see how it can happen with a distributed system. (That connectionist models *can* perform one trial learning will be shown in chapter 13 which demonstrates a model of the role of the hippocampus in episodic memory formation.)

At the other extreme there are many areas of knowledge acquisition, such as learning to play tennis or learning to talk, where acquisition of new knowledge is gradual, and accompanied by the modification or loss of previous patterns. As your tennis serve improves or you learn to pronounce the language correctly you *want* to lose some aspects of your old response patterns because they were inaccurate. Later it is difficult to recall when a specific piece of information was added to the database. Such a pattern, where new information is inextricably interwoven with old, occurs naturally with a distributed system, but not with a localist one. Distinctions between different sorts of knowledge representation occur in many models of the cognitive system. Most of the connectionist learning algorithms we will look at are more appropriate for modelling the latter sort of acquisition and representation than the former.

3 *Pattern association*

This chapter will introduce the architecture and properties of one specific kind of network, a pattern associator, and the operation of one particular learning rule, the Hebb rule. During training a pattern associator is presented with pairs of patterns. If learning is successful then the network will subsequently recall one of the patterns at output when the other is presented at input. After training, a pattern associator can also respond to novel inputs, generalising from its experience with similar patterns. Pattern associators are tolerant of noisy input and resistant to internal damage. They are capable of extracting the central tendency or prototype from a set of similar examples.

The first two chapters introduced general principles which are shared by all connectionist networks. In this and the next four chapters we will look in detail at the structure and properties of a variety of specific network architectures: pattern associators, autoassociators, competitive nets and recurrent nets. Networks can be trained in a variety of ways. In this chapter we will look at one particular learning rule, the Hebb rule. In chapter 4 we will look at the Delta rule and in chapter 5 at backpropagation.

The examples and the networks in these early chapters have deliberately been kept very simple so that the principles involved in their operation can be seen at work. It may seem that the problems they solve are so trivial that they have little to do with human cognition. Don't worry. In chapters 8–14 we will look at networks which have been scaled up to the point where they can simulate realistic aspects of human behaviour. For example, in chapter 8 we will look at a model with roughly 1000 processing units and 200 000 connections which learns to pronounce all the monosyllabic words in the English language. In chapter 13 we will look at a model of episodic memory formation in the hippocampus with about 4000 processing units and about half a million connections. Exactly the same principles will be at work in these networks as in the examples which follow.

The architecture and operation of a pattern associator

A fundamental task for the nervous system is to discover the structure of the world by finding what is correlated with what. That is, to learn to associate one stimulus

with another. For example, one stimulus might be the taste of chocolate and the other its appearance. Initially there is no connection between these two for a child. After the association between the patterns of neural firing caused by the two stimuli has been made, the sight of chocolate can recall responses originally associated with its taste. If the taste of chocolate caused salivation, then the sight of chocolate, which was initially neutral, could produce the same response. This form of pattern association was made famous by Pavlov and his dogs in some of the earliest investigations in experimental psychology. But similar mechanisms underlie far more than the control of autonomic nervous system responses like salivation. Learning that the letter string YACHT is pronounced /y/ /o/ /t/, that Mary has blonde hair or how hard you have to push the steering wheel of a car when you want to go round a corner are all examples of pattern association.

A pattern association network

A simple network which can perform pattern association is shown in figure 3.1. The upper network is drawn as a stylised version of a real neural network with axons from two sets of neurons synapsing onto the dendrites of a third. The lower figure is drawn in the standard format for a connectionist network, with two sets of computational units joined by weighted connections. These two ways of representing a pattern association network may look quite different but they perform exactly the same task. They are simply notational variants. By showing them together, the way that two different conventions represent what a network might compute should become clear.

In the network in figure 3.1(a), the horizontal rectangular bars represent the dendrites of a set of neurons labelled r_{1-4}. During learning two patterns are presented to the network simultaneously. The first pattern is the output of a set of neurons which represents, say, the taste of chocolate. In figure 3.1(a) this pattern (P_1) is (1 1 0 0). It reaches the dendrites of neurons r_{1-4} via unmodifiable synapses (represented by the symbol —<), forcing the neurons r_{1-4} into the same pattern of firing (1 1 0 0). The second pattern, P_2 (1 0 1 0 1 0), is the output of a set of neurons which represents the sight of chocolate. This reaches r_{1-4} along the axons shown by the vertical lines running downwards across the dendrites. This input connects to the dendrites of neurons r_{1-4} through the modifiable synapses represented by the black blobs. The synapse between axon j and dendrite i is w_{ij}. Learning (i.e. forming the association between P_1 and P_2) takes place by modification of these synapses.

The process of pattern association in a single layer connectionist network (i.e. one with a single layer of modifiable connections between input and output) is shown in figure 3.1(b). P_2 (1 0 1 0 1 0) is presented to the input units. P_1 (1 1 0 0) is the pattern which the net has to learn to produce on the output units in response to the input of

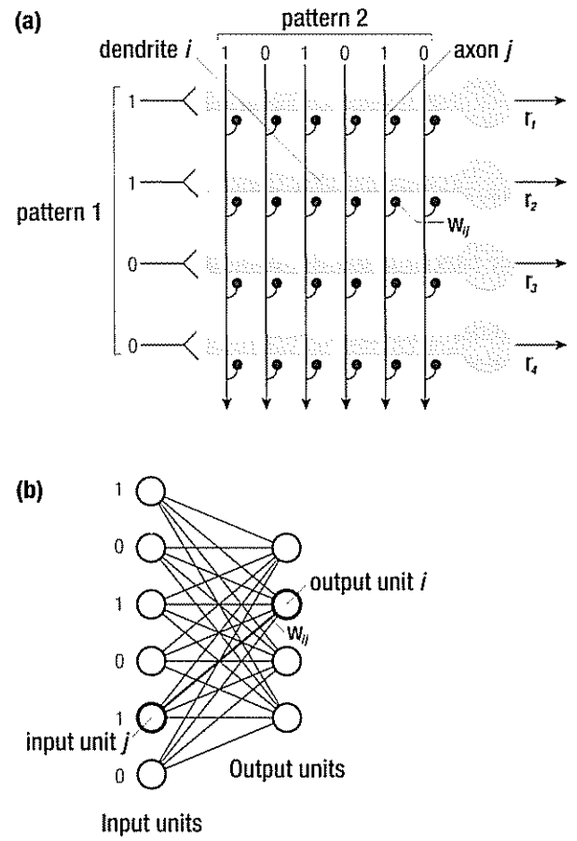

Figure 3.1 A network for performing simple pattern association. (a) The network is represented as it might appear in the nervous system with axons synapsing onto dendrites of cells r_1 to r_4. (b) The network is shown in the conventional connectionist format with an input presented to one set of units (indexed by j) producing an output on another set indexed by i.

P_2. The connection between input unit j and output unit i is represented by the line labelled w_{ij}. Pattern association takes place by modifying the strength of the connections between input units and output units.

The differences between these two ways of representing the same process should be carefully noted: First, the synapse at the junction of axon j and dendrite i in the upper figure has become a connecting line (shown in bold) between two processing units in the lower figure. Second, the way that pattern 1 reaches the network is explicit in the upper figure. In the lower figure it is a signal presented to the output units by an invisible 'teacher'. Despite these differences in presentation, the processes described are conceptually the same. In this chapter we will describe the process of learning with terms appropriate to figure 3.1(a)—axon, output neuron dendrite and

synaptic strength. In the next chapter we will use their equivalents in figure 3.1(b)—input line, output unit and connection weight.

The Hebb rule

During the learning phase the two patterns which are to be associated, P_1 and P_2, are presented to the network simultaneously. Hebbian learning can be implemented in the network in the upper part of figure 3.1 by applying the following rule: If there is activity on input axon j when neuron i is active then the strength of the connection, w_{ij}, between axon j and dendrite i is increased.[1] (Neuron i will be active if the ith axon in P_1 is On.) In the network in figure 3.1(b) Hebbian learning would be implemented, following the simultaneous presentation of P_2 at input and P_1 at output, by strengthening the connections between those units which are On in both the input pattern and the output pattern.

The Hebb rule for weight change (Δw_{ij}) can be expressed formally as:

$$\Delta w_{ij} = \varepsilon\, a_i\, a_j \tag{3.1}$$

where ε is a learning rate constant which specifies how much a synapse alters on any one pairing of P_1 and P_2,

a_i is the activity of element i in P_1,

and a_j is the activity of element j in P_2.

The Hebb rule is expressed in multiplicative form to reflect the idea that, for a synapse to increase in strength, both presynaptic activity (from P_2) and postsynaptic activity (from P_1) must be present.

Learning with the Hebb rule

The following simple examples will demonstrate the basic principles of the operation of a pattern associator. They will show how the weight changes brought about by applying Hebbian learning give rise to some surprising and biologically useful properties. For simplicity the neurons will be treated as binary with an activity of 0 or 1.

A four neuron pattern associator with six input axons carrying P_2 and four input axons carrying P_1 is represented in figure 3.2. The sight of chocolate is represented by P_2 (1 0 1 0 1 0) and the taste by P_1 (1 1 0 0). The 6×4 weight matrix in figure 3.2 corresponds directly to the 6×4 matrix of synapses in figure 3.1(a). The first row of weights in figure 3.2 shows the strength of the connections from axons 1 to 6 of P_2 to

[1] That such an operation could be the basis for useful learning in neural structures was first suggested by the neurophysiologist Donald Hebb, hence Hebbian learning. Hebb's contribution to connectionism will be described in chapter 15.

		P_2					
		1	0	1	0	1	0
		↓	↓	↓	↓	↓	↓
	1 →	0	0	0	0	0	0
	1 →	0	0	0	0	0	0
P_1	0 →	0	0	0	0	0	0
	0 →	0	0	0	0	0	0

Weight matrix

Figure 3.2 The weight matrix in the pattern associator before learning takes place.

output neuron 1. Their equivalents in the network in figure 3.1(b) are the connections from the six input units to output unit 1. The second row of weights in figure 3.2 shows the connections from the six input axons in P_2 to output neuron 2; these are represented in figure 3.1(b) by the connections to the second output unit from the six input units, and so on. The synaptic weights are initially all 0 because no learning has yet occurred.

After pairing P_1 and P_2 for one learning trial, the synaptic weights are incremented according to the Hebb rule expressed in equation 3.1. That is, every synapse which connects an active axon in P_2 to an active dendrite will be strengthened. So, for example, the 1st, 3rd and 5th weights in the first row are incremented because the first element in P_1 and the 1st, 3rd and 5th elements in P_2 are all active. But the weights in the third row are not incremented because the third element in P_1 is inactive. Similarily the weights in the second column are not incremented because the second element in P_2 is inactive. The overall result following a single learning trial is shown in figure 3.3 (if $\varepsilon = 1$).

		P_2					
		1	0	1	0	1	0
		↓	↓	↓	↓	↓	↓
	1 →	1	0	1	0	1	0
	1 →	1	0	1	0	1	0
P_1	0 →	0	0	0	0	0	0
	0 →	0	0	0	0	0	0

Weight matrix

Figure 3.3 Weight matrix after one learning trial pairing (1 1 0 0) and (1 0 1 0 1 0).

Recall from a Hebb trained matrix

When learning has taken place, the effectiveness of the memory created is tested by presenting a recall pattern (P_R) to the matrix, on its own, on the axons which originally carried P_2. The consequence should be the recall of the pattern P_1 at output. That is, the sight of chocolate should recall the taste of chocolate. The net input to output neuron i is:

$$\text{netinput}_i = \Sigma_j a_j w_{ij} \qquad (3.2)$$

Σ_j indicates that the sum is over all the input axons indexed by j (i.e. over all the axons in P_R).

To discover what happens during recall we present P_R to the matrix of weights which were acquired on the learning trial. The net input to the four output neurons is found by applying the operation defined in equation 3.2 to each neuron in turn. That is, the activity on each of the input lines to neuron i is multiplied by the weight of the corresponding synapse and these products are summed down the dendrite. So, if the recall cue (1 0 1 0 1 0) is presented to the weight matrix in figure 3.3, the net input to neuron 1 will be $(1\times1 + 0\times0 + 1\times1 + 0\times0 + 1\times1 + 0\times0) = 3$ (figure 3.4).

The resulting pattern of net input to $r_{1\to4}$ is (3 3 0 0). If the output neurons have a binary threshold activation function (see figure 1.4(d) in chapter 1) with a threshold of 2, and we set the activity level of the neuron to 1 if it exceeds the threshold and 0 if it does not, then the output activity pattern is (1 1 0 0). Thus the pattern associator has correctly recalled P_1 when P_2 is presented as a recall cue.

Learning different associations on the same weight matrix

Application of the Hebb rule strengthens every connection between an axon and a dendrite where both are active. At recall that connection is then used to activate the

	P_R						
	1	0	1	0	1	0	
	↓	↓	↓	↓	↓	↓	
	1	0	1	0	1	0	→ 3
	1	0	1	0	1	0	→ 3
	0	0	0	0	0	0	→ 0
	0	0	0	0	0	0	→ 0
	Weight matrix						netinput$_i$

Figure 3.4 Recall from the matrix after presentation of the pattern (1 0 1 0 1 0).

		P_2					
		1	1	0	0	0	1
		↓	↓	↓	↓	↓	↓
	0 →	0	0	0	0	0	0
	1 →	1	1	0	0	0	1
P_1	0 →	0	0	0	0	0	0
	1 →	1	1	0	0	0	1
		Weight matrix					

Figure 3.5 Weights acquired during one learning trial pairing (0 1 0 1) and (1 1 0 0 0 1).

dendrite when the axon in the recall cue is active. So it is hardly surprising that the pattern associator is capable of retrieving a pattern which was presented during the learning phase. But what happens if we try to store different associations on the same matrix? Can they still be recalled correctly or will they interfere?

Let us use the Hebb rule to store the association between the appearance and taste of apricots on the same connections. In this case P_1 is (0 1 0 1) and P_2 is (1 1 0 0 0 1). Equation 3.1 shows that the weights acquired by applying the Hebb rule during this learning trial will be as shown in figure 3.5.

Superimposing the weights acquired during learning to associate the taste and appearance of apricots on those acquired learning to associate the taste and appearance of chocolate (i.e. summing the weight matrices in figures 3.3 and 3.5) produces the combined weight matrix shown in figure 3.6.

The crucial test now is to see what happens when we present each P_2 in turn as a recall cue to the combined weight matrix. Presenting (1 1 0 0 0 1) produces the result (1 4 0 3) as shown in figure 3.7.

With the binary output thresholds set to 2 the activity pattern (0 1 0 1) is produced on the output neurons. So recall is correct. The appearance of apricots recalls the taste of apricots. If the pattern (1 0 1 0 1 0) is presented to the combined matrix the result will be as shown in figure 3.8.

With the threshold set at 2, the activity pattern (1 1 0 0) is produced. Thus the

1	0	1	0	1	0
2	1	1	0	1	1
0	0	0	0	0	0
1	1	0	0	0	1
Weight matrix					

Figure 3.6 The combined weight matrix after learning two different P_1, P_2 pairs.

			P_R				
1	1	0	0	0	1		
↓	↓	↓	↓	↓	↓		
1	0	1	0	1	0	→	1
2	1	1	0	1	1	→	4
0	0	0	0	0	0	→	0
1	1	0	0	0	1	→	3
			Weight matrix				netinput_i

Figure 3.7 The result of presenting the recall cue (1 1 0 0 0 1) to the combined weight matrix.

			P_R				
1	0	1	0	1	0		
↓	↓	↓	↓	↓	↓		
1	0	1	0	1	0	→	3
2	1	1	0	1	1	→	4
0	0	0	0	0	0	→	0
1	1	0	0	0	1	→	1
			Weight matrix				netinput_i

Figure 3.8 Recall from the combined matrix with the cue (1 0 1 0 1 0).

appearance of chocolate still recalls the taste of chocolate even though the same connections are now also storing the association of the appearance and taste of apricots.

This simple demonstration shows that accurate recall of different associations from a pattern associator trained with the Hebb rule is possible even when the associations are stored on the same connections. However, with only two associations some interference between the patterns is already apparent—the recalled patterns do not exactly match the original patterns. In this example the differences have been removed by the threshold setting of the activation function. But there will obviously be a limit to the number of associations which can be stored in such a matrix before interference between the patterns becomes a problem. The effect of increasing the number of memories in a more realistic pattern associator will be

explored in the section on the simulation of episodic memory formation in the hippocampus in chapter 13.

Recall reflects the similarity of retrieval pattern and stored patterns

Many operations within a connectionist network can be understood by treating the input patterns as vectors and the weights in the network as a matrix. To make connectionism accessible to the non-mathematical reader we have avoided explanations in terms of vectors. However, there is one crucial result, which explains retrieval in pattern associators (and also the operation of competitive networks described in chapter 6) that is most easily understood in terms of vector operations.[2] Before we can have the result we need the concept of the *dot product* (or *inner product*) of two vectors. A vector is an ordered set of numbers, such as [2 0 1 3]. The dot product of two vectors is found by taking the sum of the products of the numbers in equivalent positions in the two vectors. That is, the sum of the product of the first number in vector 1 and the first number in vector 2, the product of the second number in each vector and so on. If vector 1 was [2 0 1 3] and vector 2 was [1 1 0 4], the dot product would be $(2 \times 1 + 0 \times 1 + 1 \times 0 + 4 \times 3) = 14$.

The set of activities which constitute an input to a network is an ordered set of numbers. So they can be treated as the vector **p**. The activities are represented in equation 3.2 by a_j. An example is the set of activities [1 0 1 0 1 0] on the input lines representing the appearance of chocolate. The weights of the connections between the input and output neuron i, represented in equation 3.2 by w_{ij}, are another ordered set of numbers. So they can also be treated as a vector, $\mathbf{w}_i$. An example is the set of weights to output neuron 2 in the joint matrix in the previous example, [2 1 1 0 1 1]. Equation 3.2 shows that the net input to output neuron i is found by summing the product of the first activity in the input pattern and the first weight on the dendrite of output neuron i ($a_1 w_{i1}$), the second activity in the input pattern and the second weight ($a_2 w_{i2}$), and so on until the products of all pairs of activities and weights have been summed. So netinput_i is the dot product of the vector representing the input pattern and the vector representing the weights of the connections from each input line to output neuron i.

The crucial result is this: For any given set of numbers representing the activities in one vector and the weights in the other, the more similar the numbers in vector 1 are to the numbers *in the corresponding positions* in vector 2, the larger will be the dot

[2] A more detailed analysis is given in Appendix 2. What follows is a simplification intended to help the understanding of a general result.

product. An intuitive grasp of the result can be obtained by taking the dot products of a set of vectors with 12 binary bits (1 or 0), half on and half off. Imagine that p is an input pattern vector and w_1, w_2, w_3 and w_4 are the vectors representing the weights on output units 1–4; w_1 is identical to p; w_2, w_3 and w_4 have patterns which are increasingly less like p:

1 0 0 0 0 0 1 1 1 1 1 p
1 0 0 0 0 0 1 1 1 1 1 w_1
0 1 0 0 0 0 1 0 1 1 1 1 w_2
0 0 1 1 1 1 0 1 1 0 0 0 w_3
0 1 1 1 1 1 1 0 0 0 0 0 w_4

The dot product is taken by summing the product of corresponding elements in the two vectors. So where there is a 1 in the equivalent position on the two vectors, 1 will be added to the dot product; when there is a 1 in one vector and a 0 in the equivalent position in the other, nothing will be added. So the maximum dot product will be obtained when the two vectors are identical (p and w_1). It will have a value of 6, each positive element contributing 1 to the dot product. Every dissimilarity between the vectors reduces the dot product.[3] The dot product of p and w_2, which are quite similar, is reduced to 4 because there are four bits in pattern w_2 which do not match those in p. The dot product between p and w_3, which have little in common, is reduced to 2. p and w_4 have no elements in common so their dot product is 0. p and w_4 are said to be *orthogonal*.

During recall each output neuron in a pattern associator computes the dot product between the input pattern and its weight vector. What this does is to compute how *similar* the input pattern vector p is to the weight vector w_i which is stored on its dendrite. If this similarity is great, netinput_i will be large and the neuron will be turned On. If they are dissimilar, netinput_i will be small and the neuron will remain Off. Hebbian learning increments weights between elements of P_1 and P_2 which are both On. Over a series of learning trials therefore, a cumulative record of the correlation between individual elements in the different pairs of patterns builds up in the weight matrix. Thus the operation of a pattern associator trained by the Hebb rule can be summed up in the following way:

Learning—if neuron *i* is activated by P_1, an increment Δw_i, that has the same pattern as P_2, is added to the weight vector of neuron *i*.

Recall—since the patterns presented during learning determine the weight vector on neuron *i*, the output of neuron *i* at recall reflects the similarity of the recall cue to the patterns presented during learning.

[3]The dot product is not the only way to calculate the similarity of two vectors. For example, zero elements in corresponding positions will contribute to the similarity of two vectors but not to their dot product.

Properties of pattern associators

Generalisation

During recall, pattern associators generalise. That is, if a recall cue is similar to a pattern that has been learnt already, a pattern associator will produce a similar response to the new pattern as it would to the old. This occurs because the recall operation on each neuron involves computing the dot product of the recall vector p_R with the synaptic weight vector w_i. The result, netinput_i, thus reflects the similarity of the current input to the previously learnt input patterns. Small differences in netinput_i for similar input patterns can be removed by a threshold at output. In the world faced by real biological systems, recall cues are rarely identical to the patterns experienced during learning. So a mechanism which automatically generalises across slight differences in input pattern has obvious adaptive value.

Generalisation can be illustrated with the simple binary pattern associator we have considered already. What happens when it is presented with the recall cue (1 1 0 1 0 0) which is similar, but not identical, to the pattern (1 1 0 0 0 1) which it learnt before?

With a threshold set at 2, the output pattern (0 1 0 1) would be recalled in response to the cue (1 1 0 1 0 0) (figure 3.9) just as it was to the cue (1 1 0 0 0 1). The network has treated the new pattern as a noisy version of one it already knew, and reproduced the response it learnt to that. The sight of a slightly bruised apricot still recalls the taste of an apricot. Generalisation has occurred.

Fault tolerance

Even if some of the synapses on neuron *i* are damaged after learning, netinput_i following the presentation of a recall cue may still be a good approximation to the

			P_R				
1	1	0	1	0	0		
↓	↓	↓	↓	↓	↓		
1	0	1	0	1	0	→	1
2	1	1	0	1	1	→	3
0	0	0	0	0	0	→	0
1	1	0	0	0	1	→	2
			Weight matrix				netinput_i

Figure 3.9 Recall from the combined matrix when the recall cue (1 1 0 1 0 0) is presented.

			P_R				
1	1	0	0	0	1		
↓	↓	↓	↓	↓	↓		
1	0	1	0	1	0	→	1
2	1	1	0	×	1	→	4
0	0	0	0	0	0	→	0
1	×	0	0	0	1	→	2
			Weight matrix				netinput_i

Figure 3.10 Recall from a damaged matrix after presentation of the cue (1 1 0 0 0 1).

correct value. This is because netinput_i represents the *correlation* of $\mathbf{p}_R$ with $\mathbf{w}_i$. Provided the pattern carrying the recall cue consists of a reasonably large number of axons the correlation will not be greatly affected by a few missing items. After passing through the binary threshold activation function, the result may well be correct recall. The same result is achieved if some of the input axons carrying the recall cue are lost or damaged. Since real nervous systems are continually losing cells, fault tolerance is of great adaptive value.

We can illustrate this with the example shown in figure 3.10; x indicates a damaged synapse which now has no effect. Whatever information was stored in that synapse during learning has been lost. Presentation of the recall cue (1 1 0 0 0 1) to this matrix produces the net input pattern (1 4 0 2).

With the binary output threshold set to 2 this would produce the output pattern (0 1 0 1), the same as that produced by the complete matrix. The small difference between the net input to the output neurons produced by the damaged matrix and the complete one (shown in figure 3.7) illustrates graceful degradation. Minor damage will cause a small change in response to many inputs, rather than a total loss of some memories and no effect on others.

The importance of distributed representations for pattern associators

A distributed representation is one in which the activity of all the elements in the pattern is used to encode a particular stimulus. Comparing (1 0 1 0 1 0), (1 1 0 0 0 1) and the other P_2s which could be represented by a pattern of six 1s and 0s, we need to know the state of most of the elements to know which stimulus is being represented. No one element can be used to identify the stimulus. Since the information about which stimulus is present is distributed over the population of elements, this is called a distributed representation.

A *local representation* is one in which all the information about a particular stimulus is provided by the activity of one element in the pattern. One stimulus might be represented by the pattern (1 0 0 0 0 0), another by the pattern (0 1 0 0 0 0) and so on. The activity of element 1 would indicate that stimulus 1 was present, and of element 2 that stimulus 2 was present. If a particular element is active, we know that the stimulus represented by that element is present. If elements are taken to be equivalent to neurons, such coding is said to involve 'grandmother cells' because, in an extreme expression of localist coding, one neuron might represent a stimulus in the environment as complex and specific as one's grandmother. Where the activity of a number of cells must be taken into account in order to represent a stimulus the representation is described as using ensemble or distributed encoding.

The properties of generalisation and graceful degradation are only achieved if the representations are distributed. The recall operation involves computing the dot product of the input pattern vector $\mathbf{p}_R$ with the weight vector $\mathbf{w}_i$. This only allows netinput_i to reflect the *similarity* of the current input pattern to a previously learned input pattern if many elements of $\mathbf{p}_R$ are active. If local encoding were used [(1 0 0 0 0 0), (0 1 0 0 0 0) etc.] then a new $\mathbf{p}_R$ pattern would produce a dot product of either 1 or 0. Generalization and graceful degradation rely on a continuous range of dot products. They are dependent on distributed representations, for then the dot product can reflect *similarity* even when some elements of the vectors involved are altered.

Prototype extraction and noise removal

If a set of similar P_2s is paired with the same P_1 (e.g. a set of apricots which all look a little different are all associated with the taste of apricot) the weight vector becomes, with scaling, the average of the set of vectors appropriate for the individual P_2s. When tested at recall, the output of the memory is then best (i.e. the dot product is highest) to the *average* input pattern vector. If the average is thought of as a prototype, and the individual P_2s noisy exemplars of it, then even though the prototype vector itself may never have been seen, the strongest output of the network is to the prototype. Recognition of a prototype which has never been seen is a feature of human memory performance which will be explored in greater detail in chapter 4. This phenomenon is an automatic consequence of learning in a pattern associator with distributed representations.

Speed

Recall is very fast in a real neuronal network. The input firings which represent the recall cue P_R are applied simultaneously to the synapses, so netinput_i can be

accumulated in one or two time constants of the dendrite (e.g. 10–20 ms). If the threshold of the cell is exceeded, it fires. Thus in no more than a few tens of milliseconds all the output neurons of the pattern associator which will be turned On by a particular input pattern will be firing. The time taken to switch the output neurons On will be largely independent of the number of axons or dendrites in the pattern associator. This is very different from a conventional digital computer. Computing net input in equation 3.2 with one of these would involve successive multiplication and addition operations for each weight in the matrix. The time to compute the output pattern would increase in proportion to the product of the number of axons and the number of dendrites.

The pattern associator performs parallel computation in two senses. One is that for a single neuron, the separate contributions of the activity of each axon multiplied by the synaptic weight are computed in parallel and added simultaneously. The second is that this is performed in parallel for all neurons in the network. These types of neuronal network operate fast in the brain because they perform parallel processing.

Learning is also potentially fast in a pattern associator. A single pairing of P_1 and P_2 could, in principle, enable the association to be learnt. There is no need to repeat the pairing over many trials in order to discover the appropriate mapping. This is important for biological systems. A single co-occurrence of two events may provide the only opportunity to learn something which could have life-saving consequences. Although repeated pairing with small variations of the vectors produces the useful properties of prototype extraction and noise reduction, the properties of generalization and graceful degradation can be obtained with just one pairing. We will see one trial learning at work in the simulation of episodic memory formation in the hippocampus in chapter 13.

Interference is not necessarily a bad thing

In the elementary examples in this chapter we have demonstrated that independent events *can* be stored on the same connections of a distributed memory. However, it is obvious that there will be limitations on the number that can be stored without interference. Interference sounds like an undesirable property of a memory system. In the localised storage systems like computer discs and telephone directories which we are accustomed to, it *is* undesirable. So 'limitations to interference free storage' may sound like a problem. But remember that interference between responses to different but similar input patterns is the basis of many important properties of distributed memories. In this chapter we saw that it allowed generalisation, noise reduction and prototype extraction. In later chapters we will see how it enables a network to perform the generalisations which underlie many cognitive processes. The fact that interference is a property of connectionist pattern associator memories is of interest, for, unlike strictly localised forms of storage, interference is a major property of human memory. One reason that interference is tolerated in biological memory is presumably that the ability to generalize between stimuli is more useful than 100% accurate memory of specific past events.

Further reading

More detailed analysis of pattern associators and the Hebb rule can be found in Rolls and Treves (1997) and Hertz *et al.* (1991). Specific issues which go beyond the introductory account here are the types of pattern which can be easily learnt by one-layer pattern associators, the capacity of pattern associators trained with the Hebb rule and modifications to the Hebb rule which allow for decrease in synaptic weights as well as for increase.

Training a pattern associator with **tlearn**

Open the project called **pa** in the **Chapter Three** folder/directory. This project uses the **tlearn** network configuration file **pa.cf** that defines a network with 6 input units and 4 output units identical to that shown in figure 3.1(b). All the connections are set to zero. When you open the **pa** project, you will also open the **pa.data** and **pa.teach** files that will be used to train the network. These files are shown in figure 3.11. The two distributed data patterns defined in the **pa.data** file are those used to represent the appearance of chocolate and apricot earlier in the chapter. Similarly the two distributed teach patterns defined in the **pa.teach** file are those used to represent the taste of chocolate and apricot. You will use these files to train the network defined by the **pa.cf** file. You can confirm that you have the right network architecture by opening the Network Architecture display.

Training the network

In this exercise, you will train the network to associate the two sets of patterns described above. First, select Training Options... from the Network menu. The Training Options... dialogue box should appear as in figure 3.12. Set the number of training sweeps to 2 (we will train the network just once on each input pattern) and set the learning rate parameter to 10.0. If the other options in the dialogue box are not as shown, change them so they are. Now Train the network from the Network menu. The **tlearn** Status display will indicate when training is complete.

Testing the network

There are several ways to test whether the network has learnt the correct output patterns. First, open the Node Activities display as shown in figure 3.13. The top row