

4 Autoassociation

This chapter will examine the performance of a particular form of pattern associator, an autoassociator, trained with the Delta rule. An autoassociator reproduces at output the same pattern that was presented at input. It has many basic properties in common with the pattern associator taught with the Hebb rule. It can store independent memories on the same set of connections and it has the desirable property of 'cleaning up' incomplete or noisy inputs. When individual experiences, drawn from a set of different categories, are stored in an autoassociator memory, prototypical instances of each category are formed automatically. This offers a possible solution to the problem of how humans succeed in categorising the world without explicit teaching.

The architecture and operation of an autoassociator

The pattern associator described in chapter 3 learnt to produce a specified pattern at output in response to a particular pattern at input. Autoassociation is a special case of pattern association. The aim of the autoassociator is to reproduce the *same* pattern at output that was present at input. At first sight this may seem singularly pointless. Why have a complex structure if all it does is to reproduce at output the pattern which is already there at input? We shall see that an autoassociator network which has learnt about the statistical structure of previous inputs can perform some very useful operations on subsequent inputs.

Architecture

A typical structure for an autoassociator is shown in figure 4.1. It consists of a set of 8 processing units (numbered 1–8), each with a dendrite and an output line. The external input to the autoassociator comes from some previous stage of processing. The output lines take the transformation of this signal produced by the autoassociator on to the next stage of processing.

The difference between the structure of an autoassociator and that of the pattern associator discussed in the previous chapter is that the output line of each unit is

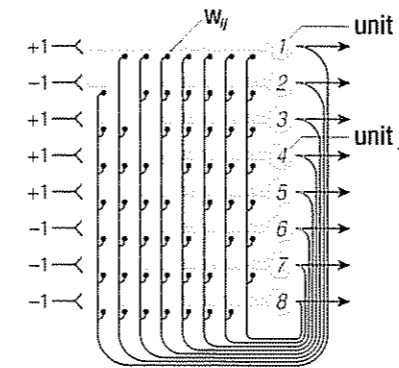


Figure 4.1 An eight unit autoassociator. (Based on McClelland and Rumelhart 1985)

connected back to the dendrites of the other units. These are called *recurrent* connections. They are represented by the loop back from the output of each unit which connects to the dendrites of all the other units. At each meeting of recurrent feedback from one unit and the dendrite of another there is a modifiable connection. (Networks with recurrent connections have some special properties which will be described in chapter 7. In this chapter we will emphasise the general properties which are similar to those of the pattern associator described in the previous chapter even though the architecture of the network and the learning rule used to train it are different.)

The net input to unit i consists of the external input (extinput_i) and the internal input (intinput_i) generated by feedback from other units within the autoassociator. The internal input is calculated in the usual way, summing the products of the activity of each unit connected to unit i and the strength of the connection between them:

$$\text{netinput}_i = \text{extinput}_i + \sum_j a_j w_{ij} \quad (4.1)$$

where a_j is the activity of unit j . In this case j indexes the units other than i in the autoassociator,

w_{ij} is the strength of the connection between the recurrent input from unit j and the dendrite of unit i .

An input is presented to the autoassociator by clamping a pattern of stimulation, the external input, onto the dendrites. This is shown as a pattern of +1s and -1s in figure 4.1. This could represent the output from a prior stage of sensory processing following the presentation of some stimulus such as a word or a face. The external input produces an activity level in each unit determined by an activation function like one of those shown in figure 1.4 in chapter 1. Via the internal feedback connections, this activity, weighted by the strength of the appropriate connection, then starts to produce an internal input to all the other units. The net input to a unit is now the

sum of the external and internal inputs. This produces a new level of activity in the unit, which feeds back to all the other units, changing their net input, and consequently their activity level, and so on. There is a danger that activity in a system with positive feedback can continue to grow. The use of a non-linear activation function such as the sigmoid shown in figure 1.4(c) ensures that the activity of each unit cannot grow beyond a fixed maximum value. The autoassociator is allowed to run for a number of cycles until it reaches a steady state where there is no further change in the internal input.

Learning with the Delta rule

The aim of the autoassociator is to reproduce at output the pattern presented at input. This is achieved during the learning phase by changing the weights so that the internal input to each unit matches the external input. This can be done by calculating the difference between the internal and external inputs and changing the weights of the connections in the direction which will reduce the difference. This is an example of the application of the Delta rule described in chapter 1.¹ If the internal input to a given unit is less than the external then the weights of connections carrying a positive input to the unit are increased, and those carrying a negative input are reduced. If the internal input is greater than the external, vice versa. The difference between the external and internal inputs to unit i is δ_i . So:

$$\delta_i = \text{extinput}_i - \text{intinput}_i \quad (4.2)$$

The rule for the weight change (Δw_{ij}) in the connection between unit i and the recurrent input from unit j is:

$$\Delta w_{ij} = \varepsilon \delta_i a_j \quad (4.3)$$

where ε is a constant which determines how large the weight change is on any individual trial,

δ_i is the error on unit i ,

and a_j is the activity of unit j .

The inclusion of a_j in the weight change rule is a way of apportioning blame for δ_i . δ_i represents the failure of the internal input to unit i , $\sum_j a_j w_{ij}$, to match the external input. By making the weight change on a connection proportional to a_j , the activity of the unit sending input along that connection, the changes are concentrated on those connections where they will have most effect on the internal input to i .

This learning rule follows a different principle from the Hebb rule used with the pattern associator in chapter 3 (compare equation 4.3 with equation 3.1). The Hebb

¹The aim of chapters 3 and 4 is to introduce a variety of architectures and learning rules. It is, of course, possible to train an autoassociator with the Hebb rule or a pattern associator with the Delta rule.

rule operated by strengthening connections between units which were both active. The rule represented by equation 4.3 starts by defining a desired state of affairs ($\delta_i = 0$) and operates by strengthening connections which would lead to a reduction in the difference between the desired and the actual state of affairs.

Chapter 5 will explore learning with the Delta rule in greater detail, in particular seeing how it can be used to train networks with more than one layer of modifiable connections. For now, we will see how the Delta rule works with a simple network. The weight matrix which results from the application of this rule turns out to have many of the same general properties as those which we saw resulting from use of the Hebb learning rule in chapter 3.

Properties of autoassociator memories

To demonstrate some of the properties of an autoassociator memory which apply to central problems in cognition we will follow some parts of a simulation which is described in greater detail in McClelland and Rumelhart (1985, 1986). The examples will be explored in the **tlearn** exercises at the end of the chapter. First we will use the simulation to show that, just like the pattern associator trained with the Hebb rule, an autoassociator trained with the Delta rule can store the individual memories of different events on a single matrix of connections. Then we will show that, again like the pattern associator, it has the biologically important properties of noise resistance and pattern completion if the recall cues for the memories are distorted.

What an autoassociator learns

The McClelland and Rumelhart exploration of autoassociator memory began with the 8 unit autoassociator shown in figure 4.1. Each unit had a maximum activity level of +1 and a minimum activity level of -1. Initially the weights were set to zero. They presented it with the external input pattern (+1 -1 +1 -1 +1 +1 -1 -1). Figure 4.2(a) shows the response of the autoassociator. Since the weights are zero there is no internal input to each unit. The unit activity, ± 0.5 in each case, reflects the result of feeding the external input value of ± 1 into the activation function. (The activation function was a sigmoid like that in figure 1.4(c) of chapter 1, but the range of activity it permitted was -1 to +1 rather than 0 to 1.)

What is the result of applying the learning algorithm represented by equation 4.3? Take unit 1. The change in the weights from units 2-8 to unit 1, Δw_{12} to Δw_{18} , will be equal to the product of ε , δ_1 and a_2 to a_8 respectively. Since the external input = 1 and the internal input = 0, equation 4.2 shows that δ_1 is positive. So the weights of the connections between unit 1 and units which have positive activity produced by the external input pattern (units 3, 5 and 6) will receive a positive increment, of a

(a)	External input	Initial unit activity	Unit
	+1	+0.5	1
	-1	-0.5	2
	+1	+0.5	3
	-1	-0.5	4
	+1	+0.5	5
	+1	+0.5	6
	-1	-0.5	7
	-1	-0.5	8

(b)	External input 1	Unit activity after training	Unit
	+1	+0.7	1
	-1	-0.7	2
	+1	+0.7	3
	-1	-0.7	4
	+1	+0.7	5
	+1	+0.7	6
	-1	-0.7	7
	-1	-0.7	8

(c)	External input 2	Unit activity after training	Unit
	+1	+0.7	1
	-1	-0.7	2
	+1	+0.7	3
	-1	-0.7	4
	+1	+0.7	5
	+1	+0.7	6
	-1	-0.7	7
	-1	-0.7	8

Figure 4.2 (a) The activity of the units in an untrained network like that shown in figure 4.1 after presentation of an external pattern. (b) The activity after training with the external input pattern. (c) The activity after training with a second external input pattern (Based on McClelland and Rumelhart 1985.)

size determined by ϵ . Since these weights started at 0 they will become positive. Weights of connections between unit 1 and units with a negative activity (units 2, 4, 7 and 8) will receive a negative increment. Since they started at 0 they will become negative.

On the next trial the input to unit 1 will consist of the external input plus, since the weights are now non-zero, an internal input. The internal input is given by $\sum_j a_j w_{ij}$, the sum of the product of activity level and weight for each unit connected to unit 1. The units which have a positive activity level (3, 5 and 6) are connected to unit 1 by positive weights; the units which have a negative activity level (2, 4, 7 and 8) are connected by negative weights. So every unit makes a positive contribution to the

internal input to unit 1. (If both a_j and w_{ij} are negative, $a_j w_{ij}$ is positive.) Since the aim of the autoassociator is to produce an internal input which matches the external input, and the external input to unit 1 is positive, the learning algorithm in equation 4.3 has produced a step in the right direction.

Now consider unit 2. Initially δ_2 is negative because the external input = -1 and the internal input = 0. So application of equation 4.3 will lead to the opposite result to that achieved for unit 1. The weights of connections to unit 2 from units which have a positive activity level (1, 3, 5 and 6) will be negative, and the weights of connections from units which have a negative activity level (4, 7 and 8) will be positive. Taking the product of these weights and the unit activities ($\sum_j a_j w_{ij}$) will produce a negative internal input to unit 2. So again the desired result will have been achieved since the external input to unit 2 is negative.

The learning rule develops a set of weights such that the internal input starts to match the external. So δ_i , the difference between external and internal inputs to unit i , will become smaller. Consequently the weight changes on each learning trial will become smaller. If the network achieves a set of weights such that the internal input matches the external input, $\delta_i = 0$ and learning (i.e. weight change) stops. Homing in on a set of weights which can perform the task, with gradually smaller changes of the weights, is typical of learning with the Delta rule. It is unlike learning with the Hebb rule where only one learning trial may be required to establish the necessary connections.

Acquiring a set of weights which would allow the internal input generated by pattern 1 to match the external input is straightforward. Autoassociation is potentially problematic because the same set of connections must also ensure that an appropriate internal input develops when a second, quite different pattern is presented. If the new pattern is (+1 +1 -1 -1 -1 +1 -1 +1) the weights must now provide a positive internal input to units 1, 2, 6 and 8, and a negative internal input to units 3, 4, 5 and 7. In McClelland and Rumelhart's simulation these two patterns were presented to the autoassociator ten times. After each presentation the weights were changed according to equation 4.3. Figures 4.2(b) and (c) show that the learning algorithm developed a set of weights which produced a larger activity in the units than to the external input alone (0.7 vs 0.5). Thus the internal input has mimicked the external output in both cases. Just like the pattern associator trained with the Hebb rule, application of the Delta rule leads to a set of weights which can store an appropriate response to different patterns on the same connections.

Figure 4.3 shows how the network manages to produce the correct response to both patterns. Each part of the figure shows a weight matrix, that is, the weights of the connections in the autoassociator. The figure shows what the autoassociator shown in figure 4.1 would look like if the dendrites and feedback lines had been removed and only the modifiable connections remained. Each value in the matrix is

the weight of the corresponding connection. (To understand this example the numerical values of the weights are not important, only the sign. So, for clarity, the numerical values have been omitted.) The eight rows in the matrix correspond to the input lines to the dendrites of the 8 units in the autoassociator; the columns show the weight of the connection from each other unit to the unit given by the row. (As usual the receiving units are indexed by i and the sending units by j .) Thus row 1 shows the weights of the recurrent connections from units 2–8 to unit 1. Column 1 shows the value of the weights of the connections from unit 1 to units 2–8. (Cells on the diagonal of the matrix are empty because there are no recurrent connections from any unit to itself—see figure 4.1.)

The matrix in figure 4.3(a) shows the weights that the autoassociator acquires on learning trials on which pattern 1 is presented. The aim of the learning rule is to ensure that the internal input to each unit matches the external input. Unit 1 receives a positive external input from pattern 1, so the set of weights from the other units (i.e. the values in row 1) should have a polarity which will generate a positive internal input to unit 1 when pattern 1 is presented. Pattern 1 also makes units 3, 5 and 6 positive. Their state is thus positively correlated with the desired state of unit 1. The positive weights between these units and unit 1 (i.e. weights 3, 5 and 6 in row 1) will tend to turn unit 1 on if units 3, 5 or 6 are active. In contrast, pattern 1 is making units 2, 4, 7 and 8 negative. The correlation between the state of these units and the state of unit 1 is negative. So negative weights develop at the connections between units 2, 4, 7 and 8 and the input line to unit 1. If any of these units are active, the negative weights will ensure that they also make a positive internal input to unit 1.

The second row shows the weights of the connections to unit 2 from units 1 and 3–8. The external input to unit 2 is negative, so the pattern of weight polarity is the reverse of the pattern to unit 1. There are positive weights to unit 2 from those units which have negative activity, and negative weights from those units which have positive activity. The third row shows connections from each unit to unit 3. Like unit 1 this has a positive external input in pattern 1. Thus the polarity of the weights from each unit to unit 3 is the same as it was to unit 1. And so on. Overall it can be seen that, just as with the pattern associator taught with the Hebb rule, a positive weight in the autoassociator reflects a positive correlation between the states of the two units it connects. In this case, because we have used -1 rather than 0 for non-positive inputs, negative weights have also developed to represent negative correlations.

Storage of different memories on the same connections

Figure 4.3(a) shows the matrix of weights which the autoassociator acquired as it learnt how to reproduce pattern 1. Figure 4.3(b) shows the weight matrix which the autoassociator acquired as it learnt to reproduce pattern 2. This is just like the matrix

(a)	External Input 1	unit, →	1	2	3	4	5	6	7	8	Unit, ↓
											1
	+			-	+	-	+	+	-	-	2
	-					+	-	-	+	+	3
	+			+	-		+	+	-	-	4
	-			-	+		-	-	+	+	5
	+			+	-	+		+	-	-	6
	+			+	-	+			-	-	7
	-			-	+	-	+	-		+	8
	-			-	+	-	+	-	-	+	
											Weight matrix
(b)	External Input 2	unit, →	1	2	3	4	5	6	7	8	Unit, ↓
											1
	+			+	-	-	-	+	-	+	2
	+			+	-	-	-	+	-	+	3
	-			-	-		+	+	-	+	4
	-			-	-	+		+	+	-	5
	+			+	+	-	-	-	-	+	6
	-			-	-	+	+	+	-	-	7
	+			+	+	-	-	-	+	-	8
											Weight matrix
(c)		unit, →	1	2	3	4	5	6	7	8	Unit, ↓
											1
						-		+	-		2
										+	3
								+		-	4
						+				-	5
											6
								+			7
						+		-			8
											Weight matrix

Figure 4.3 (a) and (b) The weight matrix acquired by the autoassociator when it has learnt to reproduce the external input pattern shown on the left. (c) The combined weight matrix formed when the two matrices acquired during learning to reproduce the two individual patterns are superimposed. (Based on McClelland and Rumelhart 1985)

acquired to pattern 1, except that the pattern of positive and negative weights is different because the correlations between elements of the two input patterns is different. When the autoassociator is required to learn both patterns the weights learnt in response to input patterns 1 and 2 are superimposed on the same connections. The effect of learning both patterns is to produce the overall weight matrix shown in figure 4.3(c).

Figures 4.2(b) and (c) showed that the autoassociator produces the correct response when either pattern 1 or 2 is presented. Inspection of the combined matrix shows how it can do this. Where the weights in the two individual matrices [figures 4.3(a) and (b)] are opposite in sign they cancel out. Zero valued weights are represented by a dot. Where the weights are the same sign the combined matrix develops a weight with the corresponding sign. Thus for example, the connection represented in row 1 column 2 of the combined matrix is zero because there is a negative correlation between elements 2 and 1 in pattern 1 but a positive correlation in pattern 2. Row 1, column 4 in the combined matrix has a negative weight because there is a negative correlation between elements 4 and 1 in both patterns 1 and 2.

It should be clear why coherent memory retrieval is possible despite superimposition of information about independent patterns. Each weight in the combined matrix represents something that is true about both pattern 1 and pattern 2. As the matrix learns more patterns the picture is, obviously, going to become more complex. But the fact that the weights can take any value allows the matrix to keep track of all the input correlations as more patterns are presented. If in all the patterns that are presented elements 1 and 4 are negatively correlated, the weight in row 1, column 4 will acquire a large negative value. If there is a negative correlation between elements 1 and 4 in many but not all patterns, the value will still be negative but with a lower value because the positive correlations will have reduced it. If there are only marginally more patterns in which the correlation is negative rather than positive, the value will still be negative, but now, because the positive and negative correlations will nearly cancel out, the magnitude will be close to zero. Each weight in the autoassociator reflects the correlation between the states of the two units it connects in the patterns which it has learnt—positive weights for positive correlations, negative weights for negative correlations. Now it is possible to see why this network has the properties of pattern completion and noise resistance.

Pattern completion

Figure 4.4 shows what happens when an incomplete fragment of pattern 1 is presented to the autoassociator rather than the pattern which it learnt. Part (a) shows the response to the full pattern (+1 -1 +1 -1 +1 +1 -1 -1). Part (b) shows the response to the partial pattern (+1 -1 +1 -1). Units 5–8 adopt the same polarity in response to the incomplete input (although at lower magnitude) as they did when the complete pattern was presented. The reason units 5–8 produce this pattern is that in pattern 1, positive activity in unit 5 is correlated with positive activity in units 1 and 3 and negative activity in units 2 and 4. These facts were recorded in the weights between units 1–4 and 5 during the learning trials with pattern 1 (see row 5 of matrix (a) in figure 4.3). These facts are still preserved (up to a point) in the com-

(a)	External Input	Unit Activity	Unit
	+1	+0.7	1
	-1	-0.7	2
	+1	+0.7	3
	-1	-0.7	4
	+1	+0.7	5
	+1	+0.7	6
	-1	-0.7	7
	-1	-0.7	8

(b)	External Input	Unit Activity
	+1	+0.6
	-1	-0.6
	+1	+0.6
	-1	-0.6
		+0.4
		+0.4
		-0.4
		-0.4

(c)	External Input	Unit Activity
	+1	+0.6
	-1	-0.6
	+1	+0.6
	-1	-0.6
	+1	+0.6
	+1	+0.6
	-1	-0.6
	+1	+0.1

Figure 4.4 (a) The response of the network to an external input pattern. (b) The response to a fragment of this pattern. (c) The response to a noisy version of the pattern. (Based on McClelland and Rumelhart 1985.)

bined matrix (see row 5 in matrix (c) of figure 4.3). When the pattern (+ - + -) is presented to units 1–4, the internal input from these units achieves a positive activity level in unit 5. Similarly, *mutatis mutandis*, for units 6–8. The level of activity is less than it would be if the complete pattern had been presented because the external input is less for the fragment than for the complete pattern.

Noise resistance

Figure 4.4(c) shows what happens when a 'noisy' version of pattern 1 is presented to the autoassociator. When the pattern (+1 -1 +1 -1 +1 +1 -1 +1) is presented rather than pattern 1 (the polarity of the last input has been changed) the autoassociator nearly changes the polarity of the 8th unit to what it would have been if pattern 1 had been presented. It does this rather than reproducing the value in the

external input for the same reason that it completes the partial pattern. When the noisy pattern is presented, the internal input to unit 8 from units 1–7 is trying to make unit 8 negative. This is nearly enough to overcome the positive external input.

It should now be clear why an autoassociator memory stage might be included in a processing system. It is not merely reproducing the input. If the external input is incomplete or noisy the internal input can clean it up before passing it on to the next stage of processing. Given that the brain is a noisy environment where signals are often distorted, either by random cell firing or by random cell death, this has obvious benefits. It is also the case that many signals reaching the brain *are* noisy versions of a prototype. Every example of a spoken word that you hear is slightly different because of the accent of the speaker, or the context it appears in, or the background noises which are superimposed upon it. Generally the distortions are not interesting in themselves—it is the prototype that the distortion came from which you wish to identify.

Despite its usefulness in some situations there is a cost to these cleaning up operations. The ‘incomplete’ or ‘noisy’ input in figure 4.4 may be neither—it may be a new signal, conveying some information different from that conveyed by pattern 1. Of course, the autoassociator cannot know which is the case. If an external input which it has not met before is presented to the autoassociator it will be altered to look more like whichever of its previously learnt signals the new one is most similar to.² If an unreliable transmission system is operating in a predictable world the inclusion of an autoassociation stage may be good system design—a novel signal is more likely to come from erroneous transmission of a known signal than from a novel signal. But it does force any part of the system beyond the autoassociator to see the present through a distorting filter of past experiences. This is, surely, a human characteristic. Everyone will recognise in other people (if not in themselves) a tendency to interpret novel experiences, not as they really are, but through a filter of preconceptions, determined by past experiences.

A dramatic example of the ability of an autoassociator to recall complete memories from partial or noisy cues, with more realistic stimuli than those used in figure 4.4, is shown in figure 4.5. Hertz *et al.* (1991) stored seven images in an auto-associative network. At recall they presented the network with either a noisy (upper left) or a partial (lower left) retrieval cue. In each case the complete original image was retrieved despite the poverty of the stimulus.

²This is an example of a general problem for autoassociators. If they are trained on patterns A, B and A+B they will have difficulty in recalling either A or B. Both will tend to complete and produce the output A+B. One way to reduce this problem is to recode the input pattern vectors, minimising their overlap, with a competitive network before they are presented to the autoassociator. (Competitive networks will be described in chapter 6.) We will see an example of this process at work in chapter 13 in a model of episodic memory formation in the hippocampus

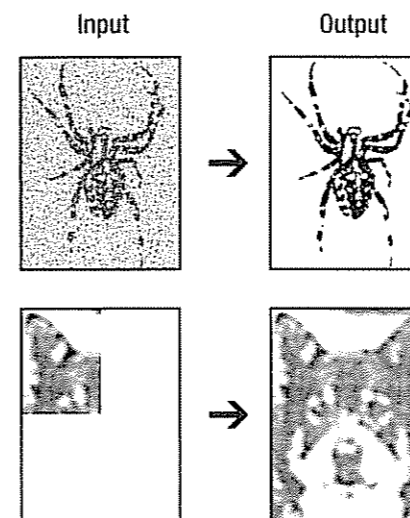


Figure 4.5 Recall of a complete image by an autoassociator given a noisy or partial cue. (Based on Hertz *et al.* 1991.)

Forming categories and prototypes from individual experiences

Experience comes in isolated incidents. One day a large brown Alsatian bites your leg; the next day a small sandy coloured terrier bites your ankle. By forming a memory of these experiences you can learn to keep your legs away from large brown Alsatians and your ankles away from small sandy coloured terriers. That's a good start. But on their own the memories will not suggest appropriate behaviour when you meet a medium sized black Rottweiler. The individual memories would be more useful if they could be pooled to form the *category* <dog>. Then the information <may bite> can be attached to the category. Provided you can recognise the black Rottweiler as an example of the *category*, you can benefit from your previous experiences with other members and take appropriate action even though this particular dog has not bitten you before. The human cognitive system is very good at recognising the similarities in individual experiences and forming general categories from them. People have no trouble recognising that Rottweilers are dogs even if they have never seen one before. Having formed a category we also know what a typical member of that category is like. Everyone would agree that while labradors and dachshunds are dogs, a labrador is a typical dog, but a dachshund is, frankly, a bit peculiar. How do people form categories from individual experiences?

Tutored category formation is not a problem. If a child has been taught a set of category names, dog, cat and bird, say, and is initially told which category to put each new experience in ('That's a dog', 'That's a cat' and so on) it is easy to see how each new example of an animal that it sees could be assigned to one of the categories.

Subsequent experiences could be compared to the examples already in each group and categorised accordingly. But *untutored* category formation seems to pose a serious problem. If the child has no categories to start with, why should it partition experiences in any particular way, or, indeed, at all? To form the category <dog> seems to require that you know that categorisation of dogs would be useful and that you know what sort of experiences you should put in the category before you start. In short, untutored category formation seems to require the cognitive system to pull itself up by its own bootstraps. And yet, as everyone knows from observation of the intellectual development of children, and many experiments on category and prototype formation confirm, untutored category and prototype formation seems to be an inevitable consequence of entering experiences into human memory.

Discovering a prototype from exemplars with an autoassociator

McClelland and Rumelhart simulated the problem of forming categories and prototypes from individual experiences by imagining a child learning about dogs. The child sees many different dogs, all basically similar in appearance, although different in detail, but with unrelated names. We know that a child would form the category <dog> from such experiences, would know what a typical dog looked like and would recognise which new experiences could plausibly be dogs. Can a connectionist network do this if it is given no information about the existence of the category?

McClelland and Rumelhart used an autoassociator like that in figure 4.1 but with 24 units. Each experience of a dog was presented to it as an external input consisting of a string of 24 binary digits. The pattern of the first 8 digits represented the signal which would be generated by the dog's name; the next 16 represented the signal generated by its physical appearance. Despite their identifiably different appearance, alsatians, spaniels and labradors all look more like each other than like most other things in the world around the child. But their names are no more similar than any other words chosen at random from the English language. Since there is no prototypical name for different dogs, the first 8 elements, the *name* part of each pattern, were a random sequence of +1s and -1s. To represent the visual similarity of different dogs, the *visual* part of each pattern (elements 9-24) was a distortion of a particular pattern (+1 -1 +1 +1 -1 -1 -1 -1 +1 +1 +1 +1 -1 -1 -1) which represented the appearance of a prototypical dog. The distortions corresponding to the varying appearance of individual dogs were made by randomly changing the sign of each of the 16 elements with a probability of 0.2.

50 different patterns were constructed, corresponding to the child meeting 50 different dogs. The weights in the autoassociator were initialised to random values

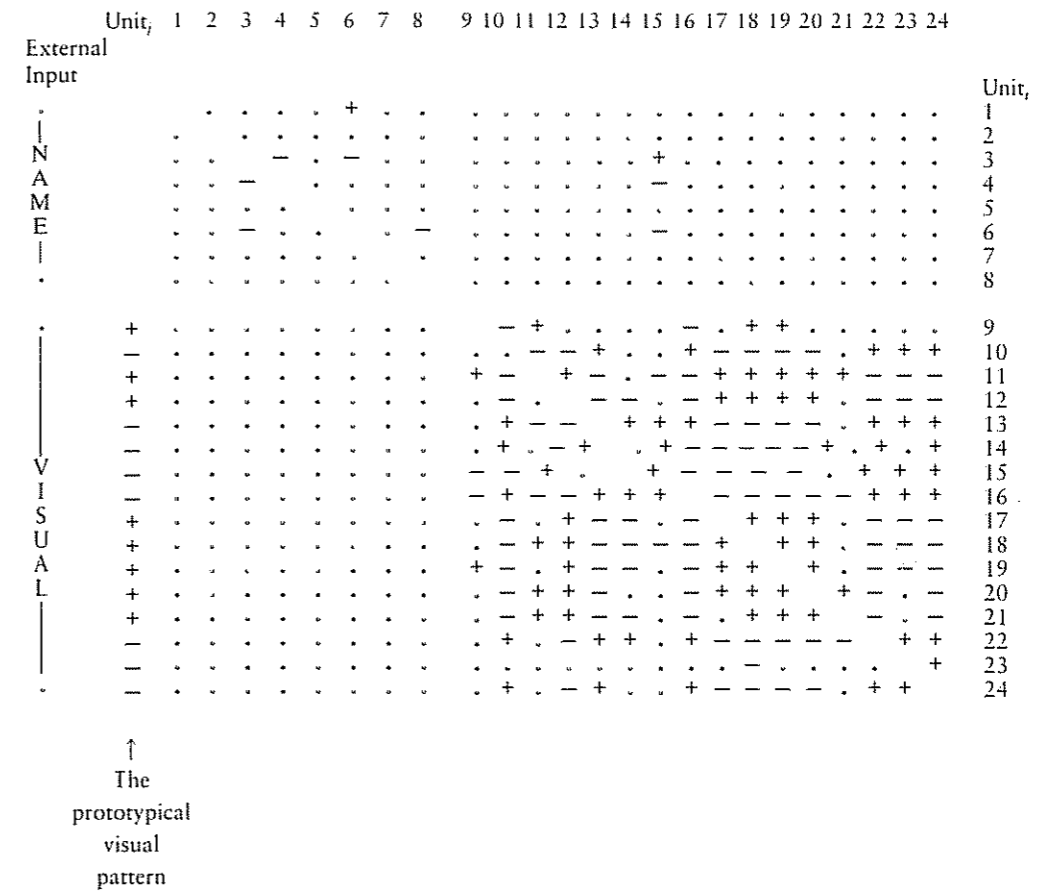


Figure 4.6 The weight matrix which develops as a 24 unit autoassociator learns about the names and appearance of 50 dogs. The first 8 elements in the input string represent the name of the dog; elements 9-24 represent its visual appearance. The appearance of each dog is a random variation of the prototypical pattern. The rows show the weights from each other unit to unit *i*. Weights with a value close to zero are indicated by dots. (Based on McClelland and Rumelhart 1985.)

with a mean of zero. Each pattern was presented as an external input to the matrix, activity was allowed to cycle until a settled pattern was reached, and then the weights were changed in the manner represented by equation 4.3. Figure 4.6 shows the weights which the matrix acquires at the end of these learning experiences. Since the autoassociator has 24 units this matrix has 24x23 weights (no unit synapses to itself). As before, the rows represent the dendrites of each unit, with the weights of the connections from each other unit to the unit represented in that row. Units 1-8 receive the part of the external input which represents the name; units 9-24 receive the part which represents the visual image.

The matrix can be imagined as consisting of four blocks. (To help identify them a

blank strip has been left around them in the matrix.) An 8×8 block in the top left hand corner shows the weights of connections from elements in the *name* part of the input pattern (units 1–8) to each other. The 16×16 block in the bottom right hand part of the matrix shows the weights of elements in the *visual* part of the external input pattern (units 9–24) to each other. The two rectangular 8×16 blocks show the weights of the connections from *name* to *visual* elements and vice versa.

It is immediately apparent that most of the cells in the blocks of the matrix representing the weights involving the *name* elements are close to zero. Remember that a weight in the autoassociator represents the correlation between the corresponding elements in the input, summed across all the 50 different input patterns. Since the 'name' elements in different patterns are random strings of +1s and -1s the correlation between any particular pair of elements summed across all input patterns is close to zero.

In contrast, for the block of the matrix corresponding to correlations between elements in the *visual* part of the pattern, positive and negative weights develop. Since all the *visual* patterns are derived from a common prototype, weights corresponding to the correlations between input elements in the prototypical pattern begin to emerge, despite the random variations from pattern to pattern. This can be seen by checking the weights along each input line against the value for that unit in the prototypical pattern shown to the left of the matrix. Consider, for example, the second element in the prototypical pattern. It is negative. Positive connections have developed to unit 10, which takes this element as its external input, from units 13, 16, 22, 23 and 24. These units have the same polarity as unit 10 in the prototypical pattern. Negative connections have developed from units 11, 12, 17, 18, 19 and 20 which have the opposite polarity to unit 10 in the original pattern. The external input which would give the strongest response on unit 10 would be the prototypical pattern itself because the whole of the internal input would be trying to push it in the same direction as the external input. Note that this happens despite the fact that the autoassociator has never had a chance to learn the prototypical pattern directly, because it has never been presented. The matrix has 'discovered' the structure of the prototypical pattern on which the different inputs were based. It did this not by some unspecified, high-level 'prototype extraction process' but as an automatic consequence of the way that the learning rule changed the weights after each stimulus presentation.

This result echoes a well-known experiment of Posner and Keele (1968). They presented subjects with patterns generated by random distortion of a prototypical pattern. Later the subjects were asked to judge whether patterns were old (i.e. in the first set) or new (i.e. a new set of random distortions). When presented with the prototype (which they had *not* seen before) subjects were as likely to claim it was old as they were when presented with a pattern they had seen before.

Learning different prototypes on the same matrix

The previous section showed how an autoassociator memory can come to represent the central tendency of a set of related examples without experiencing the prototype itself. Since people appear to do this, it is an important property of a memory system to simulate. But the simulation avoids the central problem which would face the human cognitive system in the real world. In the previous simulation the auto-associator was only shown dogs. So the fundamental problem of stimulus categorisation had been solved for it. Some part of the complete system would have to know what a dog was so that only examples of dogs were directed to this particular memory store. This can be done *after* the category is formed, but how did the category come to be formed in the first place? If this demonstration is to be a viable model of prototype extraction in people it must be capable of extracting prototypes when exemplars from different categories are presented to the same memory without any indication that they came from any predefined category.

McClelland and Rumelhart explored the autoassociator's ability to do this with a simulation in which they imagined that the child was learning about dogs, cats and bagels. They produced a visual prototype for each of these (a specific pattern for the units 9–24 which codes the prototypical object's appearance). The prototypes for dog and cat were quite similar but that for bagel unlike either of them. The prototypical patterns are shown in figure 4.7. The dog and cat prototype patterns share the same value for 12 of the 16 input elements. The bagel pattern is unlike either of them, sharing only 8 of the 16 elements with either of the other patterns (the number which would be expected by chance for two inputs that were not visually related.) The name input elements were left blank to simulate untutored visual experience.

As before, they then generated 50 distortions of each prototype to represent specific experiences of dogs, cats and bagels. These were presented to the matrix, with the weights incremented after each presentation. The 'probes' in figure 4.7 are test inputs applied to the autoassociator after the learning trials. In each case it is an incomplete pattern, resembling a distinctive part of one of the three prototypical input patterns. Figure 4.7 shows that pattern completion takes place for each fragment. In each case all 16 of the visual units in the autoassociator adopt the pattern of activity corresponding to the visual prototype pattern from which the fragment was taken.

As we saw earlier, pattern completion requires that the autoassociator has stored the information about the complete pattern from which the fragment was taken. So figure 4.7 shows that the autoassociator has stored a representation of all three prototypes simultaneously. It has done this without being given any information during acquisition that the patterns came from different prototypes. Thus the auto-

'Dog'	Probe	R	'Cat'	Probe	R	'Bagel'	Probe	R	Unit
+1		+0.3	+1		+0.3	+1		+0.2	9
-1		-0.3	-1		-0.3	+1		+0.3	10
+1		+0.3	+1		+0.3	-1		-0.4	11
+1		+0.3	+1		+0.3	+1		+0.3	12
-1		-0.3	-1		-0.3	-1		-0.3	13
-1		-0.4	-1		-0.3	+1		+0.3	14
-1		-0.3	-1		-0.3	+1		+0.3	15
-1		-0.3	-1		-0.3	-1		-0.3	16
+1	+1	+0.6	+1	+1	+0.6	+1	+1	+0.6	17
+1	+1	+0.5	-1	-1	-0.5	-1	-1	-0.6	18
+1	+1	+0.6	+1	+1	+0.6	-1	-1	-0.6	19
+1	+1	+0.5	-1	-1	-0.5	+1	+1	+0.6	20
+1		+0.3	+1		+0.3	+1		+0.3	21
-1		-0.2	+1		+0.2	+1		+0.3	22
-1		-0.3	-1		-0.3	+1		+0.3	23
-1		-0.2	+1		+0.2	-1		-0.3	24

Figure 4.7 'Dog', 'Cat' and 'Bagel' are the prototypical patterns from which the three sets of exemplars for training the matrix are derived. *Probe* shows the pattern fragments with which the matrix was tested after learning. *R* shows the activity pattern of the autoassociator to these inputs (Based on McClelland and Rumelhart 1985.)

associator offers a solution to one of the central mysteries of cognition. With the right memory architecture, mere experience is enough to ensure that appropriate categorisation of stimulus events will take place without an homunculus to tell the system what would be an appropriate classification.³ We will see an example of this in action in the section on vocabulary development in chapter 9.

Further reading

In this chapter we have looked at some of the central points covered by the McClelland and Rumelhart simulation. The original article investigates the ability of autoassociators to mimic a wider range of memory phenomena. An alternative approach to concept formation in connectionist networks can be followed in Dienes (1992).

Autoassociation exercises with tlearn

Open the project called **aa** in the **Chapter Four** folder/directory. This project uses the **tlearn** network configuration file **aa.cf** that defines a network similar to that shown in figure 4.1, i.e. 8 output units with feedback connections from each unit to

³This ability is not restricted to autoassociators. In chapter 6 we will see that competitive nets also perform untutored categorisation of inputs.

Figure 4.8 **aa.cf** and **aa.data** files.

the input of each other unit. All the connections are initially set to zero. Do not worry about the details of the **aa.cf** (network configuration) file. It simply reveals the gymnastics we have performed to make **tlearn** behave like an autoassociator.⁴ To see what the network looks like and work out how it simulates the autoassociator in figure 4.1 use the Network Architecture display. (It is easier to see what is happening if you activate the bias button.)

When you open the **aa** project, you will also open the **aa.data** and **aa.teach** files that will be used to train the network. These files are shown in figure 4.8. The **aa.data** file contains the two patterns presented to the autoassociator in the section 'Properties of autoassociator memories' (1 -1 1 -1 1 1 -1 -1) and (1 1 -1 -1 -1 1 1 -1 1). Each pattern is repeated 10 times for each of the cycles of processing necessary for the output units to reach a stable level of activity. The **aa.teach** file is identical to the **aa.data** file since we are doing autoassociation.

Training the network

Activate the Training Options dialogue box from the Network menu. Use the expanded version of the Training Options dialogue box as shown in figure 4.9. If you do not see this, press the More button. Set the number of training sweeps to 20 so

⁴**tlearn** does not implement the autoassociation learning algorithm described in the earlier part of this chapter. We have simulated the autoassociator by creating feedback connections to other units via a bank of 8 linear units that keep an updated record of the activities of the output units on the previous cycle. This is equivalent to having 8 units with feedback connections to each other.