7 Supervised Hebbian Learning

Objectives

The Hebb rule was one of the first neural network learning laws. It was proposed by Donald Hebb in 1949 as a possible mechanism for synaptic modification in the brain and since then has been used to train artificial neural networks.

In this chapter we will use the linear algebra concepts of the previous two chapters to explain why Hebbian learning works. We will also show how the Hebb rule can be used to train neural networks for pattern recognition.

Theory and Examples

Donald O. Hebb was born in Chester, Nova Scotia, just after the turn of the century. He originally planned to become a novelist, and obtained a degree in English from Dalhousie University in Halifax in 1925. Since every first-rate novelist needs to have a good understanding of human nature, he began to study Freud after graduation and became interested in psychology. He then pursued a master's degree in psychology at McGill University, where he wrote a thesis on Pavlovian conditioning. He received his Ph.D. from Harvard in 1936, where his dissertation investigated the effects of early experience on the vision of rats. Later he joined the Montreal Neurological Institute, where he studied the extent of intellectual changes in brain surgery patients. In 1942 he moved to the Yerkes Laboratories of Primate Biology in Florida, where he studied chimpanzee behavior.

In 1949 Hebb summarized his two decades of research in *The Organization of Behavior* [Hebb49]. The main premise of this book was that behavior could be explained by the action of neurons. This was in marked contrast to the behaviorist school of psychology (with proponents such as B. F. Skinner), which emphasized the correlation between stimulus and response and discouraged the use of any physiological hypotheses. It was a confrontation between a top-down philosophy and a bottom-up philosophy. Hebb stated his approach: "The method then calls for learning as much as one can about what the parts of the brain do (primarily the physiologist's field), and relating the behavior as far as possible to this knowledge (primarily for the psychologist); then seeing what further information is to be had about how the total brain works, from the discrepancy between (1) actual behavior and (2) the behavior that would be predicted from adding up what is known about the action of the various parts."

The most famous idea contained in *The Organization of Behavior* was the postulate that came to be known as Hebbian learning:

Hebb's Postulate

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

This postulate suggested a physical mechanism for learning at the cellular level. Although Hebb never claimed to have firm physiological evidence for his theory, subsequent research has shown that some cells do exhibit Hebbian learning. Hebb's theories continue to influence current research in neuroscience.

As with most historic ideas, Hebb's postulate was not completely new, as he himself emphasized. It had been foreshadowed by several others, including Freud. Consider, for example, the following principle of association stated by psychologist and philosopher William James in 1890: "When two brain processes are active together or in immediate succession, one of them, on reoccurring tends to propagate its excitement into the other."

Linear Associator

Linear Associator

Hebb's learning law can be used in combination with a variety of neural network architectures. We will use a very simple architecture for our initial presentation of Hebbian learning. In this way we can concentrate on the learning law rather than the architecture. The network we will use is the *linear associator*, which is shown in Figure 7.1. (This network was introduced independently by James Anderson [Ande72] and Teuvo Kohonen [Koho72].)

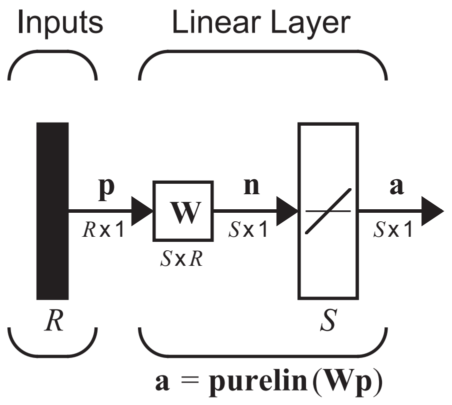


Figure 7.1 Linear Associator

The output vector $\mathbf{a}$ is determined from the input vector $\mathbf{p}$ according to:

$$\mathbf{a} = \mathbf{W}\mathbf{p}, \tag{7.1}$$

or

$$a_i = \sum_{j=1}^{R} w_{ij} p_j. \tag{7.2}$$

Associative Memory

The linear associator is an example of a type of neural network called an *associative memory*. The task of an associative memory is to learn Q pairs of prototype input/output vectors:

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \dots, \{\mathbf{p}_Q, \mathbf{t}_Q\}. \tag{7.3}$$

In other words, if the network receives an input $\mathbf{p} = \mathbf{p}_q$ then it should produce an output $\mathbf{a} = \mathbf{t}_q$, for $q = 1, 2, \dots, Q$. In addition, if the input is changed slightly (i.e., $\mathbf{p} = \mathbf{p}_q + \delta$) then the output should only be changed slightly (i.e., $\mathbf{a} = \mathbf{t}_q + \varepsilon$).

The Hebb Rule

How can we interpret Hebb's postulate mathematically, so that we can use it to train the weight matrix of the linear associator? First, let's rephrase the postulate: If two neurons on either side of a synapse are activated simultaneously, the strength of the synapse will increase. Notice from Eq. (7.2) that the connection (synapse) between input p_j and output a_i is the weight w_{ij}. Therefore Hebb's postulate would imply that if a positive p_j produces a positive a_i then w_{ij} should increase. This suggests that one mathematical interpretation of the postulate could be

The Hebb Rule

$$w_{ij}^{new} = w_{ij}^{old} + \alpha\, f_i(a_{iq}) g_j(p_{jq}) , \qquad (7.4)$$

where p_{jq} is the jth element of the qth input vector $\mathbf{p}_q$; a_{iq} is the ith element of the network output when the qth input vector is presented to the network; and α is a positive constant, called the learning rate. This equation says that the change in the weight w_{ij} is proportional to a product of functions of the activities on either side of the synapse. For this chapter we will simplify Eq. (7.4) to the following form

$$w_{ij}^{new} = w_{ij}^{old} + \alpha a_{iq} p_{jq} . \qquad (7.5)$$

Note that this expression actually extends Hebb's postulate beyond its strict interpretation. The change in the weight is proportional to a product of the activity on either side of the synapse. Therefore, not only do we increase the weight when both p_j and a_i are positive, but we also increase the weight when they are both negative. In addition, this implementation of the Hebb rule will decrease the weight whenever p_j and a_i have opposite sign.

The Hebb rule defined in Eq. (7.5) is an *unsupervised* learning rule. It does not require any information concerning the target output. In this chapter we are interested in using the Hebb rule for supervised learning, in which the target output is known for each input vector. (We will revisit the unsupervised Hebb rule in Chapter 13.) For the *supervised* Hebb rule we substitute the target output for the actual output. In this way, we are telling the algorithm what the network *should* do, rather than what it is currently doing. The resulting equation is

$$w_{ij}^{new} = w_{ij}^{old} + t_{iq} p_{jq} , \qquad (7.6)$$

where t_{iq} is the ith element of the qth target vector $\mathbf{t}_q$. (We have set the learning rate α to one, for simplicity.)

Notice that Eq. (7.6) can be written in vector notation:

$$\mathbf{W}^{new} = \mathbf{W}^{old} + \mathbf{t}_q \mathbf{p}_q^T . \qquad (7.7)$$

If we assume that the weight matrix is initialized to zero and then each of the Q input/output pairs are applied once to Eq. (7.7), we can write

$$\mathbf{W} = \mathbf{t}_1\mathbf{p}_1^T + \mathbf{t}_2\mathbf{p}_2^T + \cdots + \mathbf{t}_Q\mathbf{p}_Q^T = \sum_{q=1}^{Q} \mathbf{t}_q\mathbf{p}_q^T . \quad (7.8)$$

This can be represented in matrix form:

$$\mathbf{W} = \begin{bmatrix} \mathbf{t}_1 & \mathbf{t}_2 & \dots & \mathbf{t}_Q \end{bmatrix} \begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \\ \vdots \\ \mathbf{p}_Q^T \end{bmatrix} = \mathbf{T}\mathbf{P}^T , \quad (7.9)$$

where

$$\mathbf{T} = \begin{bmatrix} \mathbf{t}_1 & \mathbf{t}_2 & \dots & \mathbf{t}_Q \end{bmatrix}, \quad \mathbf{P} = \begin{bmatrix} \mathbf{p}_1 & \mathbf{p}_2 & \dots & \mathbf{p}_Q \end{bmatrix} . \quad (7.10)$$

Performance Analysis

Let's analyze the performance of Hebbian learning for the linear associator. First consider the case where the $\mathbf{p}_q$ vectors are orthonormal (orthogonal and unit length). If $\mathbf{p}_k$ is input to the network, then the network output can be computed

$$\mathbf{a} = \mathbf{W}\mathbf{p}_k = \left(\sum_{q=1}^{Q} \mathbf{t}_q\mathbf{p}_q^T \right) \mathbf{p}_k = \sum_{q=1}^{Q} \mathbf{t}_q (\mathbf{p}_q^T\mathbf{p}_k) . \quad (7.11)$$

Since the $\mathbf{p}_q$ are orthonormal,

$$\begin{aligned} (\mathbf{p}_q^T\mathbf{p}_k) &= 1 \qquad q = k \\ &= 0 \qquad q \neq k . \end{aligned} \quad (7.12)$$

Therefore Eq. (7.11) can be rewritten

$$\mathbf{a} = \mathbf{W}\mathbf{p}_k = \mathbf{t}_k . \quad (7.13)$$

The output of the network is equal to the target output. This shows that, if the input prototype vectors are orthonormal, the Hebb rule will produce the correct output for each input.

But what about non-orthogonal prototype vectors? Let's assume that each $\mathbf{p}_q$ vector is unit length, but that they are not orthogonal. Then Eq. (7.11) becomes

$$\mathbf{a} = \mathbf{W}\mathbf{p}_k = \mathbf{t}_k + \overset{\text{Error}}{\boxed{\sum_{q \neq k} \mathbf{t}_q (\mathbf{p}_q^T \mathbf{p}_k)}} . \qquad (7.14)$$

Because the vectors are not orthogonal, the network will not produce the correct output. The magnitude of the error will depend on the amount of correlation between the prototype input patterns.

As an example, suppose that the prototype input/output vectors are

$$\left\{ \mathbf{p}_1 = \begin{bmatrix} 0.5 \\ -0.5 \\ 0.5 \\ -0.5 \end{bmatrix}, \mathbf{t}_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \right\} \qquad \left\{ \mathbf{p}_2 = \begin{bmatrix} 0.5 \\ 0.5 \\ -0.5 \\ -0.5 \end{bmatrix}, \mathbf{t}_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\} . \qquad (7.15)$$

(Check that the two input vectors are orthonormal.)

The weight matrix would be

$$\mathbf{W} = \mathbf{T}\mathbf{P}^T = \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 0.5 & -0.5 & 0.5 & -0.5 \\ 0.5 & 0.5 & -0.5 & -0.5 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \end{bmatrix} . \qquad (7.16)$$

If we test this weight matrix on the two prototype inputs we find

$$\mathbf{W}\mathbf{p}_1 = \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} 0.5 \\ -0.5 \\ 0.5 \\ -0.5 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix} , \qquad (7.17)$$

and

$$\mathbf{W}\mathbf{p}_2 = \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} 0.5 \\ 0.5 \\ -0.5 \\ -0.5 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} . \qquad (7.18)$$

Success!! The outputs of the network are equal to the targets.

Now let's revisit the *apple* and *orange* recognition problem described in Chapter 3. Recall that the prototype inputs were

$$\mathbf{p}_1 = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} (orange) \qquad \mathbf{p}_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} (apple) . \tag{7.19}$$

(Note that they are not orthogonal.) If we normalize these inputs and choose as desired outputs -1 and 1, we obtain

$$\left\{ \mathbf{p}_1 = \begin{bmatrix} 0.5774 \\ -0.5774 \\ -0.5774 \end{bmatrix}, \mathbf{t}_1 = \begin{bmatrix} -1 \end{bmatrix} \right\} \qquad \left\{ \mathbf{p}_2 = \begin{bmatrix} 0.5774 \\ 0.5774 \\ -0.5774 \end{bmatrix}, \mathbf{t}_2 = \begin{bmatrix} 1 \end{bmatrix} \right\} . \tag{7.20}$$

Our weight matrix becomes

$$\mathbf{W} = \mathbf{T}\mathbf{P}^T = \begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} 0.5774 & -0.5774 & -0.5774 \\ 0.5774 & 0.5774 & -0.5774 \end{bmatrix} = \begin{bmatrix} 0 & 1.1548 & 0 \end{bmatrix} . \tag{7.21}$$

So, if we use our two prototype patterns,

$$\mathbf{W}\mathbf{p}_1 = \begin{bmatrix} 0 & 1.1548 & 0 \end{bmatrix} \begin{bmatrix} 0.5774 \\ -0.5774 \\ -0.5774 \end{bmatrix} = \begin{bmatrix} -0.6668 \end{bmatrix} , \tag{7.22}$$

$$\mathbf{W}\mathbf{p}_2 = \begin{bmatrix} 0 & 1.1548 & 0 \end{bmatrix} \begin{bmatrix} 0.5774 \\ 0.5774 \\ -0.5774 \end{bmatrix} = \begin{bmatrix} 0.6668 \end{bmatrix} . \tag{7.23}$$

The outputs are close, but do not quite match the target outputs.

Pseudoinverse Rule

When the prototype input patterns are not orthogonal, the Hebb rule produces some errors. There are several procedures that can be used to reduce these errors. In this section we will discuss one of those procedures, the pseudoinverse rule.

Recall that the task of the linear associator was to produce an output of $\mathbf{t}_q$ for an input of $\mathbf{p}_q$. In other words,

$$\mathbf{W}\mathbf{p}_q = \mathbf{t}_q \qquad q = 1, 2, \ldots , Q . \tag{7.24}$$

If it is not possible to choose a weight matrix so that these equations are exactly satisfied, then we want them to be approximately satisfied. One approach would be to choose the weight matrix to minimize the following performance index:

$$F(\mathbf{W}) = \sum_{q=1}^{Q} \|\mathbf{t}_q - \mathbf{W}\mathbf{p}_q\|^2 . \tag{7.25}$$

If the prototype input vectors $\mathbf{p}_q$ are orthonormal and we use the Hebb rule to find **W**, then $F(\mathbf{W})$ will be zero. When the input vectors are not orthogonal and we use the Hebb rule, then $F(\mathbf{W})$ will be not be zero, and it is not clear that $F(\mathbf{W})$ will be minimized. It turns out that the weight matrix that will minimize $F(\mathbf{W})$ is obtained by using the pseudoinverse matrix, which we will define next.

First, let's rewrite Eq. (7.24) in matrix form:

$$\mathbf{W}\mathbf{P} = \mathbf{T} , \tag{7.26}$$

where

$$\mathbf{T} = \begin{bmatrix} \mathbf{t}_1 & \mathbf{t}_2 & \dots & \mathbf{t}_Q \end{bmatrix}, \mathbf{P} = \begin{bmatrix} \mathbf{p}_1 & \mathbf{p}_2 & \dots & \mathbf{p}_Q \end{bmatrix} . \tag{7.27}$$

Then Eq. (7.25) can be written

$$F(\mathbf{W}) = \|\mathbf{T} - \mathbf{W}\mathbf{P}\|^2 = \|\mathbf{E}\|^2 , \tag{7.28}$$

where

$$\mathbf{E} = \mathbf{T} - \mathbf{W}\mathbf{P} , \tag{7.29}$$

and

$$\|\mathbf{E}\|^2 = \sum_i \sum_j e_{ij}^2 . \tag{7.30}$$

Note that $F(\mathbf{W})$ can be made zero if we can solve Eq. (7.26). If the **P** matrix has an inverse, the solution is

$$\mathbf{W} = \mathbf{T}\mathbf{P}^{-1} . \tag{7.31}$$

However, this is rarely possible. Normally the $\mathbf{p}_q$ vectors (the columns of **P**) will be independent, but R (the dimension of $\mathbf{p}_q$) will be larger than Q (the number of $\mathbf{p}_q$ vectors). Therefore, **P** will not be a square matrix, and no exact inverse will exist.

Pseudoinverse Rule

It has been shown [Albe72] that the weight matrix that minimizes Eq. (7.25) is given by the *pseudoinverse rule*:

$$\mathbf{W} = \mathbf{T}\mathbf{P}^{+}, \tag{7.32}$$

where $\mathbf{P}^{+}$ is the Moore-Penrose pseudoinverse. The pseudoinverse of a real matrix $\mathbf{P}$ is the unique matrix that satisfies

$$\begin{aligned} \mathbf{P}\mathbf{P}^{+}\mathbf{P} &= \mathbf{P}, \\ \mathbf{P}^{+}\mathbf{P}\mathbf{P}^{+} &= \mathbf{P}^{+}, \\ \mathbf{P}^{+}\mathbf{P} &= (\mathbf{P}^{+}\mathbf{P})^{T}, \\ \mathbf{P}\mathbf{P}^{+} &= (\mathbf{P}\mathbf{P}^{+})^{T}. \end{aligned} \tag{7.33}$$

When the number, R, of rows of $\mathbf{P}$ is greater than the number of columns, Q, of $\mathbf{P}$, and the columns of $\mathbf{P}$ are independent, then the pseudoinverse can be computed by

$$\mathbf{P}^{+} = (\mathbf{P}^{T}\mathbf{P})^{-1}\mathbf{P}^{T}. \tag{7.34}$$

To test the pseudoinverse rule (Eq. (7.32)), consider again the apple and orange recognition problem. Recall that the input/output prototype vectors are

$$\left\{\mathbf{p}_1 = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix}, \mathbf{t}_1 = \begin{bmatrix} -1 \end{bmatrix}\right\} \qquad \left\{\mathbf{p}_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}, \mathbf{t}_2 = \begin{bmatrix} 1 \end{bmatrix}\right\}. \tag{7.35}$$

(Note that we do not need to normalize the input vectors when using the pseudoinverse rule.)

The weight matrix is calculated from Eq. (7.32):

$$\mathbf{W} = \mathbf{T}\mathbf{P}^{+} = \begin{bmatrix} -1 & 1 \end{bmatrix} \left(\begin{bmatrix} 1 & 1 \\ -1 & 1 \\ -1 & -1 \end{bmatrix} \right)^{+}, \tag{7.36}$$

where the pseudoinverse is computed from Eq. (7.34):

$$\mathbf{P}^{+} = (\mathbf{P}^{T}\mathbf{P})^{-1}\mathbf{P}^{T} = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}^{-1} \begin{bmatrix} 1 & -1 & -1 \\ 1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} 0.25 & -0.5 & -0.25 \\ 0.25 & 0.5 & -0.25 \end{bmatrix}. \tag{7.37}$$

This produces the following weight matrix:

$$\mathbf{W} = \mathbf{T}\mathbf{P}^{+} = \begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} 0.25 & -0.5 & -0.25 \\ 0.25 & 0.5 & -0.25 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}. \tag{7.38}$$

Let's try this matrix on our two prototype patterns.

$$\mathbf{W}\mathbf{p}_1 = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} = \begin{bmatrix} -1 \end{bmatrix} \tag{7.39}$$

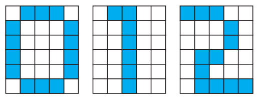
$$\mathbf{W}\mathbf{p}_2 = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \end{bmatrix} \tag{7.40}$$

The network outputs exactly match the desired outputs. Compare this result with the performance of the Hebb rule. As you can see from Eq. (7.22) and Eq. (7.23), the Hebbian outputs are only close, while the pseudoinverse rule produces exact results.

Application

Now let's see how we might use the Hebb rule on a practical, although greatly oversimplified, pattern recognition problem. For this problem we will use a special type of associative memory — the autoassociative memory. In an *autoassociative memory* the desired output vector is equal to the input vector (i.e., $\mathbf{t}_q = \mathbf{p}_q$). We will use an autoassociative memory to store a set of patterns and then to recall these patterns, even when corrupted patterns are provided as input.

Autoassociative Memory


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

The patterns we want to store are shown to the left. (Since we are designing an autoassociative memory, these patterns represent the input vectors and the targets.) They represent the digits {0, 1, 2} displayed in a 6X5 grid. We need to convert these digits to vectors, which will become the prototype patterns for our network. Each white square will be represented by a "-1", and each dark square will be represented by a "1". Then, to create the input vectors, we will scan each 6X5 grid one column at a time. For example, the first prototype pattern will be

$$\mathbf{p}_1 = \begin{bmatrix} -1 & 1 & 1 & 1 & 1 & -1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & -1 & \ldots & 1 & -1 \end{bmatrix}^T. \tag{7.41}$$

The vector $\mathbf{p}_1$ corresponds to the digit "0", $\mathbf{p}_2$ to the digit "1", and $\mathbf{p}_3$ to the digit "2". Using the Hebb rule, the weight matrix is computed

$$\mathbf{W} = \mathbf{p}_1\mathbf{p}_1^T + \mathbf{p}_2\mathbf{p}_2^T + \mathbf{p}_3\mathbf{p}_3^T . \tag{7.42}$$

(Note that $\mathbf{p}_q$ replaces $\mathbf{t}_q$ in Eq. (7.8), since this is autoassociative memory.)

Because there are only two allowable values for the elements of the prototype vectors, we will modify the linear associator so that its output elements can only take on values of "-1" or "1". We can do this by replacing the linear transfer function with a symmetrical hard limit transfer function. The resulting network is displayed in Figure 7.2.

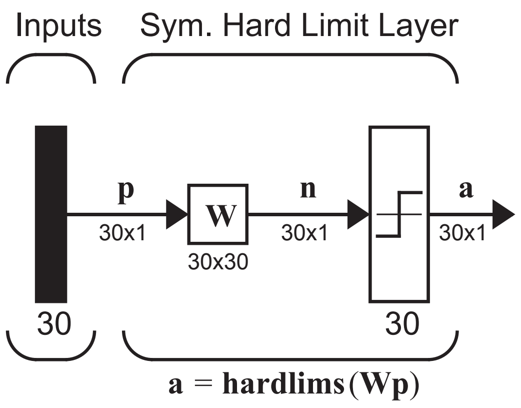


Figure 7.2 Autoassociative Network for Digit Recognition

Now let's investigate the operation of this network. We will provide the network with corrupted versions of the prototype patterns and then check the network output. In the first test, which is shown in Figure 7.3, the network is presented with a prototype pattern in which the lower half of the pattern is occluded. In each case the correct pattern is produced by the network.

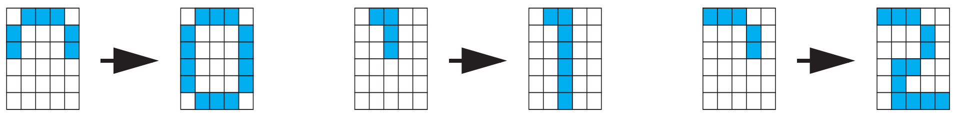


Figure 7.3 Recovery of 50% Occluded Patterns

In the next test we remove even more of the prototype patterns. Figure 7.4 illustrates the result of removing the lower two-thirds of each pattern. In this case only the digit "1" is recovered correctly. The other two patterns produce results that do not correspond to any of the prototype patterns. This is a common problem in associative memories. We would like to design networks so that the number of such spurious patterns would be minimized. We will come back to this topic again in Chapter 18, when we discuss recurrent associative memories.

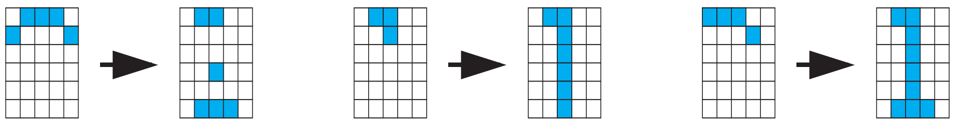


Figure 7.4 Recovery of 67% Occluded Patterns

In our final test we will present the autoassociative network with noisy versions of the prototype pattern. To create the noisy patterns we will randomly change seven elements of each pattern. The results are shown in Figure 7.5. For these examples all of the patterns were correctly recovered.

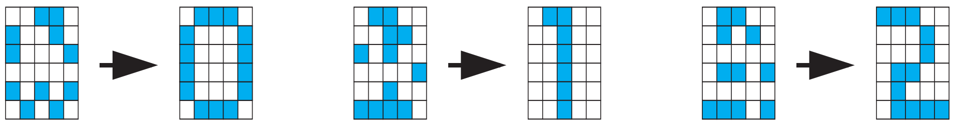


Figure 7.5 Recovery of Noisy Patterns

To experiment with this type of pattern recognition problem, use the Neural Network Design Demonstration Supervised Hebb (`nnd7sh`).

Variations of Hebbian Learning

There have been a number of variations on the basic Hebb rule. In fact, many of the learning laws that will be discussed in the remainder of this text have some relationship to the Hebb rule.

One of the problems of the Hebb rule is that it can lead to weight matrices having very large elements if there are many prototype patterns in the training set. Consider again the basic rule:

$$\mathbf{W}^{new} = \mathbf{W}^{old} + \mathbf{t}_q \mathbf{p}_q^T . \tag{7.43}$$

A positive parameter α, called the learning rate, can be used to limit the amount of increase in the weight matrix elements, if the learning rate is less than one, as in:

$$\mathbf{W}^{new} = \mathbf{W}^{old} + \alpha \mathbf{t}_q \mathbf{p}_q^T . \tag{7.44}$$

We can also add a decay term, so that the learning rule behaves like a smoothing filter, remembering the most recent inputs more clearly:

$$\mathbf{W}^{new} = \mathbf{W}^{old} + \alpha \mathbf{t}_q \mathbf{p}_q^T - \gamma \mathbf{W}^{old} = (1-\gamma)\mathbf{W}^{old} + \alpha \mathbf{t}_q \mathbf{p}_q^T , \tag{7.45}$$

where γ is a positive constant less than one. As γ approaches zero, the learning law becomes the standard rule. As γ approaches one, the learning law quickly forgets old inputs and remembers only the most recent patterns. This keeps the weight matrix from growing without bound.

The idea of filtering the weight changes and of having an adjustable learning rate are important ones, and we will discuss them again in Chapters 10, 12, 15, 16, 18 and 19.

If we modify Eq. (7.44) by replacing the desired output with the difference between the desired output and the actual output, we get another important learning rule:

$$\mathbf{W}^{new} = \mathbf{W}^{old} + \alpha(\mathbf{t}_q - \mathbf{a}_q)\mathbf{p}_q^T . \tag{7.46}$$

This is sometimes known as the delta rule, since it uses the difference between desired and actual output. It is also known as the Widrow-Hoff algorithm, after the researchers who introduced it. The delta rule adjusts the weights so as to minimize the mean square error (see Chapter 10). For this reason it will produce the same results as the pseudoinverse rule, which minimizes the sum of squares of errors (see Eq. (7.25)). The advantage of the delta rule is that it can update the weights after each new input pattern is presented, whereas the pseudoinverse rule computes the weights in one step, after all of the input/target pairs are known. This sequential updating allows the delta rule to adapt to a changing environment. The delta rule will be discussed in detail in Chapter 10.

The basic Hebb rule will be discussed again, in a different context, in Chapter 13. In the present chapter we have used a supervised form of the Hebb rule. We have assumed that the desired outputs of the network, $\mathbf{t}_q$, are known, and can be used in the learning rule. In the unsupervised Hebb rule, which is discussed in Chapter 13, the *actual* network output is used instead of the *desired* network output, as in:

$$\mathbf{W}^{new} = \mathbf{W}^{old} + \alpha\mathbf{a}_q\mathbf{p}_q^T , \tag{7.47}$$

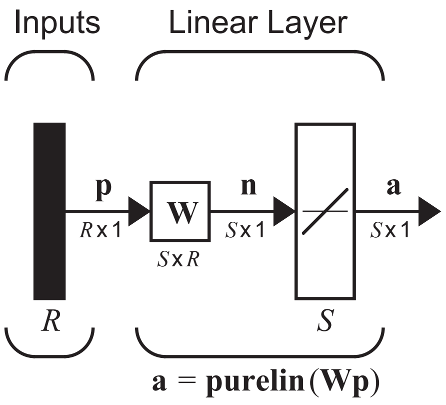
where $\mathbf{a}_q$ is the output of the network when $\mathbf{p}_q$ is given as the input (see also Eq. (7.5)). This unsupervised form of the Hebb rule, which does not require knowledge of the desired output, is actually a more direct interpretation of Hebb's postulate than is the supervised form discussed in this chapter.

Summary of Results

Hebb's Postulate

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

Linear Associator



The Hebb Rule

$$w_{ij}^{new} = w_{ij}^{old} + t_{qi}p_{qj}$$

$$\mathbf{W} = \mathbf{t}_1\mathbf{p}_1^T + \mathbf{t}_2\mathbf{p}_2^T + \cdots + \mathbf{t}_Q\mathbf{p}_Q^T$$

$$\mathbf{W} = \begin{bmatrix} \mathbf{t}_1 & \mathbf{t}_2 & \ldots & \mathbf{t}_Q \end{bmatrix} \begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \\ \vdots \\ \mathbf{p}_Q^T \end{bmatrix} = \mathbf{T}\mathbf{P}^T$$

Pseudoinverse Rule

$$\mathbf{W} = \mathbf{T}\mathbf{P}^+$$

When the number, R, of rows of $\mathbf{P}$ is greater than the number of columns, Q, of $\mathbf{P}$ and the columns of $\mathbf{P}$ are independent, then the pseudoinverse can be computed by

$$\mathbf{P}^{+} = (\mathbf{P}^{T}\mathbf{P})^{-1}\mathbf{P}^{T}.$$

Variations of Hebbian Learning

Filtered Learning

(See Chapter 14)

$$\mathbf{W}^{new} = (1-\gamma)\mathbf{W}^{old} + \alpha\mathbf{t}_q\mathbf{p}_q^T$$

Delta Rule

(See Chapter 10)

$$\mathbf{W}^{new} = \mathbf{W}^{old} + \alpha(\mathbf{t}_q - \mathbf{a}_q)\mathbf{p}_q^T$$

Unsupervised Hebb

(See Chapter 13)

$$\mathbf{W}^{new} = \mathbf{W}^{old} + \alpha\mathbf{a}_q\mathbf{p}_q^T$$

Solved Problems

P7.1 Consider the linear associator shown in Figure P7.1.

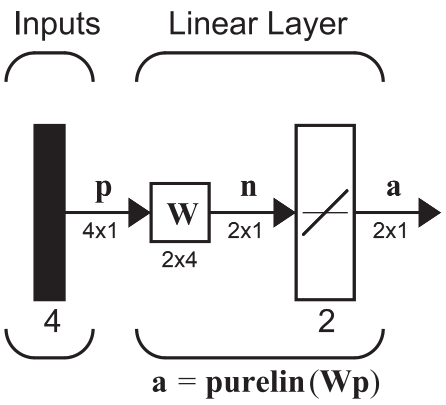


Figure P7.1 Single-Neuron Perceptron

Let the input/output prototype vectors be

$$\left\{\mathbf{p}_1 = \begin{bmatrix}1\\-1\\1\\-1\end{bmatrix}, \mathbf{t}_1 = \begin{bmatrix}1\\-1\end{bmatrix}\right\} \qquad \left\{\mathbf{p}_2 = \begin{bmatrix}1\\1\\-1\\-1\end{bmatrix}, \mathbf{t}_2 = \begin{bmatrix}1\\1\end{bmatrix}\right\}.$$

i. Use the Hebb rule to find the appropriate weight matrix for this linear associator.

ii. Repeat part (i) using the pseudoinverse rule.

iii. Apply the input $\mathbf{p}_1$ to the linear associator using the weight matrix of part (i), then using the weight matrix of part (ii).

i. The first step is to create the **P** and **T** matrices of Eq. (7.10):

$$\mathbf{P} = \begin{bmatrix}1 & 1\\-1 & 1\\1 & -1\\-1 & -1\end{bmatrix}, \qquad \mathbf{T} = \begin{bmatrix}1 & 1\\-1 & 1\end{bmatrix}.$$

Then the weight matrix can be computed using Eq. (7.9):

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

ii. For the pseudoinverse rule we use Eq. (7.32):

$$\mathbf{W} = \mathbf{T}\mathbf{P}^+.$$

Since the number of rows of $\mathbf{P}$, four, is greater than the number of columns of $\mathbf{P}$, two, and the columns of $\mathbf{P}$ are independent, then the pseudoinverse can be computed by Eq. (7.34):

$$\mathbf{P}^+ = (\mathbf{P}^T\mathbf{P})^{-1}\mathbf{P}^T.$$

$$\mathbf{P}^+ = \left(\begin{bmatrix} 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ -1 & 1 \\ 1 & -1 \\ -1 & -1 \end{bmatrix} \right)^{-1} \begin{bmatrix} 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \end{bmatrix} = \left(\begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix} \right)^{-1} \begin{bmatrix} 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{4} & 0 \\ 0 & \frac{1}{4} \end{bmatrix} \begin{bmatrix} 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} \frac{1}{4} & -\frac{1}{4} & \frac{1}{4} & -\frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} \end{bmatrix}$$

The weight matrix can now be computed:

$$\mathbf{W}^p = \mathbf{T}\mathbf{P}^+ = \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{4} & -\frac{1}{4} & \frac{1}{4} & -\frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & 0 & 0 & -\frac{1}{2} \\ 0 & \frac{1}{2} & -\frac{1}{2} & 0 \end{bmatrix}.$$

iii. We now test the two weight matrices.

$$\mathbf{W}^h\mathbf{p}_1 = \begin{bmatrix} 2 & 0 & 0 & -2 \\ 0 & 2 & -2 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 4 \\ -4 \end{bmatrix} \neq \mathbf{t}_1$$

$$\mathbf{W}^p\mathbf{p}_1 = \begin{bmatrix} \frac{1}{2} & 0 & 0 & -\frac{1}{2} \\ 0 & \frac{1}{2} & -\frac{1}{2} & 0 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \mathbf{t}_1$$

Why didn't the Hebb rule produce the correct results? Well, consider again Eq. (7.11). Since $\mathbf{p}_1$ and $\mathbf{p}_2$ are orthogonal (check that they are) this equation can be written

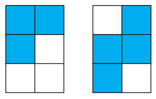
$$\mathbf{W}^h\mathbf{p}_1 = \mathbf{t}_1(\mathbf{p}_1^T\mathbf{p}_1),$$

but the $\mathbf{p}_1$ vector is not normalized, so $(\mathbf{p}_1^T\mathbf{p}_1) \neq 1$. Therefore the output of the network will not be $\mathbf{t}_1$.

The pseudoinverse rule, on the other hand, is guaranteed to minimize

$$\sum_{q=1}^{2} \left\| \mathbf{t}_q - \mathbf{W}\mathbf{p}_q \right\|^2,$$

which in this case can be made equal to zero.

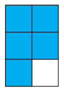

$\mathbf{p}_1$ $\mathbf{p}_2$

P7.2 Consider the prototype patterns shown to the left.

i. Are these patterns orthogonal?

ii. Design an autoassociator for these patterns. Use the Hebb rule.

iii. What response does the network give to the test input pattern, $\mathbf{p}_t$, shown to the left?


$\mathbf{p}_t$

i. The first thing we need to do is to convert the patterns into vectors. Let's assign any solid square the value 1 and any open square the value -1. Then to convert from the two-dimensional pattern to a vector we will scan the pattern column by column. (We could use rows if we wished.) The two prototype vectors then become:

$$\mathbf{p}_1 = \begin{bmatrix} 1 & 1 & -1 & 1 & -1 & -1 \end{bmatrix}^T \qquad \mathbf{p}_2 = \begin{bmatrix} -1 & 1 & 1 & 1 & 1 & -1 \end{bmatrix}^T.$$

To test orthogonality we take the inner product of $\mathbf{p}_1$ and $\mathbf{p}_2$:

$$\mathbf{p}_1^T\mathbf{p}_2 = \begin{bmatrix} 1 & 1 & -1 & 1 & -1 & -1 \end{bmatrix}\begin{bmatrix} -1 \\ 1 \\ 1 \\ 1 \\ 1 \\ -1 \end{bmatrix} = 0 \cdot$$

Therefore they are orthogonal. (Although they are not normalized since

$$\mathbf{p}_1^T\mathbf{p}_1 = \mathbf{p}_2^T\mathbf{p}_2 = 6\,.)$$

ii. We will use an autoassociator like the one in Figure 7.2, except that the number of inputs and outputs to the network will be six. To find the weight matrix we use the Hebb rule:

$$\mathbf{W} = \mathbf{T}\mathbf{P}^T,$$

where

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

Therefore the weight matrix is

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

iii. To apply the test pattern to the network we convert it to a vector:

$$\mathbf{p}_t = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & -1 \end{bmatrix}^T.$$

The network response is then

$$\mathbf{a} = \mathbf{hardlims}(\mathbf{W}\mathbf{p}_t) = \mathbf{hardlims}\left(\begin{bmatrix} 2 & 0 & -2 & 0 & -2 & 0 \\ 0 & 2 & 0 & 2 & 0 & -2 \\ -2 & 0 & 2 & 0 & 2 & 0 \\ 0 & 2 & 0 & 2 & 0 & -2 \\ -2 & 0 & 2 & 0 & 2 & 0 \\ 0 & -2 & 0 & -2 & 0 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ -1 \end{bmatrix}\right)$$

$$\mathbf{a} = \mathbf{hardlims}\left(\begin{bmatrix} -2 \\ 6 \\ 2 \\ 6 \\ 2 \\ -6 \end{bmatrix}\right) = \begin{bmatrix} -1 \\ 1 \\ 1 \\ 1 \\ 1 \\ -1 \end{bmatrix} = \mathbf{p}_2 .$$

Is this a satisfactory response? How would we want the network to respond to this input pattern? The network should produce the prototype pattern that is closest to the input pattern. In this case the test input pattern, $\mathbf{p}_t$, has a Hamming distance of 1 from $\mathbf{p}_2$, and a distance of 2 from $\mathbf{p}_1$. Therefore the network did produce the correct response. (See Chapter 3 for a discussion of Hamming distance.)

Note that in this example the prototype vectors were not normalized. This did not cause the same problem with network performance that we saw in Problem P7.1, because of the *hardlims* nonlinearity. It forces the network output to be either 1 or -1. In fact, most of the interesting and useful properties of neural networks are due to the effects of nonlinearities.

P7.3 **Consider an autoassociation problem in which there are three prototype patterns (shown below as $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$). Design autoassociative networks to recognize these patterns, using both the Hebb rule and the pseudoinverse rule. Check their performance on the test pattern $\mathbf{p}_t$ shown below.**

$$\mathbf{p}_1 = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \qquad \mathbf{p}_2 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \\ 1 \\ -1 \\ 1 \end{bmatrix} \qquad \mathbf{p}_3 = \begin{bmatrix} -1 \\ 1 \\ -1 \\ 1 \\ 1 \\ -1 \\ 1 \end{bmatrix} \qquad \mathbf{p}_t = \begin{bmatrix} -1 \\ 1 \\ -1 \\ -1 \\ 1 \\ -1 \\ 1 \end{bmatrix}$$

```
» 2 + 2
ans =
     4
```

This problem is a little tedious to work out by hand, so let’s use MATLAB. First we create the prototype vectors.

```
p1=[ 1  1 -1 -1  1  1  1]';
p2=[ 1  1  1 -1  1  -1  1]';
p3=[-1  1 -1  1  1 -1  1]';
P=[p1 p2 p3];
```

Now we can compute the weight matrix using the Hebb rule.

```
wh=P*P';
```

To check the network we create the test vector.

```
pt=[-1  1 -1 -1  1 -1  1]';
```

The network response is then calculated.

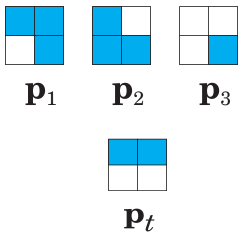
```
ah=hardlims(wh*pt);
ah'
ans =
      1     1    -1    -1     1    -1     1
```

Notice that this response does not match any of the prototype vectors. This is not surprising since the prototype patterns are not orthogonal. Let’s try the pseudoinverse rule.

```
pseu=inv(P'*P)*P';
wp=P*pseu;
ap=hardlims(wp*pt);
ap'
ans =
     -1     1    -1     1     1    -1     1
```

Note that the network response is equal to $\mathbf{p}_3$. Is this the correct response? As usual, we want the response to be the prototype pattern closest to the input pattern. In this case $\mathbf{p}_t$ is a Hamming distance of 2 from both $\mathbf{p}_1$ and $\mathbf{p}_2$, but only a distance of 1 from $\mathbf{p}_3$. Therefore the pseudoinverse rule produces the correct response.

Try other test inputs to see if there are additional cases where the pseudoinverse rule produces better results than the Hebb rule.



P7.4 Consider the three prototype patterns shown to the left.

i. Use the Hebb rule to design a perceptron network that will recognize these three patterns.

ii. Find the response of the network to the pattern $\mathbf{p}_t$ shown to the left. Is the response correct?

i. We can convert the patterns to vectors, as we did in previous problems, to obtain:

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

We now need to choose the desired output vectors for each prototype input vector. Since there are three prototype vectors that we need to distinguish, we will need two elements in the output vector. We can choose the three desired outputs to be:

$$\mathbf{t}_1 = \begin{bmatrix} -1 \\ -1 \end{bmatrix} \qquad \mathbf{t}_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \qquad \mathbf{t}_3 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}.$$

(Note that this choice was arbitrary. Any distinct combination of 1 and -1 could have been chosen for each vector.)

The resulting network is shown in Figure P7.2.

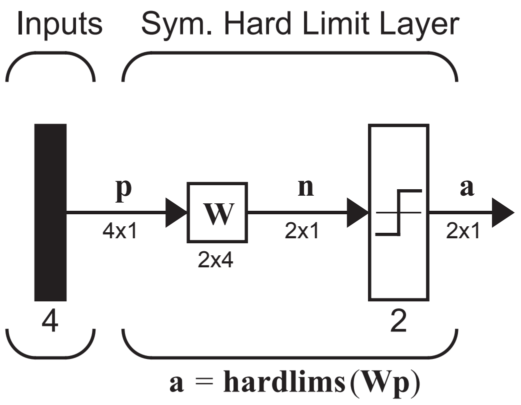


Figure P7.2 Perceptron Network for Problem P7.4

The next step is to determine the weight matrix using the Hebb rule.

$$\mathbf{W} = \mathbf{T}\mathbf{P}^T = \begin{bmatrix} -1 & -1 & 1 \\ -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ -1 & -1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} -3 & -1 & -1 & -1 \\ 1 & 3 & -1 & -1 \end{bmatrix}$$

ii. The response of the network to the test input pattern is calculated as follows.

$$\mathbf{a} = \mathbf{hardlims}(\mathbf{W}\mathbf{p}_t) = \mathbf{hardlims}\left(\begin{bmatrix} -3 & -1 & -1 & -1 \\ 1 & 3 & -1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix}\right)$$

$$= \mathbf{hardlims}\left(\begin{bmatrix} -2 \\ -2 \end{bmatrix}\right) = \begin{bmatrix} -1 \\ -1 \end{bmatrix} \rightarrow \mathbf{p}_1 .$$

So the response of the network indicates that the test input pattern is closest to $\mathbf{p}_1$. Is this correct? Yes, the Hamming distance to $\mathbf{p}_1$ is 1, while the distance to $\mathbf{p}_2$ and $\mathbf{p}_3$ is 3.

P7.5 Suppose that we have a linear autoassociator that has been designed for Q orthogonal prototype vectors of length R using the Hebb rule. The vector elements are either 1 or -1.

i. Show that the Q prototype patterns are eigenvectors of the weight matrix.

ii. What are the other (R - Q) eigenvectors of the weight matrix?

i. Suppose the prototype vectors are:

$$\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_Q .$$

Since this is an autoassociator, these are both the input vectors and the desired output vectors. Therefore

$$\mathbf{T} = \begin{bmatrix} \mathbf{p}_1 & \mathbf{p}_2 & \dots & \mathbf{p}_Q \end{bmatrix} \qquad \mathbf{P} = \begin{bmatrix} \mathbf{p}_1 & \mathbf{p}_2 & \dots & \mathbf{p}_Q \end{bmatrix} .$$

If we then use the Hebb rule to calculate the weight matrix we find

$$\mathbf{W} = \mathbf{T}\mathbf{P}^T = \sum_{q=1}^{Q} \mathbf{p}_q \mathbf{p}_q^T ,$$

from Eq. (7.8). Now, if we apply one of the prototype vectors as input to the network we obtain

$$\mathbf{a} = \mathbf{W}\mathbf{p}_k = \left(\sum_{q=1}^{Q} \mathbf{p}_q \mathbf{p}_q^T\right)\mathbf{p}_k = \sum_{q=1}^{Q} \mathbf{p}_q(\mathbf{p}_q^T \mathbf{p}_k) .$$

Because the patterns are orthogonal, this reduces to

$$\mathbf{a} = \mathbf{p}_k(\mathbf{p}_k^T \mathbf{p}_k) .$$

And since every element of $\mathbf{p}_k$ must be either -1 or 1, we find that

$$\mathbf{a} = \mathbf{p}_k R .$$

To summarize the results:

$$\mathbf{W}\mathbf{p}_k = R\mathbf{p}_k ,$$

which implies that $\mathbf{p}_k$ is an eigenvector of $\mathbf{W}$ and R is the corresponding eigenvalue. Each prototype vector is an eigenvector with the same eigenvalue.

ii. Note that the repeated eigenvalue R has a Q-dimensional eigenspace associated with it: the subspace spanned by the Q prototype vectors. Now consider the subspace that is orthogonal to this eigenspace. Every vector in this subspace should be orthogonal to each prototype vector. The dimension of the orthogonal subspace will be $R - Q$. Consider the following arbitrary basis set for this orthogonal space:

$$\mathbf{z}_1, \mathbf{z}_2, \ldots , \mathbf{z}_{R-Q} .$$

If we apply any one of these basis vectors to the network we obtain:

$$\mathbf{a} = \mathbf{W}\mathbf{z}_k = \left(\sum_{q=1}^{Q} \mathbf{p}_q \mathbf{p}_q^T\right)\mathbf{z}_k = \sum_{q=1}^{Q} \mathbf{p}_q(\mathbf{p}_q^T \mathbf{z}_k) = 0 ,$$

since each $\mathbf{z}_k$ is orthogonal to every $\mathbf{p}_q$. This implies that each $\mathbf{z}_k$ is an eigenvector of $\mathbf{W}$ with eigenvalue 0.

To summarize, the weight matrix $\mathbf{W}$ has two eigenvalues, R and 0. This means that any vector in the space spanned by the prototype vectors will be amplified by R, whereas any vector that is orthogonal to the prototype vectors will be set to 0. We will revisit this concept when we discuss the performance of the Hopfield network in Chapter 18.

P7.6 The networks we have used so far in this chapter have not included a bias vector. Consider the problem of designing a perceptron network (Figure P7.3) to recognize the following patterns:

$$\mathbf{p}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \qquad \mathbf{p}_2 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}.$$

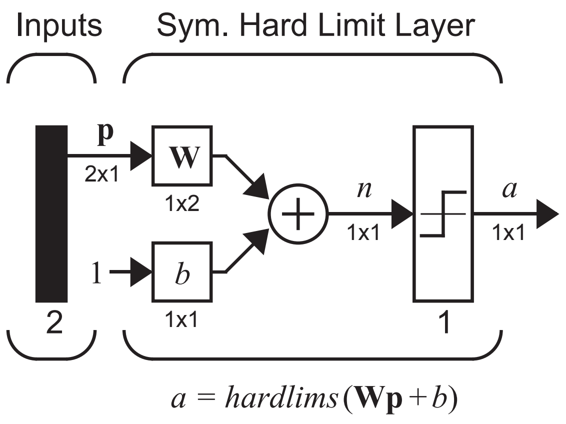


Figure P7.3 Single-Neuron Perceptron

i. Why is a bias required to solve this problem?

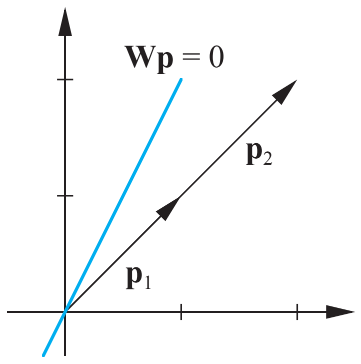
ii. Use the pseudoinverse rule to design a network with bias to solve this problem.

i. Recall from Chapters 3 and 4 that the decision boundary for the perceptron network is the line defined by:

$$\mathbf{W}\mathbf{p} + b = 0.$$

If there is no bias, then $b = 0$ and the boundary is defined by:

$$\mathbf{W}\mathbf{p} = 0,$$



which is a line that must pass through the origin. Now consider the two vectors, $\mathbf{p}_1$ and $\mathbf{p}_2$, which are given in this problem. They are shown graphically in the figure to the left, along with an arbitrary decision boundary that passes through the origin. It is clear that no decision boundary that passes through the origin could separate these two vectors. Therefore a bias is required to solve this problem.

ii. To use the pseudoinverse rule (or the Hebb rule) when there is a bias term, we should treat the bias as another weight, with an input of 1 (as is shown in all of the network figures). We then augment the input vectors with a 1 as the last element:

$$\mathbf{p}'_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \qquad \mathbf{p}'_2 = \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix}.$$

Let's choose the desired outputs to be

$$t_1 = 1 \qquad t_2 = -1,$$

so that

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

We now form the pseudoinverse matrix:

$$\mathbf{P}^{+} = \left(\begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 1 & 2 \\ 1 & 1 \end{bmatrix} \right)^{-1} \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 5 \\ 5 & 9 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 1 \end{bmatrix} = \begin{bmatrix} -0.5 & -0.5 & 2 \\ 0.5 & 0.5 & -1 \end{bmatrix}.$$

The augmented weight matrix is then computed:

$$\mathbf{W}' = \mathbf{TP}^{+} = \begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} -0.5 & -0.5 & 2 \\ 0.5 & 0.5 & -1 \end{bmatrix} = \begin{bmatrix} -1 & -1 & 3 \end{bmatrix}.$$

We can then pull out the standard weight matrix and bias:

$$\mathbf{W} = \begin{bmatrix} -1 & -1 \end{bmatrix} \qquad b = 3.$$

The decision boundary for this weight and bias is shown in the Figure P7.4. This boundary does separate the two prototype vectors.

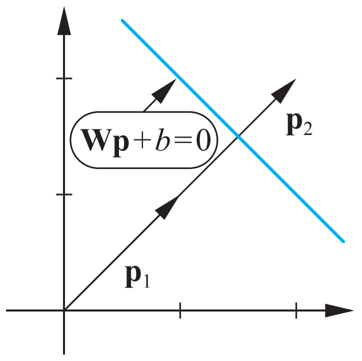


Figure P7.4 Decision Boundary for Solved Problem P7.6

P7.7 In all of our pattern recognition examples thus far, we have represented patterns as vectors by using "1" and "-1" to represent dark and light pixels (picture elements), respectively. What if we were to use "1" and "0" instead? How should the Hebb rule be changed?

First, let's introduce some notation to distinguish the two different representations (usually referred to as the bipolar {-1, 1} representation and the binary {0, 1} representation). The bipolar representation of the prototype input/output vectors will be denoted

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

and the binary representation will be denoted

$$\{\mathbf{p}'_1, \mathbf{t}'_1\}, \{\mathbf{p}'_2, \mathbf{t}'_2\}, \ldots, \{\mathbf{p}'_Q, \mathbf{t}'_Q\}.$$

The relationship between the two representations is given by:

$$\mathbf{p}'_q = \frac{1}{2}\mathbf{p}_q + \frac{1}{2}\mathbf{1} \qquad \mathbf{p}_q = 2\mathbf{p}'_q - \mathbf{1},$$

where $\mathbf{1}$ is a vector of ones.

Next, we determine the form of the binary associative network. We will use the network shown in Figure P7.5. It is different than the bipolar associative network, as shown in Figure 7.2, in two ways. First, it uses the *hardlim* nonlinearity rather than *hardlims*, since the output should be either 0 or 1. Secondly, it uses a bias vector. It requires a bias vector because all binary vectors will fall into one quadrant of the vector space, so a boundary that passes through the origin will not always be able to divide the patterns. (See Problem P7.6.)

The next step is to determine the weight matrix and the bias vector for this network. If we want the binary network of Figure P7.5 to have the same effective response as a bipolar network (as in Figure 7.2), then the net input, $\mathbf{n}$, should be the same for both networks:

$$\mathbf{W}'\mathbf{p}' + \mathbf{b} = \mathbf{W}\mathbf{p} .$$

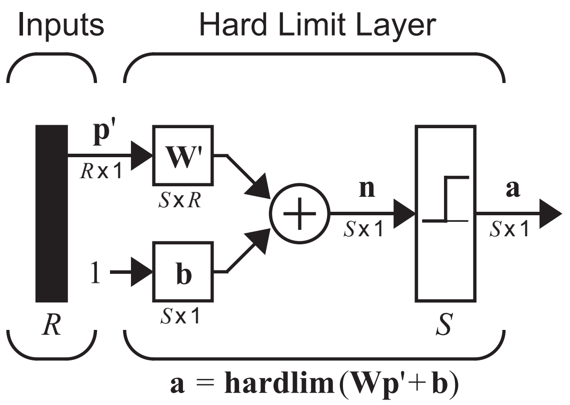


Figure P7.5 Binary Associative Network

This will guarantee that whenever the bipolar network produces a “1” the binary network will produce a “1”, and whenever the bipolar network produces a “-1” the binary network will produce a “0”.

If we then substitute for $\mathbf{p}'$ as a function of $\mathbf{p}$ we find:

$$\mathbf{W}'\left(\frac{1}{2}\mathbf{p} + \frac{1}{2}\mathbf{1}\right) + \mathbf{b} = \frac{1}{2}\mathbf{W}'\mathbf{p} + \frac{1}{2}\mathbf{W}'\mathbf{1} + \mathbf{b} = \mathbf{W}\mathbf{p} .$$

Therefore, to produce the same results as the bipolar network, we should choose

$$\mathbf{W}' = 2\mathbf{W} \qquad \mathbf{b} = -\mathbf{W}\mathbf{1} ,$$

where $\mathbf{W}$ is the bipolar weight matrix.

Epilogue

We had two main objectives for this chapter. First, we wanted to introduce one of the most influential neural network learning rules: the Hebb rule. This was one of the first neural learning rules ever proposed, and yet it continues to influence even the most recent developments in network learning theory. Second, we wanted to show how the performance of this learning rule could be explained using the linear algebra concepts discussed in the two preceding chapters. This is one of the key objectives of this text. We want to show how certain important mathematical concepts underlie the operation of all artificial neural networks. We plan to continue to weave together the mathematical ideas with the neural network applications, and hope in the process to increase our understanding of both.

We will again revisit the Hebb rule in Chapters 15 and 21. In Chapter 21 we will use the Hebb rule in the design of a *recurrent* associative memory network — the Hopfield network.

The next two chapters introduce some mathematics that are critical to our understanding of the two learning laws covered in Chapters 10 and 11. Those learning laws fall under a subheading called *performance* learning, because they attempt to optimize the performance of the network. In order to understand these performance learning laws, we need to introduce some basic concepts in optimization. As with the material on the Hebb rule, our understanding of these topics in optimization will be greatly aided by our previous work in linear algebra.

Further Reading

[Albe72] A. Albert, *Regression and the Moore-Penrose Pseudoinverse*, New York: Academic Press, 1972.

Albert's text is the major reference for the theory and basic properties of the pseudoinverse. Proofs are included for all major pseudoinverse theorems.

[Ande72] J. Anderson, "A simple neural network generating an interactive memory," *Mathematical Biosciences*, vol. 14, pp. 197–220, 1972.

Anderson proposed a "linear associator" model for associative memory. The model was trained, using a generalization of the Hebb postulate, to learn an association between input and output vectors. The physiological plausibility of the network was emphasized. Kohonen published a closely related paper at the same time [Koho72], although the two researchers were working independently.

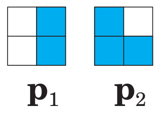
[Hebb49] D. O. Hebb, *The Organization of Behavior*, New York: Wiley, 1949.

The main premise of this seminal book is that behavior can be explained by the action of neurons. In it, Hebb proposes one of the first learning laws, which postulated a mechanism for learning at the cellular level.

[Koho72] T. Kohonen, "Correlation matrix memories," *IEEE Transactions on Computers*, vol. 21, pp. 353–359, 1972.

Kohonen proposed a correlation matrix model for associative memory. The model was trained, using the outer product rule (also known as the Hebb rule), to learn an association between input and output vectors. The mathematical structure of the network was emphasized. Anderson published a closely related paper at the same time [Ande72], although the two researchers were working independently.

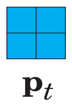
Exercises



E7.1 Consider the prototype patterns given to the left.

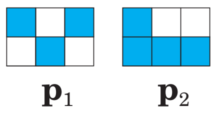
i. Are $\mathbf{p}_1$ and $\mathbf{p}_2$ orthogonal?

ii. Use the Hebb rule to design an autoassociator network for these patterns.



iii. Test the operation of the network using the test input pattern $\mathbf{p}_t$ shown to the left. Does the network perform as you expected? Explain.

E7.2 Repeat Exercise E7.1 using the pseudoinverse rule.



E7.3 Use the Hebb rule to determine the weight matrix for a perceptron network (shown in Figure E7.1) to recognize the patterns shown to the left.

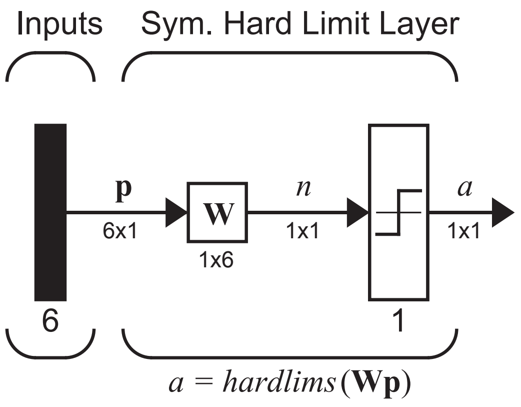


Figure E7.1 Perceptron Network for Exercise E7.3

E7.4 In Problem P7.7 we demonstrated how networks can be trained using the Hebb rule when the prototype vectors are given in binary (as opposed to bipolar) form. Repeat Exercise E7.1 using the binary representation for the prototype vectors. Show that the response of this binary network is equivalent to the response of the original bipolar network.

E7.5 Show that an autoassociator network will continue to perform if we zero the diagonal elements of a weight matrix that has been determined by the Hebb rule. In other words, suppose that the weight matrix is determined from:

$$\mathbf{W} = \mathbf{P}\mathbf{P}^T - Q\mathbf{I},$$

where Q is the number of prototype vectors. (Hint: show that the prototype vectors continue to be eigenvectors of the new weight matrix.)

E7.6 We have three input/output prototype vector pairs:

$$\left\{\mathbf{p}_1 = \begin{bmatrix}1\\0\end{bmatrix}, t_1 = 1\right\}, \left\{\mathbf{p}_2 = \begin{bmatrix}1\\1\end{bmatrix}, t_2 = -1\right\}, \left\{\mathbf{p}_3 = \begin{bmatrix}0\\1\end{bmatrix}, t_3 = 1\right\}.$$

i. Show that this problem cannot be solved unless the network uses a bias.

ii. Use the pseudoinverse rule to design a network for these prototype vectors. Verify that the network correctly transforms the prototype vectors.

E7.7 Consider the reference patterns and targets given below. We want to use these data to train a linear associator network.

$$\left\{\mathbf{p}_1 = \begin{bmatrix}2\\4\end{bmatrix}, t_1 = \begin{bmatrix}26\end{bmatrix}\right\} \qquad \left\{\mathbf{p}_2 = \begin{bmatrix}4\\2\end{bmatrix}, t_2 = \begin{bmatrix}26\end{bmatrix}\right\} \qquad \left\{\mathbf{p}_3 = \begin{bmatrix}-2\\-2\end{bmatrix}, t_3 = \begin{bmatrix}-26\end{bmatrix}\right\}$$

i. Use the Hebb rule to find the weights of the network.

ii. Find and sketch the decision boundary for the network with the Hebb rule weights.

iii. Use the pseudo-inverse rule to find the weights of the network. Because the number, R, of rows of $\mathbf{P}$ is less than the number of columns, Q, of $\mathbf{P}$, the pseudoinverse can be computed by $\mathbf{P}^{+} = \mathbf{P}^{T}(\mathbf{P}\mathbf{P}^{T})^{-1}$.

iv. Find and sketch the decision boundary for the network with the pseudo-inverse rule weights.

v. Compare (discuss) the decision boundaries and weights for each of the methods (Hebb and pseudo-inverse).

E7.8 Consider the three prototype patterns shown in Figure E7.2.

i. Are these patterns orthogonal? Demonstrate.

ii. Use the Hebb rule to determine the weight matrix for a linear autoassociator to recognize these patterns.

iii. Draw the network diagram.

iv. Find the eigenvalues and eigenvectors of the weight matrix. (Do **not** solve the equation $|\mathbf{W} - \lambda\mathbf{I}| = 0$. Use an analysis of the Hebb rule.)

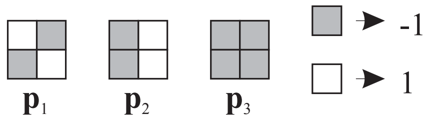


Figure E7.2 Prototype Patterns for Exercise E7.8

E7.9 Suppose that we have the following three reference patterns and their targets.

$$\left\{\mathbf{p}_1 = \begin{bmatrix} 3 \\ 6 \end{bmatrix}, t_1 = \begin{bmatrix} 75 \end{bmatrix}\right\} \quad \left\{\mathbf{p}_2 = \begin{bmatrix} 6 \\ 3 \end{bmatrix}, t_2 = \begin{bmatrix} 75 \end{bmatrix}\right\} \quad \left\{\mathbf{p}_3 = \begin{bmatrix} -6 \\ 3 \end{bmatrix}, t_3 = \begin{bmatrix} -75 \end{bmatrix}\right\}$$

i. Draw the network diagram for a linear associator network that could be trained on these patterns.

ii. Use the Hebb rule to find the weights of the network.

iii. Find and sketch the decision boundary for the network with the Hebb rule weights. Does the boundary separate the patterns? Demonstrate.

iv. Use the pseudo-inverse rule to find the weights of the network. Describe the difference between this boundary and the Hebb rule boundary.

E7.10 We have the following input/output pairs:

$$\left\{\mathbf{p}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, t_1 = \begin{bmatrix} 1 \end{bmatrix}\right\} \quad \left\{\mathbf{p}_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, t_2 = \begin{bmatrix} -1 \end{bmatrix}\right\}$$

i. Use the Hebb rule to determine the weight matrix for the perceptron network shown in Figure E7.3.

ii. Plot the resulting decision boundary. Is this a "good" decision boundary? Explain.

iii. Repeat part i. using the Pseudoinverse rule.

iv. Will there be any difference in the operation of the network if the Pseudoinverse weight matrix is used? Explain.

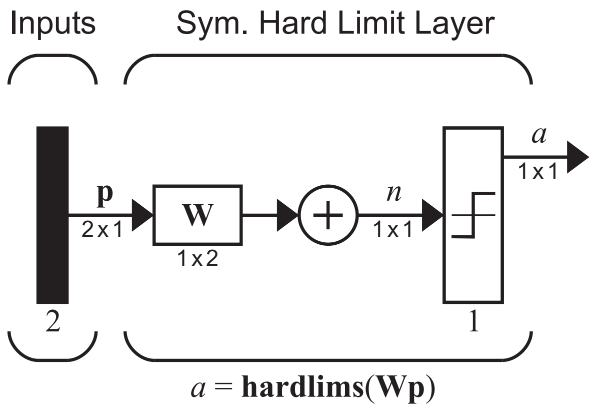


Figure E7.3 Network for Exercise E7.10

```
» 2 + 2
ans =
     4
```

E7.11 One question we might ask about the Hebb and pseudoinverse rules is: How many prototype patterns can be stored in one weight matrix? Test this experimentally using the digit recognition problem that was discussed on page 7-10. Begin with the digits “0” and “1”. Add one digit at a time up to “6”, and test how often the correct digit is reconstructed after randomly changing 2, 4 and 6 pixels.

i. First use the Hebb rule to create the weight matrix for the digits “0” and “1”. Then randomly change 2 pixels of each digit and apply the noisy digits to the network. Repeat this process 10 times, and record the percentage of times in which the correct pattern (without noise) is produced at the output of the network. Repeat as 4 and 6 pixels of each digit are modified. The entire process is then repeated when the digits “0”, “1” and “2” are used. This continues, one digit at a time, until you test the network when all of the digits “0” through “6” are used. When you have completed all of the tests, you will be able to plot three curves showing percentage error versus number of digits stored, one curve each for 2, 4 and 6 pixel errors.

ii. Repeat part (i) using the pseudoinverse rule, and compare the results of the two rules.