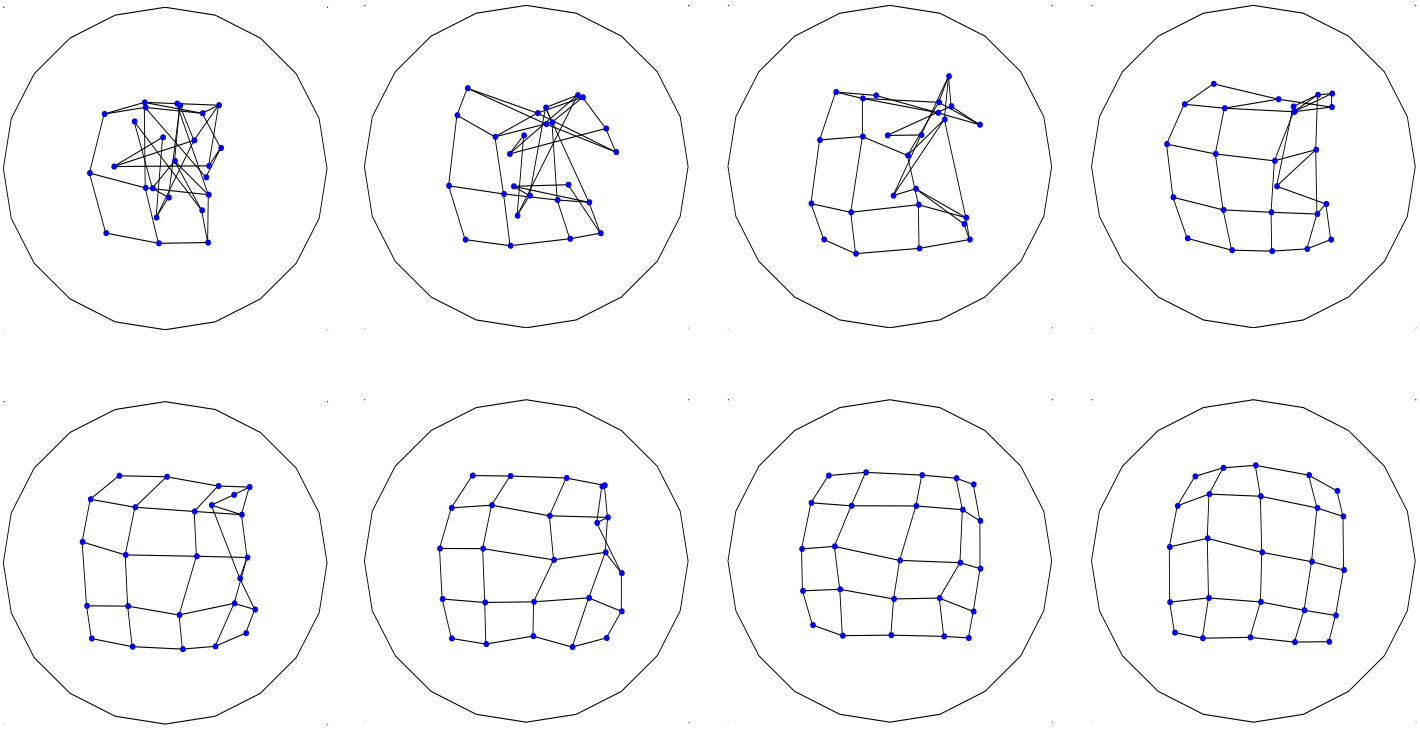


Figure 16.10 Self-Organization, 250 Iterations per Diagram

In this example, the input vectors were generated with equal probability from any point in the input space. Therefore, the neurons classify roughly equal areas of the input space.

Figure 16.11 provides more examples of input regions and the resulting feature maps after self-organization.

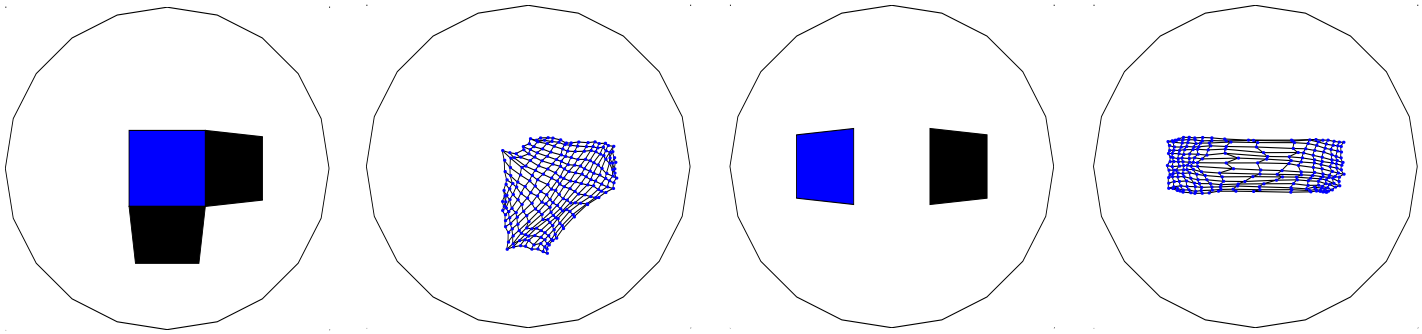


Figure 16.11 Other Examples of Feature Map Training

Occasionally feature maps can fail to properly fit the topology of their input space. This usually occurs when two parts of the net fit the topology of separate parts of the input space, but the net forms a twist between them. An example is given in Figure 16.12.

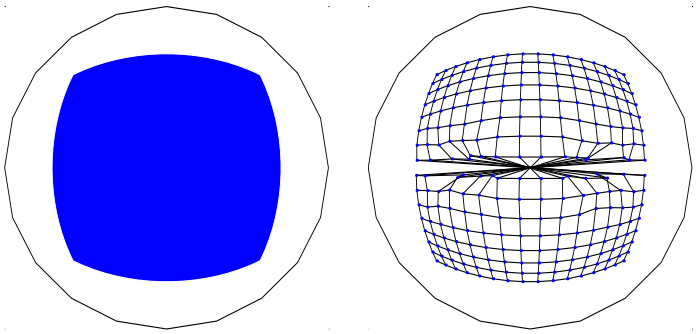


Figure 16.12 Feature Map with a Twist

It is unlikely that this twist will ever be removed, because the two ends of the net have formed stable classifications of different regions.

## Improving Feature Maps

So far, we have described only the most basic algorithm for training feature maps. Now let's consider several techniques that can be used to speed up the self-organizing process and to make it more reliable.

One method to improve the performance of the feature map is to vary the size of the neighborhoods during training. Initially, the neighborhood size, $d$, is set large. As training progresses, $d$ is gradually reduced, until it only includes the winning neuron. This speeds up self-organizing and makes twists in the map very unlikely.

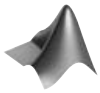
The learning rate can also be varied over time. An initial rate of 1 allows neurons to quickly learn presented vectors. During training, the learning rate is decreased asymptotically toward 0, so that learning becomes stable.

(We discussed the use of this technique for competitive layers earlier in the chapter.)

Another alteration that speeds self-organization is to have the winning neuron use a larger learning rate than the neighboring neurons.

Finally, both competitive layers and feature maps often use an alternative expression for net input. Instead of using the inner product, they can directly compute the distance between the input vector and the prototype vectors. The advantage of using the distance is that input vectors do not need to be normalized. This alternative net input expression is introduced in the next section on LVQ networks.

Other enhancements to the SOFM are described in Chapter 26, including a batch version of the SOFM learning rule. That chapter is a case study of using the SOFM for clustering.



*To experiment with feature maps use the Neural Network Design Demonstrations 1-D Feature Maps* (`nnd14fm1`) *and 2-D Feature Maps* (`nnd14fm2`).

## Learning Vector Quantization

The final network we will introduce in this chapter is the learning vector quantization (LVQ) network, which is shown in Figure 16.13. The LVQ network is a hybrid network. It uses both unsupervised and supervised learning to form classifications.

In the LVQ network, each neuron in the first layer is assigned to a class, with several neurons often assigned to the same class. Each class is then assigned to one neuron in the second layer. The number of neurons in the first layer, $S^1$, will therefore always be at least as large as the number of neurons in the second layer, $S^2$, and will usually be larger.

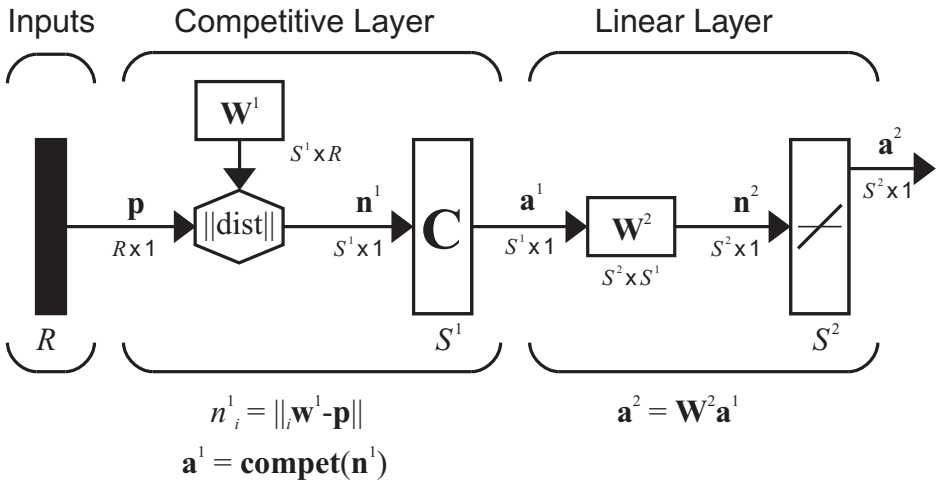



Figure 16.13 LVQ Network

As with the competitive network, each neuron in the first layer of the LVQ network learns a prototype vector, which allows it to classify a region of the input space. However, instead of computing the proximity of the input and weight vectors by using the inner product, we will simulate the LVQ networks by calculating the distance directly. One advantage of calculating the distance directly is that vectors need not be normalized. When the vectors are normalized, the response of the network will be the same, whether the inner product is used or the distance is directly calculated.

The net input of the first layer of the LVQ will be

$$n_i^1 = -\left\| {}_i\mathbf{w}^1 - \mathbf{p} \right\|, \tag{16.25}$$

or, in vector form,

$$\mathbf{n}^1 = -\begin{bmatrix} \left\| {}_1\mathbf{w}^1 - \mathbf{p} \right\| \\ \left\| {}_2\mathbf{w}^1 - \mathbf{p} \right\| \\ \vdots \\ \left\| {}_{S^1}\mathbf{w}^1 - \mathbf{p} \right\| \end{bmatrix}. \tag{16.26}$$

The output of the first layer of the LVQ is

$$\mathbf{a}^1 = \mathbf{compet}(\mathbf{n}^1). \tag{16.27}$$

Therefore the neuron whose weight vector is closest to the input vector will output a 1, and the other neurons will output 0.

Thus far, the LVQ network behaves exactly like the competitive network (at least for normalized vectors). There is a difference in interpretation, however. In the competitive network, the neuron with the nonzero output indicates which class the input vector belongs to. For the LVQ network, the winning neuron indicates a *subclass*, rather than a class. There may be several different neurons (subclasses) that make up each class.

Subclass

The second layer of the LVQ network is used to combine subclasses into a single class. This is done with the $\mathbf{W}^2$ matrix. The columns of $\mathbf{W}^2$ represent subclasses, and the rows represent classes. $\mathbf{W}^2$ has a single 1 in each column, with the other elements set to zero. The row in which the 1 occurs indicates which class the appropriate subclass belongs to.

$$(w_{ki}^2 = 1) \Rightarrow \text{subclass } i \text{ is a part of class } k \tag{16.28}$$

The process of combining subclasses to form a class allows the LVQ network to create complex class boundaries. A standard competitive layer has

the limitation that it can only create decision regions that are convex. The LVQ network overcomes this limitation.

## LVQ Learning

The learning in the LVQ network combines competitive learning with supervision. As with all supervised learning algorithms, it requires a set of examples of proper network behavior:

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}.$$

Each target vector must contain only zeros, except for a single 1. The row in which the 1 appears indicates the class to which the input vector belongs. For example, if we have a problem where we would like to classify a particular three-element vector into the second of four classes, we can express this as

$$\left\{\mathbf{p}_1 = \begin{bmatrix}\sqrt{1/2}\\0\\\sqrt{1/2}\end{bmatrix}, \mathbf{t}_1 = \begin{bmatrix}0\\1\\0\\0\end{bmatrix}\right\}. \tag{16.29}$$

Before learning can occur, each neuron in the first layer is assigned to an output neuron. This generates the matrix $\mathbf{W}^2$. Typically, equal numbers of hidden neurons are connected to each output neuron, so that each class can be made up of the same number of convex regions. All elements of $\mathbf{W}^2$ are set to zero, except for the following:

$$\text{If hidden neuron } i \text{ is to be assigned to class } k, \text{ then set } w^2_{ki} = 1. \tag{16.30}$$

Once $\mathbf{W}^2$ is defined, it will never be altered. The hidden weights $\mathbf{W}^1$ are trained with a variation of the Kohonen rule.

The LVQ learning rule proceeds as follows. At each iteration, an input vector $\mathbf{p}$ is presented to the network, and the distance from $\mathbf{p}$ to each prototype vector is computed. The hidden neurons compete, neuron $i^*$ wins the competition, and the $i^*$ th element of $\mathbf{a}^1$ is set to 1. Next, $\mathbf{a}^1$ is multiplied by $\mathbf{W}^2$ to get the final output $\mathbf{a}^2$, which also has only one nonzero element, $k^*$, indicating that $\mathbf{p}$ is being assigned to class $k^*$.

The Kohonen rule is used to improve the hidden layer of the LVQ network in two ways. First, if $\mathbf{p}$ is classified correctly, then we move the weights ${}_{i^*}\mathbf{w}^1$ of the winning hidden neuron toward $\mathbf{p}$.

$${}_{i^*}\mathbf{w}^1(q) = {}_{i^*}\mathbf{w}^1(q-1) + \alpha(\mathbf{p}(q) - {}_{i^*}\mathbf{w}^1(q-1)), \text{ if } a^2_{k^*} = t_{k^*} = 1 \tag{16.31}$$

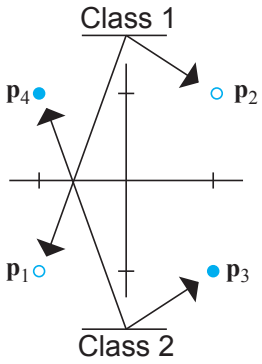
Second, if $\mathbf{p}$ was classified incorrectly, then we know that the wrong hidden neuron won the competition, and therefore we move its weights ${}_{i^*}\mathbf{w}^1$ *away* from $\mathbf{p}$.

$${}_{i^*}\mathbf{w}^1(q) = {}_{i^*}\mathbf{w}^1(q-1) - \alpha(\mathbf{p}(q) - {}_{i^*}\mathbf{w}^1(q-1)), \text{ if } a_{k^*}^2 = 1 \neq t_{k^*} = 0 \qquad (16.32)$$

The result will be that each hidden neuron moves toward vectors that fall into the class for which it forms a subclass and away from vectors that fall into other classes.

Let's take a look at an example of LVQ training. We would like to train an LVQ network to solve the following classification problem:

$$\text{class 1:}\left\{\mathbf{p}_1 = \begin{bmatrix}-1\\-1\end{bmatrix}, \mathbf{p}_2 = \begin{bmatrix}1\\1\end{bmatrix}\right\}, \text{class 2:}\left\{\mathbf{p}_3 = \begin{bmatrix}1\\-1\end{bmatrix}, \mathbf{p}_4 = \begin{bmatrix}-1\\1\end{bmatrix}\right\}, \qquad (16.33)$$




as illustrated by the figure in the left margin. We begin by assigning target vectors to each input:

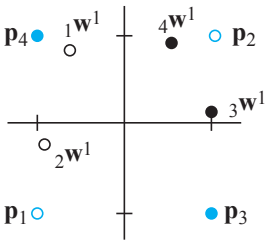
$$\left\{\mathbf{p}_1 = \begin{bmatrix}-1\\-1\end{bmatrix}, \mathbf{t}_1 = \begin{bmatrix}1\\0\end{bmatrix}\right\}, \left\{\mathbf{p}_2 = \begin{bmatrix}1\\1\end{bmatrix}, \mathbf{t}_2 = \begin{bmatrix}1\\0\end{bmatrix}\right\}, \qquad (16.34)$$

$$\left\{\mathbf{p}_3 = \begin{bmatrix}1\\-1\end{bmatrix}, \mathbf{t}_3 = \begin{bmatrix}0\\1\end{bmatrix}\right\}, \left\{\mathbf{p}_4 = \begin{bmatrix}-1\\1\end{bmatrix}, \mathbf{t}_4 = \begin{bmatrix}0\\1\end{bmatrix}\right\}. \qquad (16.35)$$

We now must choose how many subclasses will make up each of the two classes. If we let each class be the union of two subclasses, we will end up with four neurons in the hidden layer. The output layer weight matrix will be

$$\mathbf{W}^2 = \begin{bmatrix}1 & 1 & 0 & 0\\0 & 0 & 1 & 1\end{bmatrix}. \qquad (16.36)$$

$\mathbf{W}^2$ connects hidden neurons 1 and 2 to output neuron 1. It connects hidden neurons 3 and 4 to output neuron 2. Each class will be made up of two convex regions.




The row vectors in $\mathbf{W}^1$ are initially set to random values. They can be seen in the diagram at left. The weights belonging to the two hidden neurons that define class 1 are marked with hollow circles. The weights defining class 2 are marked with solid circles. The values for these weights are

$${}_1\mathbf{w}^1 = \begin{bmatrix}-0.543\\0.840\end{bmatrix}, {}_2\mathbf{w}^1 = \begin{bmatrix}-0.969\\-0.249\end{bmatrix}, {}_3\mathbf{w}^1 = \begin{bmatrix}0.997\\0.094\end{bmatrix}, {}_4\mathbf{w}^1 = \begin{bmatrix}0.456\\0.954\end{bmatrix}. \qquad (16.37)$$

At each iteration of the training process, we present an input vector, find its response, and then adjust the weights. In this case we will begin by presenting $\mathbf{p}_3$.

$$\mathbf{a}^1 = \mathbf{compet}(\mathbf{n}^1) = \mathbf{compet}\left(\begin{bmatrix} -\|{}_1\mathbf{w}^1 - \mathbf{p}_3\| \\ -\|{}_2\mathbf{w}^1 - \mathbf{p}_3\| \\ -\|{}_3\mathbf{w}^1 - \mathbf{p}_3\| \\ -\|{}_4\mathbf{w}^1 - \mathbf{p}_3\| \end{bmatrix}\right) \tag{16.38}$$

$$= \mathbf{compet}\left(\begin{bmatrix} -\|[-0.543\ \ 0.840]^T - [1\ \ -1]^T\| \\ -\|[-0.969\ \ -0.249]^T - [1\ \ -1]^T\| \\ -\|[0.997\ \ 0.094]^T - [1\ \ -1]^T\| \\ -\|[0.456\ \ 0.954]^T - [1\ \ -1]^T\| \end{bmatrix}\right) = \mathbf{compet}\left(\begin{bmatrix} -2.40 \\ -2.11 \\ -1.09 \\ -2.03 \end{bmatrix}\right) = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

The third hidden neuron has the closest weight vector to $\mathbf{p}_3$. In order to determine which class this neuron belongs to, we multiply $\mathbf{a}^1$ by $\mathbf{W}^2$.

$$\mathbf{a}^2 = \mathbf{W}^2\mathbf{a}^1 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{16.39}$$

This output indicates that $\mathbf{p}_3$ is a member of class 2. This is correct, so ${}_3\mathbf{w}^1$ is updated by moving it toward $\mathbf{p}_3$.

$${}_3\mathbf{w}^1(1) = {}_3\mathbf{w}^1(0) + \alpha(\mathbf{p}_3 - {}_3\mathbf{w}^1(0)) \tag{16.40}$$

$$= \begin{bmatrix} 0.997 \\ 0.094 \end{bmatrix} + 0.5\left(\begin{bmatrix} 1 \\ -1 \end{bmatrix} - \begin{bmatrix} 0.997 \\ 0.094 \end{bmatrix}\right) = \begin{bmatrix} 0.998 \\ -0.453 \end{bmatrix}$$

The diagram on the left side of Figure 16.14 shows the weights after ${}_3\mathbf{w}^1$ was updated on the first iteration. The diagram on the right side of Figure 16.14 shows the weights after the algorithm has converged.

The diagram on the right side of Figure 16.14 also indicates how the regions of the input space will be classified. The regions that will be classified as class 1 are shown in gray, and the regions that will be classified as class 2 are shown in blue.

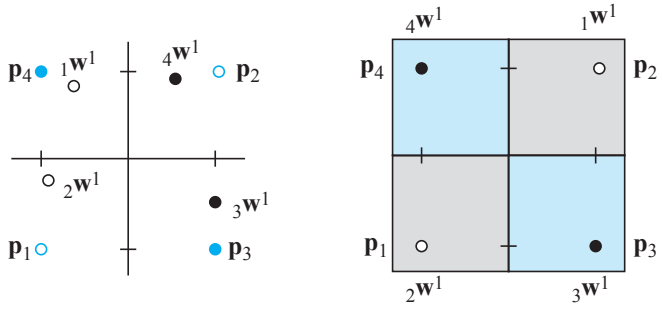



Figure 16.14 After First and Many Iterations

## Improving LVQ Networks (LVQ2)

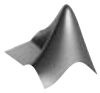
The LVQ network described above works well for many problems, but it does suffer from a couple of limitations. First, as with competitive layers, occasionally a hidden neuron in an LVQ network can have initial weight values that stop it from ever winning the competition. The result is a dead neuron that never does anything useful. This problem is solved with the use of a "conscience" mechanism, a technique discussed earlier for competitive layers, and also presented in Exercise E16.4.

Secondly, depending on how the initial weight vectors are arranged, a neuron's weight vector may have to travel through a region of a class that it doesn't represent, to get to a region that it does represent. Because the weights of such a neuron will be repulsed by vectors in the region it must cross, it may not be able to cross, and so it may never properly classify the region it is being attracted to. This is usually solved by applying the following modification to the Kohonen rule.

If the winning neuron in the hidden layer incorrectly classifies the current input, we move its weight vector away from the input vector, as before. However, we also adjust the weights of the closest neuron to the input vector that does classify it properly. The weights for this second neuron should be moved toward the input vector.

When the network correctly classifies an input vector, the weights of only one neuron are moved toward the input vector. However, if the input vector is incorrectly classified, the weights of two neurons are updated, one weight vector is moved away from the input vector, and the other one is moved toward the input vector. The resulting algorithm is called *LVQ2*.

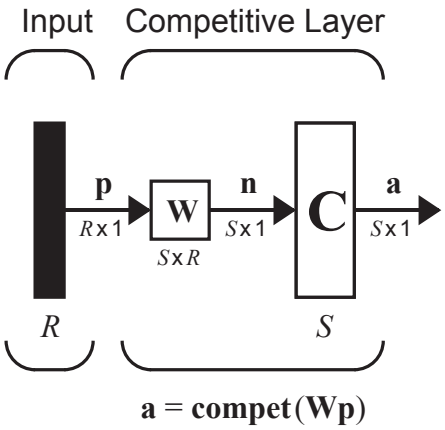
LVQ2



*To experiment with LVQ networks use the Neural Network Design Demonstrations LVQ1 Networks* (`nnd14lv1`) *and LVQ2 Networks* (`nnd14lv2`).

# Summary of Results

## Competitive Layer

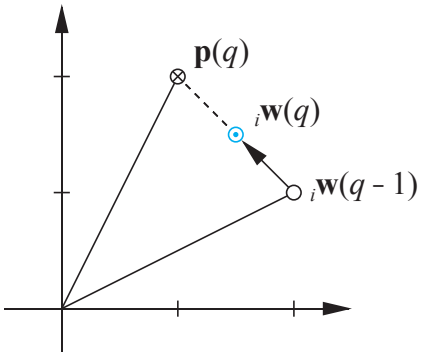



### Competitive Learning with the Kohonen Rule

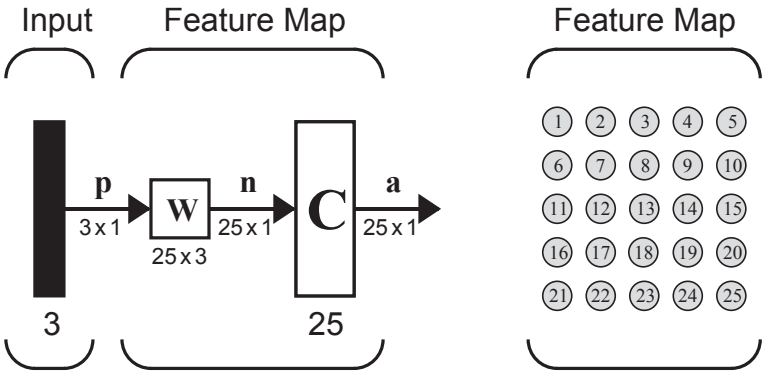
$$_{i^*}\mathbf{w}(q) = {}_{i^*}\mathbf{w}(q-1) + \alpha(\mathbf{p}(q) - {}_{i^*}\mathbf{w}(q-1)) = (1-\alpha){}_{i^*}\mathbf{w}(q-1) + \alpha\mathbf{p}(q)$$

$$_{i^*}\mathbf{w}(q) = {}_{i^*}\mathbf{w}(q-1) \qquad i \neq i^*,$$

where $i^*$ is the winning neuron.



# Self-Organizing Feature Map



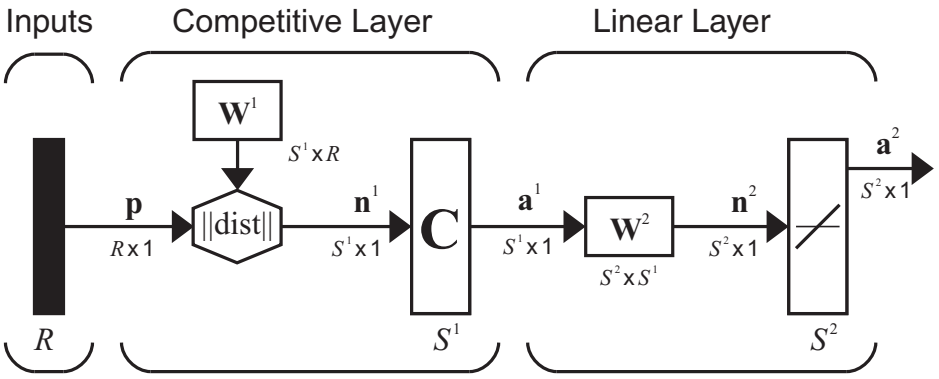

$$\mathbf{a} = \mathbf{compet}(\mathbf{Wp})$$

## Self-Organizing with the Kohonen Rule

$$
\begin{aligned}
{}_i\mathbf{w}(q) &= {}_i\mathbf{w}(q-1) + \alpha(\mathbf{p}(q) - {}_i\mathbf{w}(q-1)) \\
&= (1-\alpha){}_i\mathbf{w}(q-1) + \alpha\mathbf{p}(q)
\end{aligned}
\qquad i \in N_{i^*}(d)
$$

$$N_i(d) = \{j, d_{ij} \le d\}$$

# LVQ Network




$$n^1_i = \|{}_i\mathbf{w}^1 - \mathbf{p}\|$$

$$\mathbf{a}^1 = \mathbf{compet}(\mathbf{n}^1)$$

$$\mathbf{a}^2 = \mathbf{W}^2\mathbf{a}^1$$

$(w^2_{ki} = 1) \Rightarrow$ subclass $i$ is a part of class $k$

## LVQ Network Learning with the Kohonen Rule

$${}_{i^*}\mathbf{w}^1(q) = {}_{i^*}\mathbf{w}^1(q-1) + \alpha(\mathbf{p}(q) - {}_{i^*}\mathbf{w}^1(q-1)), \text{ if } a^2_{k^*} = t_{k^*} = 1$$

$${}_{i^*}\mathbf{w}^1(q) = {}_{i^*}\mathbf{w}^1(q-1) - \alpha(\mathbf{p}(q) - {}_{i^*}\mathbf{w}^1(q-1)), \text{ if } a^2_{k^*} = 1 \neq t_{k^*} = 0$$

# Solved Problems

**P16.1 Figure P16.1 shows several clusters of normalized vectors.**

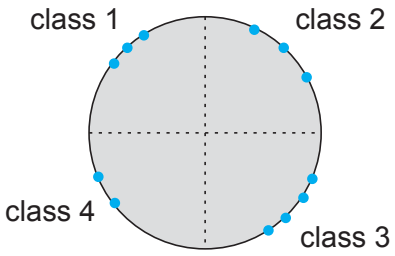



Figure P16.1 Clusters of Input Vectors for Problem P16.1

**Design the weights of the competitive network shown in Figure P16.2, so that it classifies the vectors according to the classes indicated in the diagram and with the minimum number of neurons.**

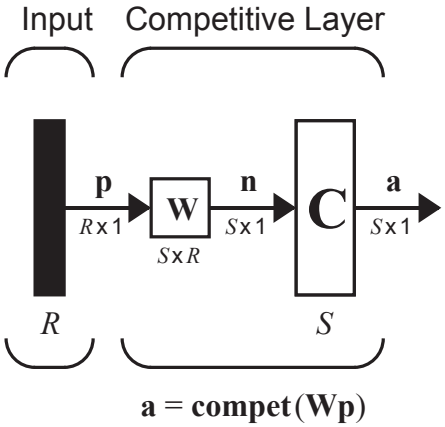



Figure P16.2 Competitive Network for Problem P16.1

**Redraw the diagram showing the weights you chose and the decision boundaries that separate the region of each class.**

Since there are four classes to be defined, the competitive layer will need four neurons. The weights of each neuron act as prototypes for the class that neuron represents. Therefore, for each neuron we will choose a prototype vector that appears to be approximately at the center of a cluster.

Classes 1, 2 and 3 each appear to be roughly centered at a multiple of $45°$. Given this, the following three vectors are normalized (as is required for the competitive layer) and point in the proper directions.

$$ {}_1\mathbf{w} = \begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}, \; {}_2\mathbf{w} = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}, \; {}_3\mathbf{w} = \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix} $$

The center of the fourth cluster appears to be about twice as far from the vertical axis as it is from the horizontal axis. The resulting normalized weight vector is

$$ {}_4\mathbf{w} = \begin{bmatrix} -2/\sqrt{5} \\ -1/\sqrt{5} \end{bmatrix}. $$

The weight matrix $\mathbf{W}$ for the competitive layer is simply the matrix of the transposed prototype vectors:

$$ W = \begin{bmatrix} {}_1\mathbf{w}^T \\ {}_2\mathbf{w}^T \\ {}_3\mathbf{w}^T \\ {}_4\mathbf{w}^T \end{bmatrix} = \begin{bmatrix} -1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \\ -2/\sqrt{5} & -1/\sqrt{5} \end{bmatrix}. $$

We get Figure P16.3 by drawing these weight vectors with arrows and bisecting the circle between each adjacent weight vector to get the class regions.

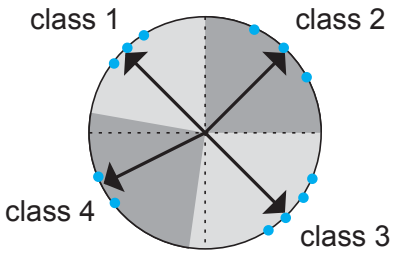



Figure P16.3 Final Classifications for Problem P16.1

**P16.2 Figure P16.4 shows three input vectors and three initial weight vectors for a three-neuron competitive layer. Here are the values of the input vectors:**

$$ \mathbf{p}_1 = \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \mathbf{p}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \mathbf{p}_3 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}. $$

**The initial values of the three weight vectors are**

$$ {}_1\mathbf{w} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, {}_2\mathbf{w} = \begin{bmatrix} -2/\sqrt{5} \\ 1/\sqrt{5} \end{bmatrix}, {}_3\mathbf{w} = \begin{bmatrix} -1/\sqrt{5} \\ 2/\sqrt{5} \end{bmatrix}. $$

**Calculate the resulting weights found after training the competitive layer with the Kohonen rule and a learning rate $\alpha$ of 0.5, on the following series of inputs:**

$$\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3.$$

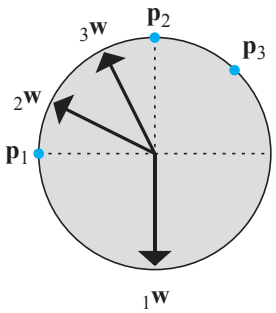



Figure P16.4 Input Vectors and Initial Weights for Problem P16.2

First we combine the weight vectors into the weight matrix $\mathbf{W}$.

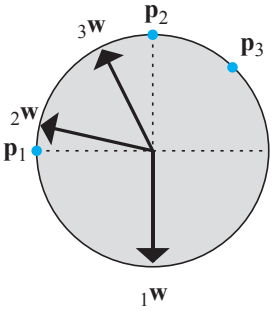
$$\mathbf{W} = \begin{bmatrix} 0 & -1 \\ -2/\sqrt{5} & 1/\sqrt{5} \\ -1/\sqrt{5} & 2/\sqrt{5} \end{bmatrix}$$

Then we present the first vector $\mathbf{p}_1$.

$$\mathbf{a} = \mathbf{compet}(\mathbf{W}\mathbf{p}_1) = \mathbf{compet}\left(\begin{bmatrix} 0 & -1 \\ -2/\sqrt{5} & 1/\sqrt{5} \\ -1/\sqrt{5} & 2/\sqrt{5} \end{bmatrix}\begin{bmatrix} -1 \\ 0 \end{bmatrix}\right) = \mathbf{compet}\left(\begin{bmatrix} 0 \\ 0.894 \\ 0.447 \end{bmatrix}\right) = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

The second neuron responded, since ${}_2\mathbf{w}$ was closest to $\mathbf{p}_1$. Therefore, we will update ${}_2\mathbf{w}$ with the Kohonen rule.

$${}_2\mathbf{w}^{new} = {}_2\mathbf{w}^{old} + \alpha(\mathbf{p}_1 - {}_2\mathbf{w}^{old}) = \begin{bmatrix} -2/\sqrt{5} \\ 1/\sqrt{5} \end{bmatrix} + 0.5\left(\begin{bmatrix} -1 \\ 0 \end{bmatrix} - \begin{bmatrix} -2/\sqrt{5} \\ 1/\sqrt{5} \end{bmatrix}\right) = \begin{bmatrix} -0.947 \\ 0.224 \end{bmatrix}$$




The diagram at left shows that the new ${}_2\mathbf{w}$ moved closer to $\mathbf{p}_1$.

We will now repeat this process for $\mathbf{p}_2$.

$$\mathbf{a} = \mathbf{compet}(\mathbf{W}\mathbf{p}_2) = \mathbf{compet}\left(\begin{bmatrix} 0 & -1 \\ -0.947 & 0.224 \\ -1/\sqrt{5} & 2/\sqrt{5} \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix}\right) = \mathbf{compet}\left(\begin{bmatrix} -1 \\ 0.224 \\ 0.894 \end{bmatrix}\right) = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

The third neuron won, so its weights move closer to $\mathbf{p}_2$.

$${}_3\mathbf{w}^{new} = {}_3\mathbf{w}^{old} + \alpha(\mathbf{p}_2 - {}_3\mathbf{w}^{old}) = \begin{bmatrix} -1/\sqrt{5} \\ 2/\sqrt{5} \end{bmatrix} + 0.5\left(\begin{bmatrix} 0 \\ 1 \end{bmatrix} - \begin{bmatrix} -1/\sqrt{5} \\ 2/\sqrt{5} \end{bmatrix}\right) = \begin{bmatrix} -0.224 \\ 0.947 \end{bmatrix}$$

We now present $\mathbf{p}_3$.

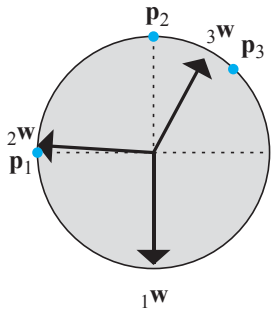
$$\mathbf{a} = \mathbf{compet}(\mathbf{W}\mathbf{p}_3) = \mathbf{compet}\left(\begin{bmatrix} 0 & -1 \\ -0.947 & 0.224 \\ -0.224 & 0.947 \end{bmatrix}\begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}\right)$$

$$= \mathbf{compet}\left(\begin{bmatrix} -0.707 \\ -0.512 \\ 0.512 \end{bmatrix}\right) = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

The third neuron wins again.

$${}_3\mathbf{w}^{new} = {}_3\mathbf{w}^{old} + \alpha(\mathbf{p}_2 - {}_3\mathbf{w}^{old}) = \begin{bmatrix} -0.224 \\ 0.947 \end{bmatrix} + 0.5\left(\begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} - \begin{bmatrix} -0.224 \\ 0.947 \end{bmatrix}\right) = \begin{bmatrix} 0.2417 \\ 0.8272 \end{bmatrix}$$

After presenting $\mathbf{p}_1$ through $\mathbf{p}_3$ again, neuron 2 will again win once and neuron 3 twice. The final weights are




$$\mathbf{W} = \begin{bmatrix} 0 & -1 \\ -0.974 & 0.118 \\ 0.414 & 0.8103 \end{bmatrix}.$$

The final weights are also shown in the diagram at left.

Note that ${}_2\mathbf{w}$ has almost learned $\mathbf{p}_1$, and ${}_3\mathbf{w}$ is directly between $\mathbf{p}_2$ and $\mathbf{p}_3$. The other weight vector, ${}_1\mathbf{w}$, was never updated. The first neuron, which never won the competition, is a dead neuron.

**P16.3 Consider the configuration of input vectors and initial weights shown in Figure P16.5. Train a competitive network to cluster**

**these vectors using the Kohonen rule with learning rate $\alpha = 0.5$. Find graphically the position of the weights after all of the input vectors (in the order shown) have been presented once.**

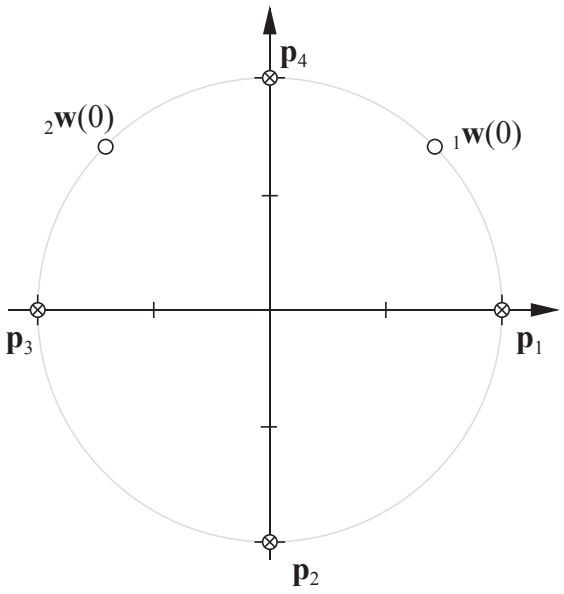



Figure P16.5 Input Vectors and Initial Weights for Problem P16.3

This problem can be solved graphically, without any computations. The results are displayed in Figure P16.6.

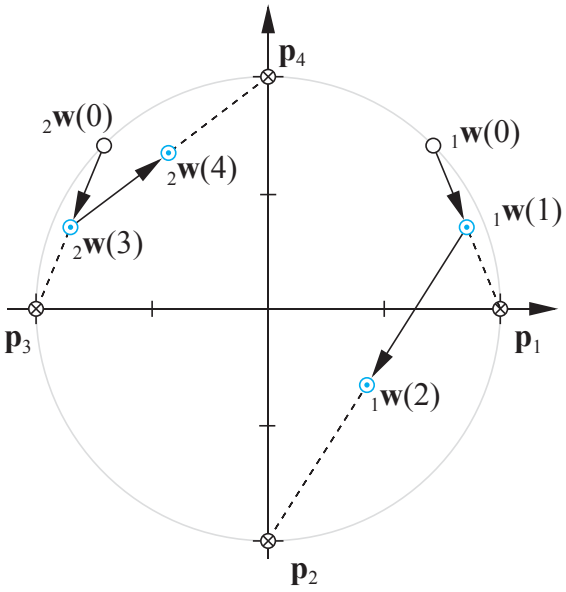



Figure P16.6 Solution for Problem P16.3

The input vector $\mathbf{p}_1$ is presented first. The weight vector $_1\mathbf{w}$ is closest to $\mathbf{p}_1$, therefore neuron 1 wins the competition and $_1\mathbf{w}$ is moved halfway to

$\mathbf{p}_1$, since $\alpha = 0.5$. Next, $\mathbf{p}_2$ is presented, and again neuron 1 wins the competition and ${}_1\mathbf{w}$ is moved halfway to $\mathbf{p}_2$. During these first two iterations, ${}_2\mathbf{w}$ is not changed.

On the third iteration, $\mathbf{p}_3$ is presented. This time ${}_2\mathbf{w}$ wins the competition and is moved halfway to $\mathbf{p}_3$. On the fourth iteration, $\mathbf{p}_4$ is presented, and neuron 2 again wins. The weight vector ${}_2\mathbf{w}$ is moved halfway to $\mathbf{p}_4$.

If we continue to train the network, neuron 1 will classify the input vectors $\mathbf{p}_1$ and $\mathbf{p}_2$, and neuron 2 will classify the input vectors $\mathbf{p}_3$ and $\mathbf{p}_4$. If the input vectors were presented in a different order, would the final classification be different?

**P16.4 So far in this chapter we have only talked about feature maps whose neurons are arranged in two dimensions. The feature map shown in Figure P16.7 contains nine neurons arranged in one dimension.**

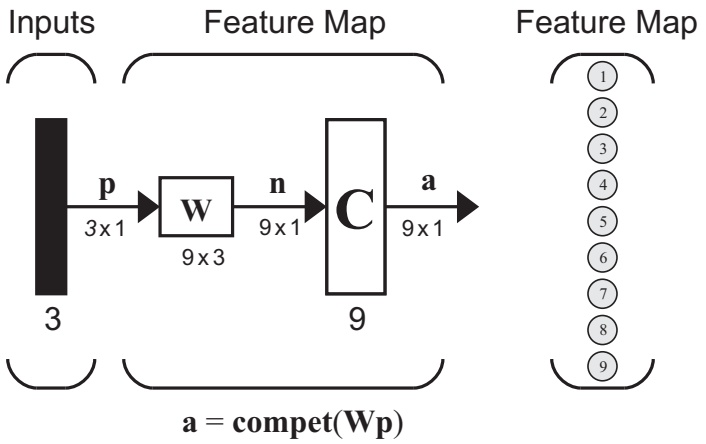



Figure P16.7 Nine-Neuron Feature Map

**Given the following initial weights, draw a diagram of the weight vectors, with lines connecting weight vectors of neighboring neurons.**

$$\mathbf{W} = \begin{bmatrix} 0.41 & 0.45 & 0.41 & 0 & 0 & 0 & -0.41 & -0.45 & -0.41 \\ 0.41 & 0 & -0.41 & 0.45 & 0 & -0.45 & 0.41 & 0 & -0.41 \\ 0.82 & 0.89 & 0.82 & 0.89 & 1 & 0.89 & 0.82 & 0.89 & 0.82 \end{bmatrix}^T$$

**Train the feature map for one iteration, on the vector below, using a learning rate of 0.1 and a neighborhood of radius 1. Redraw the diagram for the new weight matrix.**

$$\mathbf{p} = \begin{bmatrix} 0.67 \\ 0.07 \\ 0.74 \end{bmatrix}$$

The feature map diagram for the initial weights is given in Figure P16.8.

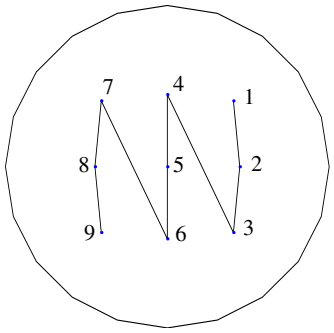



Figure P16.8 Original Feature Map

We start updating the network by presenting $\mathbf{p}$ to the network.

$$\mathbf{a} = \mathbf{compet}(\mathbf{Wp})$$

$$= \mathbf{compet}\left(\begin{bmatrix} 0.41 & 0.45 & 0.41 & 0 & 0 & 0 & -0.41 & -0.45 & -0.41 \\ 0.41 & 0 & -0.41 & 0.45 & 0 & -0.45 & 0.41 & 0 & -0.41 \\ 0.82 & 0.89 & 0.82 & 0.89 & 1 & 0.89 & 0.82 & 0.89 & 0.82 \end{bmatrix}^T \begin{bmatrix} 0.67 \\ 0.07 \\ 0.74 \end{bmatrix}\right)$$

$$= \mathbf{compet}(\begin{bmatrix} 0.91 & 0.96 & 0.85 & 0.70 & 0.74 & 0.63 & 0.36 & 0.36 & 0.3 \end{bmatrix}^T)$$

$$= \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$$

The second neuron won the competition. Looking at the network diagram, we see that the second neuron's neighbors, at a radius of 1, include neurons 1 and 3. We must update each of these neurons' weights with the Kohonen rule.

$$ {}_1\mathbf{w}(1) = {}_1\mathbf{w}(0) + \alpha(\mathbf{p} - {}_1\mathbf{w}(0)) = \begin{bmatrix} 0.41 \\ 0.41 \\ 0.82 \end{bmatrix} + 0.1\left( \begin{bmatrix} 0.67 \\ 0.07 \\ 0.74 \end{bmatrix} - \begin{bmatrix} 0.41 \\ 0.41 \\ 0.82 \end{bmatrix} \right) = \begin{bmatrix} 0.43 \\ 0.37 \\ 0.81 \end{bmatrix} $$

$$ {}_2\mathbf{w}(1) = {}_2\mathbf{w}(0) + \alpha(\mathbf{p} - {}_2\mathbf{w}(0)) = \begin{bmatrix} 0.45 \\ 0 \\ 0.89 \end{bmatrix} + 0.1\left( \begin{bmatrix} 0.67 \\ 0.07 \\ 0.74 \end{bmatrix} - \begin{bmatrix} 0.45 \\ 0 \\ 0.89 \end{bmatrix} \right) = \begin{bmatrix} 0.47 \\ 0.01 \\ 0.88 \end{bmatrix} $$

$$ {}_3\mathbf{w}(1) = {}_3\mathbf{w}(0) + \alpha(\mathbf{p} - {}_3\mathbf{w}(0)) = \begin{bmatrix} 0.41 \\ -0.41 \\ 0.82 \end{bmatrix} + 0.1\left( \begin{bmatrix} 0.67 \\ 0.07 \\ 0.74 \end{bmatrix} - \begin{bmatrix} 0.41 \\ -0.41 \\ 0.82 \end{bmatrix} \right) = \begin{bmatrix} 0.43 \\ -0.36 \\ 0.81 \end{bmatrix} $$

Figure P16.9 shows the feature map after the weights were updated.

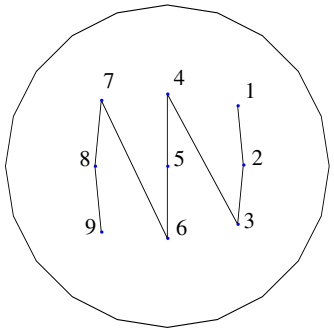



Figure P16.9 Feature Map after Update

**P16.5 Given the LVQ network shown in Figure P16.10 and the weight values shown below, draw the regions of the input space that make up each class.**

$$ \mathbf{W}^1 = \begin{bmatrix} 0 & 0 \\ 1 & -1 \\ 1 & 1 \\ -1 & 1 \\ -1 & -1 \end{bmatrix}, \mathbf{W}^2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix} $$

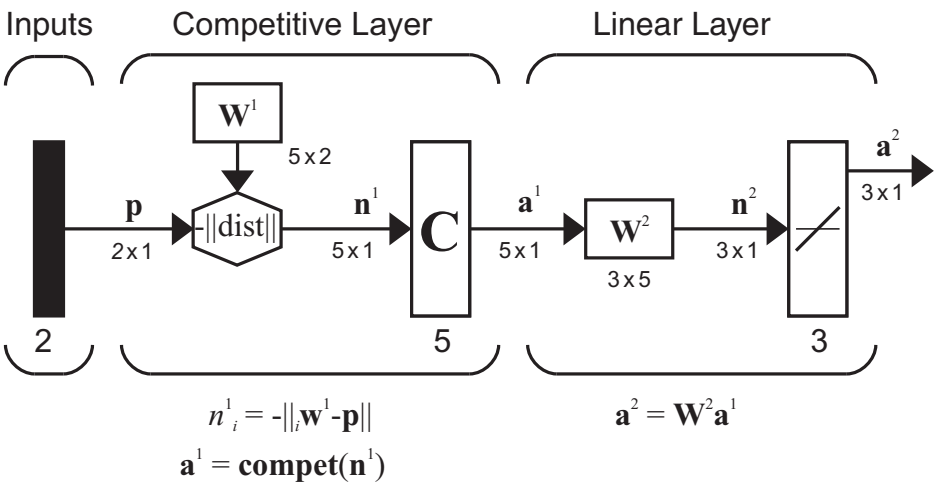



Figure P16.10 LVQ Network for Problem P16.5

We create the diagram shown in Figure P16.11 by marking each vector ${}_i\mathbf{w}$ in $\mathbf{W}^1$ according to the index $k$ of the corresponding nonzero element in the $i$th column of $\mathbf{W}^2$, which indicates the class.

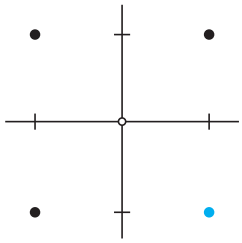


Figure P16.11 Prototype Vectors Marked by Class

The decision boundaries separating each class are found by drawing lines between each pair of prototype vectors, perpendicular to an imaginary line connecting them and equidistant from each vector.

In Figure P16.12, each convex region is colored according to the weight vector it is closest to.

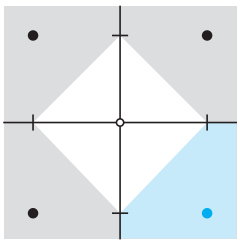


Figure P16.12 Class Regions and Decision Boundaries

**P16.6 Design an LVQ network to solve the classification problem shown in Figure P16.13. The vectors in the diagram are to be classified into one of three classes, according to their color.**

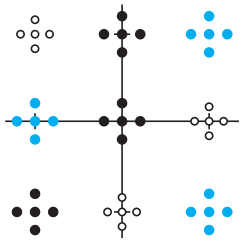


Figure P16.13 Classification Problem

**When the design is complete, draw a diagram showing the region for each class.**

We will begin by noting that since LVQ networks calculate the distance between vectors directly, instead of using the inner product, they can classify vectors that are not normalized, such as those above.

Next we will identify each color with a class:

- Class 1 will include all white dots.
- Class 2 will include all black dots.
- Class 3 will include all blue dots.

Now we can choose the dimensions of the LVQ network. Since there are three classes, the network must have three neurons in its output layer. There are nine subclasses (i.e., clusters). Therefore the hidden layer must have nine neurons. This gives us the network shown in Figure P16.14.

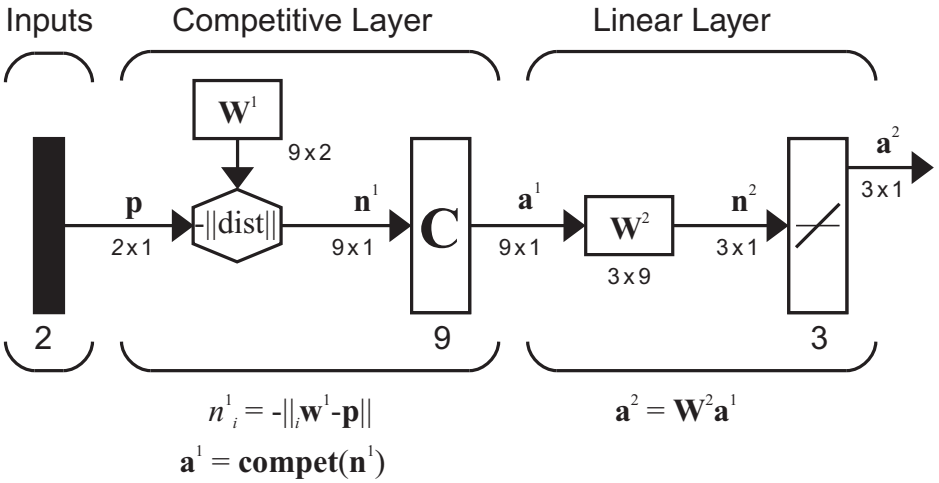



Figure P16.14 LVQ Network for Problem P16.6

We can now design the weight matrix $\mathbf{W}^1$ of the first layer by setting each row equal to a transposed prototype vector for one cluster. Picking prototype vectors at the center of each cluster gives us the following values:

$$\mathbf{W}^1 = \begin{bmatrix} -1 & 0 & 1 & -1 & 0 & 1 & -1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & -1 & -1 & -1 \end{bmatrix}^T .$$

Now each neuron in the first layer will respond to a different cluster.

Next we choose $\mathbf{W}^2$ so that each subclass is connected to the appropriate class. To do this we use the following rule:

$$\text{If subclass } i \text{ is to be assigned to class } k, \text{ then set } w_{ki}^2 = 1 .$$

For example, the first subclass is the top-left cluster in the vector diagram. The vectors in this cluster are white, so they belong in the first class. Therefore we should set $w_{1,1}^2$ to one.

Once we have done this for all nine subclasses we end up with these values:

$$\mathbf{W}^2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} .$$

We can test the network by presenting a vector to it. Here we calculate the output of the first layer for $\mathbf{p} = \begin{bmatrix} 1 & 0 \end{bmatrix}^T$:

$$\mathbf{a}^1 = \mathbf{compet}(\mathbf{n}^1) = \mathbf{compet}\left( \begin{bmatrix} -\sqrt{5} \\ -\sqrt{2} \\ -1 \\ -2 \\ -1 \\ 0 \\ -\sqrt{5} \\ -\sqrt{2} \\ -1 \end{bmatrix} \right) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} .$$

The network says that the vector we presented is in the sixth subclass. Let's see what the second layer says.

$$\mathbf{a}^2 = \mathbf{W}^2\mathbf{a}^1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

The second layer indicates that the vector is in class 1, as indeed it is. The diagram of class regions and decision boundaries is shown in Figure P16.15.

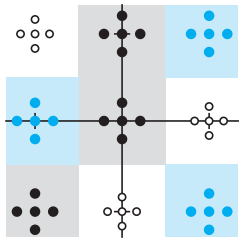


Figure P16.15 Class Regions and Decision Boundaries

**P16.7 Competitive layers and feature maps require that input vectors be normalized. But what if the available data is not normalized?**

**One way to handle such data is simply to normalize it before giving it to the network. This has the disadvantage that the vector magnitude information, which may be important, is lost.**

**Another solution is to replace the inner product expression usually used to calculate net input,**

$$\mathbf{a} = \mathbf{compet}(\mathbf{Wp}),$$

**with a direct calculation of distance,**

$$n_i = -\|_i\mathbf{w} - \mathbf{p}\| \text{ and } \mathbf{a} = \mathbf{compet}(\mathbf{n}),$$

**as is done with the LVQ network. This works and saves the magnitude information.**

**However, a third solution is to append a constant of 1 to each input vector before normalizing it. Now the change in the added element will preserve the vector magnitude information.**

**Normalize the following vectors using this last method:**

$$\mathbf{p}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \mathbf{p}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \mathbf{p}_3 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

First we add an extra element with value 1 to each vector.

$$\mathbf{p'}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \mathbf{p'}_2 = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}, \mathbf{p'}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Then we normalize each vector.

$$\mathbf{p''}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} / \left\| \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right\| = \begin{bmatrix} 1/\sqrt{3} \\ 1/\sqrt{3} \\ 1/\sqrt{3} \end{bmatrix}$$

$$\mathbf{p''}_2 = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} / \left\| \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \right\| = \begin{bmatrix} 0 \\ 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$$

$$\mathbf{p''}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} / \left\| \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right\| = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Now the third element of each vector contains magnitude information, since it is equal to the inverse of the magnitude of the extended vectors.

# Epilogue

In this chapter we have demonstrated how the associative instar learning rule of Chapter 15 can be combined with competitive networks, similar to the Hamming network of Chapter 3, to produce powerful self-organizing networks. By combining competition with the instar rule, each of the prototype vectors that are learned by the network become representative of a particular class of input vector. Thus the competitive networks learn to divide their input space into distinct classes. Each class is represented by one of the prototype vectors (rows of the weight matrix).

Three types of networks, all developed by Tuevo Kohonen, were discussed in this chapter. The first is the standard competitive layer. Its simple operation makes it a practical network for many problems.

The self-organizing feature map is very similar to the competitive layer, but more closely models biological on-center/off-surround networks. The result is a network that not only learns to classify input vectors, but also learns the topology of the input space.

The third network, the LVQ network, uses both unsupervised and supervised learning to recognize clusters. It uses a second layer to combine multiple convex regions into classes that can have any shape. LVQ networks can even be trained to recognize classes made up of multiple unconnected regions.

Chapters 18 and 19 will build on the networks presented in this chapter. For example, Chapter 18 will carry out a more detailed examination of lateral inhibition, on-center/off-surround networks and the biology that inspired them. In Chapter 19 we discuss a modification to the standard competitive network (called adaptive resonance theory), which solves the weight stability problem that we discussed in this chapter.

Chapter 22 presents practical tips for training competitive networks, and Chapter 26 is a case study of using self organizing feature maps on a real-world clustering problem.

# Further Reading

[FrSk91] J. Freeman and D. Skapura, *Neural Networks: Algorithms, Applications, and Programming Techniques*, Reading, MA: Addison-Wesley, 1991.

This text contains code fragments for network algorithms, making the details of the networks clear.

[Koho87] T. Kohonen, *Self-Organization and Associative Memory*, 2nd Ed., Berlin: Springer-Verlag, 1987.

Kohonen introduces the Kohonen rule and several networks that use it. It provides a complete analysis of linear associative models and gives many extensions and examples.

[Hech90] R. Hecht-Nielsen, *Neurocomputing*, Reading, MA: Addison-Wesley, 1990.

This book contains a section on the history and mathematics of competitive learning.

[RuMc86] D. Rumelhart, J. McClelland et al., *Parallel Distributed Processing*, vol. 1, Cambridge, MA: MIT Press, 1986.

Both volumes of this set are classics in neural network literature. The first volume contains a chapter describing competitive layers and how they learn to detect features.

# Exercises

**E16.1** Suppose that the weight matrix for layer 2 of the Hamming network is given by

$$\mathbf{W}^2 = \begin{bmatrix} 1 & -\frac{3}{4} & -\frac{3}{4} \\ -\frac{3}{4} & 1 & -\frac{3}{4} \\ -\frac{3}{4} & -\frac{3}{4} & 1 \end{bmatrix}.$$

This matrix violates Eq. (16.6), since

$$\varepsilon = \frac{3}{4} > \frac{1}{S-1} = \frac{1}{2}.$$

Give an example of an output from Layer 1 for which Layer 2 will fail to operate correctly.

**E16.2** Consider the input vectors and initial weights shown in Figure E16.1.

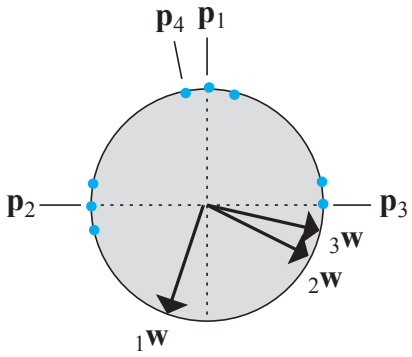



Figure E16.1 Cluster Data Vectors

**i.** Draw the diagram of a competitive network that could classify the data above so that each of the three clusters of vectors would have its own class.

**ii.** Train the network graphically (using the initial weights shown) by presenting the labeled vectors in the following order:

$$\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4.$$

Recall that the competitive transfer function chooses the neuron with the lowest index to win if more than one neuron has the same

net input. The Kohonen rule is introduced graphically in Figure 16.3.

**iii.** Redraw the diagram in Figure E16.1, showing your final weight vectors and the decision boundaries between each region that represents a class.

**E16.3** Train a competitive network using the following input patterns:

$$\mathbf{p}_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \mathbf{p}_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \mathbf{p}_3 = \begin{bmatrix} -1 \\ -1 \end{bmatrix}.$$

**i.** Use the Kohonen learning law with $\alpha = 0.5$, and train for one pass through the input patterns. (Present each input once, in the order given.) Display the results graphically. Assume the initial weight matrix is

$$\mathbf{W} = \begin{bmatrix} \sqrt{2} & 0 \\ 0 & \sqrt{2} \end{bmatrix}.$$

**ii.** After one pass through the input patterns, how are the patterns clustered? (In other words, which patterns are grouped together in the same class?) Would this change if the input patterns were presented in a different order? Explain.

**iii.** Repeat part (i) using $\alpha = 0.25$. How does this change affect the training?

**E16.4** Earlier in this chapter the term "conscience" was used to refer to a technique for avoiding the dead neuron problem plaguing competitive layers and LVQ networks.

Neurons that are too far from input vectors to ever win the competition can be given a chance by using adaptive biases that get more negative each time a neuron wins the competition. The result is that neurons that win very often start to feel "guilty" until other neurons have a chance to win.

Figure E16.2 shows a competitive network with biases. A typical learning rule for the bias $b_i$ of neuron $i$ is

$$b_i^{new} = \begin{cases} 0.9 b_i^{old}, \text{ if } i \neq i^* \\ b_i^{old} - 0.2, \text{ if } i = i^* \end{cases}.$$

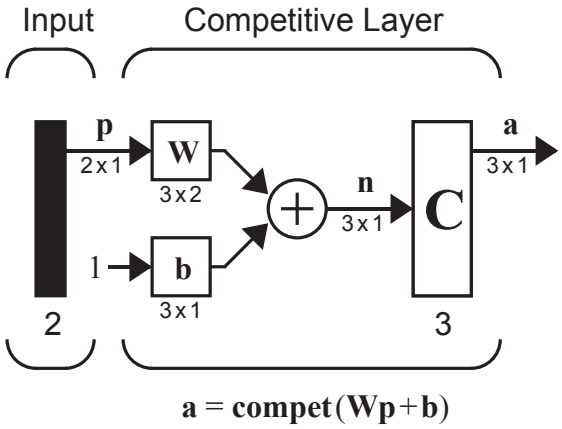



Figure E16.2 Competitive Layer with Biases

**i.** Examine the vectors in Figure E16.3. Is there any order in which the vectors can be presented that will cause ${}_1w$ to win the competition and move closer to one of the vectors? (Note: assume that adaptive biases are *not* being used.)

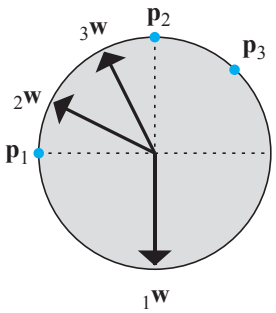



Figure E16.3 Input Vectors and Dead Neuron

**ii.** Given the input vectors and the initial weights and biases defined below, calculate the weights (using the Kohonen rule) and the biases (using the above bias rule). Repeat the sequence shown below until neuron 1 wins the competition.

$$\mathbf{p}_1 = \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \mathbf{p}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \mathbf{p}_3 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$$

$${}_1\mathbf{w} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, {}_2\mathbf{w} = \begin{bmatrix} -2/\sqrt{5} \\ -1/\sqrt{5} \end{bmatrix}, {}_3\mathbf{w} = \begin{bmatrix} -1/\sqrt{5} \\ -2/\sqrt{5} \end{bmatrix}, b_1(0) = b_2(0) = b_3(0) = 0$$

Sequence of input vectors: $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \ldots$

**iii.** How many presentations occur before ${}_1\mathbf{w}$ wins the competition?

**E16.5** The net input expression for LVQ networks calculates the distance between the input and each weight vector directly, instead of using the inner product. The result is that the LVQ network does not require normalized input vectors. This technique can also be used to allow a competitive layer to classify nonnormalized vectors. Such a network is shown in Figure E16.4.

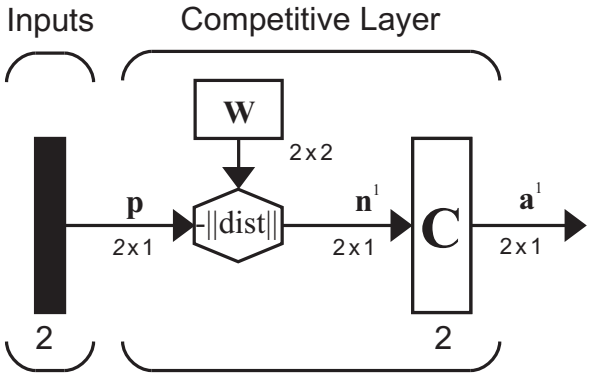



Figure E16.4 Competitive Layer with Alternate Net Input Expression

Use this technique to train a two-neuron competitive layer on the (nonnormalized) vectors below, using a learning rate, $\alpha$, of 0.5.

$$\mathbf{p}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \mathbf{p}_2 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, \mathbf{p}_3 = \begin{bmatrix} -2 \\ -2 \end{bmatrix}$$

Present the vectors in the following order:

$$\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_1.$$

Here are the initial weights of the network:

$${}_1\mathbf{w} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, {}_2\mathbf{w} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

**E16.6** Repeat E16.5 for the following inputs and initial weights. Show the movements of the weights graphically for each step. If the network is trained for a large number of iterations, how will the three vectors be clustered in the final configuration?

$$\mathbf{p}_1 = \begin{bmatrix} 2 \\ 0 \end{bmatrix}, \mathbf{p}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \mathbf{p}_3 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

$${}_1\mathbf{w} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, {}_2\mathbf{w} = \begin{bmatrix} -1 \\ 0 \end{bmatrix}.$$

**E16.7** We have a competitive learning problem, where the input vectors are

$$\mathbf{p}_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \mathbf{p}_2 = \begin{bmatrix} 0 \\ 2 \end{bmatrix}, \mathbf{p}_3 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \mathbf{p}_4 = \begin{bmatrix} 2 \\ 2 \end{bmatrix},$$

and the initial weight matrix is

$$\mathbf{W} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}.$$

**i.** Use the Kohonen learning law to train a competitive network using a learning rate of $\alpha = 0.5$. (Present each vector once, in the order shown.) Use the modified competitive network of Figure E16.4, which uses negative distance, instead of inner product.

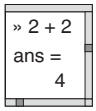
**ii.** Display the results of part i graphically, as in Figure 16.3. (Show all four iterations.)

**iii.** Where will the weights eventually converge (approximately)? Explain. Sketch the approximate final decision boundaries.

**E16.8** Show that the modified competitive network of Figure E16.4, which computes distance directly, will produce the same results as the standard competitive network, which uses the inner product, when the input vectors are normalized.

**E16.9** We would like a classifier that divides the interval of the input space defined below into five classes.

$$0 \le p_1 \le 1$$




**i.** Use MATLAB to randomly generate 100 values in the interval shown above with a uniform distribution.

**ii.** Square each number so that the distribution is no longer uniform.

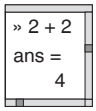
**iii.** Write a MATLAB M-file to implement a competitive layer. Use the M-file to train a five-neuron competitive layer on the squared values

until its weights are fairly stable.

**iv.** How are the weight values of the competitive layer distributed? Is there some relationship between how the weights are distributed and how the squared input values are distributed?

**E16.10** We would like a classifier that divides the square region defined below into sixteen classes of roughly equal size.

$$0 \le p_1 \le 1\ ,\ 2 \le p_2 \le 3$$




**i.** Use MATLAB to randomly generate 200 vectors in the region shown above.

**ii.** Write a MATLAB M-file to implement a competitive layer with Kohonen learning. Calculate the net input by finding the distance between the input and weight vectors directly, as is done by the LVQ network, so the vectors do not need to be normalized. Use the M-file to train a competitive layer to classify the 200 vectors. Try different learning rates and compare performance.

**iii.** Write a MATLAB M-file to implement a four-neuron by four-neuron (two-dimensional) feature map. Use the feature map to classify the same vectors. Use different learning rates and neighborhood sizes, then compare performance.

**E16.11** We want to train the following 1-D feature map (which uses distance instead of inner product to compute the net input):

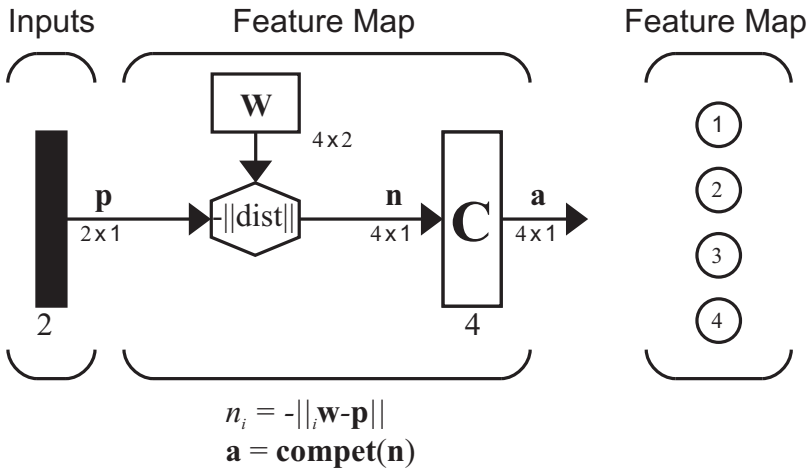



Figure E16.5 1-D Feature Map for Exercise E16.11

The initial weight matrix is $\mathbf{W}(0) = \begin{bmatrix} 2 & -1 & -1 & 1 \\ 2 & 1 & -2 & 0 \end{bmatrix}^T$.

**i.** Plot the initial weight vectors as dots, and connect the neighboring weight vectors as lines (as in Figure 16.10, except that this is a 1-D feature map).

**ii.** The following input vector is applied to the network. Perform one iteration of the feature map learning rule. (You can do this graphically.) Use a neighborhood size of 1 and a learning rate of $\alpha = 0.5$.

$$\mathbf{p}_1 = \begin{bmatrix} -2 & 0 \end{bmatrix}^T$$

**iii.** Plot the new weight vectors as dots, and connect the neighboring weight vectors as lines.

**E16.12** Consider the following feature map, where distance is used instead of inner product to compute the net input.

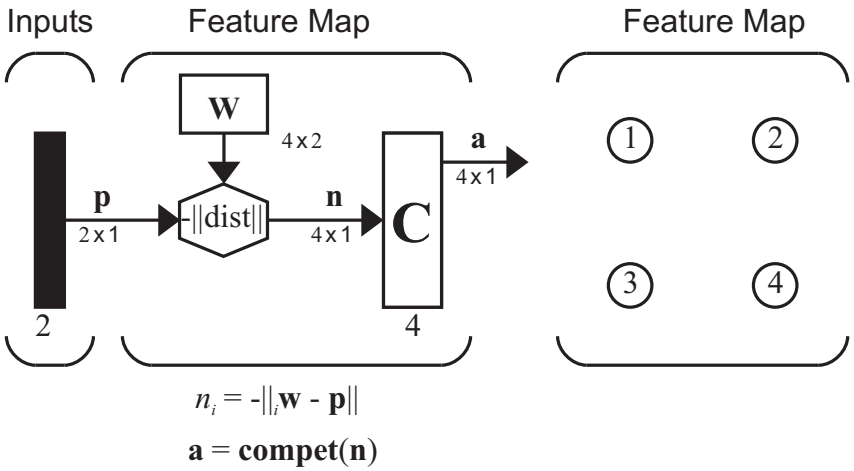



Figure E16.6 2-D Feature Map for Exercise E16.12

The initial weight matrix is

$$\mathbf{W} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}^T$$

**i.** Plot the initial weights, and show their topological connections, as in Figure 16.10.

**ii.** Apply the input $\mathbf{p} = \begin{bmatrix} -1 & 1 \end{bmatrix}^T$, and perform one iteration of the feature map learning rule, with learning rate of $\alpha = 0.5$, and neighborhood radius of 1.

**iii.** Plot the weights after the first iteration, and show their topological connections.

**E16.13** An LVQ network has the following weights:

$$\mathbf{W}^1 = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix}, \mathbf{W}^2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

**i.** How many classes does this LVQ network have? How many subclasses?

**ii.** Draw a diagram showing the first-layer weight vectors and the decision boundaries that separate the input space into subclasses.

**iii.** Label each subclass region to indicate which class it belongs to.

**E16.14** We would like an LVQ network that classifies the following vectors according to the classes indicated:

$$\text{class 1: } \left\{ \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}, \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} \right\}, \text{ class 2: } \left\{ \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} \right\}, \text{ class 3: } \left\{ \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix} \right\}.$$

**i.** How many neurons are required in each layer of the LVQ network?

**ii.** Define the weights for the first layer.

**iii.** Define the weights for the second layer.

**iv.** Test your network for at least one vector from each class.

**E16.15** We would like an LVQ network that classifies the following vectors according to the classes indicated:

$$\text{class 1: } \left\{ \mathbf{p}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \mathbf{p}_2 = \begin{bmatrix} 0 \\ 2 \end{bmatrix} \right\}, \text{ class 2: } \left\{ \mathbf{p}_3 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \mathbf{p}_4 = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \right\}$$

**i.** Could this classification problem be solved by a perceptron? Explain your answer.

**ii.** How many neurons must be in each layer of an LVQ network that can classify the above data, given that each class is made up of two convex-shaped subclasses?

**iii.** Define the second-layer weights for such a network.

**iv.** Initialize the first-layer weights of the network to all zeros and calculate the changes made to the weights by the Kohonen rule (with a learning rate $\alpha$ of 0.5) for the following series of vectors:

$$\mathbf{p}_4, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_1, \mathbf{p}_2.$$

**v.** Draw a diagram showing the input vectors, the final weight vectors and the decision boundaries between the two classes.

**E16.16** An LVQ network has the following weights and training data.

$$\mathbf{W}^1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}, \mathbf{W}^2 = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

$$\left\{\mathbf{p}_1 = \begin{bmatrix} -2 \\ 2 \end{bmatrix}, \mathbf{t}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}\right\}, \left\{\mathbf{p}_2 = \begin{bmatrix} 2 \\ 0 \end{bmatrix}, \mathbf{t}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}\right\}, \left\{\mathbf{p}_3 = \begin{bmatrix} 2 \\ -2 \end{bmatrix}, \mathbf{t}_3 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}\right\},$$

$$\left\{\mathbf{p}_4 = \begin{bmatrix} -2 \\ 0 \end{bmatrix}, \mathbf{t}_4 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}\right\}$$

**i.** Plot the training data input vectors and weight vectors (as in Figure 16.14).

**ii.** Perform four iterations of the LVQ learning rule, with learning rate $\alpha = 0.5$, as you present the following sequence of input vectors: $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$, $\mathbf{p}_4$ (one iteration for each input). Do this graphically, on a separate diagram from part i.

**iii.** After completing the iterations in part ii, on a new diagram, sketch the regions of the input space that make up each subclass and each class. Label each region to indicate which class it belongs to.

**E16.17** An LVQ network has the following weights:

$$\mathbf{W}^1 = \begin{bmatrix} 0 & 1 & -1 & 0 & 0 & -1 & -1 \\ 0 & 0 & 0 & 1 & -1 & -1 & 1 \end{bmatrix}^T, \mathbf{W}^2 = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

**i.** How many classes does this LVQ network have? How many subclasses?

**ii.** Draw a diagram showing the first-layer weight vectors and the decision boundaries that separate the input space into subclasses.

**iii.** Label each subclass region to indicate which class it belongs to.

**iv.** Suppose that an input $\mathbf{p} = \begin{bmatrix} 1 & 0.5 \end{bmatrix}^T$ from Class 1 is presented to the network. Perform one iteration of the LVQ algorithm, with $\alpha = 0.5$.

**E16.18** An LVQ network has the following weights:

$$\mathbf{W}^1 = \begin{bmatrix} 0 & 0 & 2 & 1 & 1 & -1 \\ 0 & 2 & 2 & 1 & -1 & -1 \end{bmatrix}^T, \quad \mathbf{W}^2 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

**i.** How many classes does this LVQ network have? How many subclasses?

**ii.** Draw a diagram showing the first-layer weight vectors and the decision boundaries that separate the input space into subclasses.

**iii.** Label each subclass region to indicate which class it belongs to.

**iv.** Perform one iteration of the LVQ algorithm, with the following input/target pair: $\mathbf{p} = \begin{bmatrix} -1 & -2 \end{bmatrix}^T$, $\mathbf{t} = \begin{bmatrix} 1 & 0 \end{bmatrix}^T$. Use learning rate $\alpha = 0.5$.