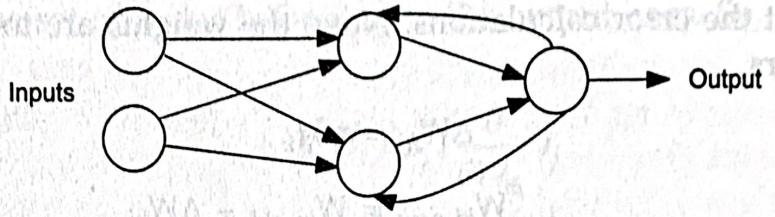

Figure 17.13 A feedback network.

Recurrent Neural Networks (RNN)

They belong to the category of feedback networks, where there are connections between the input, output and the hidden layers, and there can be interconnections between the nodes as well. Many's the time, it is seen that the feedback networks are actually referred to as *recurrent networks*. A recurrent network has at least one cyclic path. For training the network, many variants have been proposed in the recurrent networks and are in practice. Owing to the connections in backward direction also, they provide an internal memory. Figure 17.14 depicts a recurrent network. Today, it is most commonly used for speech recognition. Bayesian and RNN are also combined for time series modelling.

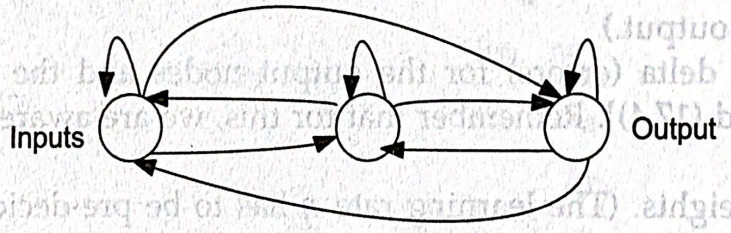

Figure 17.14 Recurrent network.

17.5 PATTERN ASSOCIATIVE NETWORKS

In case of pattern associative network, a pattern itself is generated at the output, given an input pattern. A weight matrix is used in the neural networks, which are used as associative memories. The process of learning or storing the desired patterns is actually a process to generate the weight matrix. (This is discussed further in the chapter). So, pattern associative networks have associative memory. The pattern associative networks are categorised into two tasks—auto-associative and hetero-associative.

Auto-associative

In auto-associative, the patterns are stored in the network (associative memory). The network corrects the inputs. When a new pattern is given as an input, comparison (association) is done with the ones that are stored and a nearest matching is given as an output. With an auto-associative network, a distorted image can be corrected. We can say that the noise is erased off from the pattern. A very fundamental task that is done by these networks is character recognition. Figure 17.15 shows an auto-associative network model. You must have noted by now that this learning, hence, belongs to unsupervised category.

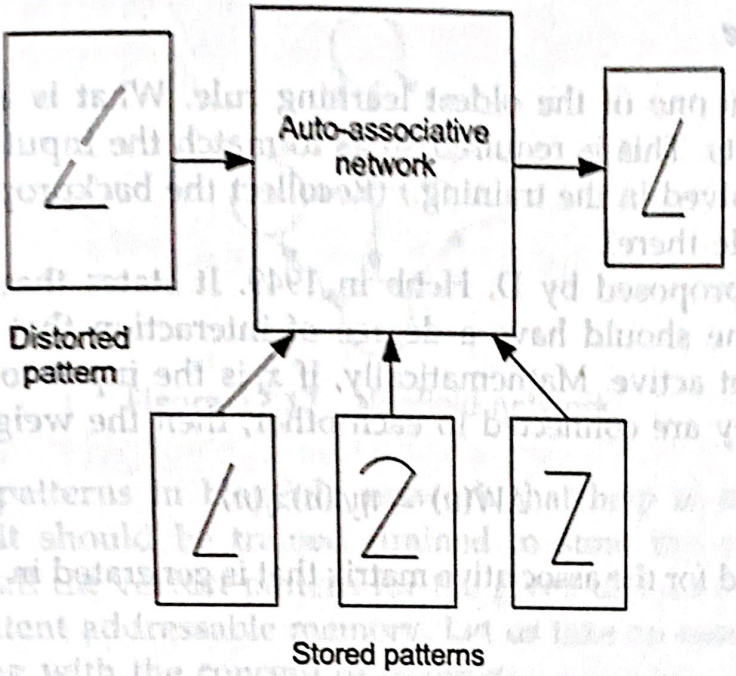

Figure 17.15 Auto-associative network model.

Hetero-associative

Unlike auto-associative where the patterns are stored in the memory, here the association is stored. It is better understood from Figure 17.16. So, the output is altogether a different pattern.

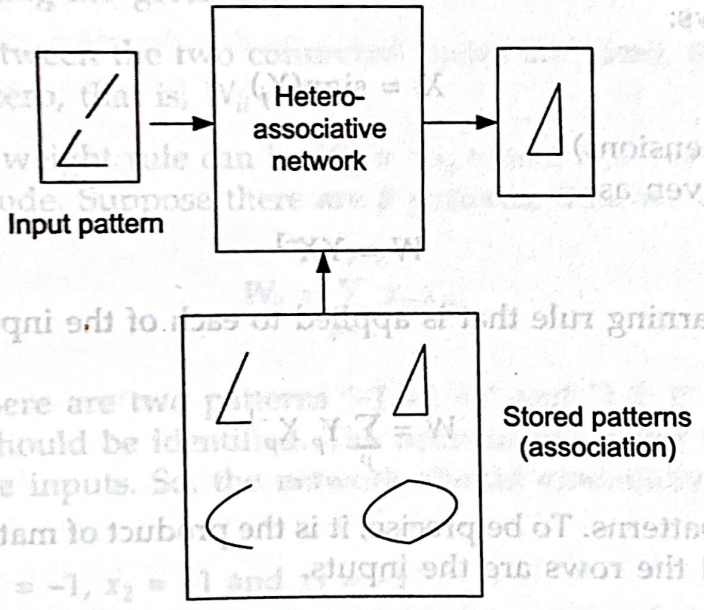

Figure 17.16 Hetero-associative network model.

Mathematically, we can say that in association, we are having X_i as the input pattern and Y_i as the output pattern. The mapping $X_i \rightarrow Y_i$ is the task to be carried out where $i \in \{1, 2, 3, ..., n\}$ that are the stored patterns. In case of auto-associative, the output and the input have same dimensionality, whereas it is not the case with hetero-associative.

Hebbian Learning Rule

Hebbian learning rule is one of the oldest learning rule. What is a learning rule? A rule that adjusts the weights. This is required so as to match the input to the target required. (Naturally. this is involved in the training.) (Recollect the backpropagation; we have used a generalised delta rule there).

Hebb's rule was proposed by D. Hebb in 1949. It states that two neurons that are active at the same time should have a degree of interaction that is higher than that of the ones which are not active. Mathematically, if x_i is the input to neuron i and y_j is the output of j, where they are connected to each other, then the weight is given as:

$$\Delta W(n) = \eta y_i(n) x_i(n)$$

This rule is required for the associative matrix that is generated in the pattern association. Let us find it out.

17.5.1 Associative Memory

In case of associative memory, we are actually considering the calculations of W. We know that we want the mapping from $X \rightarrow Y$. For the association to be established, we can rewrite it as $XW = Y$. If the network is auto-associative, it can be simply written as $XW = X$.

If the output belongs to the set {–1, 1}, then a simple sign function can be applied. It can be given as follows:

$$X_j = \text{sign}(Y_j)$$

where, $j = 1$ to k (dimensions)

Now, W can be given as

$$W = YX^{-1}$$

By the Hebbian learning rule that is applied to each of the input-output neuron pair, we can have

$$W = \sum_P Y_p X_p^{-1}$$

where, P refers to the patterns. To be precise, it is the product of matrix where the columns are output vectors and the rows are the inputs.

17.5.2 Hopfield Network

It belongs to the category of auto-associative networks and is a feedback network. It consists of a single layer. The neurons in the layer are fully connected. The training stage of Hopfield network is less complicated and the network values go through state transitions. Figure 17.17 represents the Hopfield network.

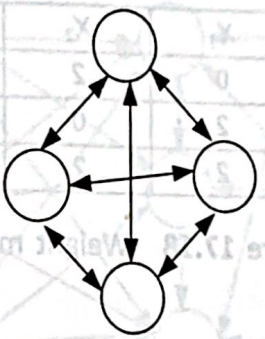

Figure 17.17 Hopfield network.

There are stored patterns in Hopfield network that help in the correction of input patterns. To correct, it should be trained (trained to store the patterns) and later on, should be used to recall the correct pattern for the given distorted pattern. The Hopfield network works as content addressable memory. Let us take an example and see how this correction occurs along with the concept of training.

Training

In training, the Hopfield network learns a pattern. For it to learn, it has to determine the weight matrix. (Just before we have discussed about the association matrix.) There are different learning rules that can be used for the determination of the matrix. The one we discuss is again inspired by Hebb's rule and is the simplest one! Two conditions that are generally fixed in training are given below:

1. The weights between the two connected nodes are same, that is, $W_{ij} = W_{ji}$
2. Self weight is zero, that is, $W_{ii} = 0$.

Hence, the simple weight rule can be $W_{ij} = x_i x_j$ where x_i is the input to ith node and x_j is the input to jth node. Suppose there are p patterns, then we can generalise it as

$$W_{ij} = \sum_{p=1}^{P} x_{ip} x_{jp}$$

Let us say that there are two patterns '–1 –1 –1' and '1 1 1' and we want them to be trained and they should be identified. The neurons are going to be 3. Let us assume x_1, x_2 and x_3 to be the inputs. So, the network should essentially correct the pattern as follows:

For pattern 1: $x_1 = -1$, $x_2 = -1$ and $x_3 = -1$

For pattern 2: $x_1 = 1$, $x_2 = 1$ and $x_3 = 1$

$W_{12} = -1*-1 + 1*1 = 2$

$W_{23} = -1* -1 + 1*1 = 2$

$W_{34} = -1* -1 + 1*1 = 2$

So, the weights here are now same, that is, all are equal to 2.

Thus, the weight matrix is given below in Figure 17.18.

	X_1	X_2	X_3
X_1	0	2	2
X_2	2	0	2
X_3	2	2	0

Figure 17.18 Weight matrix.

Recall:

Now, let us see how it would correct the input pattern '1 -1 1'. A node is selected at random for update here. Say node 2 is randomly selected for update, and then its net input is

$$W_{12} * x_1 + W_{23} * x_3 + x_2$$

So, we get

$$(2 * 1) + (2 * 1) + (-1) = 2 + 2 - 1 = 3$$

Suppose sign function is used, then sgn(3) will give us 1. Thus, the node value is changed giving us (1, 1, 1). Similarly, if node 1 is selected, then the net input will be $(2 * -1) + (2 * 1) + 1 = -2 + 2 + 1 = 1$. Then, sgn(1) = 1. Hence, it remains unchanged giving us the corrected pattern (1, 1, 1). Remember that there are various ways to select the nodes; sometimes, it is as simple as even/odd. This updation of the node takes place till it converges to the one that is present.

There is a concept of energy function. It measures the simulation that is required for a neuron to fire. This energy is required to decide upon the weight so that it is proportional to the patterns. Here, depending on the patterns that are to be stored, the weight computation is done on the correlations between the input values. Though in the example, we have used a simple weight calculation, generally the weights are computed as follows:

$$W_{ij} = \frac{1}{P} \sum_{p=1}^{P} v_{pi} v_{pj}$$

where v_{pi} and v_{pj} are the values for ith and jth parameter, respectively, in the input pattern p.

What could be the drawbacks of this network? Suppose that the network is presented with some pattern it has not learnt. Will it always repair it? If the pattern stored is too large, then there is a possibility that an altogether new spurious pattern could be generated or the case could occur that the network would not converge.

17.6 COMPETITIVE LEARNING

Competitive learning is an unsupervised learning, where the idea behind this learning is to determine weights to update during the learning. This update is done only for a winning neuron in the whole process of learning. So, the neurons compete with each other and only one wins. It is winners take it all approach. Figure 17.19 shows a simple competitive learning network.

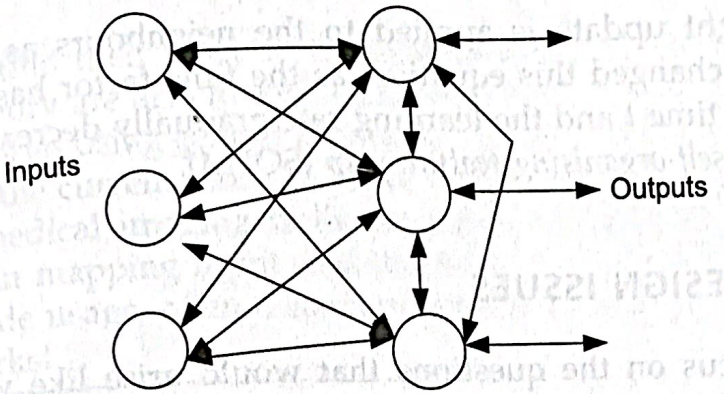

Figure 17.19 A competitive network.

For competitive learning, the competitive nodes are represented in the form of weight vectors. This means a competitive output node y will be formed by the weights connected with the input nodes and this dimension is same as that of the inputs. It is to be noted that the initial weights are randomly selected between 0 and 1. Consider a competitive node $y_a = \{w_{1y}, w_{2y}, ..., w_{ny}\}$ where n is the number of input dimensions. Assume we have m output nodes. Let i be some input pattern. When an input is presented, the distance between the input and the competitive weight vectors is computed. This is most often the Euclidean distance that is used. So, $d(i, y_a)$, where, $a = \{1, 2, ..., m\}$ and m is the number of outputs is computed. For every output competitive vector, this is calculated and the one with the lowest value wins. Suppose y_j wins. That means it is closest to the pattern. Then, the weights associated with this are updated by the rule.

$$y_{j\_new} = y_{j\_old} + \eta(i - y_{j\_old})$$

where y_{j_old} is the previous value and y_{j_new} is the new weight for y_j.

How long will it be done? Again, till the network converges. In competitive learning, it is observed that after the training is over, a clustering mechanism is found in the learning process, where the patterns are grouped.

17.7 SELF-ORGANISING MAP (SOM)

Now, come to the last point in this chapter, i.e., self-organising maps. They are unsupervised networks based on the competitive learning. They were proposed by Kohonen in 1980s and are also called *Kohonen networks*. As said, they are based on the competitive learning in which the winner is found out. But a slight change in SOM is done. Instead of updating the weights of just the winner, the neighbours are also given a share and the adjustments take place. An SOM network actually transforms the input space into a 2D feature map that maintains the topological order. Often, this property of SOM is used to visualise the high dimensional data. SOM makes use of pre-determined topology to form the neighbours in order to carry out propagation. So, the topological distance between the winner and the other nodes is the criterion for this propagation.

The initial weights are selected at random as before, and in the learning process, the weight vector actually moves towards the centroid of the subsets of an input pattern. The weight update rule is given below:

$$y_j(t + 1) = y_j(t) + \eta(t)\ (i(t) - y_j(t))$$