UNIT 9 HOPFIELD NETWORKS

Structure **Page No.**

9.1 INTRODUCTION

One of the most important functions of human brain is the laying down and recall of memories. More over the studies in neurodynamics indicate the presence of short-term and long-term memory. The short-term memory contributes in processing routine tasks while long-term memory helps in storing the lesson from the past experience. Our memories mostly function in a manner better termed as **associative** or **content-addressable**. That is, a memory does not exist in some isolated fashion, located in a particular set of neurons. All memories are in some sense strings of memories. For example, we remember a place or a person by some event. Sometimes more than the facial features of a person it is the voice or the peculiarity of the name that impresses our memory. Thus, memories are stored in *association* with one another.

The neural network researchers contend that there is a basic mismatch between the standard technology used by computers and the technology of the human brain for processing information. Feed-forward neural networks are well studied for obvious reasons of simplicity in understanding and modelling the computation. Feed forward network is acyclic. The artificial neurons or the perceptrons in a feed-forward neural network model have no cyclic connections. The input data is processed and passed to the outputs but not vive-versa.

Recurrent networks can have connections that go back from the output nodes and can also have arbitrary connections between any nodes. That is a recurrent neural network has at least one feedback loop. For example, in a single layer recurrent network each neuron may have a feedback connection to each of the other neuron or it may even have self-feedback loops.

As learning using perceptrons is a process of modifying the values of the synaptic weights and the threshold or the activation function using samples of training data. A group of (connected) perceptrons are trained on sample input-output pairs until it learns to compute the correct function. Once a feed-forward network is trained, its state is fixed which is not altered by any subsequent input data until the network is retrained. This indicates that a feed-forward neural network does not have memory. In a neurobiological context memory refers to “the relatively enduring neural alterations induced by the interaction of an organism with its environment”. Learning tasks lead us to assume existence of memory. Therefore, the alterations indicate presence of memory and for its usefulness memory needs to be accessible.

Associative memories have a very faint similarity to that of the human brain’s ability to associate patterns. The architecture of a neural network with associative memory depends on its capability of association. An associate memory is a storehouse of associated patterns encoded in some form. When this storehouse is triggered or incited with a pattern the associated pattern pair is recalled or output. An associative memory network is designed by storing/recording several ideal patterns into the network's stable state. When a pattern (even with noise or distorted representation of stored patterns) is input to the storehouse, the network is expected to reach one of the stored patterns and retrieve it as an output. These neural network models are categorized as association models and are also known as content addressable or auto-associative neural networks. All the definitions will be discussed in Sec. 9.2. In Sec. 9.3 we shall discuss Hopfield networks. In Sec. 9.4, we shall discuss the structure of the Hopfield networks, in detail and we shall extend our discussion on the functionality of the Hopfield networks in Sec. 9.5. The storage capacity of this network is discussed in Sec. 9.6.

Objectives

After reading this unit you should be able to:

- identify and design general association and recurrent models of neural network;
- model Hopfield networks;
- train Hopfield networks for a given set of patterns;
- give a trained network to run the Hopfield for a test pattern with noise may require rigorous computation specially if dimension of the vectors is high.

9.2 RELATED DEFINITIONS

Let us recall various definitions as given below:

Recurrent Networks: The interconnection scheme of the recurrent networks is feedback connection or loops as shown below in Fig. 1.

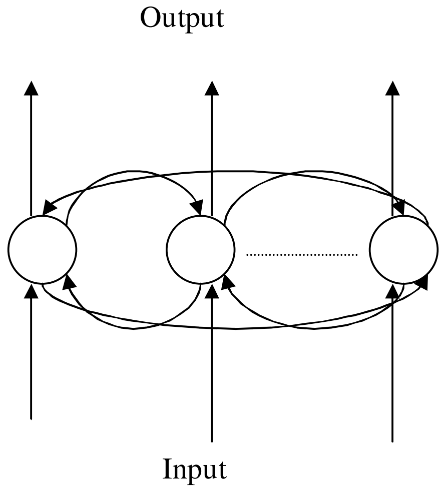


Fig. 1: Recurrent network

Associative Memory: An associative memory is a brain-like distributed memory that learns by association. Association is one of the basic features of the human memory and is prevalent in most models of cognition. Associative memory can be categorized as auto associative memory and Hetero associative memory. These are defined in the following:

i) **Autoassociation**: In autoassociation a neural network is required to store a set of patterns (vectors) by repeatedly presenting them to the network. The network is subsequently presented a partial description or noisy version of an original pattern stored in it, and the task is to recall or more specifically ***retrieve*** that particular pattern.

ii) **Heteroassociation**: In heteroassociation an arbitrary set of input patterns is associated (paired) with another set of arbitrary set of output patterns. The task of retrieval of patterns however is similar i.e. on input of a stored pattern or a distorted version of the already stored pattern the original pattern coupled with the given input is recalled.

Let x_k be the key pattern (vector) applied to an associative memory and y_k be the memorized pattern (vector). The pattern association performed by the network is described by,

$$x_k \rightarrow y_k, \ k = 1, \ 2, \ldots, \ q$$

where q is the number of patterns stored in the network. The key pattern x_k acts as the stimulus that not only determines the storage location of the memorized pattern y_k, but also holds the key for its retrieval.

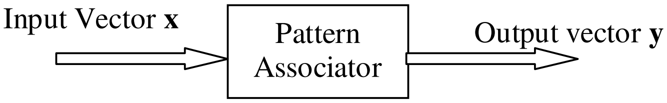


Fig. 2: Input-output relation of the pattern associator

Auto-associative networks consist of one layer of neural elements (units) that are all interconnected, although self-connections are usually disallowed. The neural elements are nonlinear, with states bounded from zero to one – so called two-state-neurons. Any state, whether initial or desired, will be represented by some pattern of zeros and ones over the units. Recurrence and nonlinearity lie at the heart of auto-associative network behaviour. Recurrence allows desired states to emerge via interactions among the units, and the nonlinearity prevents the whole network from running away, and instead encourages the network to settle into a stable state. The auto-associative network will, in most cases, relax from any initial state into the desired state closest to it.

The interconnections between units can be trained according to Hebbian rules (proposed in 1949) - "*when one cell repeatedly assists in firing another, the axon of the first cell develops synaptic knobs (or enlarges them if they already exist) in contact with the soma of the second cell.*" Hebbian rules are *local*, in the sense that they depend only upon the activity of neurons pre- and post-synaptic to any particular connection. The original rule proposed by Hebb specified an increase in synaptic strength whenever pre- and post-synaptic neurons were active together.

Therefore in an association net, if we compare two patterns components (vectors or pixels) within many patterns and find that they are frequently in the same state then the arc weight between the two perceptrons must be increased (should be positive) and if they are frequently in different states, then the arc weight between the two perceptrons must be decreased (should be negative).

There are two phases involved in the operation of an associative memory:

- **Storage phase:** This refers to the training of the network.

- **Recall/Retrieval phase:** This involves the retrieval of a memorized pattern in response to the presentation of a noisy or distorted version of a key pattern to the network.

Matrix Representation: For the computation a matrix representation is a practical tool.

Let $X =$ matrix of input patterns, where each ROW is a pattern.

So $x_{k,i} =$ the i-th bit of the k-th pattern.

Let $Y =$ matrix of output patterns, where each ROW is a pattern. So $y_{k,j} =$ the j-th bit of the k-th pattern. Then, average correlation between two input patterns i and j across all patterns is:

$$w_{i,j} = 1/P(x_{1,i}y_{1,j} + x_{2,i}y_{2,j} + \cdots + x_{p,i}\,y_{p,j}) \qquad (1)$$

All weights for an associative memory network model therefore can be calculated by the following:

$$W = X^T Y \qquad (2)$$

Therefore, for a set of input patterns: $IP_1, IP_2, \ldots, IP_p$, and output patters $OP_1, OP_2, \ldots, OP_p$,

$$X = \begin{array}{c} IP_1 \\ IP_2 \\ : \\ IP_p \end{array} \begin{bmatrix} x_{1,1} & \cdots & x_{1,n} \\ x_{2,1} & \cdots & x_{2,n} \\ : & : & : \\ x_{p,1} & \cdots & x_{p,n} \end{bmatrix}$$

$$X^T = \begin{array}{c} \\ \\ \\ \\ \\ \end{array}\begin{array}{cccc} IP_1 & IP_2 & \cdots & IP_p \\ \hline x_{1,1} & x_{2,1} & \cdots & x_{p,1} \\ : & : & \cdots & : \\ x_{1,i} & x_{2,i} & \cdots & x_{p,i} \\ : & : & \cdots & : \\ x_{1,n} & x_{2,n} & \cdots & x_{p,n} \end{array}$$

$$Y = \begin{array}{c} OP_1 \\ OP_2 \\ : \\ OP_p \end{array} \begin{bmatrix} y_{1,1} & \cdots y_{1,j} \cdots & y_{1,n} \\ y_{2,1} & \cdots y_{2,j\ldots} & y_{2,n} \\ : & : & : \\ y_{p,1} & \cdots y_{p,j\ldots} & y_{p,n} \end{bmatrix}$$

From the above sets of vectors (matrices) X^T and Y, it is obvious that Eqn. (1) is the dot product of i-th row of X^T and the j-th column of Y. But since the output vector in case of autoassociation is one of the earlier memorized vectors it is one of vectors in X itself and therefore the weight calculation autoassociation networks is,

$$W = X^T X \qquad (3)$$

Consider the following procedure to explain the associative memory network model.

Input: Pattern (often noisy/corrupt)

Output: Corresponding pattern (complete/relatively noise-free)

Process:

1. Load input pattern onto highly-interconnected neurons (a trained network on given set of library patterns).
2. Run the neurons until they reach a steady state.
3. Read output off the states of the neurons.

Subsequently Fig. 3 and fig. 4 give a graphical illustration of the procedure.

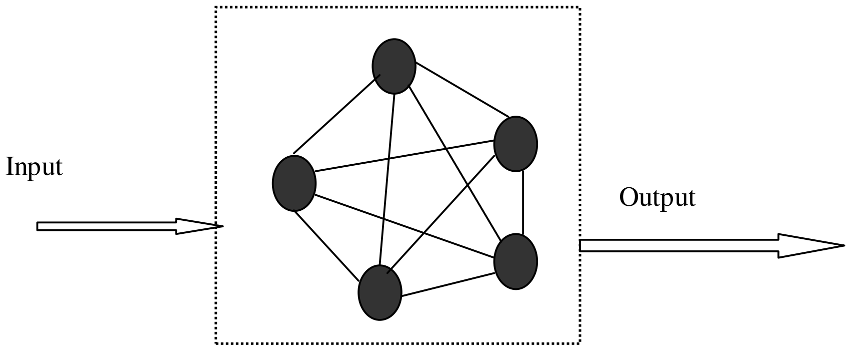


Fig. 3: Trained associative network

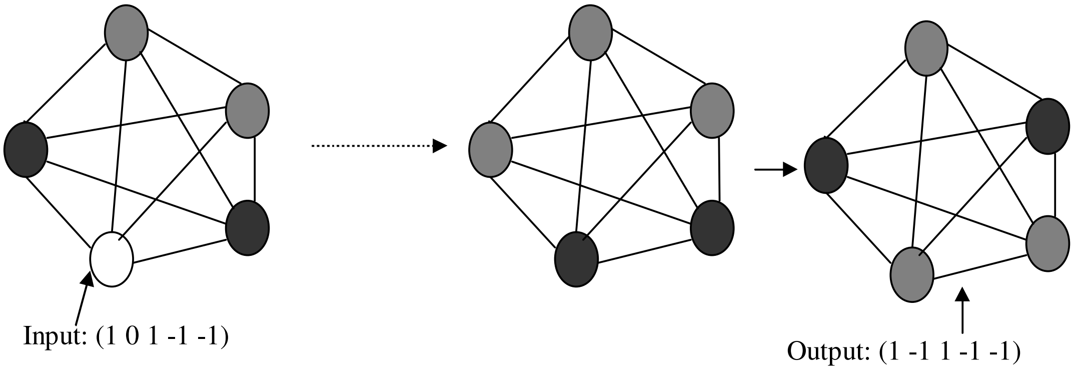


Fig. 4: Running an input pattern

Now, let us discuss Hopfield networks in the following section.

9.3 HOPFIELD NETWORKS

The Hopfield Net proposed by J.J. Hopfield in 1982 is a neural network that is a lot simpler to understand than the Multi Layer Perceptron. For a start, it only has one layer of nodes. Unlike the MLP, it doesn't make one of its outputs go high to show which of the patterns most closely matches the input. Instead it takes the input pattern, which is assumed to be one of the library patterns which has been changed or corrupted somehow, and tries to reconstruct the correct pattern from the input that you give it.

Hopfield networks are Recurrent neural network model that possesses auto-associative property. Therefore the network is fully interconnected with the exception that no neuron has any connection to itself. Thus, the Hopfield network is a form of artificial neural network that serve as content-addressable memory systems with binary threshold units. They are guaranteed to converge to a stable state. The aim of the Hopfield network is to recognize a partial or distorted input pattern as one of its previously memorized vectors and to output the perfect and complete memorized version.

Thus the Hopfield networks proposed as a theory of memory has the following features:

1. **Distributed Representation:** A memory is stored as a pattern of activation across a set of processing elements. Further, the memories can be superimposed on one another; different memories are represented by different patterns over the same set of processing elements.
2. **Distributed Asynchronous Controls:** Each processing element makes decisions based only on its own local situation. All these local actions add up to a global solution.
3. **Content-addressable Memory:** A number of patterns can be stored in a network. To retrieve a pattern, we need to only specify a portion of it. The network automatically finds the closest match.
4. **Fault Tolerance:** If a few processing elements misbehave or fails completely, the network will still function properly.

The main concept underlying the Hopfield network is that a single network of interconnected, binary-valued neurons can store multiple stable states, the library patterns also called *attractors*, similar to the brain – the full pattern can be recovered if the network is presented with only partial information just as described in associative memories. Furthermore, there is a degree of stability in the system, if just a few of the connections between neurons are severed, the recalled memory is not too badly corrupted, the network can respond with a "best guess". The nodes in the Hopfield network are vast simplifications of real neurons, they can only exist in one of two possible **"states"** - {firing, not firing} or $\{0, 1\}$ or $\{1, -1\}$. Every node is connected to every other node with some strength (synaptic weights). At any instant of time a node will change its state (i.e. start or stop firing) depending on the inputs it receives from the other nodes.

If we start the system off with any general pattern of firing and non-firing nodes then this pattern will in general change with time. To see this think of starting the network with just one firing node. This will send a signal to all the other nodes via its connections so that a short time later some of these other nodes will fire. These new firing nodes will then excite others after a further short time interval and a whole cascade of different firing patterns will occur. One might imagine that the firing pattern of the network would change in a complicated perhaps random way with time. The crucial property of the Hopfield network which renders it useful for simulating memory recall is the following: it guarantees that the pattern will settle down after a long enough time to some fixed pattern. Certain nodes will be always **"on"** and others **"off"**. Furthermore, it is possible to arrange that these stable firing patterns of the network correspond to the desired memories we wish to store.

Example 1: Suppose we have trained our Hopfield Net on the three patterns given in fig. 5:

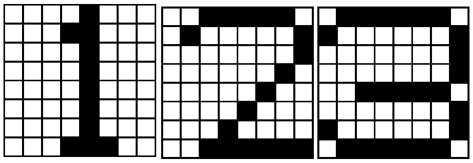


Fig. 5

Next we input a pattern which is a bit like one of these, say the left most one given in the Fig. 6. Leave the network to run. It gradually alters the pattern we give it until it has reconstructed one of the originals ones, the right most one.

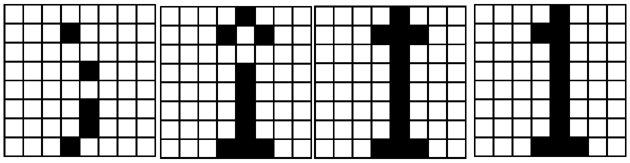


Fig. 6

The Hopfield Net is left to iterate (loop round and round) until it doesn't change any more. Then it *should* match one of the input patterns. The program should then compare the reconstructed pattern with the library of originals, and see which one matches.

9.4 STRUCTURE OF HOPFIELD NETWORKS

The structure of the Hopfield networks model is shown in Fig. 7.

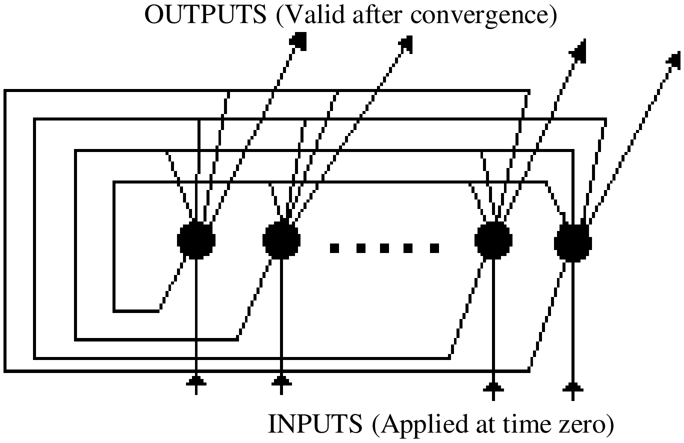

OUTPUTS (Valid after convergence)
INPUTS (Applied at time zero)

Fig. 7: Hopfield Network Model

The inputs to the Hopfield Net are at the bottom. The nodes produce an output which comes out at bottom and is fed back into all the nodes except the one that produced it (i.e. all the nodes refer to each other, but not to themselves) and uses this to produce the next output. The final output of the nodes is extracted at the top. For a network of N neurons, the state of the network is denoted by the vector,

$$\mathbf{s} = (s_1, s_2, \dots, s_N)^T$$

Each $s_j = \pm 1$, the state of neuron j represents the one bit of information, and the $N \times 1$ state vector $\mathbf{s}$ represents a binary word of N bits of information. As the units/neurons have a binary threshold units, the dynamical rule has the possibility of two definitions for a_i the activation of neuron i :

$$a_i = \begin{cases} 1 & \text{if } \sum_j w_{ij}s_j \geq \theta_i, \\ -1 & \text{otherwise.} \end{cases} \tag{4}$$

or,

$$a_i = \begin{cases} 1 & \text{if } \sum_j w_{ij}s_j \geq \theta_i, \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

where, w_{ij} is the connection (synaptic) weight from neuron i to neuron j as in Eqn. (1), s_j is the state of neuron (unit) j and θ_i is the threshold of the neuron i. The above relation in Eqns. (4) and (5) may be written in the compact form,

$$a_i = \text{sgn}[\Sigma w_{ij}\, s_j] \text{ for all } j = 1, 2, \ldots, N$$

where, sgn is the signum function. If a_i is zero then the action can be arbitrary. However, the neuron I may remain in the previous state regardless of whether it is on or off. The feedback network with no self-looping, Hopfield networks also have the following two restrictions on due to the kinds of interconnection that prevails:

i) $w_{ii} = 0$ for all neurons i. i.e. there is no loop/connection to itself, and

ii) $w_{ij} = w_{ji}$ for all neurons. i.e. the connections are symmetric.

The Hopfield nets also have a scalar value associated with each state of the network referred to as the energy, E of the network. Energy is defined as below,

$$E = -\sum_{i<j} w_{ij}s_i s_j + \sum_i \theta_i s_i \tag{6}$$

The reason for this is somewhat technical but we can proceed by analogy. Imagine a ball rolling on some bumpy surface. We imagine the position of the ball at any instant to represent the activity of the nodes in the network. Memories will be represented by special patterns of node activity corresponding to wells in the surface. Thus, if the ball is let go, it will execute some complicated motion but we are certain that eventually it will end up in one of the wells of the surface. We can think of the height of the surface as representing the energy of the ball. We know that the ball will seek to minimize its energy by seeking out the lowest spots on the surface-the wells.

In the language of memory recall, if we start/ initiate the network with a pattern for firing, the network must approximate one of the "stable firing patterns" in the memories. However, it will "under its own steam" end up in a nearby (with respect to the threshold) well in the energy surface thereby recalling the original perfect memory.

In the following section, we shall focus on the functionality of Hopfield networks.

9.5 THE FUNCTIONALITY OF HOPFIELD NETWORKS

As mentioned earlier that Hopfield nets are a special case of associative networks or content-addressable memory nets. Therefore, Hopfield nets too have the corresponding two phases namely, Storage phase and the Retrieval phase.

Storage Phase: Let us have M, N-dimensional vectors to be stored in the Hopfield nets, say $P = \{p_k \mid k = 1, 2, \ldots, M\}$. The m vectors compose the library or the fundamental memories, representing the patterns to be memorized by the network. Let $p_{k,I}$ denote the i-th element of the fundamental memory p_k. According to the Hebb's rule the synaptic weights from neuron i to j can be computed using Eqn. (1) i.e.

$$w_{ij} = 1/N(p_{1,i}\; p_{1,j} + p_{2,i}\; p_{2,j} + \cdots + p_{M,i}\; p_{M,j}) \tag{7}$$

and $w_{ii} = 0$ for all neurons $i = 1, 2, \ldots, M$. Let W denotes the $N \times N$ synaptic weight matrix or connectivity matrix of the network.

$$W = \frac{1}{N}\sum_{k}^{M} p_k p_k^T - MI \tag{8}$$

where, $p_k p_k{}^T$ denoted the dot product of the k-th vector and its transpose. I is the identity matrix. From the above computation the following three points are reconfirmed:

- The output of each neuron in the network is fed back to all other neurons.
- There is no self-looping in the network.
- The weight matrix of network is symmetric.

Retrieval Phase: This phase is initiated on input of an N-dimensional vector as input say $\mathbf{p}_t$, to the trained network. This is also called a probe. Typically the probe has elements ± 1 and is thought to be a noisy or incomplete version of any of the attractors already in the library/ fundamental memory. The information retrieval starts with a dynamical rule in which each neuron i of the network randomly but at some fixed rate examines its activation function a_i as a result of the connectivity its neighbouring neurons. While examining if a_i is greater than zero then the state is changed to 1 else remain in the previous state. But if the value of a_i is less than zero then switch state to -1. However, if the value of a_i is equal to zero then the state is left in its previous state regardless of whether it was in on or off state. The updating of the states of the neurons is deterministic but selection of the neurons for updating states is done randomly in serial (asynchronous) or synchronous. Starting with the test input or the probe vector the network must finally reach a state of stability, or produce time invariant state vector as the output pattern i.e. $\mathbf{y}$, whose individual elements satisfy the following stability condition,

$$y_i = \operatorname{sgn}\left(\sum_{i=1}^{N} w_{j,i} p_{ti}\right), \; j = 1, 2, \ldots, N. \tag{9}$$

or the matrix form,

$$\mathbf{y} = \operatorname{sgn}(\mathbf{W p_t}) \tag{10}$$

Example 2: Consider the following example:

i) Input Patterns $M = 3, N = 4, 3$ 4-dimensional vector.

IP1	1	1	1	–1
IP2	1	1	–1	1
IP3	– 1	1	1	–1

ii) The average correlation across the three patterns to design and train the Hopfield network.

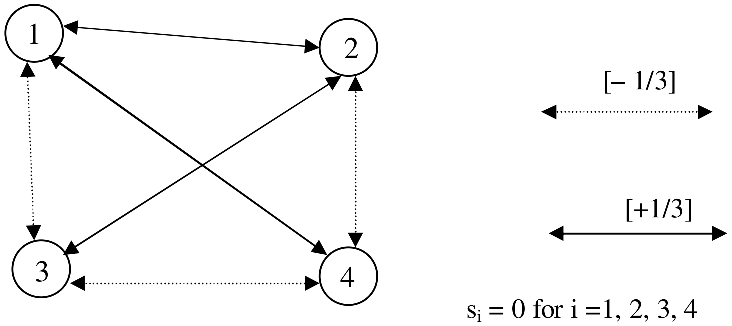
$w_{i,j}$	IP1	IP2	IP3	Avg
$w_{1,2}$	1	1	−1	1/3
$w_{1,3}$	1	−1	−1	−1/3
$w_{1,4}$	−1	1	1	1/3
$w_{2,3}$	1	−1	1	1/3
$w_{2,4}$	−1	1	−1	−1/3
$w_{3,4}$	−1	−1	−1	−1

$$X = \begin{bmatrix} 1 & 1 & 1 & -1 \\ 1 & 1 & -1 & 1 \\ -1 & 1 & 1 & -1 \end{bmatrix}$$

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

The weight for the auto association network is $W = X^T X = \begin{bmatrix} 3 & 1 & -1 & 1 \\ 1 & 3 & 1 & -1 \\ -1 & 1 & 3 & -3 \\ 1 & -1 & -3 & 3 \end{bmatrix}$

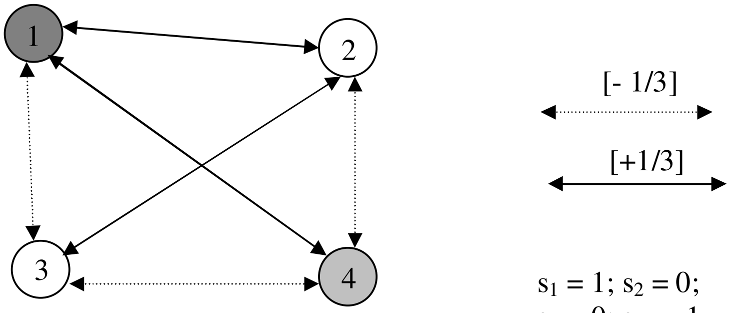
Here it is clear that in the connectivity matrix $w_{2,4} = w_{4,2}$ or in general the lower and the upper triangles of the product matrix represents the 6 weights $w_{i,j} = w_{j,i}$. We can scale the weights by dividing them by $p = 3$. For the iterative purposes it is easier to scale by p at the end instead of scaling the entire weight matrix W prior to testing. Now let us construct the Hopfield net and training it with the inputs IP_1, IP_2, IP_3.



$s_i = 0$ for i =1, 2, 3, 4

Fig. 8: Hopfield network

iii) Now consider the testing input = (1 0 0 –1).



$s_1 = 1$; $s_2 = 0$;
$s_3 = 0$; $s_4 = -1$

Fig. 9: With input (1 0 0 –1)

iv) Synchronous iteration to update all the nodes.

$$(1\ 0\ 0\ -1)\begin{pmatrix} 3 & 1 & -1 & 1 \\ 1 & 3 & 1 & -1 \\ -1 & 1 & 3 & -3 \\ 1 & -1 & -3 & 3 \end{pmatrix} = (2\ 2\ 2\ -2)$$

Table 1

Inputs					
Node	1	2	3	4	Output
1	**1**	0	0	-1/3	1
2	1/3	**0**	0	1/3	1
3	-1/3	0	**0**	1	1
4	1/3	0	0	**-1**	-1

The diagonal represents the values from the input layer in the above table depicting the result of the synchronous iteration modeled by the preceding computation.

Now scaling the output by dividing by $p = 3$ and using the threshold $\theta = 0$, we obtain the vector $(2/3\ 2/3\ 2/3\ -2/3) \Rightarrow (1\,1\,1\,-1)$ the same as IP_1 an attractor in the library. Consider the following figure to display the processing in the Hopfield net to reach a stable state again with the output vector as computed earlier. The graphical representation of the output is given in Fig. 10.

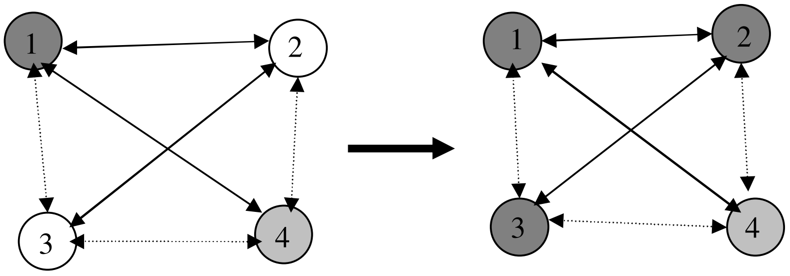


Fig. 10: (1 0 0 –1) → (1 1 1 –1)

Now, try some exercises.

E1) Run the above Hopfield network for the test input vector $(1\;0\;0\;0)$.

E2) Draw the graph of the input to output vector obtained in E1).

The smart thing about the Hopfield network is that there exists a rather simple way of setting up the connections between nodes in such a way that any desired set of patterns can be made "stable firing patterns". Thus any set of memories can be burned into the network at the beginning. Then if we kick the network off with any old set of node activity we are *guaranteed* that a "memory" will be recalled. Not too surprisingly, the memory that is recalled is the one which is "closest" to the starting pattern. In other words, we can give the network a corrupted image or memory and the network will "all by itself" try to reconstruct the perfect image. Of course, if the input image is sufficiently poor, it may recall the incorrect memory–the network can become "confused"–just like the human brain. We know that when we try to remember someone's telephone number we will sometimes produce the wrong one! Notice also that the network is reasonably robust–if we change a few connection strengths just a little the recalled images are "roughly right". We don't lose any of the images completely.

Example 3: Consider a Hopfield network whose weight matrix is given by,

$$\mathrm{w} = \frac{1}{3}\begin{bmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \\ 2 & -2 & 0 \end{bmatrix}$$

Consider a test input vectors $\mathrm{pt}_1 = (1\;-1\;1)$ and $\mathrm{pt}_2 = (-1\;1\;-1)$.

Using the Eqn. (9) and then its transpose results in the desired vector.

$$\text{Therefore,}\quad \mathbf{Wpt}_1^{\mathbf{T}} = \frac{1}{3}\begin{bmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \\ 2 & -2 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} = \frac{1}{3}\begin{bmatrix} 4 \\ -4 \\ 4 \end{bmatrix}$$

$$\mathbf{y}_1 = \mathrm{Sgn}\;(\mathbf{Wpt}_1) = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}$$

Since the CM or the weight matrix is symmetric we can also directly get the vector as by the product of vector and the matrix, i.e. **ptW**

$$\mathbf{y}_2 = \mathrm{sgn}\,(\mathbf{pt}_2\mathbf{W}) = (-1\;1\;-1)\frac{1}{3}\begin{bmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \\ 2 & -2 & 0 \end{bmatrix} = (-1\;1\;-1)$$

Eqn. (10) is also known as **alignment condition**. The state vectors that satisfy the alignment condition are stable states. Therefore, the given patterns are stables state vector too.

If we consider the following vectors as probes for this example, $(-1\;-1\;1)$, $(1, 1, 1)$ or $(1\;-1\;-1)$. The resulting output is $(1\;-1\;1)$. Note that each of the probes had single error compared to the stored memory.

Similarly, consider another set of test patterns to input to the Hopfield network $(1\,1\,-1), (-1\,-1\,-1)$ or $(-1\,1\,1)$. The resulting output vector in this case is computed to be $(-1\,1\,-1)$. In this case too the test patterns are noted to have single error.

Now, try the following exercises.

E3) Run the Hopfield network of Example 2 for the test input vector $(1\,1\,0\,0)$. Also, draw the graphical representation of the output.

E4) Consider the set of pattern vectors P. Obtain the connectivity matrix (CM) for the patterns in P (four patterns).

$$P = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

In the following Section, we shall discuss the storage capacity Hopfield Networks.

9.6 HOPFIELD NETWORKS: STORAGE CAPACITY

There are two major limitations of Hopfield networks. One is the tendency to local minima, which is good for association but bad for optimisation. The other limitation is on the network capacity. For a network of N binary nodes, the capacity limit is of the order of N rather than 2^N. If the patterns or the vectors are N-dimensional then there must be network of N binary nodes.

However, the capacity is equal to the relationship between the number of patterns that can be stored & retrieved without error to the size of the network. Let M be the number of patterns to be stored or fundamental memories. So long as the storage capacity is of the network is not overloaded i.e., M is small compared to N.

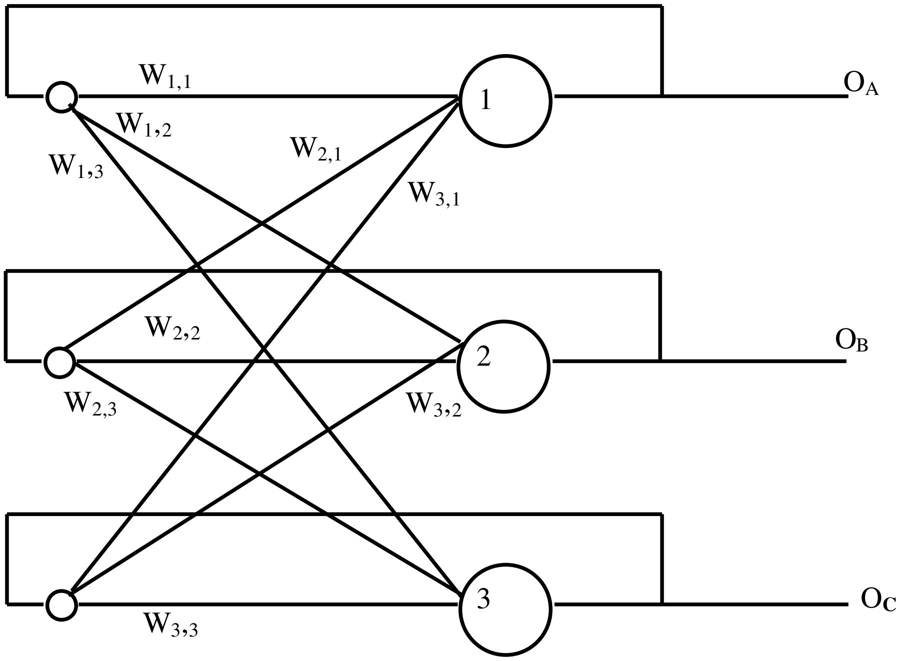
$$\text{Capacity} = \frac{M}{N} \text{ or, } \frac{M}{\text{Number of weights}}$$

If we use the following definition of 100% correct retrieval, when any of the stored patterns is entered completely (no noise), then that same pattern is returned by the network; i.e. The pattern is a stable attractor. A detailed proof shows that a Hopfield network of N nodes can achieve 100% correct retrieval on P patterns if: $P < N/(4*\ln(N))$.

N	Max P
10	1
100	5
1000	36
10000	271
10^{11}	10^{9}

In general, as more patterns are added to a network, the average correlations will be less likely to match the correlations in any particular pattern. Hence, the likelihood of retrieval error will increase. Just as it happens in case of human memory. When the information is loaded/crammed into a memory, or patterns to store recall is slower than when there are less patterns to remember. The key to perfect recall is selective ignorance!

Example 4: A three neuron network trained with three patterns.



Let’s say we’d like to train three patterns:

Pattern number one: □□■ $O_{A(1)} = -1\ O_{B(1)}\ = -1\ O_{C(1)} = -1$

Pattern number two: ■□□ $O_{A(2)} = 1\ O_{B(21)}\ = -1\ O_{C(2)} = -1$

Pattern number three: □■■ $O_{A(3)} = -1\ O_{B(3)}\ = 1\ O_{C(3)} = 1$

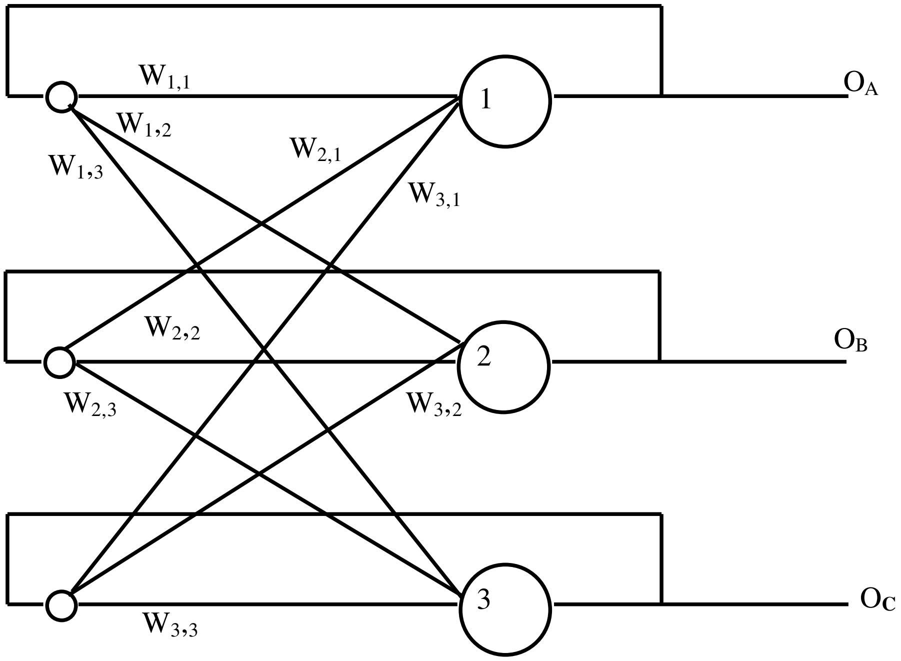
$$W_{1,1} = 0$$
$$W_{1,2} = O_{A(1)} \times O_{B(1)} + O_{A(2)} \times O_{B(2)} + O_{A(3)} \times O_{B(3)} = (-1)\times(-1) + 1\times(-1) + (-1)\times 1 = -1$$
$$W_{1,3} = O_{A(1)} \times O_{C(1)} + O_{A(2)} \times O_{C(2)} + O_{A(3)} \times O_{C(3)} = (-1)\times 1 + 1\times(-1) + (-1)\times 1 = \quad -3$$

$$W_{2,2} = 0$$
$$W_{2,1} = O_{B(1)} \times O_{A(1)} + O_{B(2)} \times O_{A(2)} + O_{B(3)} \times O_{A(3)} = (-1)\times(-1) + (-1)\times 1 + 1\times(-1) = -1$$
$$W_{2,3} = O_{B(1)} \times O_{C(1)} + O_{B(2)} \times O_{C(2)} + O_{B(3)} \times O_{C(3)} = (-1)\times 1 + (-1)\times(-1) + 1\times 1 = \quad 1$$

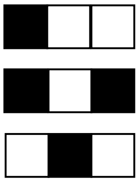
$$W_{3,3} = 0$$
$$W_{3,1} = O_{C(1)} \times O_{A(1)} + O_{C(2)} \times O_{A(2)} + O_{C(3)} \times O_{A(3)} = 1\times(-1) + (-1)\times 1 + 1\times(-1) = \quad -3$$
$$W_{3,2} = O_{C(1)} \times O_{B(1)} + O_{C(2)} \times O_{B(2)} + O_{C(3)} \times O_{B(3)} = 1\times(-1) + (-1)\times(-1) + 1\times 1 = \quad 1$$

Now try the following exercise.

E5) Train this network with the three patterns shown.



Patterns:



Applications of the Hopfield Network

1. The Hopfield network can be used as an effective interface between analog and digital devices, where the input signals to the network are analog and the output signals are discrete values.
2. Associative memory is a major application of the Hopfield network.
3. The energy minimization ability of the Hopfield network is used to solve optimization problems.
4. The various applications of Hopfield networks are as given below. Hopfield network remembers cues from the past and does not complicate the training procedure.
5. The Hopfield network can be used for converting analog signals into the digital format, for associative memory.

Now, let us summarize this unit.

9.7 SUMMARY

The summary of this unit includes the four basic operations to design and run a Hopfield network. These four operations are–learning, initialization (of the network), iteration until convergence, and finally outputting the output vector.

i) **Learning/Training :** Let $p_1, p_2, \ldots, p_m$ be the N-dimensional patterns to be stored. Using the dot product rule, the Hebbian postulate of learning, the synaptic weights of the entire recurrent network prohibiting self-looping is computed by,

$$w_{i,j} = \begin{cases} \dfrac{1}{N}\sum_{k=1}^{M} p_{k,i}p_{k,j} & i \neq j \\ 0 & i = j \end{cases}$$

ii) **Initialization:** Let $\mathbf{p}_t$ be the n-dimensional vector probe or the test input to the Hopfield network. The initialization is carried out by,

$$s_i(0) = p_{i,t},\ i = 1, 2, \ldots, N$$

where $s_i(0)$ is the state of the i-th neuron at time $n = 0$, and $p_{i,t}$ the i-th element of the test input p_t.

iii) **Iteration until convergence:** Update the elements of the state vector $s(n)$ synchronously or asynchronously using the rule,

$$s_i(n+1) = \operatorname{sgn}\left[\sum_{j=1}^{N} w_{ij}s_i(n)\right],\ i = 1, 2, \ldots, N$$

Repeat the iteration until the state of the neurons do not change i.e. the state vector s remains unchanged.

iv) **Outputting:** Let s_{fixed} denote the stable states computed at the end of above iterative step. The resulting pattern OP of the network is,

$$OP = s_{\text{fixed}}$$

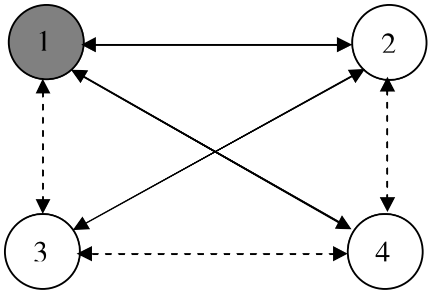
The storage phase uses the learning/training operation. The subsequent operations initialization, iteration until convergence and outputting are applied in the retrieval phase.

9.8 SOLUTIONS/ANSWERS

E1) This test pattern too has elements other than ± 1. Using the Eqn. (4) or **ptW** and dividing the resultant by 3 the vector obtained is $(1\,1\,-1\,1)$. The calculation is given as below.

Input = (1 0 0 0)

The corresponding graphical representation of the Hopfield network is given as



The synchronous iteration to update the nodes is given by

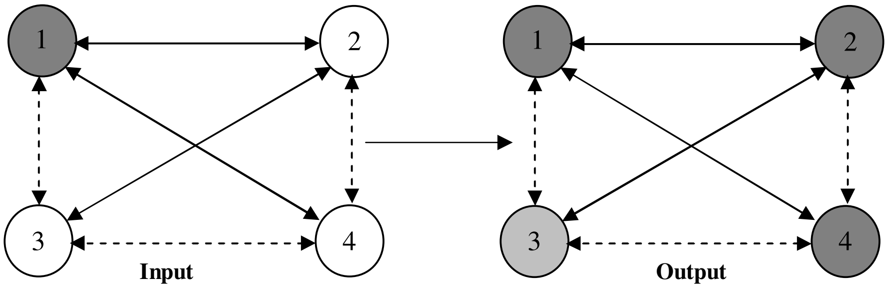
$$\begin{pmatrix}1 & 0 & 0 & 0\end{pmatrix}\begin{pmatrix}3 & 1 & -1 & 1\\ 1 & 3 & 1 & -1\\ -1 & 1 & 3 & -3\\ 1 & -1 & -3 & 3\end{pmatrix}$$

$$=\begin{pmatrix}3 & 1 & -1 & 1\end{pmatrix}$$

Nodes	1	2	3	4	Output
1	**1**	0	0	0	1
2	1/3	**0**	0	0	1
3	– 1/3	0	**0**	0	-1
4	1/3	0	0	**0**	1

Which is a stable state and one of the input patterns stored in the library. Though the asynchronous update results in spurious output, the synchronous update gives a pattern from the fundamental memory.

E2)



E3) First of all the input definitely has some serious noise therefore the value 0 which is not there in the input patterns is present in the probe. However, use the Eqn. (4) or $\mathbf{ptW} = (1\,1\,0\,0)$, which is a stable state, but not one of the input patterns stored in the library.

E4) Here $N = 10$ and $M = 4$. Use either the Eqn. (6) or Eqn. (7) to obtain the weights for each $w_{i,j}$ for all i and $j = 1, 2, \ldots, 10$ but $i \neq j$. Substitute the values of P^T and P in Eqn. (7) we get $CM = 1/4P^TP - 10I_{10}$.

9.9 PRACTICAL ASSIGNMENT

Session 8

Write a program in ‘C’ language to find

i) The average correlation matrix for given input patterns M and N to design and train the Hopfield Network.

ii) The weight matrix.

iii) Output of the Hopfield network.

Also test your program on the input patterns given in Example 2.