

potential disadvantage in using neurons with synaptic weights which can take on only positive values. This difficulty arises in the following way. Consider a set of positive normalized input firing rates and synaptic weight vectors (in which each element of the vector can take on any value between 0.0 and 1.0). Such vectors of random values will on average be more highly aligned with the direction of the central vector $(1, 1, 1, \dots, 1)$ than with any other vector. An example can be given for the particular case of vectors evenly distributed on the positive 'quadrant' of a high-dimensional hypersphere: the average overlap (i.e. dot product) between two random vectors (e.g. a random pattern vector and a random dendritic weight vector) will be approximately 0.637 while the average overlap between a random vector and the central vector will be approximately 0.798. A consequence of this will be that if a neuron begins to learn towards several input pattern vectors it will get drawn towards the average of these input patterns which will be closer to the $1, 1, 1, \dots, 1$ direction than to any one of the patterns. As a dendritic weight vector moves towards the central vector, it will become more closely aligned with more and more input patterns so that it is more rapidly drawn towards the central vector. The end result is that in large nets of this type, many of the dendritic weight vectors will point towards the central vector. This effect is not seen so much in small systems since the fluctuations in the magnitude of the overlaps are sufficiently large that in most cases a dendritic weight vector will have an input pattern very close to it and thus will not learn towards the centre. In large systems the fluctuations in the overlaps between random vectors become smaller by a factor of $1/\sqrt{N}$ so that the dendrites will not be particularly close to any of the input patterns.

One solution to this problem is to allow the elements of the synaptic weight vectors to take negative as well as positive values. This could be implemented in the brain by feedforward inhibition. A set of vectors taken with random values will then have a reduced mean correlation between any pair, and the competitive net will be able to categorize them effectively. A system with synaptic weights which can be negative as well as positive is not physiologically plausible, but we can instead imagine a system with weights lying on a hypersphere in the positive quadrant of space but with additional inhibition which results in the cumulative effects of some input lines being effectively negative. This can be achieved in a network by using positive input vectors, positive weight vectors, and thresholding the output neurons at their mean activation. A large competitive network of this general nature does categorize well, and has been described more fully elsewhere (Bennett, 1990). In a large network with inhibitory feedback neurons, and principal cells with thresholds, the network could achieve at least in part an approximation to this type of thresholding useful in large competitive networks.

A second way in which nets with positive-only values of the elements could operate is by making the input vectors sparse and initializing the weight vectors to be sparse, or to have a reduced contact probability. (A measure of sparseness is defined in Eq. 1.4.) For relatively small net sizes simulated ($N = 100$) with patterns with a sparseness α of, for example, 0.1 or 0.2, learning onto the average vector can be avoided. However, as the net size increases, the sparseness required does become very low. In large nets, a greatly reduced contact probability between neurons (many synapses kept identically zero) would prevent learning of the average vector, thus allowing categorization to occur. Reduced contact probability will, however, prevent complete alignment of synapses with patterns, so that the performance of the network will be affected.

5 Error-correcting networks: perceptrons, the delta rule, backpropagation of error in multilayer networks, and reinforcement learning algorithms

The networks described in this chapter are capable of mapping a set of inputs to a set of required outputs using correction when errors are made. Although some of the networks are very powerful in the types of mapping they can perform, the power is obtained at the cost of learning algorithms which do not use local learning rules. A local learning rule specifies that synaptic strengths should be altered on the basis of information available locally at the synapse, for example the activity of the presynaptic and the postsynaptic neurons. Because the networks described here do not use local learning rules, their biological plausibility remains at present uncertain, although it has been suggested that perceptron learning may be implemented by the special neuronal network architecture of the cerebellum (see Chapter 9). One of the aims of future research must be to determine whether comparably difficult problems to those solved by the networks described in this chapter can be solved by biologically plausible neuronal networks.

5.1 Perceptrons and one-layer error-correcting networks

Under this heading, we describe one-layer networks taught by an error-correction algorithm. The term perceptron refers strictly to networks with binary threshold activation functions. The outputs might take the values only 1 or 0 for example. The term perceptron arose from networks designed originally to solve perceptual problems (Rosenblatt, 1961; Minsky and Papert, 1969), and these networks are referred to briefly below. If the output neurons have continuous-valued firing rates, then a more general error-correcting rule called the delta rule is used, and is introduced in this chapter. For such networks, the activation function may be linear, or it may be non-linear but monotonically increasing, without a sharp threshold, as in the sigmoid activation function (see Fig. 1.3).

5.1.1 Architecture and general description

The one-layer error-correcting network has a set of inputs which it is desired to map or classify into a set of outputs (see Fig. 5.1). During learning, an input pattern is selected, and produces output firing by activating the output neurons through modifiable synapses, which

then fire as a function of their typically non-linear activation function. The output of each neuron is then compared to a target output for that neuron given that input pattern, an error between the actual output and the desired output is determined, and the synaptic weights on that neuron are then adjusted to minimize the error. This process is then repeated for all patterns until the average error across patterns has reached a minimum. A one-layer error-correcting network can thus produce output firing for each pattern in a way that has similarities to a pattern associator. It can perform more powerful mappings than a pattern associator, but requires an error to be computed for each neuron, and for that error to affect the synaptic strength in a way that is not altogether local. A more detailed description follows.

These one-layer networks have a target for each output neuron (for each input pattern). They are thus an example of a supervised network. With the one-layer networks taught with the delta rule or perceptron learning rule described next, there is a separate teacher for each output neuron.

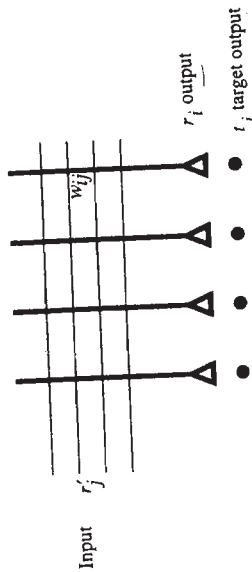


Fig. 5.1 One-layer perceptron.

5.1.2 Generic algorithm (for a one-layer network taught by error correction)

For each input pattern and desired target output:

1. Apply an input pattern to produce input firing r_j , and obtain the output activation of each neuron in the standard way by computing the dot product of the input pattern and the synaptic weight vector. The synaptic weight vector can be initially zero, or (small) random values.

$$h_i = \sum_j r_j w_{ij} \tag{5.1}$$

- where the sum is over the C input axons, r_j .
2. Apply an activation function to produce the output firing r_i .

$$r_i = f(h_i)$$

This activation function f may be sigmoid, linear, binary threshold, linear threshold, etc.

If the activation function is non-linear, this helps to classify the inputs into distinct output patterns, but a linear activation function may be used if an optimal linear mapping is desired (see Adaline and Madaline, below).

3. Calculate the difference for each cell i between the target output t_i and the actual output r_i produced by the input, which is the error Δ_i

$$\Delta_i = t_i - r_i$$

4. Apply the following learning rule, which corrects the (continuously variable) weights according to the error and the input firing r_j

$$\delta w_{ij} = k(t_i - r_i)r_j' \tag{5.2}$$

where k is a constant which determines the learning rate. This is often called the delta rule, the Widrow-Hoff rule, or the LMS (least mean squares) rule (see below).

5. Repeat steps 1-4 for all input pattern/output target pairs until the root mean square error becomes zero or reaches a minimum.

5.1.3 Variations on the single-layer error-correcting network

5.1.3.1 Rosenblatt's (1961) perceptron

The perceptron was developed by Rosenblatt (1961), and its properties and constraints were extensively analysed by Minsky and Papert (1969) (see also Nilsson, 1965). The original perceptron was conceived as a model of the eye, and consists (Fig. 5.2) of sensory cells, which connect to association or feature detector cells, which in turn connect by modifiable weights to the response cells in the output layer. The part which corresponds to the delta-rule perceptron described above is the association cell-to-response cell mapping through the modifiable weights. The association cell layer can be thought of as a preprocessor.

Each sensory cell receives as input stimulus either 1 or 0. This excitation is passed on to the association cells (or feature detector cells as in Fig. 5.2) with either a 1 or -1 multiplying factor (equivalent to a synaptic weight). If the input to the association cell exceeds 0, the cell fires and outputs 1; if not, it outputs 0. The association cell layer output is passed on to the response cells in the output layer, through modifiable weights w_{ij} , which can take any value, positive or negative. (Association cell A_j connects to response cell R_i through weight w_{ij} .) Each response cell sums its total input and if it exceeds a threshold, the response cell outputs a 1; if not, it outputs 0. Sensory input patterns are in class 1 for response cell R_i if they cause the response cell to fire, in class 0 if they do not.

The algorithm given for training the original perceptron as a pattern classifier was as follows. (The delta rule given above is a general formulation of what the following training algorithm achieves.) The perceptron is given a set of input patterns, and for each input pattern the output of the perceptron is compared with what is desired, and the weights are adjusted according to a rule of the following type (formulated for response cell R_i):

1. If a pattern is incorrectly classified (by cell R_i) in class 0 when it should be in class 1, increase by a constant all the weights (to cell R_i) from the association cells that are active.
2. If a pattern is incorrectly classified in class 1 when it should be in class 0, decrease by a constant all the weights coming from association cells that are active.
3. If a pattern is correctly classified, do not change any weights.

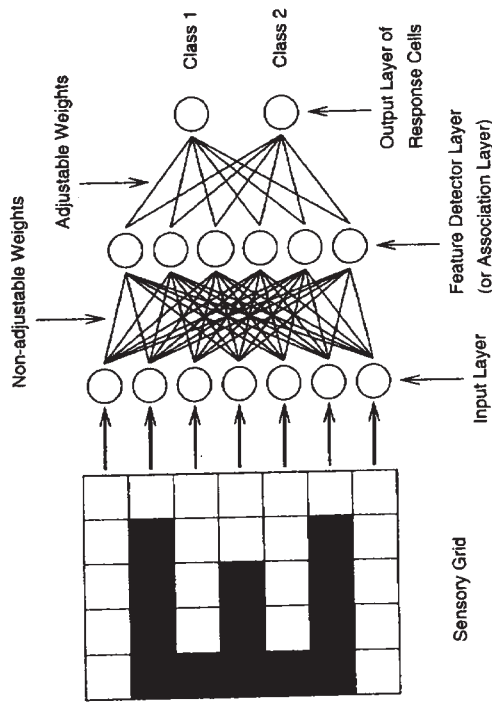


Fig. 5.2 A schematic diagram of Rosenblatt's conceptualization of a perceptron applied to vision. The sensory stimulus activates an input layer which is connected to a feature detector (or association) layer formed from combinations of the inputs. The feature detector layer is connected to the output layer (which in this case has two neurons, which indicate whether the input stimulus is in Class 1 or Class 2) by a layer of adjustable weights trained by the perceptron learning rule.

Operating under these rules, the perceptron will gradually make fewer and fewer wrong classifications and (under certain restricted conditions—see Minsky and Papert, 1969) will classify every pattern in the set correctly. In this scheme, each response cell must be told how to adjust its weights for each input pattern. The learning scheme works with single and with multiple output cells (see Fig. 5.2, in which there are two output cells), and can learn to produce different output response patterns to different input stimulus patterns. Each cell needs its own teacher to supervise the learning of that cell.

5.1.3.2 Adaline

The adaline (adaptive linear element) of Widrow and Hoff (1960) used binary (1, -1) inputs and outputs, and a binary threshold output activation function. The learning rule was expressed in the more general form of Eq. 5.2, and this rule is thus often called the Widrow-Hoff rule.

5.1.3.3 Adaptive filter

The adaptive filter of Widrow *et al.* (1976) (see Widrow and Lehr, 1990) used continuous inputs and outputs, and a linear activation function. Its rule is equivalent to Eq. 5.2 above,

$$\delta w_{ij} = k (t_i - h_i) r'_j$$

This approach was developed in engineering, and enables an optimal linear filter to be set up by learning. With multiple output cells, the filter was known as a Madaline.

In general, networks taught by the delta rule may have linear, binary threshold, or non-linear but monotonically increasing (e.g. sigmoid) activation functions, and may be taught with binary or continuous input patterns. The properties of these variations are made clear next.

5.1.3.4 Capability and limitations of single-layer error-correcting networks

Perceptrons perform pattern classification. That is, each neuron classifies the input patterns it receives into classes determined by the teacher. This is thus an example of a supervised network, with a separate teacher for each output neuron. The classification is most clearly understood if the output neurons are binary, or are strongly non-linear, but the network will still try to obtain an optimal mapping with linear or near-linear output neurons.

When each neuron operates as a binary classifier, we can consider how many input patterns p can be classified by each neuron, and the classes of pattern that can be correctly classified. The result is that the maximum number of patterns that can be correctly classified by a neuron with C inputs is

$$p_{\max} = 2C$$

when the inputs have random continuous-valued inputs, but the patterns must be linearly separable (see Hertz *et al.*, 1991). The linear separability requirement can be made clear by considering a geometric interpretation of the logical AND problem, which is linearly separable, and the XOR problem, which is not linearly separable. The truth tables for the AND and XOR functions are shown first (there are two inputs, r_1 and r_2 , and one output):

Inputs		Output	
r_1	r_2	AND	XOR
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0

For the AND function, we can plot the mapping required in a 2D graph as shown in Fig. 5.3. A line can be drawn to separate the input coordinates for which 0 is required as the output from those for which 1 is required as the output. The problem is thus linearly

separable. A neuron with two inputs can set its weights to values which draw this line through this space, and such a one-layer network can thus solve the AND function.

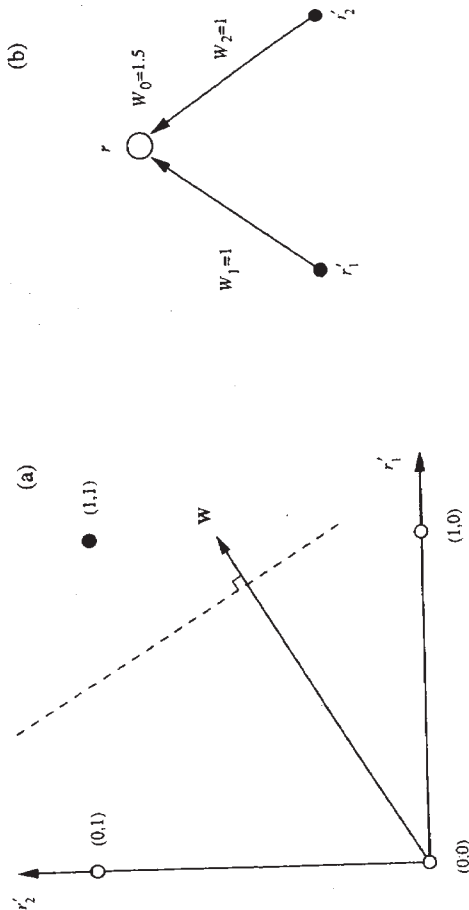


Fig. 5.3 (a) The AND function shown in a 2D space. Input values for the two neurons are shown along the two axes of the space. The outputs required are plotted at the coordinates where the inputs intersect, and the values of the output required are shown as an open circle for 0, and a filled circle for 1. The AND function is linearly separable, in that a line can be drawn in the space which separates the coordinates for which 0 output is required from those from which a 1 output is required. w shows the direction of the weight vector. (b) A one-layer neural network can set its two weights w_1 and w_2 to values which allow the output neuron to be activated only if both inputs are present. In this diagram, w_0 is used to set a threshold for the neuron, and its connected to an input with value 1. The neuron thus fires only if the threshold of 1.5 is exceeded, which happens only if both inputs to the neuron are 1.

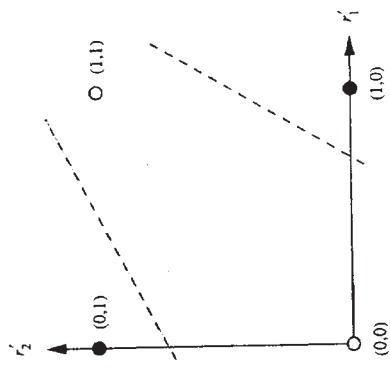


Fig. 5.4 The XOR function shown in a 2D space. Input values for the two neurons are shown along the two axes of the space. The outputs required are plotted at the coordinates where the inputs intersect, and the values of the output required are shown as an open circle for 0, and a filled circle for 1. The XOR function is not linearly separable, in that a line cannot be drawn in the space to separate the coordinates for those from which a 0 output is required from those from which a 1 output is required. A one-layer neural network cannot set its two weights to values which allow the output neuron to be activated appropriately for the XOR function.

For the XOR function, we can plot the mapping required in a 2D graph as shown in Fig. 5.4. No straight line can be drawn to separate the input coordinates for which 0 is required as the output from those for which 1 is required as the output. The problem is thus not linearly separable. For a one-layer network, no set of weights can be found that will perform the XOR, or any other non-linearly separable function.

Although the inability of one-layer networks with binary neurons to solve non-linearly separable problems is a limitation, it is not in practice a major limitation on the processing which can be performed in a neural network for a number of reasons. First, if the inputs can take continuous values, then if the patterns are drawn from a random distribution, the one-layer network can map up to $2C$ of them. Second, as described for pattern associators, the perceptron could be preceded by an expansion recoding network such as a competitive network with more output than input neurons. This effectively provides a two-layer network for solving the problem, and (non-linear) multilayer networks are in general capable of solving arbitrary mapping problems. Ways in which such multilayer networks might be trained are discussed later in this chapter.

We now return to the issue of the capacity of one-layer perceptrons, that is how many patterns p can be correctly mapped to correct binary outputs if the input patterns are linearly separable.

Output neurons with continuous values, random patterns

Before treating this case we note that if the inputs are orthogonal, then just as in the pattern associator, C patterns can be correctly classified, where there are C inputs, r'_j ($j = 1, C$), per neuron. The argument is the same as for a pattern associator.

We consider next the capacity of a one-layer error-correcting network which learns patterns drawn from a random distribution. For neurons with continuous output values, whether the activation function is linear or not, the capacity (for fully distributed inputs) is set by the criterion that the set of input patterns must be linearly independent (see Hertz *et al.*, 1991). (Three patterns are linearly independent if any one cannot be formed by addition (with scaling allowed) of the other two patterns—see Appendix A1.) Given that there can be a maximum of C linearly independent patterns in a C -dimensional space (see Appendix A1), the capacity of the perceptron with such patterns is C patterns. If we choose p random patterns with continuous values, then they will be linearly independent for $p \ll C$ (except for cases with very low probability when the randomly chosen values may not produce linearly independent patterns). (With random continuous values for the input patterns, it is very unlikely that the addition of any two, with scaling allowed, will produce a third pattern in the set.) Thus with continuous valued input patterns,

$$P_{max} = C.$$

If the inputs are not linearly independent, networks trained with the delta rule produce a least mean squares (LMS) error (optimal) solution (see below).

Output neurons with binary threshold activation functions

Let us consider here strictly defined perceptrons, that is networks with (binary) threshold output neurons, and taught by the perceptron learning procedure described above.

Capacity with fully distributed output patterns

The condition here for correct classification is that described above for the AND and XOR functions, that the patterns must be linearly separable. If we consider random continuous-valued inputs, then the capacity is

$$p_{\max} = 2C$$

(see Cover, 1965; Hertz *et al.*, 1991; this capacity is the case with C large, and the number of output neurons small). The interesting point to note here is that, even with fully distributed inputs, a perceptron is capable of learning more (fully distributed) patterns than there are inputs per neuron. This formula is in general valid for large C , but happens to hold also for the AND function illustrated above.

Sparse encoding of the patterns

If the output patterns r are sparse (but still distributed), then just as with the pattern associator, it is possible to map many more than C patterns to correct outputs. Indeed, the number of different patterns or prototypes p that can be stored is

$$p \approx C/a$$

where a is the sparseness of the target pattern t . p can in this situation be much larger than C (cf. Rolls and Treves, 1990, and Appendix A3).

Perceptron convergence theorem

It can be proved that such networks will learn the desired mapping in a finite number of steps (Block, 1962; Minsky and Papert, 1969; see Hertz *et al.*, 1991). (This of course depends on there being such a mapping, the condition for this being that the input patterns are linearly separable.) This is important, for it shows that single-layer networks can be proved to be capable of solving certain classes of problem.

As a matter of history, Minsky and Papert (1969) went on to emphasize the point that no one-layer network can correctly classify non-linearly separable patterns. Although it was clear that multilayer networks can solve such mapping problems, Minsky and Papert were pessimistic that an algorithm for training such a multilayer network would be found. Their emphasis that neural networks might not be able to solve general problems in computation, such as computing the XOR, which is a non-linearly separable mapping, resulted in a decline in research activity in neural networks. In retrospect, this was unfortunate, for humans are rather poor at solving parity problems such as the XOR (Thorpe *et al.*, 1989), yet can perform many other useful neural network operations very fast. Algorithms for training multilayer perceptrons were gradually discovered by a number of different investigators, and became widely known after the publication of the algorithm described by Rumelhart, Hinton and Williams (1986a,b). Even before this, interest in neural network pattern associators, autoassociators and competitive networks was developing (see Hinton and Anderson, 1981; Kohonen, 1977, 1988), but the acceptance of the algorithm for training multilayer perceptrons led to a great rise in interest in neural networks, partly for use in connectionist

models of cognitive function (McClelland and Rumelhart, 1986; McLeod, Plunkett and Rolls, 1998), and partly for use in applications (see Bishop, 1995).

In that perceptrons can correctly classify patterns provided only that they are linearly separable, but pattern associators are more restricted (see Chapter 2), perceptrons are more powerful learning devices than Hebbian pattern associators.

Gradient descent for neurons with continuous-valued outputs

We now consider networks trained by the delta (error correction) rule (Eq. 5.2), and having continuous valued outputs. The output activation function may be linear or non-linear, but provided that it is differentiable (in practice, does not include a sharp threshold), the network can be thought of as gradually decreasing the error on every learning trial, that is as performing some type of gradient descent down a continuous error function. The concept of gradient descent arises from defining an error ϵ for a neuron as

$$\epsilon = \sum_{\mu} (t^{\mu} - r^{\mu})^2 \quad (5.3)$$

where μ indexes the patterns learned by the neuron. The error function for a neuron in the direction of a particular weight would have the form shown in Fig. 5.5. The delta rule can be conceptualized as performing gradient descent of this error function, in that for the j th synaptic weight on the neuron

$$\delta w_j = -k \delta \epsilon / \partial w_j \quad (5.4)$$

where $\delta \epsilon / \partial w_j$ is just the slope of the error curve in the direction of w_j in Fig. 5.5. This will decrease the weight if the slope is positive and increase the weight if the slope is negative. Given (5.3), and recalling that $h = \sum_j r_j w_j$, Eq. 5.4 becomes

$$\delta w_j = -k \partial / \partial w_j \sum_{\mu} [(t^{\mu} - f(h^{\mu}))^2] = 2k \sum_{\mu} [(t^{\mu} - r^{\mu})] f'(h) r_j \quad (5.5)$$

where $f'(h)$ is the derivative of the activation function. Provided that the activation function is monotonically increasing, its derivative will be positive, and the sign of the weight change will only depend on the mean sign of the error. Equation 5.5 thus shows one way in which, from a gradient descent conceptualization, Eq. 5.2 can be derived.

With linear output neurons, this gradient descent is proved to reach the correct mapping (see Hertz *et al.*, 1991). (As with all single-layer networks with continuous-valued output neurons, a perfect solution is only found if the input patterns are linearly independent. If they are not, an optimal mapping is achieved, in which the sum of the squares of the errors is a minimum.) With non-linear output neurons (for example with a sigmoid activation function), the error surface may have local minima, and is not guaranteed to reach the optimal solution, although typically a near-optimal solution is achieved. Part of the power of this gradient descent conceptualization is that it can be applied to multilayer networks with neurons with non-linear but differentiable activation functions, for example with sigmoid activation functions (see Hertz *et al.*, 1991).

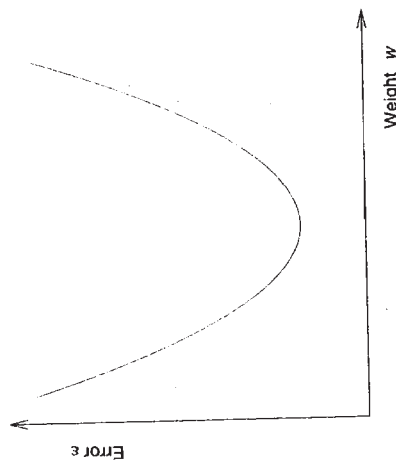


Fig. 5.5 The error (e) function for a neuron in the direction of a particular weight w .

5.1.4 Properties

The properties of single-layer networks trained with a delta rule (and of perceptrons) are similar to those of pattern associators trained with a Hebbian rule in many respects (see Chapter 2). In particular, the properties of generalization and graceful degradation are similar, provided that (for both types of network) distributed representations are used. The main differences are in the types of pattern that can be separated correctly, the learning speed (in that delta-rule networks can take advantage of many training trials to learn to separate patterns that could not be learned by Hebbian pattern associators), and in that the delta-rule network needs an error term to be supplied for each neuron, whereas an error term does not have to be supplied for a pattern associator, just an unconditioned or forcing stimulus. Given these overall similarities and differences, the properties of one-layer delta-rule networks are considered here briefly.

5.1.4.1 Generalization

During recall, delta-rule one-layer networks with non-linear output neurons produce appropriate outputs if a recall cue vector r_i^* is similar to a vector that has been learned already. This occurs because the recall operation involves computing the dot (inner) product of the input pattern vector r_i^* with the synaptic weight vector w_i , so that the firing produced, r_i , reflects the similarity of the current input to the previously learned input pattern r_i^* . Distributed representations are needed for this property. If two patterns that a delta-rule network has learned to separate are very similar, then the weights of the network will have been adjusted to force the different outputs to occur correctly. At the same time, this will mean that the way in which the network generalizes in the space between these two vectors will be very sharply defined. (Small changes in the input vector will force it to be classified one way or the other.)

5.1.4.2 Graceful degradation or fault tolerance

One-layer delta-rule networks show graceful degradation provided that the input patterns r_i^* are distributed. Just as for generalization, graceful degradation may appear to be limited if the network has had to learn to separate two very similar patterns which differ in only a few inputs r_i^* by a small amount. However, in this situation the input patterns can be thought of as being, in so far as their difference is concerned, rather non-distributed, so this is not a real exception to this general property of graceful degradation.

5.1.4.3 Prototype extraction, extraction of central tendency, and noise reduction

This occurs as for pattern autoassociators.

5.1.4.4 Speed

Recall is very fast in a one-layer pattern associator or perceptron, because it is a feedforward network (with no recurrent or lateral connections). Recall is also fast if the neuron has cell-like properties, because the stimulus input firings r_j^* ($j = 1, C$ axons) can be applied simultaneously to the synapses w_{ij} , and the activation h_i can be accumulated in one or two time constants of the dendrite (e.g. 10–20 ms). Whenever the threshold of the cell is exceeded, it fires. Thus, in effectively one time step, which takes the brain no more than 10–20 ms, all the output neurons of the delta-rule network can be firing with rates that reflect the input firing of every axon.

Learning is as fast ('one-shot') in perceptrons as in pattern associators if the input patterns are orthogonal. If the patterns are not orthogonal, so that the error-correction rule has to work to separate patterns, then the network may take many trials to achieve the best solution (which will be perfect under the conditions described above).

5.1.4.5 Non-local learning rule

The learning rule is not truly local, as it is in pattern associators, autoassociators, and competitive networks, in that with one-layer delta-rule networks, the information required to change each synaptic weight is not available in the presynaptic terminal (reflecting the presynaptic rate) and the postsynaptic activation. Instead, an error for the neuron must be computed, possibly by another neuron, and then this error must be conveyed back to the postsynaptic neuron to provide the postsynaptic error term, which together with the presynaptic rate determines how much the synapse should change, as in Eq. 5.2:

$$\delta w_{ij} = k (t_i - r_i) r_j^*$$

where $(t_i - r_i)$ is the error.

A rather special architecture would be required if the brain were to utilize delta-rule error-correcting learning. One such architecture might require each output neuron to be supplied with its own error signal by another neuron. The possibility that this is implemented in one

part of the brain, the cerebellum, is introduced in Chapter 9. Another functional architecture would require each neuron to compute its own error by subtracting its current activation by its r' inputs from another set of afferents providing the target activation for that neuron. A neurophysiological architecture and mechanism for this is not currently known.

5.1.4.6 Interference

Interference is less of a property of single-layer delta rule networks than of pattern autoassociators and autoassociators, in that delta rule networks can learn to separate patterns even when they are highly correlated. However, if patterns are not linearly independent, then the delta rule will learn a least mean squares solution, and interference can be said to occur.

5.1.4.7 Expansion recoding

As with pattern associators and autoassociators, expansion recoding can separate input patterns into a form which makes them learnable, or which makes learning more rapid with only a few trials needed, by delta rule networks. It has been suggested that this is the role of the granule cells in the cerebellum, which provide for expansion recoding by 1000:1 of the mossy fibre inputs before they are presented by the parallel fibres to the cerebellar Purkinje cells (Marr, 1969; Albus, 1971; see Chapter 9).

5.1.5 Utility of single-layer error-correcting networks in information processing by the brain

In the cerebellum, each output cell, a Purkinje cell, has its own climbing fibre, which distributes from its inferior olive cell its terminals throughout the dendritic tree of the Purkinje cell. It is this climbing fibre which controls whether learning of the r' inputs supplied by the parallel fibres onto the Purkinje cell occurs, and it has been suggested that the function of this architecture is for the climbing fibre to bring the error term to every part of the postsynaptic neuron (see Chapter 9). This rather special arrangement with each output cell apparently having its own teacher is probably unique in the brain, and shows the lengths to which the brain might need to go to implement a teacher for each output neuron. The requirement for error-correction learning is to have the neuron forced during a learning phase into a state which reflects its error while presynaptic afferents are still active, and rather special arrangements are needed for this.

5.2 Multilayer perceptrons: backpropagation of error networks

5.2.1 Introduction

So far, we have considered how error can be used to train a one-layer network using a delta rule. Minsky and Papert (1969) emphasized the fact that one-layer networks cannot solve certain classes of input-output mapping problems (as described above). It was clear then that these restrictions would not apply to the problems that can be solved by feedforward

multilayer networks, if they could be trained. A multilayer feedforward network has two or more connected layers, in which connections allow activity to be projected forward from one layer to the next, and in which there are no lateral connections within a layer. Such a multilayer network has an output layer (which can be trained with a standard delta rule using an error provided for each output neuron), and one or more hidden layers, in which the neurons do not receive separate error signals from an external teacher. (Because they do not provide the outputs of the network directly, and do not directly receive their own teaching error signal, these layers are described as hidden.) To solve an arbitrary mapping problem (in which the inputs are not linearly separable), a multilayer network could have a set of hidden neurons which would remap the inputs in such a way that the output layer can be provided with a linearly separable problem to solve using training of its weights with the delta rule. The problem was, how could the synaptic weights into the hidden neurons be trained in such a way that they would provide an appropriate representation? Minsky and Papert (1969) were pessimistic that such a solution would be found, and partly because of this, interest in computations in neural networks declined for many years. Although some work in neural networks continued in the following years (e.g. Marr, 1969, 1970, 1971; Willshaw and Longuet-Higgins, 1969, see Willshaw, 1981; von der Malsburg, 1973; Grossberg, 1976a,b; Arbib, 1964, 1987; Amari, 1982; Amari *et al.*, 1977), widespread interest in neural networks was revived by the type of approach to associative memory and its relation to human memory taken by the work described in the volume edited by Hinton and Anderson (1981), and by Kohonen (1984). Soon after this, a solution to training a multilayer perceptron using backpropagation of error became widely known (Rumelhart, Hinton and Williams, 1986a,b) (although earlier solutions had been found), and very great interest in neural networks and also in neural network approaches to cognitive processing (connectionism) developed (Rumelhart and McClelland, 1986; McClelland and Rumelhart, 1986).

5.2.2 Architecture and algorithm

An introduction to the way in which a multilayer network can be trained by backpropagation of error is described next. Then we consider whether such a training algorithm is biologically plausible. A more formal account of the training algorithm for multilayer perceptrons (sometimes abbreviated MLP) is given by Rumelhart, Hinton and Williams, 1986a,b).

Consider the two-layer network shown in Fig. 5.6. Inputs to the hidden neurons in layer A feed forward activity to the output neurons in layer B. The neurons in the network have a sigmoid activation function. One reason for such an activation function is that it is non-linear, and non-linearity is needed to enable multilayer networks to solve difficult (non-linearly separable) problems. (If the neurons were linear, the multilayer network would be equivalent to a one-layer network, which cannot solve such problems.) Neurons B1 and B2 of the output layer, B, are each trained using a delta rule and an error computed for each output neuron from the target output for that neuron when a given input pattern is being applied to the network. Consider the error which needs to be used to train neuron A1 by a delta rule. This error clearly influences the error of neuron B1 in a way which depends on the magnitude of the synaptic weight from neuron A1 to B1; and on the error of neuron B2 in a way that

depends on the magnitude of the synaptic weight from neuron A1 to B2. In other words, the error for neuron A1 depends on

the weight from A1 to B1 (w_{11}) · error of neuron B1
 + the weight from A1 to B2 (w_{21}) · error of neuron B2.

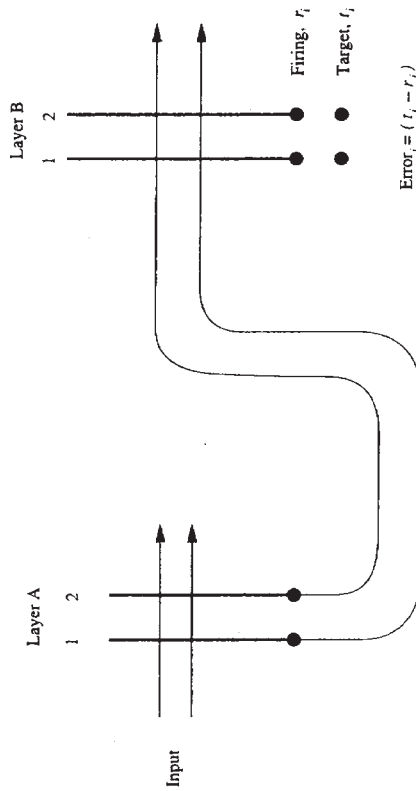


Fig. 5.6 A two-layer perceptron. Inputs are applied to layer A through modifiable synapses. The outputs from layer A are applied through modifiable synapses to layer B. Layer B can be trained using a delta rule to produce firing (r_i) which will approach the target t_i . It is more difficult to modify the weights in layer A, because appropriate error signals must be backpropagated from layer B.

In this way, the error calculation can be propagated backwards through the network to any neuron in any hidden layer, so that each neuron in the hidden layer can be trained, once its error is computed, by a delta rule. For this to work, the way in which each neuron is activated and sends a signal forward must be continuous (not binary), so that the extent to which there is an error in, for example, neuron B1 can be related back in a graded way to provide a continuously variable correction signal to previous stages. This is one of the requirements for enabling the network to descend a continuous error surface. The activation function must be non-linear (e.g. sigmoid) for the network to learn more than could be learned by a single-layer network. (Remember that a multilayer linear network can always be made equivalent to a single-layer linear network, and that there are some problems that cannot be solved by single-layer networks.) For the way in which the error of each output neuron should be taken into account to be specified in the error-correction rule, the position at which the output neuron is operating on its output activation function must also be taken into account. For this, the slope of the activation function is needed, and because the slope is needed, the activation function must be differentiable. Although we indicated use of a sigmoid activation function, other activation functions which are non-linear and monotonically increasing (and differentiable) can be used. (For further details, see Rumelhart, Hinton and Williams, 1986a, b and Hertz *et al.*, 1991).

5.2.3 Properties of multilayer networks trained by error backpropagation

5.2.3.1 Arbitrary mappings

Arbitrary mappings of non-linearly separable patterns can be achieved. For example, such networks can solve the XOR problem, and parity problems in general of which XOR is a special case. (The parity problem is to determine whether the sum of the (binary) bits in a vector is odd or even.) Multilayer feedforward backpropagation of error networks are not guaranteed to converge to the best solution, and may become stuck in local minima in the error surface. However, they generally perform very well.

5.2.3.2 Fast operation

The network operates as a feedforward network, without any recurrent or feedback processing. Thus (once it has learned) the network operates very fast, with a time proportional to the number of layers.

5.2.3.3 Learning speed

The learning speed can be very slow, taking many thousands of trials. The network learns to gradually approximate the correct input-output mapping required, but the learning is slow because of the credit assignment problem for neurons in the hidden layers. The credit assignment problem refers to the issue of how much to correct the weights of each neuron in the hidden layer. As the example above shows, the error for a hidden neuron could influence the errors of many neurons in the output layers, and the error of each output neuron reflects the error from many hidden neurons. It is thus difficult to assign credit (or blame) on any single trial to any particular hidden neuron, so an error must be estimated, and the net run until the weights of the crucial hidden neurons have become altered sufficiently to allow good performance of the network. Another factor which can slow learning is that if a neuron operates close to a horizontal part of its activation function, then the output of the neuron will depend rather little on its activation, and correspondingly the error computed to backpropagate will depend rather little on the activation of that neuron, so learning will be slow.

More general approaches to this issue suggest that the number of training trials for such a network will (with a suitable training set) be of the same order of magnitude as the number of synapses in the network (see Cortes *et al.*, 1996).

5.2.3.4 Number of hidden neurons and generalization

Backpropagation networks are generally intended to discover regular mappings between the input and output, that is mappings in which generalization will occur usefully. If there were one hidden neuron for every combination of inputs that had to be mapped to an output, then this would constitute a look-up table, and no generalization between similar inputs (or inputs not yet received) would occur. The best way to ensure that a backpropagation network learns

the structure of the problem space is to set the number of neurons in the hidden layers close to the minimum that will allow the mapping to be implemented. This forces the network not to operate as a look-up table. A problem is that there is no general rule about how many hidden neurons are appropriate, given that this depends on the types of mappings required. In practice, these networks are sometimes trained with different numbers of hidden neurons, until the minimum number required to perform the required mapping has been approximated.

A problem with this approach in a biological context is that in order to achieve their competence, backpropagation networks use what is almost certainly a learning rule which is much more powerful than those which could be implemented biologically, and achieves its excellent performance by performing the mapping through a minimal number of hidden neurons. In contrast, real neuronal networks in the brain probably use much less powerful learning rules, in which errors are not propagated backwards, and at the same time have very large numbers of hidden neurons, without the bottleneck which helps to provide backpropagation networks with their good performance. A consequence of these differences between backpropagation and biologically plausible networks may be that the way in which biological networks solve difficult problems may be rather different from the way in which backpropagation networks find mappings. Thus the solutions found by connectionist systems may not always be excellent guides to how biologically plausible networks may perform on similar problems. Part of the challenge for future work is to discover how more biologically plausible networks than backpropagation networks can solve comparably hard problems, and then to examine the properties of these networks, as a perhaps more accurate guide to brain computation.

5.2.3.5 Non-local learning rule

Given that the error for a hidden neuron is calculated by propagating backwards information based on the errors of all the output neurons to which a hidden neuron is connected, and all the relevant synaptic weights, and the activations of the output neurons to define the part of the activation function on which they are operating, it is implausible to suppose that the correct information to provide the appropriate error for each hidden neuron is propagated backwards between real neurons. A hidden neuron would have to 'know', or receive information about, the errors of all the neurons to which it is connected, and its synaptic weights to them, and their current activations. If there were more than one hidden layer, this would be even more difficult. To expand on the difficulties: first, there would have to be a mechanism in the brain for providing an appropriate error signal to each output neuron in the network. With the possible exception of the cerebellum, an architecture where a separate error signal could be provided for each output neuron is difficult to identify in the brain. Second, any retrograde passing of messages across multiple-layer forward-transmitting pathways in the brain that could be used for backpropagation seems highly implausible, not only because of the difficulty of getting the correct signal to be backpropagated, but also because retrograde signals in a multilayer net would take days to arrive, long after the end of any feedback given in the environment indicating a particular error. Third, as noted in Section 10.2, the backprojection pathways that are present in the cortex seem suited to

perform recall, and this would make it difficult for them also to have the correct strength to carry the correct error signal.

As stated above, it is a major challenge for brain research to discover whether there are algorithms that will solve comparably difficult problems to backpropagation, but with a local learning rule. Such algorithms may be expected to require many more hidden neurons than backpropagation networks, in that the brain does not appear to use information bottlenecks to help it solve difficult problems. The issue here is that much of the power of backpropagation algorithms arises because there is a minimal number of hidden neurons to perform the required mapping using a final one-layer delta-rule network. Useful generalization arises in such networks because with a minimal number of hidden neurons, the net sets the representation they provide to enable appropriate generalization. The danger with more hidden neurons is that the network becomes a look-up table, with one hidden neuron for every required output, and generalization when the inputs vary becomes poor. The challenge is to find a more biologically plausible type of network which operates with large numbers of neurons, and yet which still provides useful generalization.

5.3 Reinforcement learning

One issue with perceptrons and multilayer perceptrons that makes them generally biologically implausible for many brain regions is that a separate error signal must be supplied for each output neuron. When operating in an environment, usually a simple binary or scalar signal representing success or failure is received. Partly for this reason, there has been some interest in networks that can be taught with such a single reinforcement signal. In this section, we describe one such approach to such networks. We note that such networks are classified as supervised networks in which there is a single teacher, and that these networks attempt to perform an optimal mapping between an input vector and an output neuron or set of neurons. They thus solve the same class of problems as single and (if multilayer) multilayer perceptrons. They should be distinguished from pattern association networks in the brain, which might learn associations between previously neutral stimuli and primary reinforcers such as taste (signals which might be interpreted appropriately by a subsequent part of the brain), but do not attempt to produce arbitrary mappings between an input and an output, using a single reinforcement signal. A class of problems to which such networks might be applied are motor control problems. It was to such a problem that Barto and Sutton (Barto, 1985; Sutton and Barto, 1981) applied their reinforcement learning algorithm.

5.3.1 Associative reward-penalty algorithm of Barto and Sutton

The terminology of Barto and Sutton is followed here (see Barto, 1985).

5.3.1.1 Architecture

The architecture, shown in Fig. 5.7, uses a single reinforcement signal, r , = +1 for reward, and -1 for penalty. The inputs x_i take real (continuous) values. The output of a neuron, y , is binary, +1 or -1. The weights on the output neuron are designated w_i .

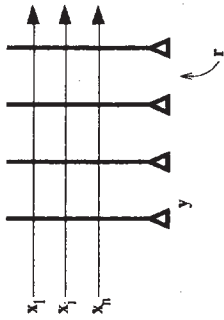


Fig. 5.7 A network trained by a single reinforcement input r . The inputs to each neuron are in the terminology used by Barto and Sutton x_i , $i = 1, n$; and y is the output of one of the output neurons.

5.3.1.2 Operation

1. An input vector is applied to the network, and produces output activation, s , in the normal way as follows:

$$s = \sum w_i x_i$$

2. where x_i is the firing of the input axon i .
The output is calculated from the activation with a noise term η included. The principle of the network is that if the added noise on a particular trial helps performance, then whatever change it leads to should be incorporated into the synaptic weights, in such a way that the next time that input occurs, the performance is improved.

$$\text{Output } y = +1 \text{ if } s + \eta > 0 \\ = -1 \text{ else}$$

where η = the noise added on each trial.

3. Learning rule. The weights are changed as follows:

$$\delta w_i = \rho(y - E[y|s])x_i \quad \text{if } r = +1 \\ \delta w_i = \rho\lambda(-y - E[y|s])x_i \quad \text{if } r = -1.$$

ρ and λ are learning rate constants. (They are set so that the learning rate is higher when positive reinforcement is received than when negative reinforcement is received.) $E[y|s]$ is the expectation of y given s (usually a sigmoidal function of s with the range ± 1). $E[y|s]$ is a (continuously varying) indication of how the unit usually responds to the current input pattern, i.e. if the actual output y is larger than normally expected, by computing $s = \sum w_i x_i$, because of the noise term, and the reinforcement is $+1$, increase the weight from x_i ; and vice versa. The expectation could be the prediction generated before the noise term is incorporated.

This network combines an associative capacity with its properties of generalization and graceful degradation, with a single 'critic' or error signal for the whole network (Barto, 1985). The network can solve difficult problems (such as balancing a pole by moving a trolley which

supports the pole from side to side, as the pole starts to topple). Although described for single-layer networks, the algorithm can be applied to multilayer networks. The learning rate is very slow, for there is a single reinforcement signal on each trial for the whole network, not a separate error signal for each neuron in the network.

An important advance in the area of reinforcement learning was the introduction of algorithms that allow for learning to occur when the reinforcement is delayed or received over a number of time steps. A solution to this problem is the addition of an adaptive critic that learns through a time difference (TD) algorithm how to predict the future value of the reinforcer. The output of the critic is used as an effective reinforcer instead of the instantaneous reinforcement being received (which reflects previous performance) (see Sutton and Barto, 1990; Barto, 1995). This is a solution to the temporal credit assignment problem. The algorithm has been applied to modelling the time course of classical conditioning (Sutton and Barto, 1990).

The reinforcement learning algorithm is certainly a move towards biological relevance, in that learning with a single reinforcer can be achieved. That single reinforcer might be broadcast throughout the system by a general projection system, such as the dopamine pathways in the brain, which distribute to large parts of the striatum and the prefrontal cortex. It is not clear yet how a biological system might store the expected output $E[y|s]$ for comparison with the actual output when noise has been added, and might take into account the sign and magnitude of the noise. Nevertheless, this is an interesting algorithm.

5.4 Contrastive Hebbian learning: the Boltzmann machine

In a move towards a learning rule which is more local than in backpropagation networks, yet which can solve similar mapping problems in a multilayer architecture, we describe briefly contrastive Hebbian learning. The multilayer architecture has forward connections through the network to the output layer, and a set of matching backprojections from the output layer through each of the hidden layers to the input layer. The forward connection strength between any pair of neurons has the same value as the backward connection strength between the same two neurons, resulting in a symmetric set of forward and backward connection strengths. An input pattern is applied to the multilayer network, and an output is computed using normal feedforward activation processing with neurons with a sigmoid (non-linear and monotonically increasing) activation function. The output firing then via the backprojections is used to create firing of the input neurons. This process is repeated until the firing rates settle down, in an iterative way (which is similar to the settling of the autoassociative nets described in Chapter 3). After settling, the correlations between any two neurons are remembered, for this type of unclamped operation, in which the output neurons fire at the rates that the process just described produces. The correlations reflect the normal presynaptic and postsynaptic terms used in the Hebb rule, e.g. $(r'_j r_i)^{uc}$, where uc refers to the unclamped condition, and as usual r'_j is the firing rate of the input neuron, and r is the activity of the receiving neuron. The output neurons are then clamped to their target values, and the iterative process just described is repeated, to produce for every pair of synapses in the network $(r'_j/r_i)^c$, where the c refers now to the clamped condition. An error-correction term for each synapse is then computed from the difference between the remembered correlation of

the unclamped and the clamped conditions, to produce a synaptic weight correction term as follows:

$$\delta w_{ij} = k[(r'_j r_i)^c - (r'_j r_i)^{uc}] \quad (5.6)$$

where k is a learning rate constant. This process is then repeated for each input pattern to output pattern to be learned. The whole process is then repeated many times with all patterns until the output neurons fire similarly in the clamped and unclamped conditions, that is until the errors have become small. Further details are provided by Hinton and Sejnowski (1986). The version described above is the mean field (or deterministic) Boltzmann machine (Peterson and Anderson, 1987; Hinton, 1989). More traditionally, a Boltzmann machine updates one randomly chosen neuron at a time, and each neuron fires with a probability that depends on its activation (Ackley, Hinton and Sejnowski, 1985; Hinton and Sejnowski, 1986). The latter version makes fewer theoretical assumptions, while the former may operate an order of magnitude faster. An application of this type of network is described by Baddeley (1995).

In terms of biological plausibility, it certainly is the case that there are backprojections between adjacent cortical areas (see Rolls, 1989a-e). Indeed, there are as many backprojections between adjacent cortical areas as there are forward projections. The backward projections seem to be more diffuse than the forward projections, in that they connect to a wider region of the preceding cortical area than the region which sends the forward projections. If the backward and the forward synapses in such an architecture were Hebb-modifiable, then there is a possibility that the backward connections would be symmetric with the forward connections. Indeed, such a connection scheme would be useful to implement top-down recall, as described in Chapter 6. What seems less biologically plausible is that after an unclamped phase of operation, the correlations between all pairs of neurons would be remembered, there would then be a clamped phase of operation with *each* output neuron clamped to the required rate for that particular input pattern, and then the synapses would be corrected by an error-correction rule that would require a comparison of the correlations between the neuronal firing of every pair of neurons in the unclamped and clamped conditions.

5.5 Conclusion

The networks described in this chapter are capable of mapping a set of inputs to a set of required outputs using correction when errors are made. Although some of the networks are very powerful in the types of mapping they can perform, the power is obtained at the cost of learning algorithms which do not use local learning rules. Because the networks described in this chapter do not use local learning rules, their biological plausibility remains at present uncertain, although it has been suggested that perceptron learning may be implemented by the special neuronal network architecture of the cerebellum (see Chapter 9). One of the aims of future research must be to determine whether comparably difficult problems to those solved by the networks described in this chapter can be solved by biologically plausible neuronal networks (see e.g. Dayan and Hinton, 1996; O'Reilly, 1996).

6 The hippocampus and memory

In this chapter we show how it is becoming possible to link quantitative neuronal network approaches to other techniques in neuroscience to develop quantitative theories about how brain systems involved in memory operate. The particular brain system considered is the hippocampus and nearby structures in the temporal lobe of the brain, which are involved in learning about new events or facts. The primary aim of this chapter is to show how it is possible to link these approaches, and to produce a theory of the operation of a brain structure. Although of course the particular theory of hippocampal function described here is subject to revision with future empirical as well as theoretical research, the aim is nevertheless to show how evidence from many different disciplines can now be combined to produce a theory of *how* a part of the brain may work. The empirical evidence required comes from: lesion studies which show what function may be performed by a brain region; single and multiple single neuron recording in the hippocampus while it is performing its normal functions, to show what information reaches the hippocampus, and how the information is represented; the external connections of the hippocampus, to help define what information reaches the hippocampus, and which brain regions the hippocampus can influence; the internal connections of the hippocampus, to show what networks are present, and to give quantitative evidence which is important in defining the computational properties of the networks; biophysical studies to define the ways in which neurons operate; and from studies of synaptic modifiability in different parts of the hippocampus and its role in learning, to define the synaptic modification rules at different hippocampal synapses which should be incorporated into models of how the hippocampus works.

6.1 What functions are performed by the hippocampus? Evidence from the effects of damage to the brain

In humans with bilateral damage to the hippocampus and nearby parts of the temporal lobe, amnesia is produced. The amnesia is anterograde, that is for events which happen after the brain damage. Although there is usually some retrograde amnesia, that is for events before the brain damage, this may not be severe, and indeed its severity appears to depend on how much damage there is to the cortical areas adjacent and connected to the hippocampus such as the entorhinal, perirhinal, and parahippocampal cortex. These effects of damage to the hippocampus indicate that the very long-term storage of information is not in the hippocampus, at least in humans.¹ On the other hand, the hippocampus does appear to be