# NEURAL NETWORKS

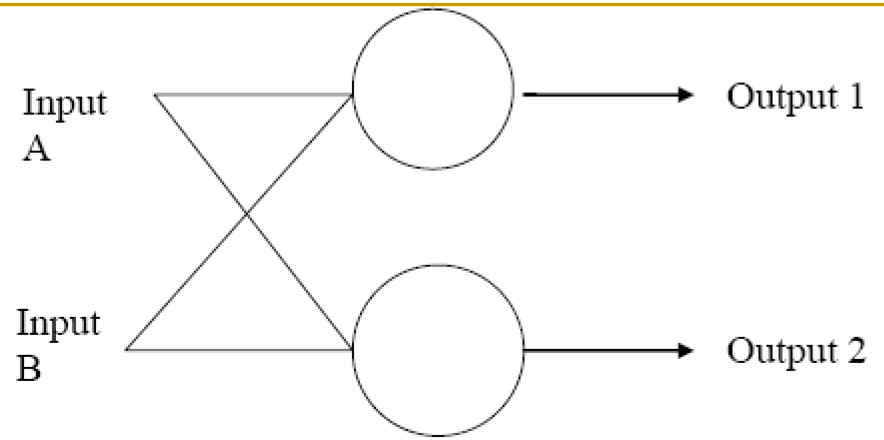
# Hopfield Neural Networks

Lecture 8

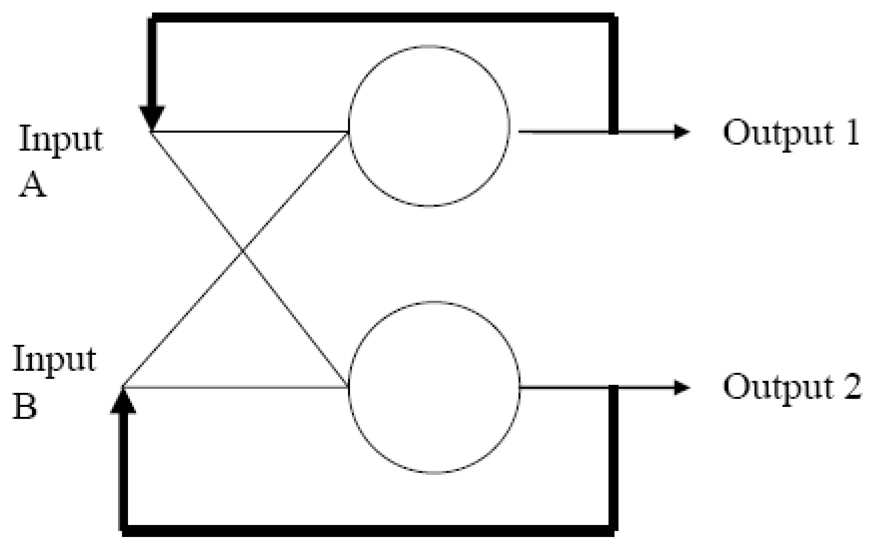
Asst. Prof.Dr. Sibel SENAN
ssenan@istanbul.edu.tr

# HOPFIELD NEURAL NETWORKS

- In 1983, a physicist called John Hopfield published the famous paper "*Neural networks and physical systems with emergent collective computational abilities*".
- What Hopfield did was to add *feedback connections* to the network (the outputs are fed back into the inputs)




Feed Forward Network




Same network with Feedback connections

**RECURRENT Network**

- The network operates in a very similar way to the feedforward ones explained earlier and the neurons have basically the same function.
- We apply inputs to A and B and calculate the outputs (as before).
- The difference is that once the output is calculated, we feed it back into the inputs again. So, we take output 1 and feed it into input A and likewise feed output 2 into input B.
- This gives us two new inputs (the previous outputs) and the process is repeated.
- We keep on doing this until the outputs don’t change any more (they remain constant).
- At this point the network is said to have *relaxed*.
- The process is shown in Figure 1.

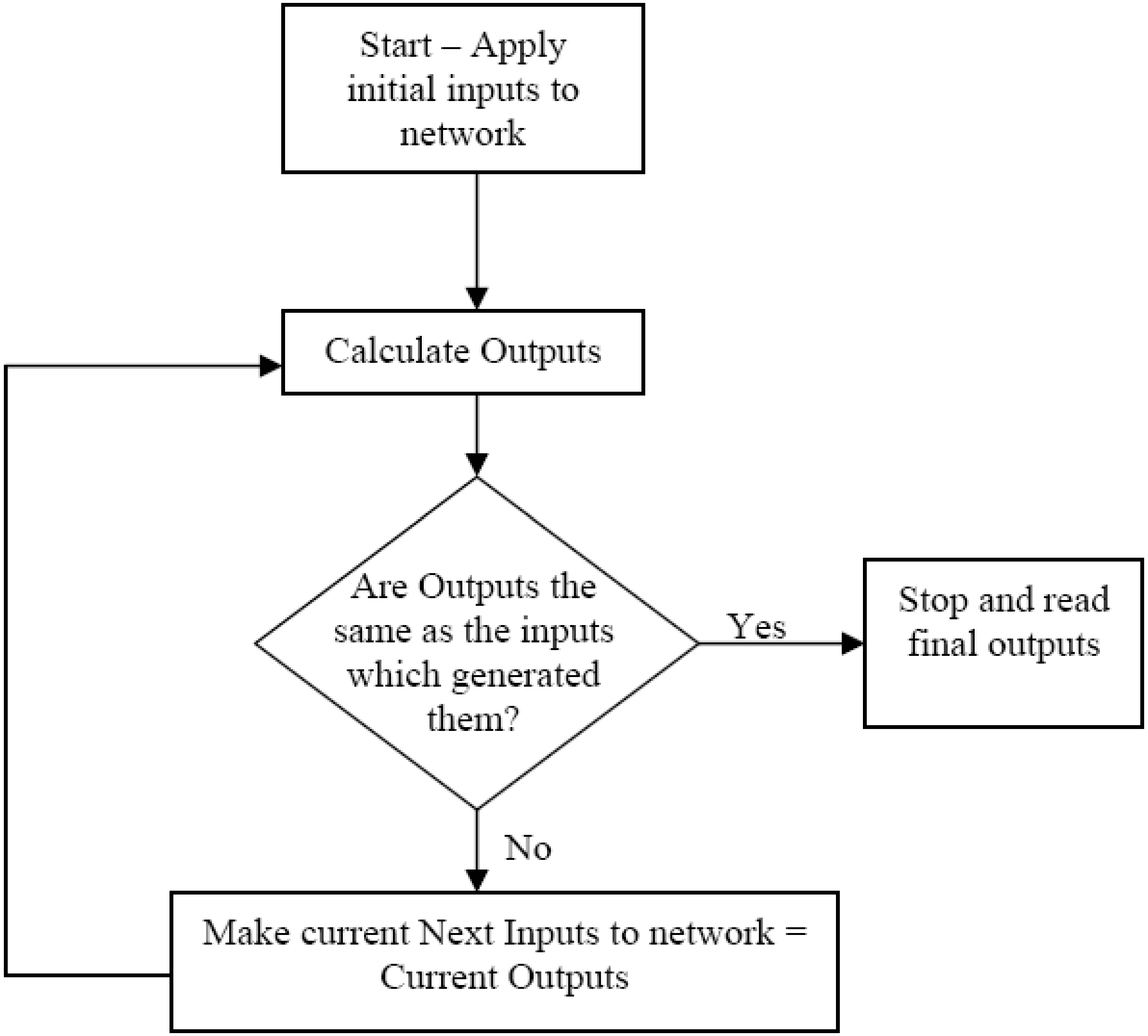
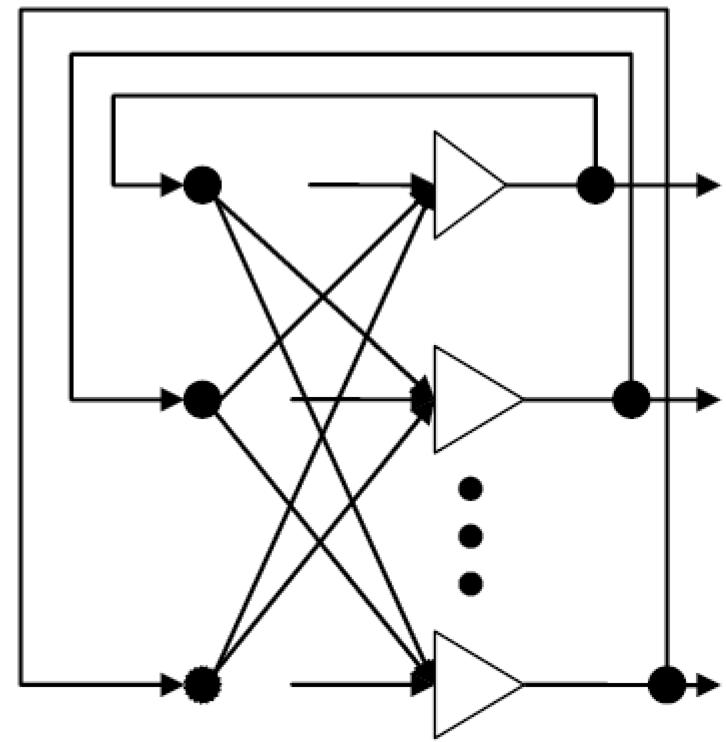



**Figure 1.** *The process of applying inputs to a feedback network*.

# HOPFIELD NEURAL NETWORKS

- The ***Hopfield Network/Model*** is a fully connected, one layer, recurrent network that deals with the basic associative memory problem:
  - Store a set of *P* binary valued patterns $\{\mathbf{t}^p\} = \{t^{ip}\}$ in such a way that when presented with a new pattern $\mathbf{s} = \{s_i\}$ the network responds by producing whichever stored pattern most closely resembles **s**.

- Hopfield neural network (HNN) is a model of auto-associative memory.
- The structure is shown in the right figure.
- It is a single layer neural network with feedbacks.



# The state-transition mechanism

Suppose that the current state of the network is

$$\mathbf{v}^k = [v_1^k, v_2^k, \ldots, v_n^k]$$

then, the next state can be calculted by

$$v_i^{k+1} = \operatorname{sgn}(net_i) = \operatorname{sgn}(\sum_{\substack{j=1 \\ j \neq i}}^{n} w_{ij} v_j^k + \theta_i)$$

where $\theta_i$ is the threshold of the $i$th neuron

- Note that the update is asynchronous. That is, one neuron is updated each time, and the update order is random.

- Note also that $w_{ij} = w_{ji}$ and $w_{ii} = 0$ for all $i$.

*Ref. Lecture slides of Qiangfu Zhao*

- A Hopfield network can *reconstruct* a pattern from a corrupted original as shown in Figure 2.

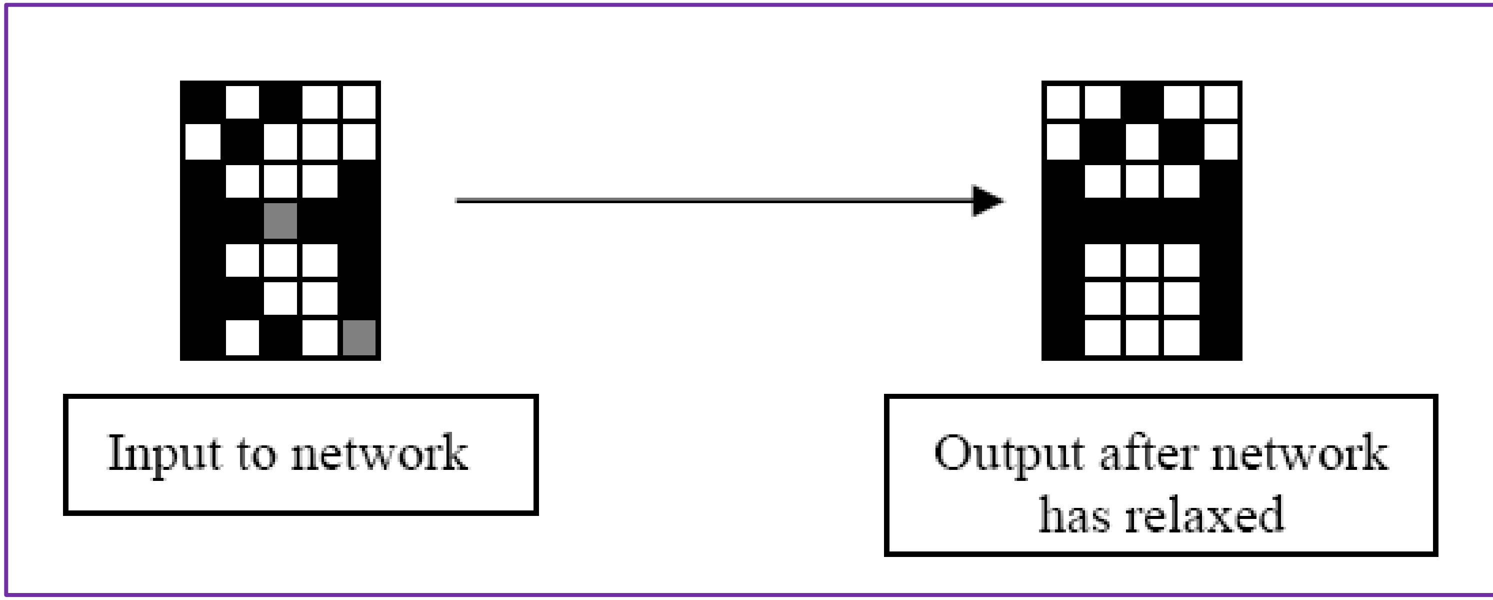



**Figure 2.** *Reconstructing a corrupted pattern.*

✓ This means that the network has been able to store the correct (uncorrupted) pattern – in other words it has a memory. Because of this these networks are sometimes called *Associative Memories* or *Hopfield Memories*.

# Why Hopfield neural network is an associative memory ?

- Starting from any initial state, the HNN will change its state until the energy function approaches to the minimum.
- The minimum point is called the ***attractor.***
- Patterns can be stored in the network in the form of attractors.
- The initial state is given as the input, and the state after convergence is the output.

*Ref. Lecture slides of Qiangfu Zhao*

# How to store the patterns ?

Suppose that we have $\boldsymbol{p}$ patterns to be stored.

We can calculate the weight matrix as follows :

$$\boldsymbol{W} = \sum_{m=1}^{p} \boldsymbol{s}^m (\boldsymbol{s}^m)^T - p\boldsymbol{I}$$

where $\boldsymbol{s}^m$ is the $m$ - $th$ pattern (a column vector), and $\boldsymbol{I}$ is the unit matrix. The thresholds of all neurons are set to zeros.

*Ref. Lecture slides of Qiangfu Zhao*

# How to store the patterns (cont.)?

If the patterns take value from $\{-1,1\}$, the weights are given by

$$w_{ij} = (1-\delta_{ij})\sum_{m=1}^{p} s_i^m s_j^m$$

where $\delta_{ij}$ is the Kronecker function defined by

$$\delta_{ij} = \begin{cases} 1 & i=j \\ 0 & \text{otherwise} \end{cases}$$

If the patterns take value from $\{0,1\}$, we have

$$w_{ij} = (1-\delta_{ij})\sum_{m=1}^{p} (2s_i^m - 1)(2s_j^m - 1)$$

*Ref. Lecture slides of Qiangfu Zhao*

# How to use a HNN ?

- Phase 1: Store all patterns into the network by finding the weight matrix as above.

- Phase 2: Recall a pattern when an input is given as the initial state.

*Ref. Lecture slides of Qiangfu Zhao*

# Phase 1: Storage

- Step 1 Initialization: $\boldsymbol{W=0}$
- Step 2 Store the *m-th* pattern $s^{(m)}$ by

$$\boldsymbol{W}=\boldsymbol{W}+\boldsymbol{s}^{(m)}(s^{(m)})^t$$

- Step 2 is repeated for all patterns.
- After all patterns are stored, set $w_{ii}=0$ for $i=0,1,\ldots,n$.

*Ref. Lecture slides of Qiangfu Zhao*

# Phase 2: Recalling

- Step 1 Present an input (key) vector to the network
  $x_j(0)$ : key vector, n=0 (time)
- Step 2 Update the elements of state vector $x(n)$ according to the rule
  $$x_j(n+1) = \mathrm{sgn}[\sum_{i=1}^{N} w_{ij} x_i(n)], \quad j=1,2,\ldots N$$
- Step 3 Repeat Step 2 until the state vector $x$ remains unchanged.
- Step 4 Let $x_{fixed}$ denote the fixed point (stable state) computed at the end of Step 3. The resulting output vector y of the network is
  $$y = x_{fixed}$$

- In the recalling phase for Step 2

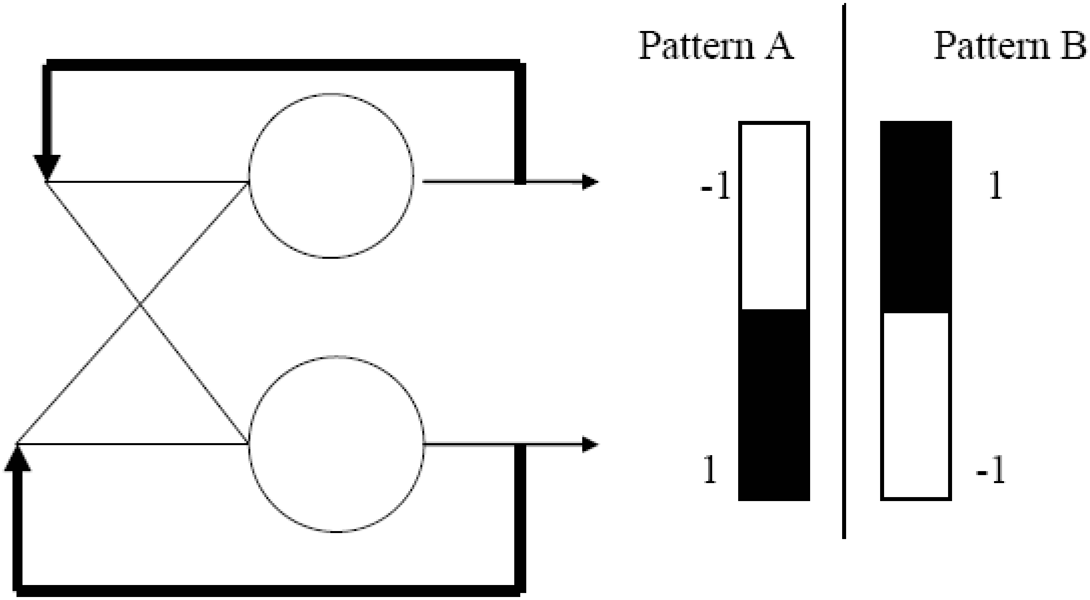
$$v_j = \sum_{i=1}^{N} w_{ij} x_i(n)$$

is local induced field for for neuron j.

- Here, neuron j modifies its state $x_j$ according to the rule :
    - If $v_j$ is greater than zero, $x_{j+1}$ will be 1
    - If $v_j$ is less than zero, $x_{j+1}$ will be -1
    - If $v_j$ is exactly zero, $x_{j+1}$ will remain in its previous state.

# ➢ Example(1) for Storage Phase

a simple Hopfield network.

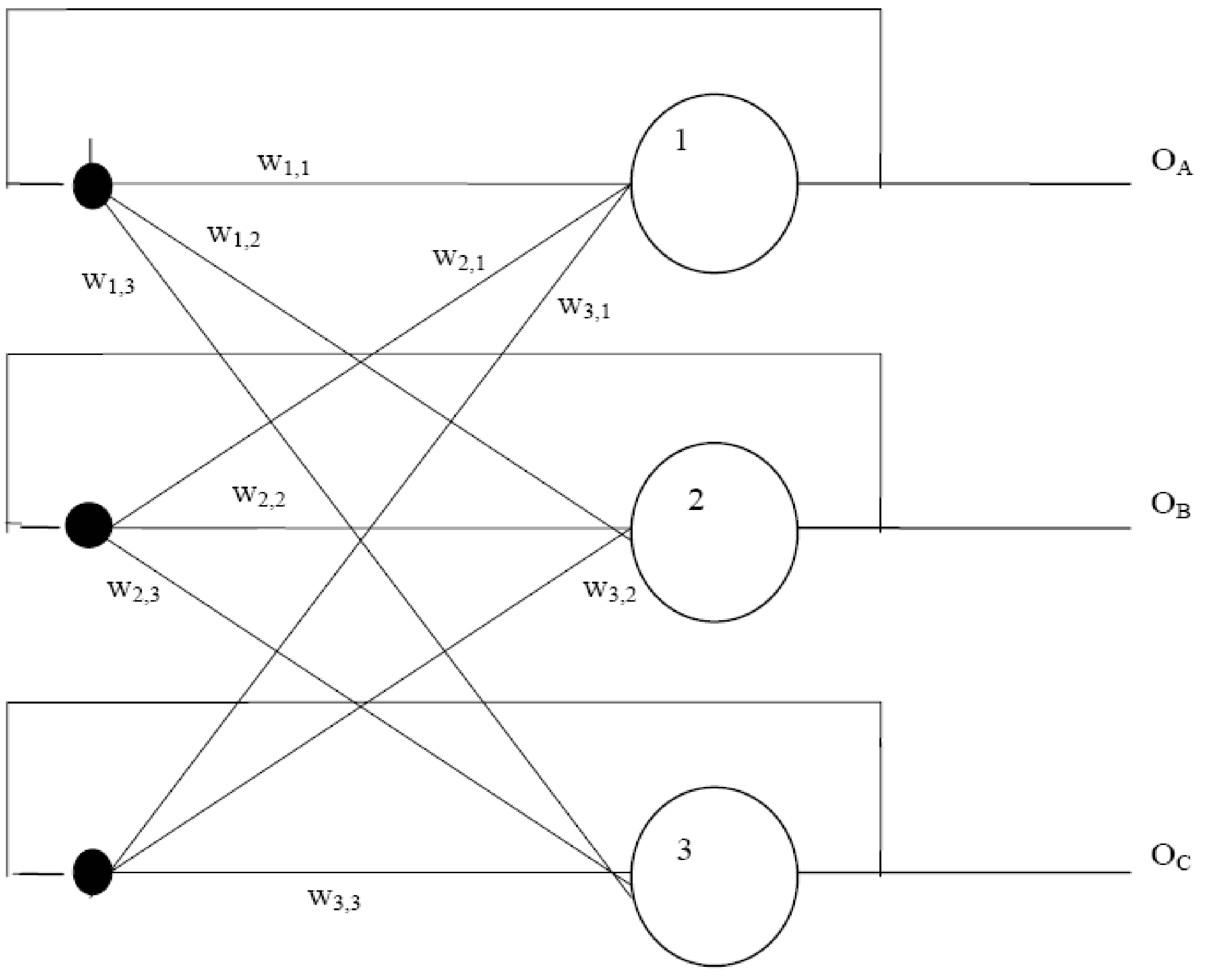



We multiply the pixel in each pattern corresponding to the index of the weight, so for $W_{1,2}$ we multiply the value of pixel 1 and pixel 2 together in each of the patterns we wish to train. We then add up the result (which in this case is -2).

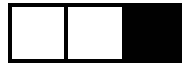
Weights which have equal indexes (like $W_{2,2}$) we make zero.

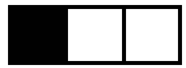
# ➢ Example(2) for Storage Phase

A three neuron network trained with three patterns.



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

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

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

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

$w_{1,1} = 0$

$w_{1,2} = O_{A(1)} \times O_{B(1)} + O_{A(2)} \times O_{B(2)} + O_{A(3)} \times O_{B(3)} = (-1) \times (-1) + 1 \times (-1) + (-1) \times 1 = -1$

$w_{1,3} = O_{A(1)} \times O_{C(1)} + O_{A(2)} \times O_{C(2)} + O_{A(3)} \times O_{C(3)} = (-1) \times 1 + 1 \times (-1) + (-1) \times 1 = -3$

$w_{2,2} = 0$

$w_{2,1} = O_{B(1)} \times O_{A(1)} + O_{B(2)} \times O_{A(2)} + O_{B(3)} \times O_{A(3)} = (-1) \times (-1) + (-1) \times 1 + 1 \times (-1) = -1$

$w_{2,3} = O_{B(1)} \times O_{C(1)} + O_{B(2)} \times O_{C(2)} + O_{B(3)} \times O_{C(3)} = (-1) \times 1 + (-1) \times (-1) + 1 \times 1 = 1$

$w_{3,3} = 0$

$w_{3,1} = O_{C(1)} \times O_{A(1)} + O_{C(2)} \times O_{A(2)} + O_{C(3)} \times O_{A(3)} = 1 \times (-1) + (-1) \times 1 + 1 \times (-1) = -3$

$w_{3,2} = O_{C(1)} \times O_{B(1)} + O_{C(2)} \times O_{B(2)} + O_{C(3)} \times O_{B(3)} = 1 \times (-1) + (-1) \times (-1) + 1 \times 1 = 1$

## ➢ Example for Recalling Phase

- For patterns [1 -1 1] and [-1 1 -1] the Memory Matrix is obtained as :

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

✓ If the key vector applied to the network is [-1 -1 1], [1 1 1] or [1 -1 -1] then the resulting ouput is the fundamental memory is [1 -1 1]. Each of these values of the key vector represents a single error, compared to the stored pattern.

✓ If the key vector applied to the network is [1 1 -1], [-1 -1 -1] or [-1 1 1] then the resulting ouput is the fundamental memory is [-1 1 -1]. Each of these values of the key vector represents a single error, compared to the stored pattern.

- The real reason Hopfield's work is important is that it shows that adding feedback connections to a network makes it more general (it can store and recall patterns as well as just recognise them).
- The network can also produce a wide range of behaviours not seen in simple feedforward types – including oscillation and even chaos.
- The method of training given above, however, can be shown to always produce a stable network - one which won't oscillate (the network is always stable providing that $W_{n,m} = W_{m,n}$ and $W_{n,n} = 0$).