Note that McClelland and Rumelhart use "j" for the sending unit and "i" for the receiving unit, but also reverse the order of the subscripts for weights (so w_ij is the weight from "j" to "i", which is sending-to-receiving).

CHAPTER 4

Learning in PDP Models: The Pattern Associator

In previous chapters we have seen how PDP models can be used as content-addressable memories and constraint-satisfaction mechanisms. PDP models are also of interest because of their learning capabilities. They learn, naturally and incrementally, in the course of processing. In this chapter, we will begin to explore learning in PDP models. We will consider two "classical" procedures for learning: the so-called Hebbian, or correlational learning rule, described by Hebb (1949) and before him by William James (1890), and the error-correcting or "delta" learning rule, as studied in slightly different forms by Widrow and Hoff (1960) and by Rosenblatt (1959).

We will also explore the characteristics of one of the most basic network architectures that has been widely used in distributed memory modeling with the Hebb rule and the delta rule. This is the *pattern associator*. The pattern associator has a set of input units connected to a set of output units by a single layer of modifiable connections that are suitable for training with the Hebb rule and the delta rule. Models of this type have been extensively studied by James Anderson (see Anderson, 1983, for a more recent review), Kohonen (1977), and many others; a number of the papers in the Hinton and Anderson (1981) volume describe models of this type. The models of past-tense learning and of case-role assignment in *PDP:18* and *PDP:19* are pattern associators trained with the delta rule. An analysis of the delta rule in pattern associator models is described in *PDP:11*.

As these works point out, one-layer pattern associators have several suggestive properties that have made them attractive as models of learning and memory. They can learn to act as content-addressable memories; they generalize the responses they make to novel inputs that are similar to the inputs that they have been trained on; they learn to extract the prototype of a set of repeated experiences in ways that are very similar to the concept-learning characteristics seen in human cognitive processes; and they degrade gracefully with damage and noise. In this chapter our aim is to help you develop a basic understanding of the characteristics of these simple parallel networks. However, it must be noted that these kinds of networks have limitations. In the next chapter we will examine these limitations and consider learning procedures that allow the same positive characteristics of pattern associators to manifest themselves in networks and overcome one important class of limitations.

We begin this chapter by presenting a basic description of the learning rules and how they work in training connections coming into a single unit. We will then apply them to learning in the pattern associator.

BACKGROUND

The Hebb Rule

In Hebb's own formulation, this learning rule was somewhat vaguely described. He suggested that when two cells fire at the same time, the strength of the connection between them should be increased. There are a variety of mathematical formulations of this principle that may be given. The simplest by far is the following:

$$\Delta w_{ij} = \epsilon a_i a_j. \tag{1}$$

Here we use ϵ to refer to the value of the learning rate parameter. This version has been used extensively in the early work of James Anderson (e.g., Anderson, 1977). If we start from all-zero weights, then expose the network to a sequence of learning events indexed by l, the value of any weight at the end of a series of learning events will be

$$w_{ij} = \epsilon \sum_l a_{il} a_{jl}. \tag{2}$$

In studying this rule, we will assume that activations are distributed around 0 and that the units in the network have activations that can be set in either of two ways: They may be *clamped* to particular values by external inputs or they may be determined by inputs via their connections to other units in the network. In the latter case, we will initially focus on the case where the units are completely linear; that is, on the case in which the activation and the output of the unit are simply set equal to the net input:

$$a_i = \sum_j a_j w_{ij}. \quad \text{[note "i" is receiving, "j" is sending]} \tag{3}$$

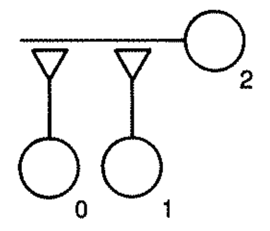
In this formulation, with the activations distributed around 0, the w_{ij} assigned by Equation 2 will be proportional to the correlation between the

activations of units i and j; normalizations can be used to preserve this correlational property when units have mean activations that vary from 0.

The correlational character of the Hebbian learning rule is at once the strength of the procedure and its weakness. It is a strength because these correlations can sometimes produce useful associative learning; that is, particular units, when active, will tend to excite other units whose activations have been correlated with them in the past. It can be a weakness, though, since correlations between unit activations often are not sufficient to allow a network to learn even very simple associations between patterns of activation.

First let's examine a positive case: a simple network consisting of two input units and one output unit (Figure 1A). Suppose that we arrange things so that by means of inputs external to this network we are able to impose patterns of activation on these units, and suppose that we use the Hebb rule (Equation 1 above) to train the connections from the two input units to the output unit. Suppose further that we use the four patterns shown in Figure 1B; that is, we present each pattern, forcing the units to

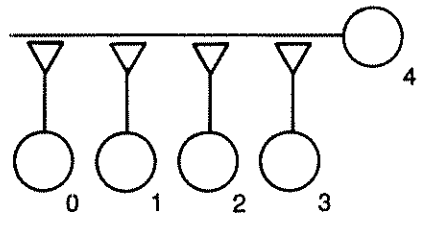
A



B

Input		Output
0	1	2
+	+	+
+	−	+
−	+	−
−	−	−

C



D

Input				Output
0	1	2	3	4
+	−	+	−	+
+	+	+	+	+
+	+	+	−	−
+	−	−	+	−

E

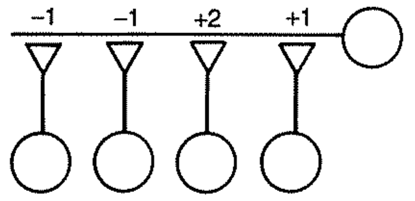


FIGURE 1. Two simple associative networks and the patterns used in training them.

the correct activation, then we adjust the strengths of the connections between the units. According to Equation 1, w_{20} (the weight on the connection to unit 2 from unit 0) will be increased in strength for each pattern by amount ϵ, which in this case we will set to 1.0. On the other hand, w_{21} will be increased by amount ϵ in one of the cases (pattern 0) and reduced by ϵ in the other case, for a net change of 0.

As a result of this training, then, this simple network would have acquired a positive connection weight to unit 2 from unit 0. This connection will now allow unit 0 to make unit 2 take on an activation value correlated with that of unit 0. At the same time, the network would have acquired a null connection from unit 1 to unit 2, capturing the fact that the activation of unit 1 has no predictive relation to the activation of unit 2. In this way, it is possible to use Hebbian learning to learn associations that depend on the correlation between activations of units in a network.

Unfortunately, the correlational learning that is possible with a Hebbian learning rule is a "unitwise" correlation, and sometimes, these unitwise correlations are not sufficient to learn correct associations between whole input patterns and appropriate responses. To see that this is so, suppose we change our network so that there are now four input units and one output unit, as shown in Figure 1C. And suppose we want to train the connections in the network so that the output unit takes on the values given in Figure 1D for each of the four input patterns shown there. In this case, the Hebbian learning procedure will not produce correct results. To see why, we need to examine the values of the weights (equivalently, the pairwise correlations of the activations of each sending unit with the receiving unit). What we see is that three of the connections end up with 0 weights because the activation of the corresponding input unit is uncorrelated with the activation of the output unit. Only one of the input units, unit 2, has a positive correlation with unit 4 over this set of patterns. This means that the output unit will make the same response to the first three patterns since in all three of these cases the third unit is on, and this is the only unit with a nonzero connection to the output unit.

Before leaving this example, we should note that there are values of the connection strengths that will do the job. One such set is shown in Figure 1E. The reader can check that this set produces the correct results for each of the four input patterns by using Equation 3.

Apparently, then, successful learning requires finding connection strengths that are not proportional to the correlations of activations of the units. How can this be done?

The Delta Rule

One answer that has occurred to many people over the years is the idea of using the difference between the desired, or *target*, activation and the

obtained activation to drive learning. The idea is to adjust the strengths of the connections so that they will tend to reduce this *difference* or *error* measure. Because the rule is driven by differences, we have tended to call it the delta rule. Others have called it the Widrow-Hoff learning rule or the least mean square (LMS) rule (Widrow & Hoff, 1960); it is related to the perceptron convergence procedure of Rosenblatt (1959).

This learning rule, in its simplest form, can be written

$$\Delta w_{ij} = \epsilon e_i a_j \quad (4)$$

where e_i, the error for unit i, is given by

$$e_i = t_i - a_i, \quad (5)$$

[t_i is target of output (receiving) unit]

the difference between the teaching input to unit i and its obtained activation.

To see how this rule works, let's use it to train the five-unit network in Figure 1C on the patterns in Figure 1D. The training regime is a little different here: For each pattern, we turn the input units on, then we see what effect they have on the output unit; its activation reflects the effects of the current connections in the network. (As before we assume the units are linear.) We compute the difference between the obtained output and the teaching input (Equation 5). Then, we adjust the strengths of the connections according to Equation 4. We will follow this procedure as we cycle through the four patterns several times, and look at the resulting strengths of the connections as we go. The network is started with initial weights of 0. The results of this process for the first cycle through all four patterns are shown in the first four rows of Figure 2.

The first time pattern 0 is presented, the response (that is, the obtained activation of the output unit) is 0, so the error is +1. This means that the changes in the weights are proportional to the activations of the input units. A value of 0.25 was used for the learning rate parameter, so each Δw is ± 0.25. These are added to the existing weights (which are 0), so the resulting weights are equal to these initial increments. When pattern 1 is presented, it happens to be uncorrelated with pattern 0, and so again the obtained output is 0. (The output is obtained by summing up the pairwise products of the inputs on the current trial with the weights obtained at the end of the preceding trial.) Again the error is +1, and since all the input units are on in this case, the change in the weight is +0.25 for each input. When these increments are added to the original weights, the result is a value of +0.5 for w_{04} and w_{24}, and 0 for the other weights. When the next pattern is presented, these weights produce an output of +1. The error is therefore −2, and so relatively larger Δw terms result. Even so, when the final pattern is presented, it produces an output of +1 as well. When the weights are adjusted to take this into account, the weight from input unit 0 is negative and the weight from unit 2 is positive; the other weights are 0. This completes the first sweep through the set of patterns. At this point,

```
Ep Pat    Input     Tgt Output Error       Delta w's            New values of w's

 0  0  1 -1  1 -1|  1  0.00  1.00| 0.25-0.25 0.25-0.25 |  0.25-0.25 0.25-0.25
 0  1  1  1  1  1|  1  0.00  1.00| 0.25 0.25 0.25 0.25 |  0.50 0.00 0.50 0.00
 0  2  1  1  1 -1| -1  1.00 -2.00|-0.50-0.50-0.50 0.50 |  0.00-0.50 0.00 0.50
 0  3  1 -1 -1  1| -1  1.00 -2.00|-0.50 0.50 0.50-0.50 | -0.50 0.00 0.50 0.00

                       tss: 10.00

 1  0  1 -1  1 -1|  1  0.00  1.00| 0.25-0.25 0.25-0.25 | -0.25-0.25 0.75-0.25
 1  1  1  1  1  1|  1  0.00  1.00| 0.25 0.25 0.25 0.25 |  0.00 0.00 1.00 0.00
 1  2  1  1  1 -1| -1  1.00 -2.00|-0.50-0.50-0.50 0.50 | -0.50-0.50 0.50 0.50
 1  3  1 -1 -1  1| -1  0.00 -1.00|-0.25 0.25 0.25-0.25 | -0.75-0.25 0.75 0.25

                       tss: 7.00

...

 3  0  1 -1  1 -1|  1  0.25  0.75| 0.19-0.19 0.19-0.19 | -0.63-0.63 1.25 0.25
 3  1  1  1  1  1|  1  0.25  0.75| 0.19 0.19 0.19 0.19 | -0.44-0.44 1.44 0.44
 3  2  1  1  1 -1| -1  0.13 -1.13|-0.28-0.28-0.28 0.28 | -0.72-0.72 1.16 0.72
 3  3  1 -1 -1  1| -1 -0.44 -0.56|-0.14 0.14 0.14-0.14 | -0.86-0.58 1.30 0.58

                       tss: 1.52

...

10  0  1 -1  1 -1|  1  0.90  0.10| 0.03-0.03 0.03-0.03 | -0.95-0.95 1.90 0.90
10  1  1  1  1  1|  1  0.90  0.10| 0.03 0.03 0.03 0.03 | -0.92-0.92 1.92 0.92
10  2  1  1  1 -1| -1 -0.85 -0.15|-0.04-0.04-0.04 0.04 | -0.96-0.96 1.89 0.96
10  3  1 -1 -1  1| -1 -0.92 -0.08|-0.02 0.02 0.02-0.02 | -0.98-0.94 1.91 0.94

                       tss: 0.05

...

20  0  1 -1  1 -1|  1  0.99  0.01| 0.00-0.00 0.00-0.00 | -1.00-1.00 1.99 0.99
20  1  1  1  1  1|  1  0.99  0.01| 0.00 0.00 0.00 0.00 | -1.00-1.00 2.00 1.00
20  2  1  1  1 -1| -1 -0.99 -0.01|-0.00-0.00-0.00 0.00 | -1.00-1.00 1.99 1.00
20  3  1 -1 -1  1| -1 -1.00 -0.00|-0.00 0.00 0.00-0.00 | -1.00-1.00 1.99 1.00

                       tss: 0.00
```

FIGURE 2. Learning with the delta rule. See text for explanation.

the values of the weights are far from perfect; if we froze them at these values, the network would produce 0 output to the first three patterns. It would produce the correct answer (an output of -1) only for the last pattern.

The correct set of weights is approached asymptotically if the training procedure is continued for several more sweeps through the set of patterns. Each of these sweeps, or *training epochs*, as we will call them henceforth, results in a set of weights that is closer to a perfect solution. To get a measure of the closeness of the approximation to a perfect solution, we can calculate an error measure for each pattern as that pattern is being processed. For each pattern, the error measure is the value of the error $(t-a)$ squared. This measure is then summed over all patterns to get a *total sum of squares* or *tss* measure. The resulting error measure, shown for each of the illustrated epochs in Figure 2, gets smaller over epochs, as do the changes in the strengths of the connections. The weights that result at

the end of 20 epochs of training are very close to the perfect solution values. With more training, the weights converge to these values.

The error-correcting learning rule, then, is much more powerful than the Hebb rule. In fact, it can be proven rather easily that the error-correcting rule will find a set of weights that drives the error as close to 0 as we want for each and every pattern in the training set, provided such a set of weights exists. Many proofs of this theorem have been given; a particularly clear one may be found in Minsky and Papert (1969).

The Linear Predictability Constraint

We have just noted that the delta rule will find a set of weights that solves a network learning problem, provided such a set of weights exists. What are the conditions under which such a set actually does exist?

Such a set of weights exists only if for each input-pattern–target-pair the target can be predicted from a weighted sum, or *linear combination*, of the activations of the input units. That is, the set of weights must satisfy

$$t_{ip} = \sum_j w_{ij} a_{jp} \tag{6}$$

for output unit i in all patterns p.

This constraint (which we called the *linear predictability constraint* in *PDP:17*) can be overcome by the use of hidden units, but hidden units cannot be trained using the delta rule as we have described it here because (by definition) there is no teacher for them. Procedures for training such units are discussed in Chapter 5.

Up to this point, we have considered the use of the Hebb rule and the delta rule for training connections coming into a single unit. We now consider how these learning rules produce the characteristics of *pattern associator* networks.

THE PATTERN ASSOCIATOR

In a pattern associator, there are two sets of units: input units and output units. There is also a matrix representing the connections from the input units to the output units. A pattern associator is really just an extension of the simple networks we have been considering up to now, in which the number of output units is greater than one and each input unit has a connection to each output unit. An example of an eight-unit by eight-unit pattern associator is shown in Figure 3.

The pattern associator is a device that learns associations between input patterns and output patterns. It is interesting because what it learns about

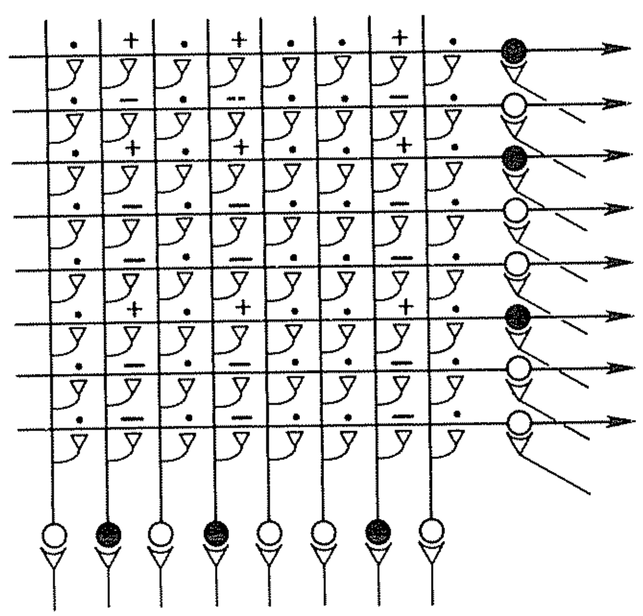


FIGURE 3. A schematic diagram of an eight-unit pattern associator. An input pattern, an output pattern, and values for the weights that will allow the input to produce the output are shown. (From *PDP:18*, p. 227.)

one pattern tends to generalize to other similar patterns. In what follows we will see how this property arises, first in the simplest possible pattern associator—a pattern associator consisting of linear units, trained by the Hebb rule.[1]

The Hebb Rule in Pattern Associator Models

To begin, let us consider the effects of training a network with a single learning trial l, involving an input pattern $\mathbf{i}_l$ and an output pattern $\mathbf{o}_l$. Assuming all the weights in the network are initially 0, we can express the value of each weight as

$$w_{ij} = \epsilon i_{jl} o_{il}. \tag{7}$$

Note that we are using the variable i_{jl} to stand for the activation of input unit j in input pattern $\mathbf{i}_l$, and we are using o_{il} to stand for the activation of output unit i in output pattern $\mathbf{o}_l$. Thus, each weight is just the product of

[1] Readers who wish to gain a better grasp on the mathematical basis of this class of models may find it worthwhile to read *PDP:9*. An in-depth analysis of the delta rule in pattern associators is in *PDP:11*.

the activation of the input unit times the activation of the output unit in the learning trial l.

Now let us present a test input pattern, $\mathbf{i}_t$, and examine the resulting output pattern it produces. Since the units are linear, the activation of output unit i when tested with input pattern $\mathbf{i}_t$ is

$$o_{it} = \sum_j w_{ij} i_{jt}. \tag{8}$$

Substituting for w_{ij} from Equation 7 yields

$$o_{it} = \sum_j \epsilon i_{jl} o_{il} i_{jt} \tag{9}$$

Since we are summing with respect to j in this last equation, we can pull out ϵ and o_{il}:

$$o_{it} = \epsilon o_{il} \sum_j i_{jl} i_{jt}. \tag{10}$$

Equation 10 says that the output at the time of test will be proportional to the output at the time of learning times the sum of the elements of the input pattern at the time of learning, each multiplied by the corresponding element of the input pattern at the time of test.

This sum of products of corresponding elements is called the *dot product*. It is very important to our analysis because it expresses the *similarity* of the two patterns $\mathbf{i}_l$ and $\mathbf{i}_t$. It is worth noting that we have already encountered an expression similar to this one in Equation 2. In that case, though, the quantity was proportional to the correlation of the activations of two *units* across an ensemble of *patterns*. Here, it is proportional to the correlation of two *patterns* across an ensemble of *units*. It is often convenient to normalize the dot product by taking out the effects of the number of elements in the vectors in question by dividing the dot product by the number of elements. We will call this quantity the *normalized dot product*. For patterns consisting of all +1s and −1s, it corresponds to the correlation between the two patterns. The normalized dot product has a value of 1 if the patterns are identical, a value of −1 if they are exactly opposite to each other, and a value of 0 if the elements of one vector are completely uncorrelated with the elements of the other.

We can rewrite Equation 10, then, replacing the summed quantity by the normalized dot product of input pattern $\mathbf{i}_l$ and input pattern $\mathbf{i}_t$, which we denote by $(\mathbf{i}_l \cdot \mathbf{i}_t)_n$:

$$o_{it} = k o_{il} (\mathbf{i}_l \cdot \mathbf{i}_t)_n \tag{11}$$

where $k = n\epsilon$ (n is the number of units). Since Equation 11 applies to all of the elements of the output pattern $\mathbf{o}_t$, we can write

$$\mathbf{o}_t = k\mathbf{o}_l (\mathbf{i}_l \cdot \mathbf{i}_t)_n. \tag{12}$$

This result is very basic to thinking in terms of patterns since it demonstrates that what is crucial for the performance of the network is the similarity relations among the input patterns—their correlations—rather than their specific properties considered as individuals.[2] Thus Equation 12 says that the output pattern produced by our network at test is a scaled version of the pattern stored on the learning trial. The magnitude of the pattern is proportional to the similarity of the learning and test patterns. In particular, if $k = 1$ and if the test pattern is identical to the training pattern, then the output at test will be identical to the output at learning.

An interesting special case occurs when the normalized dot product between the learned pattern and the test pattern is 0. In this case, the output is 0: There is no response whatever. Patterns that have this property are called *orthogonal* or *uncorrelated*; note that this is not the same as being opposite or *anticorrelated*.

To develop intuitions about orthogonality, you should compute the normalized dot products of each of the patterns b, c, d, and e below with pattern a:

a	$+ + - -$
b	$+ - + -$
c	$+ - - +$
d	$+ + + +$
e	$- - + +$

You will see that patterns b, c, and d are all orthogonal to pattern a; in fact, they are all orthogonal to each other. Pattern e, on the other hand, is not orthogonal to pattern a, but is anticorrelated with it. Interestingly, it forms an orthogonal set with patterns b, c, and d. When all the members of a set of patterns are orthogonal to each other, we call them an *orthogonal set*.

Now let us consider what happens when an entire ensemble of patterns is presented during learning. In the Hebbian learning situation, the set of weights resulting from an ensemble of patterns is just the sum of the sets of weights resulting from each individual pattern. That is, after learning trials on each of a set of input patterns $\mathbf{i}_l$, each paired with an output pattern $\mathbf{o}_l$, the value of each weight will be

$$w_{ij} = \epsilon \sum_l i_{jl} o_{il}. \tag{13}$$

Thus, the output produced by each test pattern is

$$\mathbf{o}_t = k \sum_l \mathbf{o}_l (\mathbf{i}_l \cdot \mathbf{i}_t)_n. \tag{14}$$

[2] Technically, performance depends on the similarity relations among the patterns and on their overall strength or magnitude. However, among vectors of equal strength (e.g., the vectors consisting of all +1s and −1s), only the similarity relations are important.

In words, the output of the network in response to input pattern t is the sum of the output patterns that occurred during learning, with each pattern's contribution weighted by the similarity of the corresponding input pattern to the test pattern. Three important facts follow from this:

1. If a test input pattern is orthogonal to all training input patterns, the output of the network will be 0; there will be no response to an input pattern that is completely orthogonal to all of the input patterns that occurred during learning.

2. If a test input pattern is similar to one of the learned input patterns and is uncorrelated with all the others, then the test output will be a scaled version of the output pattern that was paired with the similar input pattern during learning. The magnitude of the output will be proportional to the similarity of the test input pattern to the learned input pattern.

3. For other test input patterns, the output will always be a blend of the training outputs, with the contribution of each output pattern weighted by the similarity of the corresponding input pattern to the test input pattern.

In the exercises, we will see how these properties lead to several desirable features of pattern associator networks, particularly their ability to generalize based on similarity between test patterns and patterns presented during training.

These properties also reflect the limitations of the Hebbian learning rule; when the input patterns used in training the network do not form an orthogonal set, it is not in general possible to avoid contamination, or "cross-talk," between the response that is appropriate to one pattern and the response that occurs to the others. This accounts for the failure of Hebbian learning with the second set of training patterns considered in Figure 1. The reader can check that the input patterns we used in our first training example in Figure 1 (which was successful) were orthogonal but that the patterns used in the second example were not orthogonal.

The Delta Rule in Pattern Associator Models

Once again, the delta rule allows us to overcome the orthogonality limitation imposed by the Hebb rule. For the pattern associator case, the delta rule for a particular input-target pair $\mathbf{i}_l, \mathbf{t}_l$ is

$$\Delta w_{ij} = \epsilon (t_{il} - o_{il}) i_{jl}. \tag{15}$$

Therefore the weights that result from an ensemble of learning pairs indexed by l can be written:

$$w_{ij} = \epsilon \sum_l (t_{il} - o_{il}) i_{jl}. \tag{16}$$

It is interesting to compare this to the Hebb rule. Consider first the case where each of the learned patterns is orthogonal to every other one and is presented exactly once during learning. Then $\mathbf{o}_l$ will be $\mathbf{0}$ (a vector of all zeros) for all learned patterns l, and the above formula reduces to

$$w_{ij} = \epsilon \sum_l t_{il} i_{jl}. \tag{17}$$

In this case, the delta rule produces the same results as the Hebb rule; the teaching input simply replaces the output pattern from Equation 13. As long as the patterns remain orthogonal to each other, there will be no cross-talk between patterns. Learning will proceed independently for each pattern. There is one difference, however. If we continue learning beyond a single epoch, the delta rule will stop learning when the weights are such that they allow the network to produce the target patterns exactly. In the Hebb rule, the weights will grow linearly with each presentation of the set of patterns, getting stronger without bound.

In the case where the input patterns $\mathbf{i}_l$ are not orthogonal, the results of the two learning procedures are more distinct. In this case, though, we can observe the following interesting fact: We can read Equation 15 as indicating that the change in the weights that occurs on a learning trial is storing an association of the input pattern with the *error* pattern; that is, we are adding to each weight an increment that can be thought of as an association between the *error* for the output unit and the activation of the input unit. To see the implications of this, let's examine the effects of a learning trial with input pattern $\mathbf{i}_l$ paired with output pattern $\mathbf{t}_l$ on the output produced by test pattern $\mathbf{i}_t$. The effect of the change in the weights due to this learning trial (as given by Equation 15) will be to change the output of some output unit i by an amount proportional to the error that occurred for that unit on the learning trial, e_i, times the dot product of the learned pattern with the test pattern:

$$\Delta o_{it} = k e_{il} (\mathbf{i}_l \cdot \mathbf{i}_t)_n.$$

Here k is again equal to ϵ times the number of input units n. In vector notation, the change in the output pattern $\mathbf{o}_t$ can be expressed as

$$\Delta \mathbf{o}_t = k \mathbf{e}_l (\mathbf{i}_l \cdot \mathbf{i}_t)_n$$

Thus, the change in the output pattern at test is proportional to the error vector times the normalized dot product of the input pattern that occurred

during learning and the input pattern that occurred during test. Two facts follow from this:

1. If the input on the learning trial is identical to the input on the test trial so that the normalized dot product is 1.0 and if $k = 1.0$, then the change in the output pattern will be exactly equal to the error pattern. Since the error pattern is equal to the difference between the target and the obtained output on the learning trial, this amounts to one trial learning of the desired association between the input pattern on the training trial and the target on this trial.

2. However, if $\mathbf{i}_t$ is different from $\mathbf{i}_l$ but not completely different so that $(\mathbf{i}_l \cdot \mathbf{i}_t)_n$ is not equal to either 1 or 0, then the output produced by $\mathbf{i}_t$ will be affected by the learning trial. The magnitude of the effect will be proportional to the magnitude of $(\mathbf{i}_l \cdot \mathbf{i}_t)_n$.

The second effect—the transfer from learning one pattern to performance on another—may be either beneficial or interfering. Importantly, for patterns of all +1s and −1s, the transfer is always less than the effect on the pattern used on the learning trial itself, since the normalized dot product of two different patterns must be less than the normalized dot product of a pattern with itself. This fact plays a role in several proofs concerning the convergence of the delta rule learning procedure (see Kohonen, 1977, and *PDP:11* for further discussion).

The Linear Predictability Constraint Again

Earlier we considered the linear predictability constraint for training a single output unit. Since the pattern associator can be viewed as a collection of several different output units, the constraint applies to each unit in the pattern associator. Thus, to master a set of patterns there must exist a set of weights w_{ij} such that

$$t_{ip} = \sum_j w_{ij} i_{jp} \tag{18}$$

for all output units i for all target-input pattern pairs p.

Another way of putting this set of constraints that is appropriate for the pattern associator is as follows: An arbitrary output pattern $\mathbf{o}_p$ can be correctly associated with a particular input pattern $\mathbf{i}_p$ without ruining associations between other input-output pairs, only if $\mathbf{i}_p$ *cannot* be written as a *linear combination* of the other input patterns. A pattern that cannot be written as a linear combination of a set of other patterns is said to be *linearly independent* from these other patterns. When all the members of a set of patterns are linearly independent, we say they form a *linearly independent set*. To ensure that arbitrary associations to each of a set of input patterns can be learned, the input patterns must form a linearly independent set.

It is worth noting that the linear independence constraint is primarily a constraint on the similarity relations among input patterns. If we consider the input patterns to be representations of environmental inputs, then whether a set of weights exists that allows us to associate arbitrary responses with each environmental input depends on the way in which these environmental inputs are represented as patterns of activation over a set of input units inside the system. As long as we already have a way of representing a set of environmental inputs so that they are linearly independent, the delta rule will be able to associate any arbitrary responses with these environmental inputs.

Although this is a serious constraint, it is worth noting that there are cases in which the response that we need to make to one input pattern can be predictable from the responses that we make to other patterns with which they overlap. In these cases, the fact that the pattern associator produces a response that is a combination of the responses to other patterns allows it to produce very efficient, often rule-like solutions to the problem of mapping each of a set of input patterns to the appropriate response. We will examine this property of pattern associators in the exercises.

Nonlinear Pattern Associators

Not all pattern associator models that have been studied in the literature make use of the linear activation assumptions we have been using in this analysis. Several different kinds of nonlinear pattern associators (i.e., associators in which the output units have nonlinear activation functions) fall within the general class of pattern associator models. These nonlinearities have effects on performance, but the basic principles that we have observed here are preserved even when these nonlinearities are in place. In particular:

1. Orthogonal inputs are mutually transparent.
2. The learning process converges with the delta rule as long as there is a set of weights that will solve the learning problem; the nonlinearities that have been tried tend not to have much effect on which sets of patterns can be learned and which cannot.
3. What is learned about one pattern tends to transfer to others.

THE FAMILY OF PATTERN ASSOCIATOR MODELS

With the above as background, we turn to a brief specification of several members of the class of pattern associator models that are available through the **pa** program. These are all variants on the pattern associator theme. Each model consists of a set of input units and a set of output units. The activations of the input units are *clamped* by externally supplied input patterns. The activations of the output units are determined in a single two-phase processing cycle. First, the net input to each output unit is computed. This is the sum of the activations of the input units times the corresponding weights, plus an optional bias term associated with the output unit:

$$net_i = \sum_j w_{ij} + bias_i . \qquad (19)$$

Activation Functions

After computing the net input to each output unit, the activation of the output unit is then determined according to an activation function. Several variants are available:

- *Linear.* Here the activation of output unit i is simply equal to the net input.
- *Linear threshold.* In this variant, each of the output units is a *linear threshold unit*; that is, its activation is set to 1 if its net input exceeds 0 and is set to 0 otherwise. Units of this kind were used by Rosenblatt in his work on the perceptron (1959).
- *Stochastic.* This is the activation function used in *PDP:18* and *PDP:19*. Here, the output is set to 1, with a probability p given by the logistic function:

  $$p(o_i{=}1) = \frac{1}{1+e^{-net_i/T}} \qquad (20)$$

  This is the same activation function used in Boltzmann machines.
- *Continuous sigmoid.* In this variant, each of the output units takes on an activation that is nonlinearly related to its input according to the logistic function:

  $$o_i = \frac{1}{1+e^{-net_i/T}} \qquad (21)$$