Chapter 2

Associative Memory

2.1 Objectives

After this Chapter, you should

1. understand the terms "associative memory", "autoassociation", "heteroassociation", "recall" and "crosstalk".
2. be able to create a simple memory matrix using Hebb's rule.
3. understand the significance and reasons for crosstalk.
4. be able to use Lyapunov's method in simple situations.
5. be able to use the vector and matrix notation for simple memories.

2.2 Memory

We cannot remember an event before it happens. Therefore an event happens, some change takes place in our brains so that subsequently we can remember the event. So memory is inherently bound up in the learning process.

In this chapter, we will meet a simple example of associative memory: the type of memory which allows us to associate a particular taste with a particular sight and, at a different level, a particular tune with a particular person etc. Therefore we will develop an artificial neural network (see Figure 2.1) which performs a mapping from an input vector (e.g. a set of sights) to an output vector (e.g. a taste). The memory which links these is found in the weights which join input to output and the process of learning this memory is equivalent to changing the weights.

There are two types of association:

Autoassociation: where a pattern vector is associated with itself. This is most useful for pattern completion where a partial pattern (a pair of eyes) or a noisy pattern (a blurred image) is associated with its complete and accurate representation (the whole face).

Heteroassociation: where a vector is associated with another vector which may have different dimensionality. We may still hope that a noisy or partial input vector will retrieve the complete output vector.

This chapter relates to Haykin Chapter 3, pages 90-114. The exercises on p114 are very good.

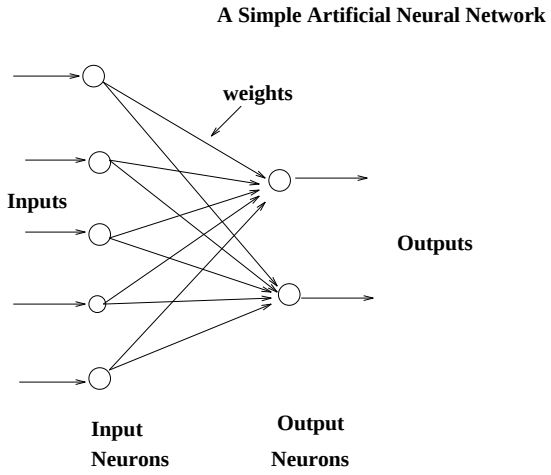

Figure 2.1: This simple two layer network can be used to hold memories associating input data with outputs.

2.3 Heteroassociation

Let us assume that the input vector, $\mathbf{x}$ is n-dimensional and the output vector, $\mathbf{y}$ is m-dimensional. Then the input data (the stimulus) is represented by the firing of n different input neurons and the output vector (the response) is held on m output neurons. There is a set of weights between the input neurons and the output neurons which is modelled by a matrix, $\mathbf{W}$ which is $m * n$. Then we have

$$\begin{aligned} \mathbf{x} &= (x_1, x_2, ..., x_n)^T \\ \mathbf{y} &= (y_1, y_2, ..., y_m)^T \end{aligned}$$

linked by $\mathbf{W}$, where w_{ij} is the weight from x_j to y_i. Then if we present $\mathbf{x}_k$, the k^{th} input pattern, to the network, we wish to recall the k^{th} output pattern, $\mathbf{y}_k$. Therefore we must have the output neurons firing in the appropriate pattern when we input $\mathbf{x}_k$. i.e. $\mathbf{y}_k = \mathbf{W}\mathbf{x}_k$ for all k.

$$\begin{bmatrix} y_{k1} \\ y_{k2} \\ \vdots \\ y_{km} \end{bmatrix} = \begin{bmatrix} w_{11}(k) & w_{12}(k) & \cdots & w_{1n}(k) \\ w_{21}(k) & w_{22}(k) & \cdots & w_{2n}(k) \\ \vdots & \vdots & \vdots & \vdots \\ w_{m1}(k) & w_{m2}(k) & \cdots & w_{mn}(k) \end{bmatrix} \begin{bmatrix} x_{k1} \\ x_{k2} \\ \vdots \\ x_{kn} \end{bmatrix} \tag{2.1}$$

Now this set of weights would, we hope, give us the required output vector (or associated memory) when we enter the k^{th} input vector. But we wish to recall p distinct patterns when presented with p distinct stimuli or input patterns. Therefore we will use an $m * n$ memory matrix, $\mathbf{M}$, defined by

$$\begin{aligned} \mathbf{M} &= \sum_{k=1}^{p} \mathbf{W}(k) \text{ where} \\ \mathbf{W}(k) &= \begin{bmatrix} w_{11}(k) & w_{12}(k) & \cdots & w_{1n}(k) \\ w_{21}(k) & w_{22}(k) & \cdots & w_{2n}(k) \\ \vdots & \vdots & \vdots & \vdots \\ w_{m1}(k) & w_{m2}(k) & \cdots & w_{mn}(k) \end{bmatrix} \end{aligned}$$

the matrix memory for the k^{th} pattern alone.

2.4 Creating the Memory Matrix

Firstly we note that we can recursively calculate the memory matrix using

$$\mathbf{M}(k) = \mathbf{M}(k-1) + \mathbf{W}(k) \tag{2.2}$$

if we define $\mathbf{M}(0)$ to be 0, the zero matrix (i.e. the memory matrix initially holds no memories). Therefore if we have a memory matrix holding k-1 patterns we can simply add in the memories for the k^{th} pattern. Thus our problem is one of calculating $\mathbf{W}(k)$, the matrix of memories of the k^{th} pattern.

The simplest way of doing this is by using Hebb's learing rule which we shall meet in detail in Chapter 5. Hebbian learning is so-called after Donald Hebb who in 1949 conjectured:

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

We generally equate the efficiency of a synapse in a biological network with the weight of a connection in an artificial neural network. Therefore the rule is written as $w_{ij}(k) = x_j y_i$. That is the weight between each input and output neuron is increased proportional to the magnitude of the simultaneous firing of these neurons. We can write this in matrix terms as

$$\begin{aligned}
\mathbf{W}(k) &= \mathbf{y}_k \mathbf{x}_k^T \\
&= \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix} \\
&= \begin{bmatrix} y_1x_1 & y_1x_2 & \cdots & y_1x_n \\ y_2x_1 & y_2x_2 & \cdots & y_2x_n \\ \vdots & \vdots & \vdots & \vdots \\ y_mx_1 & y_mx_2 & \cdots & y_mx_n \end{bmatrix}
\end{aligned}$$

Note that an n-dimensional input vector and an m-dimensional output vector give an m*n dimensional weight matrix. If we were associating the input patterns, $\mathbf{y}$ with the output patterns $\mathbf{x}$, we would have the $n * m$ weight matrix which is the transpose of the current matrix.

We can now see that the output from the system, the vector $\mathbf{y}$, is

$$\begin{aligned}
\mathbf{y} &= W(k)\mathbf{x}_k \\
&= \mathbf{y}_k \mathbf{x}_k^T \mathbf{x}_k \\
&= \mathbf{y}_k
\end{aligned}$$

if the input patterns have been normalised to length 1.[1]

Therefore we can see that using $\mathbf{W}(k)$ we get a truly associative memory which will associate the correct output vector with the k^{th} input vector.

Therefore the sequence $\mathbf{W}(\mathbf{k})$ seems to fulfill our requirements for an associative memory. However, we have ignored the possibility that when we add the successive W(k), we may get interference between sets of patterns. This interference is usually called **crosstalk**.

[1] When the input patterns have not been normalised, our outputs are in the same direction as the associated vector $\mathbf{y}_k$ but may be larger or smaller

2.5 Recall and Crosstalk

We can investigate the effect of crosstalk as follows. Let $\mathbf{M} = \sum_{k=1}^{p} \mathbf{W}(k)$. Then let the i^{th} input pattern, $\mathbf{x}_i$ evoke the response $\mathbf{y}$. Then we have

$$\begin{aligned}
\mathbf{y} &= \mathbf{M}\mathbf{x}_i \\
&= \sum_k \mathbf{W}(k)\mathbf{x}_i \\
&= \sum_k \mathbf{y}_k \mathbf{x}_k^T \mathbf{x}_i \\
&= \mathbf{y}_i \mathbf{x}_i^T \mathbf{x}_i + \sum_{k \neq i} \mathbf{y}_k \mathbf{x}_k^T \mathbf{x}_i \\
&= \mathbf{y}_i + \mathbf{c}_i
\end{aligned}$$

where $\mathbf{c}_i$ is the crosstalk due to the effect of the other patterns. These two terms can be thought of as signal and noise respectively so what we would like would be to minimise the noise so that the recalled pattern is as close to the associated pattern $\mathbf{y}_i$ as possible. One way to do this is to make the input patterns orthogonal (or as orthogonal as possible). Note that the cosine of the angle between two (normalised) input vectors is given by

$$\cos(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{x}_i . \mathbf{x}_j}{\| \mathbf{x}_i \| . \| \mathbf{x}_j \|} = \frac{\mathbf{x}_i^T \mathbf{x}_j}{\| \mathbf{x}_i \| . \| \mathbf{x}_j \|} = \mathbf{x}_i^T \mathbf{x}_j \tag{2.3}$$

since the input vectors have been normalised to have unit length, $\| \mathbf{x}_i \| = 1, \forall i$. So we can see that the crosstalk term

$$\mathbf{c}_i = \sum_{k \neq i} \mathbf{y}_k \cos(\mathbf{x}_k, \mathbf{x}_i) \tag{2.4}$$

which will be minimised when the angle between the vectors is as close to a right angle as possible and will be zero when the vectors are orthogonal. Note that if the vectors are not orthogonal, there will inevitably be some crosstalk between the vectors. We say that the associative memory works perfectly only when the input patterns form an **orthonormal** set i.e.

$$\begin{aligned}
\mathbf{x}_k^T \mathbf{x}_i &= 0 \text{ when } i \neq k \\
\mathbf{x}_i^T \mathbf{x}_i &= 1
\end{aligned}$$

2.5.1 Example

We would like to train an artificial neural network to differentiate between sets of patterns on a square grid of pixels. Consider Figure 2.2. We show 4 patterns two of which we wish to associate with pattern "one" and two of which we will identify as pattern "two".

To train a neural network to differentiate between these two sets of patterns (and we assume that we have many examples of "ones" and "twos"), we set up a neural network with 49 inputs and 2 outputs. Each of the 49 inputs corresponds to one pixel so that if the example drawn on the grid is such that pixel 18 (say) is more than half inside the figure, the 18^{th} element of the vector is a 1; if the 18^{th} pixel on the other hand is more than half outside the figure, the 18^{th} element of the vector is a 0. Therefore our input data to the neural network is a 49 dimensional vector composed only of ones and zeroes.

The output vector is only a two dimensional vector which will be $(1, 0)^T$ if the input pattern is a "one" and $(0, 1)^T$ if the input pattern is "two".

Then for the k^{th} pattern,

$$\mathbf{W}(k) = \mathbf{y}_k \mathbf{x}_k^T \tag{2.5}$$

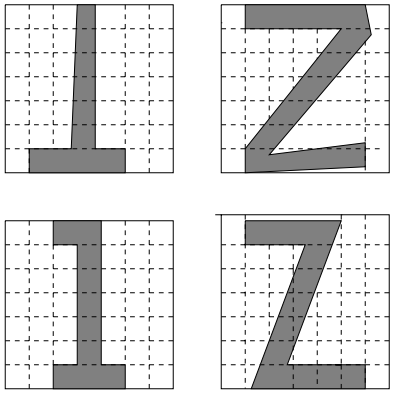

Figure 2.2: Sample patterns of ones and twos. Each pattern is converted to a 49 dimensional vector in which a dimension has the value one if more than half the pixel is contained in the figure and 0 otherwise

So if the pattern is a "one", the second row of W is all zeros while if the pattern is a "two" the first row is a row of zeroes. Also the other row in each case is simply a row containing the 49 dimensional input vector. e.g. for the first "two" pattern above

$$\mathbf{W} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 1 & \cdots & 1 & 0 \end{bmatrix} \tag{2.6}$$

So if we have p patterns

$$\begin{aligned} \mathbf{M} &= \sum_{k=1}^{p} \mathbf{W}(k) \\ &= \begin{bmatrix} \sum_{i \in P_1} \mathbf{x}_i^T \\ \sum_{i \in P_2} \mathbf{x}_i^T \end{bmatrix} \end{aligned}$$

where P_i is the set of training patterns of type i.

So the fact that the y-values are orthogonal to one another means that, in the weight matrix, the memories associated with "ones" do not interfere with the memories associated with "twos". However all the "ones" memories are interfering with each other; in other words, the memory which will be retained will be that which is formed by the crosstalk between patterns of the same type - the crosstalk is actually helpful in forming an idealised version of "one" and of "two".

Thus we can see that this matrix memory is proportional to the mean of the training patterns of each type which it meets during the network training. Therefore the ability of this type of network to recognise new examples of "ones" and "twos" depends on how typical were the examples on which it was trained. In general, if we put an input vector of a noisy "one" (see Figure 2.3) into the network we would hope that the output vector would more closely resemble $(1,0)^T$ than $(0,1)^T$ i.e. we should be able to identify which pattern the input is closest to by looking at the activation of the output vector. The strength of firing of the appropriate output neuron gives a measure of how closely the input corresponds to the prototypical input vector of that class.

More information on associative memories can be found in Haykin p90-98.

2.6 Bidirectional Autoassociative Memories

The usual aim of autoassociation is to place a memory in a network in such a way that a noisy or partial memory will cause the original complete memory to be recalled. This will not generally be the case with

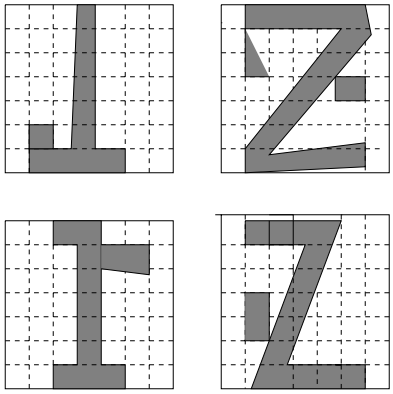

Figure 2.3: Noisy versions of the previous data which when input to the network will be classed as "ones" and "twos" as appropriate.

the simple memory outlined above since if we have created the mapping $\mathbf{x}_k \to \mathbf{y}_k$, then if we enter a noisy version of $\mathbf{x}_k$, say $\mathbf{x}'_k = \mathbf{x}_k + \mu_k$, then we will not in general retrieve the memory $\mathbf{y}_k$ but will retrieve $\mathbf{M}\mathbf{x}'_k = \mathbf{M}\mathbf{x}_k + \mathbf{M}\mu_k = \mathbf{y}_k + \mathbf{M}\mu_k$ since we have a linear memory.

One possibility is to allow the activation to pass backward and forward through the matrix M in the hope that the divergence from the original memory will decrease in time. Therefore using $\mathbf{x}_k(0)$ to denote the k^{th} input pattern at time 0, we would get a set of iteratively closer solutions:

$$\begin{aligned}
\mathbf{y}_k(0) &= \mathbf{M}\mathbf{x}_k(0) \text{ forward pass} \\
\mathbf{x}_k(1) &= \mathbf{M}^T\mathbf{y}_k(0) \text{ backward pass} \\
\mathbf{y}_k(1) &= \mathbf{M}\mathbf{x}_k(1) \text{ forward pass} \\
\mathbf{x}_k(2) &= \mathbf{M}^T\mathbf{y}_k(1) \text{ backward pass} \\
&\vdots \\
\mathbf{y}_k(t-1) &= \mathbf{M}\mathbf{x}_k(t-1) \text{ forward pass} \\
\mathbf{x}_k(t) &= \mathbf{M}^T\mathbf{y}_k(t-1) \text{ backward pass}
\end{aligned} \tag{2.7}$$

Now, this sequence will not be useful for the purely linear relationships described above since e.g. $\mathbf{x}_k(1) = \mathbf{M}^T\mathbf{y}_k(0) = \mathbf{M}^T\mathbf{M}\mathbf{x}_k(0) = \mathbf{x}_k(0)$ (see Question 4) i.e. we have gained nothing from the back and forth activation passing. Therefore we introduce a non-linear activation function at both the input and output neurons; we introduce the simplest - a threshold function. Then

$$\begin{aligned}
\mathbf{y}_{k,j} &= 1, \text{ if } \mathbf{M}_j.\mathbf{x} > 0 \\
\mathbf{y}_{k,j} &= 0, \text{ if } \mathbf{M}_j.\mathbf{x} < 0 \\
\mathbf{x}_i &= 1, \text{ if } \mathbf{M}_i^T.\mathbf{y} > 0 \\
\mathbf{x}_i &= 0, \text{ if } \mathbf{M}_i^T.\mathbf{y} < 0
\end{aligned}$$

where we have used $\mathbf{y}_{k,j}$ to designate the output of the j^{th} output neuron to the k^{th} input pattern and $\mathbf{M}_j$ is the j^{th} row of the matrix $\mathbf{M}$.

So all patterns are now binary 1/0 patterns. It will be shown by the method of Lyapunov that every matrix M is bidirectionally stable (i.e. the process 2.7 converges to a stable solution) for every input.

Student	Hard	Endur	Smart	Consc	Intu	Crea	Course	Math	Comp	Phys	EE
A_1			X	X		X	B_1	X			X
A_2	X			X	X		B_2	X	X		
A_3			X	X	X		B_3	X		X	

Table 2.1: Students attributes and their subjects.

2.6.1 An Example

Let us create a BAM for the pair of memories shown in Table 2.1 in which an X in any position identifies that the student has that quality or that the course contains that class. We wish to create a memory which associates Student i with Course i.

First we may represent each student and course by a vector e.g.

$$\begin{aligned} A_1 &= (0,0,1,1,0,1)^T & B_1 &= (1,0,0,1)^T \\ A_2 &= (1,0,0,1,1,0)^T & B_2 &= (1,1,0,0)^T \\ A_3 &= (0,0,1,1,1,0)^T & B_3 &= (1,0,1,0)^T \end{aligned}$$

The BAM is easier to construct with a bipolar coding and so we use

$$\begin{aligned} X_1 &= (-1,-1,1,1,-1,1)^T & Y_1 &= (1,-1,-1,1)^T \\ X_2 &= (1,-1,-1,1,1,-1)^T & Y_2 &= (1,1,-1,-1)^T \\ X_3 &= (-1,-1,1,1,1,-1)^T & Y_3 &= (1,-1,1,-1)^T \end{aligned}$$

We can now find correlation matrices:

$$X_1 Y_1^T = \begin{bmatrix} -1 \\ -1 \\ 1 \\ 1 \\ -1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & -1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 1 & 1 & -1 \\ -1 & 1 & 1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 \\ -1 & 1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

$$X_2 Y_2^T = \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \\ 1 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & 1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \end{bmatrix}$$

$$X_3 Y_3^T = \begin{bmatrix} -1 \\ -1 \\ 1 \\ 1 \\ 1 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & -1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} -1 & 1 & -1 & 1 \\ -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 \end{bmatrix}$$

Let us consider initially a matrix containing only the first two pairs of memories, $A_1 - B_1$ and $A_2 - B_2$.

Then we get

$$M = X_1 Y_1^T + X_2 Y_2^T = \begin{bmatrix} 0 & 2 & 0 & -2 \\ -2 & 0 & 2 & 0 \\ 0 & -2 & 0 & 2 \\ 2 & 0 & -2 & 0 \\ 0 & 2 & 0 & -2 \\ 0 & -2 & 0 & 2 \end{bmatrix} \tag{2.8}$$

which can be used in recall with

$$A_1^T M = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 & 2 & 0 & -2 \\ -2 & 0 & 2 & 0 \\ 0 & -2 & 0 & 2 \\ 2 & 0 & -2 & 0 \\ 0 & 2 & 0 & -2 \\ 0 & -2 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 2 & -4 & -2 & 4 \end{bmatrix}$$

which can be thresholded to give (1001). We can similarly feed B_1 to the network to get A_1 using $B_1^T M^T$. Alternatively we can use MB_1 e.g.

$$MB_1 = \begin{bmatrix} 0 & 2 & 0 & -2 \\ -2 & 0 & 2 & 0 \\ 0 & -2 & 0 & 2 \\ 2 & 0 & -2 & 0 \\ 0 & 2 & 0 & -2 \\ 0 & -2 & 0 & 2 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -2 \\ -2 \\ 2 \\ 2 \\ -2 \\ 2 \end{bmatrix}$$

which can again be thresholded to give A_1. In addition we can input slightly wrong stimuli and hope to get a correct response - this is the pattern completion properties of this type of network. Let us input $A_*^T = (1, 0, 1, 1, 0, 1)$. Then we have

$$\begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 & 2 & 0 & -2 \\ -2 & 0 & 2 & 0 \\ 0 & -2 & 0 & 2 \\ 2 & 0 & -2 & 0 \\ 0 & 2 & 0 & -2 \\ 0 & -2 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 2 & -2 & -2 & 2 \end{bmatrix}$$

which is again thresholded to give B_1 which can be fed back as before to retrieve A_1. So a noisy version of a pattern can be fed forward and back through the network to give an accurate version of the pattern.

Now we add in the third pattern $X_3 - Y_3$. Then M now becomes

$$\begin{bmatrix} 0 & 2 & 0 & -2 \\ -2 & 0 & 2 & 0 \\ 0 & -2 & 0 & 2 \\ 2 & 0 & -2 & 0 \\ 0 & 2 & 0 & -2 \\ 0 & -2 & 0 & 2 \end{bmatrix} + \begin{bmatrix} -1 & 1 & -1 & 1 \\ -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 3 & -1 & -1 \\ -3 & 1 & 1 & 1 \\ 1 & -3 & 1 & 1 \\ 3 & -1 & -1 & -1 \\ 1 & 1 & 1 & -3 \\ -1 & -1 & -1 & 3 \end{bmatrix}$$

Now if we try to recall B_1 from A_1 we get

$$A_1^T M = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} * \begin{bmatrix} -1 & 3 & -1 & -1 \\ -3 & 1 & 1 & 1 \\ 1 & -3 & 1 & 1 \\ 3 & -1 & -1 & -1 \\ 1 & 1 & 1 & -3 \\ -1 & -1 & -1 & 3 \end{bmatrix} = \begin{bmatrix} 3 & -5 & -1 & 3 \end{bmatrix}$$

which is again precisely B_1. Similarly with A_2, A_3 and even the noisy version of A_1, A_*. However the vectors used in the example were chosen with care, notice how the Y values are pairwise orthogonal to one another. As we add more and more memories to our matrix we will experience crosstalk and be unable to retrieve complete versions of our memories. Add in a fourth memory to the matrix M and attempt to retrieve all four memories.

2.6.2 Lyapunov's Method

Lyapunov's method is

1. Find a function which changes monotonically when the system is changed. i.e. find a function which consistently decreases (increases) *every time* the state of the network changes.
2. Show that the function is bounded below (above).

If we can find such a function then we know that the learning rules cause convergence though we cannot say to which final values convergence will take place.

For the Bidirectional Associative Memory (BAM), we choose the function E(**x**,**y**) such that

$$\begin{aligned} E(\mathbf{x}, \mathbf{y}) &= -\mathbf{x}\mathbf{M}\mathbf{y}^T \\ &= -\sum_i \sum_j x_i y_j m_{ij} \end{aligned}$$

Then E(**x**,**y**) is bounded below since $E(\mathbf{x}, \mathbf{y}) \geq -\sum_i \sum_j |m_{ij}|$.

Now consider the effect of a change at the input(**x**) neurons. Let Δx_i be the change due to the feedback from the previous output (y) neurons at the i^{th} input neuron. Then Δx_i is -1, 0 or 1 at any iteration, and so

$$\begin{aligned} \Delta E(\mathbf{x}, \mathbf{y}) &= -\Delta - \mathbf{x}\mathbf{M}\mathbf{y}^T \\ &= -\sum_i \Delta x_i \sum_j y_j m_{ij} \end{aligned}$$

Now if $\sum_j y_j m_{ij} > 0$ then $\Delta \mathbf{x}_i > 0$, and so $\Delta E(\mathbf{x}_i, \mathbf{y}) < 0$. Similarly, if $\sum_j y_j m_{ij} < 0$ then $\Delta \mathbf{x}_i < 0$, and so $\Delta E(\mathbf{x}_i, \mathbf{y}) < 0$. A similar argument applies to change at the output neurons. In other words, any change will cause E to decrease and we have seen that E is bounded below so that the network must converge. Therefore regardless of the actual identity of matrix M the network will stabilise.

Example

As an extension of the previous example, we wish to teach a network how to count. Specifically if we input x we wish it to output x+1. Therefore both input and output vectors are 49 dimensional binary vectors and we train the network to associate an input vector representing "one" with an output vector representing "two", an input vector representing "two" with an output vector representing "three" etc..

Pattern No.	Input vector	Output vector
1	$(1,0,0)^T$	$(2,3,-1)^T$
2	$(0,1,0)^T$	$(1,-1,3)^T$
3	$(0,0,1)^T$	$(0,1,-3)^T$

Table 2.2: Three input-output pairs

Now when we enter a noisy version of a "one", it will elicit a response of a noisy version of a "two" which will be fed back to the inputs to elicit a less noisy version of a "one" etc. From the above, we know that the network will settle into a stable state. Provided the initial vector is not too noisy (and so could not be mistaken for e.g. a "seven") the stable pattern will be "one"-"two".

2.7 Conclusion

We have met two simple memories and differentiated between autoassociation and heteroassociation. We can make a further distinction between the networks presented in this Chapter: the first is a static memory whereas the second network, the BAM, is of a dynamic nature - there is a certain settling time in which the activation within the network passes back and forth eventually settling, we hope, on one of the remembered patterns. Because of this we can view the activation in the network as "short term memory" while the "long term memory" is held in the weights of the network.

The major points to note from this chapter

- The memory is contained in the weights.
- The memory is distributed.
- Both the stimulus (input pattern) and response (output pattern) are high dimensional vectors
- The recalled memory consists of a pattern of activities across the output neurons.
- The memory is robust
- There may be crosstalk between individual memories (associations)

2.8 Exercises

1. Calculate the weights used in an associative memory with which we will associate the three input patterns and output patterns shown in Table 2.2. (Objectives 2, 5).

   (a) Show that there is no crosstalk. (Objectives 1,3, 5).

   (b) Calculate the weight matrix of the memory. (Objectives 2 5).

   To use this set of patterns as a backwards memory we must normalise the output patterns and threshold at the largest.

2. Construct a simple BAM with the patterns given in Table 2.3. Show that the network performs perfect autoassociation with the given patterns. Investigate what happens when we enter the patterns

   (a) $(1,1,1,-1,1,-1)^T$

   (b) $(-1,1,-1,1,-1,1)^T$

Pattern No.	Input vector	Output vector
1	$(1,-1,1,-1,1,-1)^T$	$(1,1,-1,-1)^T$
2	$(1,1,1,-1,-1,-1)^T$	$(1,-1,1,-1)^T$

Table 2.3: Two input-output vectors with which to construct a BAM

(Objectives 2, 3, 5).

3. Given a set of orthogonal m-dimensional vectors $\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n$, calculate the network memory which will perform autoassociation. What is the maximum number of vectors, n, which can be stored in such a memory? (Objectives 2, 3,5).

4. To show the need for a non-linearity in the BAM, we have assumed that $MM^T = I$, the identity matrix. Prove this. (Objective 5).

5. Explain with diagrams Lyapunov's method. (Objective 4).