CHAPTER III

Neural Networks as Associative Memory

One of the primary functions of the brain is associative memory. We associate the faces with names, letters with sounds, or we can recognize the people even if they have sunglasses or if they are somehow elder now.

Associative memories can be implemented either by using feedforward or recurrent neural networks. Such associative neural networks are used to associate one set of vectors with another set of vectors, say input and output patterns. The aim of an associative memory is, to produce the associated output pattern whenever one of the input pattern is applied to the neural network. The input pattern may be applied to the network either as input or as initial state, and the output pattern is observed at the outputs of some neurons constituting the network. According to the way that the network handles errors at the input pattern, they are classified as ***interpolative*** and ***accretive*** memory. In the interpolative memory it is allowed to have some deviation from the desired output pattern when added some noise to the related input pattern. However, in accretive memory, it is desired the output to be exactly the same as the associated output pattern, even if the input pattern is noisy. Another classification of associative memory is such that while the memory in which the associated input and output patterns differ are called ***heteroassociative*** *memory*, it is called ***autoassociative*** memory if they are the same.

In this chapter, first the basic definitions about associative memory is given and then it is explained how neural networks can be made linear associators so as to perform as interpolative memory. Next it is explained how the Hopfield network can be used as autoassociative memory and then Bipolar Associative Memory network, which is designed to operate as heteroassociative memory, is introduced.

3.1. Associative Memory

In an associative memory, we store a set of patterns $\boldsymbol{\mu}^k$, k=1...K, so that the network responds by producing whichever of the stored patterns most closely resembles the one presented to the network

Here we need a measure for defining resemblance of the patterns. For this purpose the norms that were introduced in Section 2.1 may be used. While Euclidean distance is convenient for the continuous valued pattern vectors, Hamming distance, which gives the number of mismatched components, is more appropriate for patterns with binary or bipolar entries.

Suppose that the stored patterns, which are called exemplars or memory elements, are in the form of pairs of associations, $\boldsymbol{\mu}^k=(\mathbf{u}^k,\mathbf{y}^k)$, $\mathbf{u}^k \in R^N$, $\mathbf{y}^k \in R^M$, k=1..K. According to the mapping $\varphi: R^N \rightarrow R^M$ that they implement, we distinguish the following types of associative memories:

- **Interpolative associative memory**: when $\mathbf{u}=\mathbf{u}^r$ is presented to the memory it responds by producing $\mathbf{y}^r$ of the stored association. However if $\mathbf{u}$ differs from $\mathbf{u}^r$ by an amount of $\boldsymbol{\varepsilon}$, that is if $\mathbf{u}=\mathbf{u}^r+\boldsymbol{\varepsilon}$ is presented to the memory, then the response differs from $\mathbf{y}^r$ by some amount $\boldsymbol{\varepsilon}^r$. Therefore in interpolative associative memory we have

$$\varphi(\mathbf{u}^r+\varepsilon)=\mathbf{y}^r+\varepsilon^r \quad such \quad that \quad \varepsilon=\mathbf{0}\Rightarrow\varepsilon^r=\mathbf{0}, \quad k=1..K \quad (3.1.1)$$

- **Accretive associative memory:** when $\mathbf{u}$ is presented to the memory, it responds by producing $\mathbf{y}^r$ of the stored association such that $\mathbf{u}^r$ is the one closest to $\mathbf{u}$ among $\mathbf{u}^k$, k=1..K, that is,

$$\varphi(\mathbf{u})=\mathbf{y}^r \quad such\ that \quad \mathbf{u}^r=\min_{\mathbf{u}^k} \| \mathbf{u}^k-\mathbf{u} \|, \ k=1..K \quad (3.1.2)$$

The accretive associative memory in the form given above is called **heteroassociative memory**. However if the stored exemplars are in a special form such that the desired patterns and the input patterns are the same, that is $\mathbf{y}^k=\mathbf{u}^k$ for k=1..K, then it is called **autoassociative memory**. In such a memory, whenever $\mathbf{u}$ is presented to the memory it responds by $\mathbf{u}^r$ which is the closest one to $\mathbf{u}$ among $\mathbf{u}^k$, k=1..K, that is,

$$\varphi(\mathbf{u})=\mathbf{u}^r \quad such\ that \quad \mathbf{u}^r=\min_{\mathbf{u}^k} \| \mathbf{u}^k-\mathbf{u} \| \quad k=1..K \quad (3.1.3)$$

While interpolative memories can be implemented by using feed-forward neural networks, it is more appropriate to use recurrent networks as accretive memories.

The advantage of using recurrent networks as associative memory is their convergence to one of a finite number of stable states when started at some initial state. The basic goals are :

- to be able to store as many exemplars as we need, each corresponding to a different stable state of the network,
- to have no other stable state
- to have the stable state that the network converges to be the one closest to the applied pattern

The problems that we are faced with being:

- the capacity of the network is restricted,
- depending on the number and properties of the patterns to be stored, some of the exemplar may not be the stable states,
- some spurious stable states different than the exemplars may arise by themselves
- the converged stable state may be other than the one closest to the applied pattern

One way of using recurrent neural networks as associative memory is to fix the external input of the network and present the input pattern $\mathbf{u}^r$ to the system by setting $\mathbf{x}(0)=\mathbf{u}^r$. If we relax such a network, then it will converge to the attractor $\mathbf{x}^*$ for which $\mathbf{x}(0)$ is within the basin attraction as explained in Section 2.7. If we are able to place each $\boldsymbol{\mu}^k$ as an attractor of the network by proper choice of the connection weights, then we expect the network to relax to the attractor $\mathbf{x}^*=\boldsymbol{\mu}^r$ that is related to the initial state $\mathbf{x}(0)=\mathbf{u}^r$. For a good performance of the network, we need the network to converge only to one of the stored patterns $\boldsymbol{\mu}^k$, $k=1...K$. Unfortunately, some initial states may converge to ***spurious states***, which are the undesired attractors of the network representing none of the stored patterns. Spurious states may arise by themselves depending on the model used and the patterns stored. The capacity of the neural associative memories is restricted by the size of the networks. If we increment the number of stored patterns for a fixed size neural network, spurious states arise inevitably. Sometimes, the network may converge not to a spurious state, but to a memory pattern not so close to the presented pattern. What we expect for a feasible operation is, at least for the stored memory patterns themselves, if any of them is

presented to the network by setting $\mathbf{x}(0)=\boldsymbol{\mu}^k$, then the network should stay converged to $\mathbf{x}^*=\boldsymbol{\mu}^r$ (Figure 3.1).

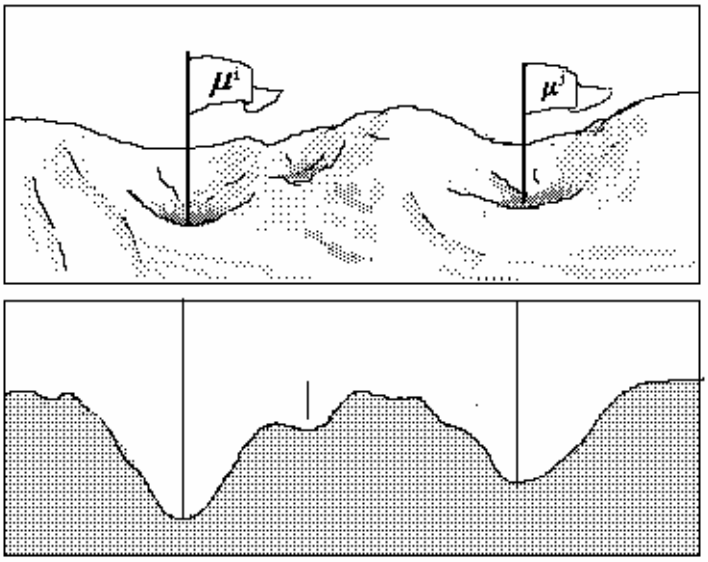

Figure 3.1 In associative memory each memory element is assigned to an attractor

A second way to use recurrent networks as associative memory, is to present the input pattern $\mathbf{u}^r$ to the system as an external input. This can be done by setting $\theta=\mathbf{u}^r$, where θ is the threshold vector whose i^{th} component is corresponding to the threshold of neuron i. After setting $\mathbf{x}(0)$ to some fixed value we relax the network and then wait until it converges to an attractor $\mathbf{x}^*$. For a good performance of the network, we desire the network to have a single attractor such that $\mathbf{x}^*=\boldsymbol{\mu}^k$ for each stored input pattern $\mathbf{u}^k$, therefore the network will converge to this attractor independent of the initial state of the network. Another solution to the problem is to have predetermined initial values, so that these initial values lie within the basin attraction of $\boldsymbol{\mu}^k$ whenever $\mathbf{u}^k$ is applied. We will consider this kind of networks in Chapter 7 in more detail, where we will examine how these recurrent networks are trained.

3.2 Linear Associators as Interpolative Memory

It is quite easy to implement interpolative associative memory when the set of input memory elements $\{\mathbf{u}^k\}$ constitutes an othonormal set of vectors, that is

$$\mathbf{u}^i \cdot \mathbf{u}^j = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \quad (3.2.1)$$

By using kronecker delta, we write simply

$$\mathbf{u}^i \cdot \mathbf{u}^j = \delta_{ij} \tag{3.2.2}$$

The mapping function $\varphi(\mathbf{u})$ defined below may be used to establish an interpolative associative memory:

$$\varphi(\mathbf{u}) = \mathbf{W}^{\mathsf{T}}\mathbf{u} \quad (3.2.3)$$

where T denotes transpose and

$$\mathbf{W} = \sum_k \mathbf{u}^k \times \mathbf{y}^k \quad (3.2.4)$$

Here the symbol $\times$ is used to denote outer product of $\mathbf{x} \in \mathrm{R}^N$ and $\mathbf{y} \in \mathrm{R}^M$, which is defined as

$$\mathbf{u}^k \times \mathbf{y}^k = \mathbf{u}^k \mathbf{y}^{k^{\mathsf{T}}} = (\mathbf{y}^k \mathbf{u}^{k^{\mathsf{T}}})^{\mathsf{T}}, \quad (3.2.5)$$

resulting in a matrix of size *N* by *M*.

By defining matrices [Haykin 94]:

$$\mathbf{U}=[\mathbf{u}^1\ \mathbf{u}^2 .. \mathbf{u}^k .. \mathbf{u}^K] \quad (3.2.6)$$

and

$$\mathbf{Y}=[\mathbf{y}^1\ \mathbf{y}^2 .. \mathbf{y}^k .. \mathbf{y}^K] \quad (3.2.7)$$

the weight matrix can be formulated as

$$\mathbf{W}^{\mathsf{T}} = \mathbf{Y}\mathbf{U}^{\mathsf{T}} \quad (3.2.8)$$

If the network is going to be used as autoassociative memory we have **Y**=**U** so,

$$\mathbf{W}^{\mathsf{T}} = \mathbf{U}\mathbf{U}^{\mathsf{T}} \tag{3.2.9}$$

For a function $\varphi(\mathbf{u})$ to constitute an interpolative associative memory, it should satisfy the condition

$$\varphi(\mathbf{u}^r)=\mathbf{y}^r \quad \textit{for } r=1..K \tag{3.2.10}$$

We can check it simply as

$$\varphi(\mathbf{u}^r) = \mathbf{W}^\top \mathbf{u}^r \tag{3.2.11}$$

which is

$$\mathbf{W}^\top \mathbf{u}^r = \mathbf{Y}\mathbf{U}^\top \mathbf{u}^r \quad (3 \tag{.2.12}$$

Since the set $\{\mathbf{u}^k\}$ is orthonormal, we have

$$\mathbf{Y}\mathbf{U}^\top \mathbf{u}^r = \sum_k \delta_{kr}\mathbf{y}^k = \mathbf{y}^r \quad (3 \tag{.2.13}$$

which results in

$$\varphi(\mathbf{u}^r) = \mathbf{Y}\mathbf{U}^\top \mathbf{u}^r = \mathbf{y}^r \quad (3.2.14)$$

as we desired.

Furthermore, if an input pattern $\mathbf{u}=\mathbf{u}^r+\boldsymbol{\varepsilon}$ different than the stored patterns is applied as input to the network, we obtain

$$\begin{aligned}\varphi(\mathbf{u}) &= \mathbf{W}^\top (\mathbf{u}^r + \varepsilon) \\ &= \mathbf{W}^\top \mathbf{u}^r + \mathbf{W}^\top \varepsilon\end{aligned} \tag{3.2.15}$$

Using equation (3.2.12) and (3.2.13) results in

$$\varphi(\mathbf{u}) = \mathbf{y}^r + \mathbf{W}^\top \varepsilon \quad (3.2.16)$$

Therefore, we have

$$\varphi(\mathbf{u}) = \mathbf{y}^r + \varepsilon^r \quad (3.2.17)$$

in the required form, where

$$\varepsilon^r = \mathbf{W}^\top \varepsilon \quad (3.2.18)$$

Such a memory can be implemented as shown in Figure 3.2 by using M neurons each having N inputs. The connection weights of neuron i is assigned value $\mathbf{W}_i$, which is the i^{th} column vector of matrix $\mathbf{W}$. Here each neuron has a linear output transfer function $f(a)=a$. When a stored pattern $\mathbf{u}^k$ is applied as input to the network, the desired value $\mathbf{y}^k$ is observed at the output of the network as:

$$\mathbf{x}^k = \mathbf{W}^T\mathbf{u}^k \quad (3.2.19)$$

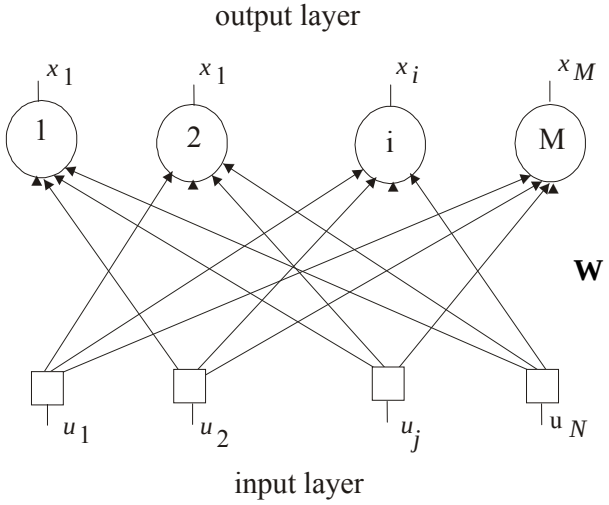

Figure 3.2 Linear Associator

Until now, we have investigated the use of linear mapping $\mathbf{YU}^T$ as associative memory, which works well when the input patterns are orthonormal. In the case the input patterns are not orthonormal, the linear associator cannot map some input patterns to desired output patterns without error. In the following we will investigate the conditions necessary to minimize the output error for the exemplar patterns. That is, for a given set of exemplars $\mu^k=(\mathbf{u}^k,\mathbf{y}^k)$, $\mathbf{u}^k \in R^N$, $\mathbf{y}^k \in R^M$, $k=1..K$, our purpose is to find a linear mapping $\mathbf{A}^*$ among $\mathbf{A}$: $R^N \rightarrow R^M$ such that:

$$\mathbf{A}^* = \min_{\mathbf{A}} \sum_k \| \mathbf{y}^k - \mathbf{A}\mathbf{u}^k \| \quad (3.2.20)$$

where ||.|| is chosen as Euclidean norm.

The problem may be reformulated by using the matrices $\mathbf{U}$ and $\mathbf{Y}$ [Haykin 94]:

$$\mathbf{A}^* = \min_{\mathbf{A}} \| \mathbf{Y} - \mathbf{A}\mathbf{U} \| \quad (3.2.21)$$

The pseudo inverse method [Kohonen 76] based on least squares estimation provides a solution for the problem in which **A*** is determined as:

$$\mathbf{A}^* = \mathbf{Y}\mathbf{U}^+ \quad (3.2.22)$$

where $\mathbf{U}^+$ is pseudo inverse of **U**.

The pseudoinverse U^+ is a matrix satisfying the condition:

$$\mathbf{U}^+\mathbf{U} = \mathbf{1} \quad (3.2.23)$$

where **1** is the identity matrix. A perfect match is obtained by using $\mathbf{A}^* = \mathbf{Y}\mathbf{U}^+$, since

$$\mathbf{A}^*\mathbf{U} = \mathbf{Y}\mathbf{U}^+\mathbf{U} = \mathbf{Y} \quad (3.2.24)$$

resulting in no error due to the fact

$$|| \mathbf{Y} - \mathbf{A}^*\mathbf{U}|| = 0 \quad (3.2.25)$$

In the case the input patterns are linearly independent, that is none of them can be obtained as a linear combinations of the others, then a matrix $\mathbf{U}^+$ satisfying Eq. (3.2.23) can be obtained by applying the formula [Golub and Van Loan 89, Haykin 94]

$$\mathbf{U}^+ = (\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}^T \quad (3.2.26)$$

Notice that for the input patterns, which are the columns of the matrix **U**, to be linearly independent, the number of columns should not be more than the number of rows, that is $K \leq N$, otherwise $\mathbf{U}^T\mathbf{U}$ will be singular and no inverse will exist. The condition $K \leq N$ means that the number of entries constituting the patterns restricts the capacity of the memory. At most N patterns can be stored in such a memory.

This memory can be implemented by a neural network for which $\mathbf{W}^T=\mathbf{Y}\mathbf{U}^+$. The desired value $\mathbf{y}^k$ appears at the output of the network as $\mathbf{x}^k$ when $\mathbf{u}^k$ is applied as input to the network:

$$\mathbf{x}^k = \mathbf{W}^T\mathbf{u}^k \quad (3.2.27)$$

as explained in the previous section.

Notice that for the special case of orthonormal patterns that we examined previously in this section, we have

$$\mathbf{U}^T\mathbf{U} = \mathbf{1} \quad (3.2.28)$$

that results in the pseudoinverse, which is in the form

$$\mathbf{U}^+ = \mathbf{U}^T \quad (3.2.29)$$

and therefore

$$\mathbf{W}^T = \mathbf{Y}\mathbf{U}^T \quad (3.2.30)$$

as we have derived previously.

3.3. Hopfield Autoassociative Memory

In section 2.9 we have examined continuous input, continuous time Hopfield network. In t his s ection we will i nvestigate how Hopfield n etwork can be used as autoassociative memory. For this purpose some modifications are done on it so that it works in discrete state space and discrete time. When discrete Hopfield network was introduced as associative memory in [Hopfield 82] it had attracted a great attention. In [Hopfield 84] it is shown that many important characteristics of the dis crete and continuos deterministic models are closely related (Figure 3.3)

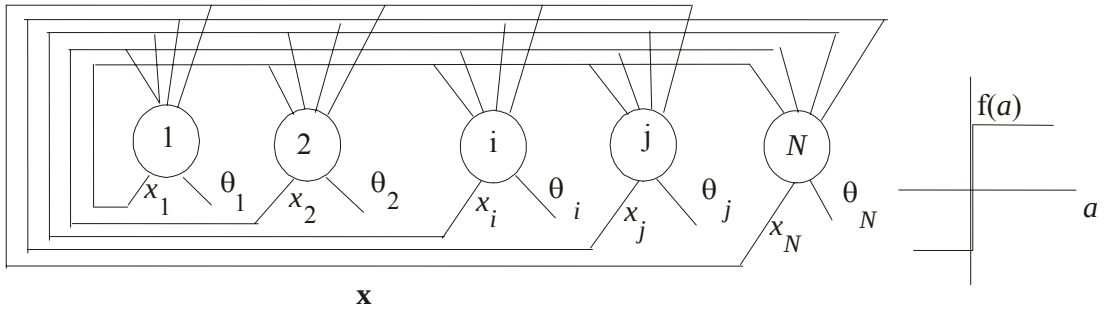

Figure 3.3 Hopfield Associative Memory

Note that, whenever the patterns to be stored in Hopfield network are from N dimensional bipolar space constituting a hypercube, that is $\mathbf{u}^k \in \{-1,1\}^N$, $k=1..K$, then it is convenient to have any stable state of the network on the corners of the hypercube. For this purpose refer to the output transfer function given by Eq. (2.9.3) and to Figure 2.7 for different values of the gain. If we let the output transfer function of the neurons in the network to have a very high gain, in the extreme case

$$f_i(a) = \lim_{\kappa \to \infty} \tanh(\kappa a) \tag{3.3.1}$$

we obtain

$$f_i(a) = \text{sign}(a) = \begin{cases} 1 & for\ a > 0 \\ 0 & for\ a = 0 \\ -1 & for\ a > 0 \end{cases} \tag{3.3.2}$$

Furthermore note that the second term of the energy function given by Eq. (2.9.7) that we repeat here for convenience:

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ji} x_j x_i + \sum_{i=1}^{N} \frac{1}{R_i} \int_0^{x_i} f^{-1}(x)dx - \sum_{i=1}^{N} \theta_i x_i \tag{3.3.3}$$

approaches to zero. Therefore the stable states of the network corresponds to the local minima of the function:

$$E = -\frac{1}{2}\sum_{i}\sum_{j} w_{ji} x_j x_i - \sum_{i} \theta_i x_i \tag{3.3.4}$$

so that they lie on the corners of the hypercube as explained previously.

In section 2.10, we have derived the discrete time approximation for continuous time Hopfield network described in section 2.9. However in this section we investigate a special case of the Hopfield network where the stable states of the network are forced to take discrete values in bipolar state space. Knowing in advance that the local minima of the energy function should take place at the corners of the N dimensional hypercube, we can get rid of the slow convergence problem due to small value of η.

For this purpose a discrete state excitation [Hopfield 82] of the network, is provided in the following:

$$x_i(k+1) = f(a_i(k)) = \begin{cases} 1 & for\ a_i(k) > 0 \\ x(k) & for\ a_i(k) = 0 \\ -1 & for\ a_i(k) > 0 \end{cases} \quad (3.3.5)$$

where $a_i(k)$ is defined in a manner similar to that we used to:

$$a_i(k) = \sum_j w_{ji} x_j(k) + \theta_i \quad (3.3.6)$$

The processing elements of the network are updated one at a time, such that all of the processing elements must be updated at the same average rate.

Note that, for any vector **x** having bipolar entries, that is $x_i \in \{-1,1\}$, we obtain the vector itself if we apply the function defined by Eq. (3.3.5) on it, that is

$$\mathbf{f}(\mathbf{x})=\mathbf{x} \quad (3.3.7)$$

Here **f** is used to denote the vector function such that the function f is applied at each entry.

For stability of the discrete Hopfield network, it is further required $w_{ii}=0$ in addition to the constraint $w_{ij}=w_{ji}$

In order to use discrete Hopfield network as autoassociative memory, its weights are fixed to

$$\mathbf{W}^T = \mathbf{U}\mathbf{U}^T \quad (3.3.8)$$

where **U** is the input pattern matrix as defined in Eq. (3.2.6). Remember that in autoassociative memory we have **Y**=**U**, where **Y** is the matrix of desired output patterns as defined in Eq (3.2.7). For the stability of the network, the diagonal entries of **W** is set to 0, that is $w_{ii}=0$, $i=1..N$

If all the states of the network are to be updated at once, then the next state of the system can be represented in the form

$$\mathbf{x}(k+1)=\mathbf{f}(\mathbf{W}^{\mathsf{T}}\mathbf{x}(k)) \quad (3.3.9)$$

For the special case if the exemplars are orthonormal, then due to fact indicated by Eqs. (3.3.7) and (3.2.13) we have

$$\mathbf{f}(\mathbf{W}^{\mathsf{T}}\mathbf{u}^{r})=\mathbf{f}(\mathbf{u}^{r})=\mathbf{u}^{r} \quad (3.3.10)$$

that means each exemplar is a stable state of the network. Whenever the initial state is set to one of the exemplar, the system remains there. However, if the initial state is set to some arbitrary input, then the network converges to one of the stored exemplars, depending on the basin of attraction in which $\mathbf{x}(0)$ lies.

However, in general, the input patterns are not orthonormal, so there is no guarantee that each exemplar is corresponding to a stable state. Therefore the problems that we mentioned in Section 3.1 arise. The capacity of the Hopfield net is less than $0.138N$ patterns, where N is the number of units in the network [Lippmann 89].

In the following we will show that the energy function always decreases as the state of the processing elements are changed one by one. Notice that:

$$\begin{aligned}\Delta E &= E(\mathbf{x}(k+1)) - E(\mathbf{x}(k)) \\ &= -\frac{1}{2}\sum_i \sum_j w_{ji}x_j(k+1)x_i(k+1) - \sum_i \theta_i x_i(k+1) \quad (3.3.11) \\ &+ \frac{1}{2}\sum_i \sum_j w_{ji}x_j(k)x_i(k) + \sum_i \theta_i x_i(k)\end{aligned}$$

Assume that the neuron that just changes state at step k is neuron p. Therefore $x_p(k+1)$ is determined by equation 3.3.5 and for all the other neurons we have $x_i(k+1)=x_i(k)$, $i\neq p$. Furthermore we have $w_{pp}=0$. Hence,

$$\Delta E = -((x_p(k+1) - x_p(k))(\sum_j w_{jp}x_j(k)) + \theta_p) \quad (3.3.12)$$

that is,

$$\Delta E = -((x_p(k+1) - x_p(k))a_p(k) \quad (3.3.13)$$

Notice that if the value of x_p remains the same, then $x_p(k+1)=x_p(k)$ so $\Delta E=0$. If they are not the same, than it is either the case $x_p(k)= -1$ and $x_p(k+1)=1$ due to fact $a_p(k)>0$,

or $x_p(k)=1$ and $x_p(k+1)= -1$ due to fact $a_p(k)<0$. Whatever the case is, if $x_p(k+1)\neq x_p(k)$ it is in a direction for which $\Delta E<0$. Therefore, for discrete Hopfield network we have

$$\Delta E \leq 0 \quad (3.14)$$

Because at each state change, the energy function decreases at least by some fixed minimum amount, and because the energy function is bounded, it reaches a minimum value in a finite number of state changes. So the Hopfield network converges to one stable state in finite time in contrary to the asymptotic convergence in the continuous Hopfield network. The schedule, in which only one unit of the discrete Hopfield network is updated at a time, is called ***asynchronous*** update. The other approach in which all the units are updated at once is called ***synchronous*** update. Although the convergence with the asynchronous update mechanism is guaranteed, it may result in a cycle of length two in synchronous update.

It should be noted that, the continuous deterministic model implies the possibility of implementing the discrete network in actual hardware because of the close relation between discrete and continuous models. However, the discrete model is often implemented through computer simulations because of its simplicity.

Exercise: Explain how can we use the Hopfield network as autoassociative memory if the states are not from bipolar space $\{-1,1\}^N$ but from binary space $\{0,1\}^N$.

3.4. Bi-directional Associative Memory

The Bi-directional Associative Memory (BAM) introduced in [Kosko 88] is a recurrent network (Figure 3.4) designed to work as heteroassociative memory [Nielsen 90]. BAM network consists of two sets of neurons whose outputs are represented by vectors $\mathbf{x} \in R^N$ and $\mathbf{v} \in R^M$ respectively, having activation defined by the pair of equations:

$$\frac{da_{x_i}}{dt} = -\alpha_i a_{x_i} + \sum_{j=1}^{N} w_{ji} f(a_{v_j}) + \theta_i \quad for\ i = 1..M \quad (3.4.1)$$

$$\frac{da_{v_j}}{dt} = -\beta_j a_{y_j} + \sum_{i=1}^{M} w_{ij} f(a_{x_i}) + \phi_j \quad for\ j = 1..N \quad (3.4.2)$$

where α_i, β_j, θ_i, ϕ_j are positive constants for all $i=1..M$, $j=1..N$, f is sigmoid function and $\mathbf{W}=[w_{ij}]$ is any $N\times M$ real matrix.

The stability of the BAM network can be proved easily by applying Cohen-Grossberg theorem on the state vector $\mathbf{z}\in \mathrm{R}^{N+M}$ defined as

$$z_i = \begin{cases} x_i & i \le M \\ v_j & j = i - M, \quad M < i \le M + N \end{cases} \quad (3.4.3)$$

that is **z** obtained through concatenation **x** and **v**.

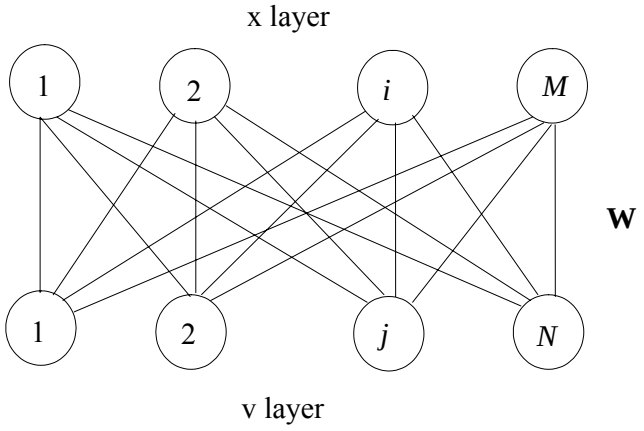

Figure 3.4: Bi-directional Associative Memory

Since BAM is a special case of the network defined by Cohen-Grossberg theorem, it has a Lyapunov Energy function as it is provided in the following:

$$\begin{aligned} E(\mathbf{x},\mathbf{v}) &= -\sum_{i=1}^{M}\sum_{j=1}^{N} w_{ij} f(a_{x_i}) f(a_{v_j}) \\ &+ \sum_{i=1}^{M} \alpha_i \int_0^{a_{x_i}} f'(a) a \, da + \sum_{j=1}^{N} \beta_i \int_0^{a_{v_i}} f'(b) b \, db \quad (3.4.4) \\ &- \sum_{i=1}^{M} f(x_i)\theta_i - \sum_{j=1}^{N} f(v_j)\phi_j \end{aligned}$$

The discrete BAM model is defined in a manner similar to discrete Hopfield network. The output functions are chosen to be $f(a)=\text{sign}(a)$ and states are excited as:

$$x_i(k+1) = f(a_{x_i}(k)) = \begin{cases} 1 & for\ a_{x_i}(k) > 0 \\ x(k) & for\ a_{x_i}(k) = 0 \\ -1 & for\ a_{x_i}(k) < 0 \end{cases} \quad (3.4.5)$$

$$v_j(k+1) = f(a_{v_j}(k)) = \begin{cases} 1 & for\ a_{v_j}(k) > 0 \\ v(k) & for\ a_{v_j}(k) = 0 \\ -1 & for\ a_{v_j}(k) < 0 \end{cases} \quad (3.4.6)$$

where

$$a_{x_i} = \sum_{j=1}^{N} w_{ji} f(a_{v_j}) + \theta_i \quad for\ i = 1..M \quad (3.4.7)$$

and

$$a_{v_j} = \sum_{i=1}^{N} w_{ij} f(a_{x_i}) + \phi_j \quad for\ j = 1..N\ . \quad (3.4.8)$$

or in compact matrix notation it is shortly

$$\mathbf{x}(k+1) = \mathbf{f}(\mathbf{W}^{\mathsf{T}}\mathbf{v}(k)) \quad (3.4.9)$$

and

$$\mathbf{v}(k+1) = \mathbf{f}(\mathbf{W}\mathbf{x}(k)). \quad (3.4.10)$$

In the discrete BAM, the energy function becomes

$$E(\mathbf{x},\mathbf{y}) = -\sum_{i=1}^{M} \sum_{j=1}^{N} w_{ij} f(a_{x_i}) f(a_{v_j}) - \sum_{i=1}^{M} f(a_{x_i})\theta_i - \sum_{j=1}^{N} f(a_{v_j})\phi_j \quad (3.4.11)$$

satisfying the condition

$$\Delta E \le 0 \quad (3.4.12)$$

which implies the stability of the system.

The weights of BAM is determined by the equation

$$\mathbf{W}^{\top}=\mathbf{Y}\mathbf{U}^{\top} \tag{3.4.13}$$

For the special case of orthonormal input and output patterns we have

$$\mathbf{f}(\mathbf{W}^{\top}\mathbf{u}^{r})=\mathbf{f}(\mathbf{Y}\mathbf{U}^{\top}\mathbf{u}^{r})=\mathbf{f}(\mathbf{y}^{r})=\mathbf{y}^{r} \tag{3.4.14}$$

and

$$\mathbf{f}(\mathbf{W}\mathbf{y}^{r})=\mathbf{f}(\mathbf{U}\mathbf{Y}^{\top}\mathbf{y}^{r})=\mathbf{f}(\mathbf{u}^{r})=\mathbf{u}^{r} \tag{3.4.15}$$

indicating that exemplars are stable states of the network. Whenever the initial state is set to one of the exemplar, the system remains there. For arbitrary initial states the network converges to one of the stored exemplars, depending on the basin of attraction in which $\mathbf{x}(0)$ lies.

For the input patterns that are not orthonormal, the network behaves as it is explained for the Hopfield network.

Exercise: Compare the special case $\mathbf{U}=\mathbf{Y}$ of BAM with Hopfield Autoassociative memory.