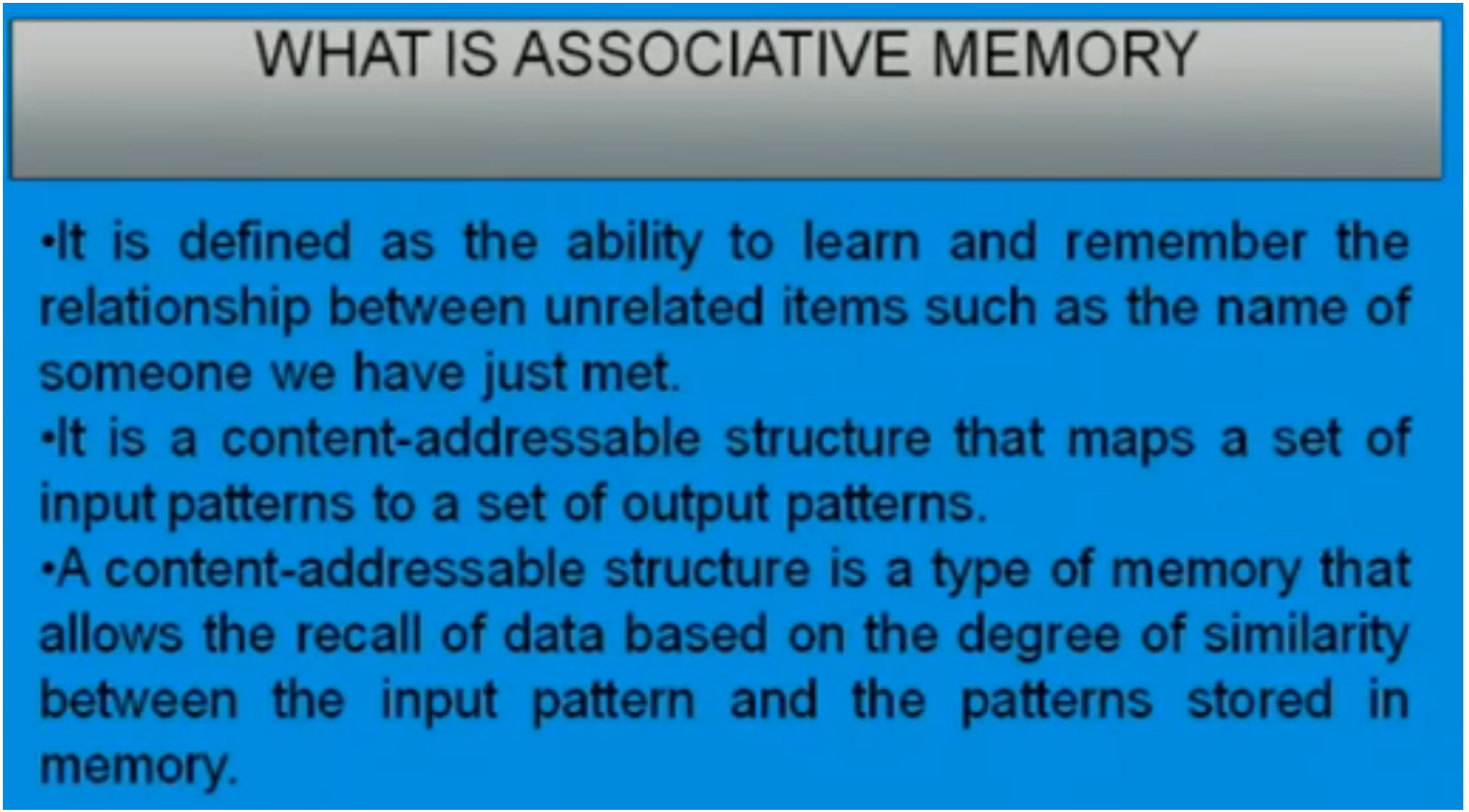
What is Unsupervised Learning?

- Most simply, it can be thought of as learning to recognise and recall things
  - Recognition – “I’ve seen that before”
  - Recall – “I’ve seen that before and I can recall more about it from memory”.
- There is no feedback or reward like there is with reinforcement learning
- There is no given answer like there is in supervised learning

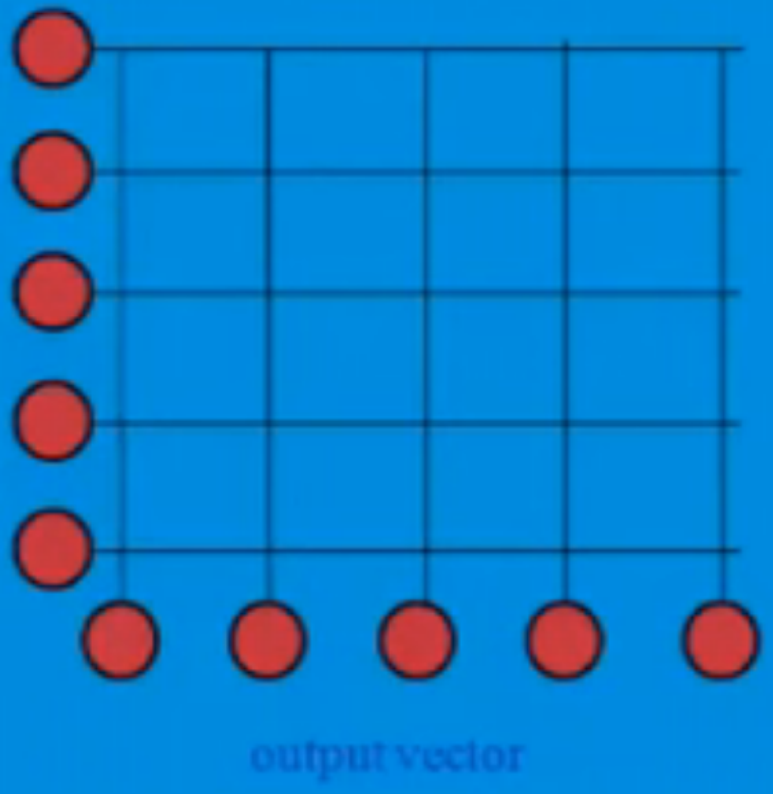


- **Auto Associative vs Hetero Associative Memory**

Source: https://www.youtube.com/watch?v=a7FFd01jz6w&t=2788s

ASSOCIATIVE MEMORY

- It is trained with input and output vectors.
  - It modifies the weights each time it is shown a pair.
- After one sweep through the training set it must "retrieve" the correct output vector for a given input vector



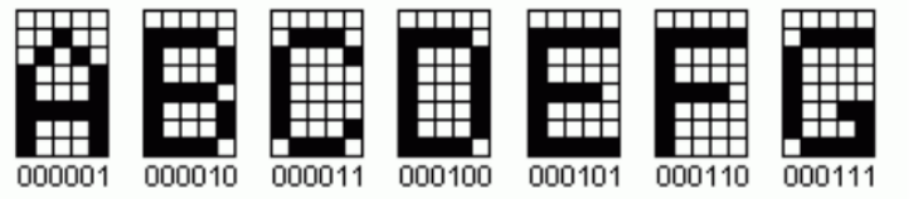
- The network is firstly trained to store a set of patterns in the form *s* : *t*.
- *s* represents the input vector and *t* the corresponding output vector.
- The network is then tested on a set of data to test its "memory" by using it to identify patterns containing incorrect or missing information.

A Simple Associative Memory
The Hopfield Network

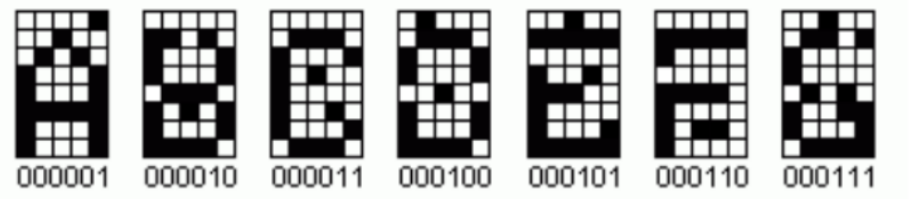
- Stores patterns in an associative memory
- Can recall a complete pattern when given only a part of that pattern as input
- Robust under noise – will recall the nearest pattern it has to the input stimulus
- Robust under damage – remove a few of the connections and it still works

Example – Learning Images

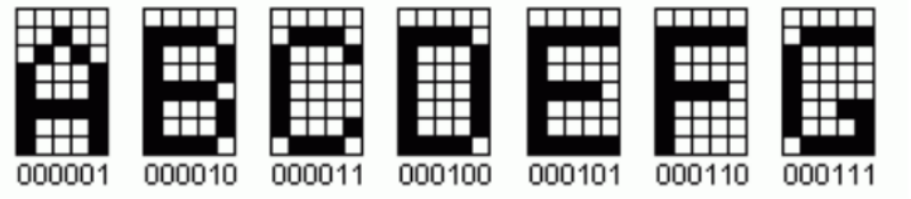
- Train a network with clean images:



- Present corrupted text as inputs:



- Network produces clean characters as outputs:



Characteristics of a Hopfield Network

- A collection of nodes, which we will call neurons (though they are really just simple mathematical functions)
- Each neuron is connected to every other neuron in the network (but not itself) – we call the connections synapses
- Synapses have a weight that is either excitatory (+ve) or inhibitory (-ve)
- Weights are symmetrical: $W_{ij} = W_{ji}$
- Neurons can be either on or off – represented as an output value of +1 or -1
- The neurons are of the McCulloch and Pitts type that we have already seen.

Hopfield Network Algorithm

1. **Assign connection weights**

$$w_{ij} = \begin{cases} \frac{1}{N}\sum_{s=0}^{M-1} x_i^s x_j^s & i \neq j \\ 0 & i = j, \end{cases}$$

where w_{ij} is the connection weight between node i and node j, and x_i^s is element i of the exemplar pattern s, and is either $+1$ or -1.
There are M patterns, from 0 to $M-1$, in total.

2. **Initialise with unknown pattern**

$$\mu_i(0) = x_i \qquad 0 \leq i \leq N-1$$

where $\mu_i(t)$ is the output of node i at time t.

3. **Iterate until convergence**

$$\mu_i(t+1) = f_h\left[\sum_{i=0}^{N-1} w_{ij}\mu_j(t)\right] \qquad 0 \leq j \leq N-1$$

The function f_h is the hard-limiting non-linearity, the step function,
Repeat the iteration until the outputs from the node remain unchanged.

Example in vector form

Hopfield Network

- Weight Matrix: $W = \begin{bmatrix} 0 & w_{12} & w_{13} & \dots & w_{1n} \\ w_{12} & 0 & w_{23} & \dots & w_{2n} \\ w_{13} & w_{23} & 0 & \dots & w_{3n} \\ \vdots & \vdots & & \ddots & \vdots \\ w_{1n} & w_{2n} & w_{3n} & \dots & 0 \end{bmatrix}$
- Inputs: $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ where x_j is either 1 or –1

Learning used in Hopfield net

Hebbian Learning

- In all ANN's, "learning" corresponds to modification of the synaptic weights
  - Many ANN's learn via a *Hebbian Learning rule*
  - From Cognitive Psychologist Donald Hebb, who surmised that learning takes place by "reinforcing connections" among learned states
- To "learn" a pattern $\mathbf{p} = [p_1, \ldots, p_n]^T$ where $p_{ij} = \pm 1$
  - Fix a "learning rate" $\varepsilon > 0$
  - Update the synaptic weights using

$$w_{ij}^{new} = w_{ij}^{old} + \varepsilon p_i p_j$$

Learning used in Hopfield net

Hebbian Learning

- In matrix form, notice that

$$\mathbf{p}\cdot\mathbf{p}^T = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{bmatrix} \begin{bmatrix} p_1, p_2, \ldots, p_n \end{bmatrix} = \begin{bmatrix} p_1p_1 & p_1p_2 & \cdots & p_1p_n \\ p_2p_1 & p_2p_2 & \cdots & p_2p_n \\ \vdots & \vdots & \ddots & \vdots \\ p_np_1 & p_np_2 & \cdots & p_np_n \end{bmatrix}$$

- Since $p_jp_j = 1$, Hebbian learning in matrix form is

$$W^{new} = W^{old} + \varepsilon\left(\mathbf{p}\cdot\mathbf{p}^T - I\right)$$

Example in vector form

Hopfield Network

- Simulation begins with $\mathbf{x}^{\text{initial}} = \left[x_1^{init}, \ldots, x_n^{init} \right]^T$

- Select neuron *i* at random and update via

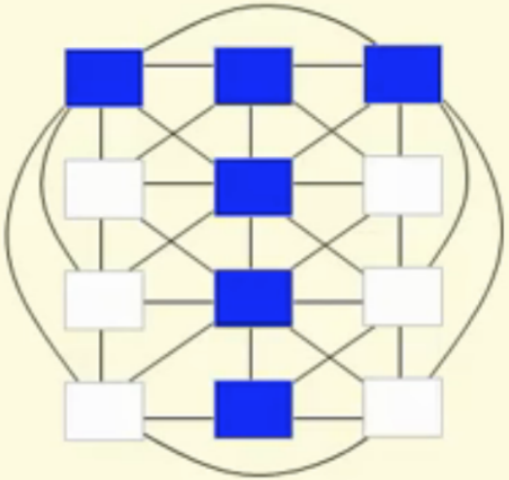
$$x_i^{new} = \sigma \left(\sum_{i \neq j} w_{ij} x_j^{old} - \theta_j \right)$$

- Repeat until (hopefully) a learned pattern is recovered

Example in vector form

Hopfield Network Example

Blue = 1
White = 0



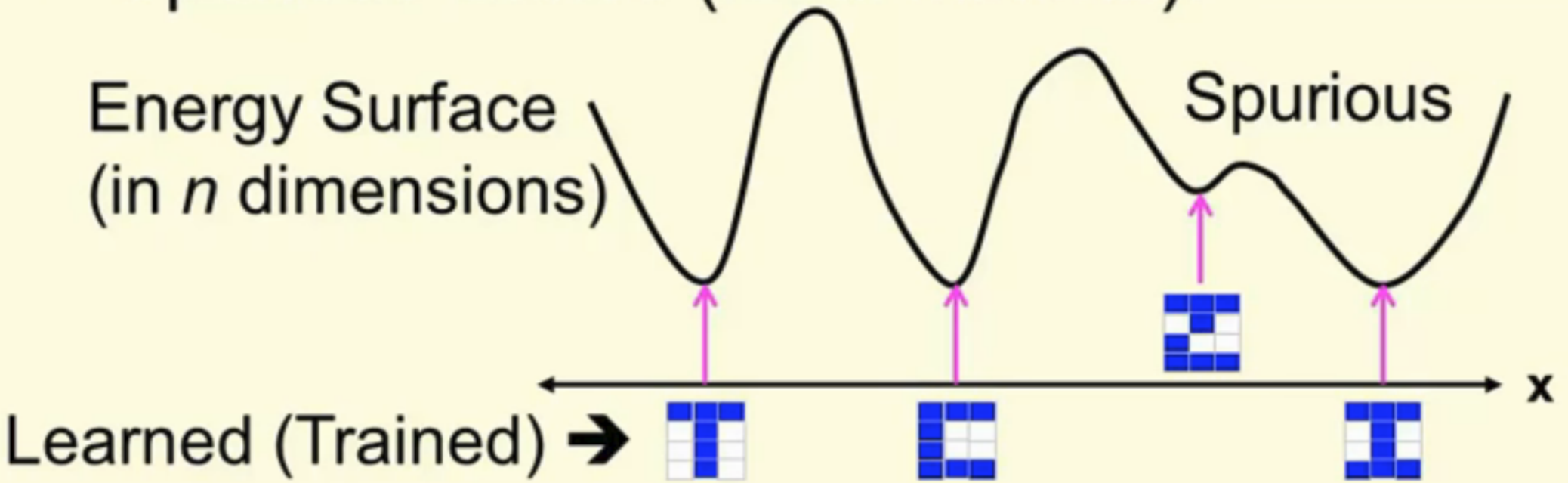
Imagine Complete Connectivity with weights $w_{ij} = w_{ji}$ between i^{th} and j^{th} neurons

Choose i^{th} neuron at random and calculate its new state

$$x_i^{new} = \sigma\left(\sum_{i \neq j} w_{ij} x_j^{old} - \theta_j\right)$$

Problems with Hopfield

- Spurious States (local minima):



- Can it correctly predict the class of any trained pattern – i.e., *f*(pattern) = class?

Another Example (from youtube tutorial)

Steps to Determine Weights

- Let us find the contribution of the pattern A = (1, 0, 1, 0):
- The binary to bipolar mapping of A = (1, 0, 1, 0) gives the vector (1, –1, 1, –1).
- Then take the transpose, and multiply:

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

Source: https://www.youtube.com/watch?v=a7FFd01jz6w&t=2788s

•Now subtract 1 from each element in the main diagonal

$$\begin{bmatrix} 0 & -1 & 1 & -1 \\ -1 & 0 & -1 & 1 \\ 1 & -1 & 0 & -1 \\ -1 & 1 & -1 & 0 \end{bmatrix}$$

•Calculate the contribution from the pattern B = (0, 1, 0, 1)
and add with W results:

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

W′ = Wx(3/2) =

0	-3	3	-3
-3	0	-3	3
3	-3	0	-3
-3	3	-3	0

•We can now optionally apply an arbitrary scalar multiplier to all the entries of the matrix

Source: https://www.youtube.com/watch?v=a7FFd01jz6w&t=2788s

Threshold Function

A = (1, 0, 1, 0)
B = (0, 1, 0, 1)

- A threshold function is defined as follows. The threshold value θ is 0.

$$f(t) = \begin{cases} 1 & \text{if } t >= \theta \\ 0 & \text{if } t < \theta \end{cases}$$

0	-3	3	-3
-3	0	-3	3
3	-3	0	-3
-3	3	-3	0

- The activation at the first node is the dot product of the input vector and the first column of the weight matrix (0 −3 3 −3)
- The dot product works out to 3 and f(3) = 1. Similarly, the dot products of the second, third, and fourth nodes are –6, 3, and –6, respectively .
- The output of the network is the vector (1, 0, 1, 0)

Source: https://www.youtube.com/watch?v=a7FFdo1jz6w&t=2788s

0	-3	3	-3
-3	0	-3	3
3	-3	0	-3
-3	3	-3	0

A = (1, 0, 1, 0)
B = (0, 1, 0, 1)

- When B is presented, the dot product obtained at the first node is –6 and the output is 0.

- The outputs for the rest of the nodes taken together with the output of the first node gives (0, 1, 0, 1), which means that the network has stable recall for B also.

- Let C = (0, 1, 0, 0) be presented to the network. The activations would be –3, 0, –3, 3, making the outputs 0, 1, 0, 1, which means that B achieves stable recall.

Source: https://www.youtube.com/watch?v=a7FFdo1jz6w&t=2788s

Asynchronous Update

- The Hopfield network is a recurrent network. This means that outputs from the network are fed back as inputs.

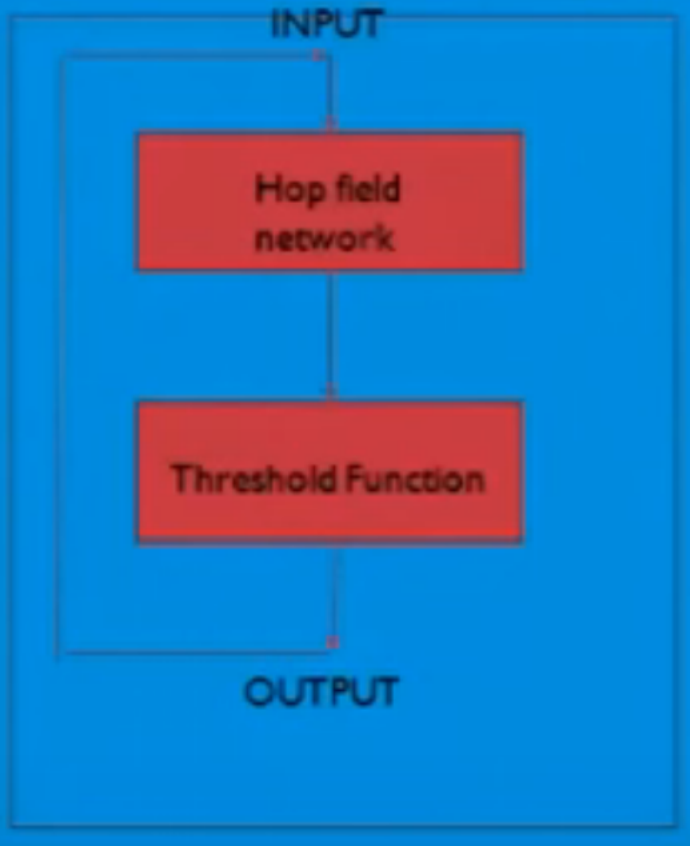


Figure 2. Feedback in the Hopfield Network

Source: https://www.youtube.com/watch?v=a7FFd01jz6w&t=2788s

CONTD.

The true operation of the Hopfield network follows the procedure below for input vector Invec and output vector Outvec:

- Apply an input, Invec, to the network, and initialize Outvec = Invec
- Start with i = 1
- Calculate Value i = DotProduct (Inveci, Columni of Weight matrix)
- Calculate Outvec i = f(Valuei) where f is the threshold function discussed previously
- Update the input to the network with component Outvec i

Source: https://www.youtube.com/watch?v=a7FFd01jz6w&t=2788s

CONTD.

Now let's apply asynchronous update for input E, (1,0,0,1)

Example of Asynchronous Update for the Hopfield Network

Step i	Invec	Column of Weight vector	Value	Outvec	Notes
0	1001			1001	initialization : set Outvec = Invec = Input pattern
1 1	1001	0 −3 3 −3	−3	0001	column 1 of Outvec changed to 0
2 2	0001	−3 0 −3 3	3	0101	column 2 of Outvec changed to 1
3 3	0101	3 −3 0 −3	−6	0101	column 3 of Outvec stays as 0
4 4	0101	−3 3 −3 0	3	0101	column 4 of Outvec stays as 1
5 1	0101	0 −3 3 −3	−6	0101	column 1 stable as 0
6 2	0101	−3 0 −3 3	3	0101	column 2 stable as 1
7 3	0101	3 −3 0 −3	−6	0101	column 3 stable as 0
8 4	0101	−3 3 −3 0	3	0101	column 4 stable as 1; stable recalled

Source: https://www.youtube.com/watch?v=a7FFd01jz6w&t=2788s

Wrong Way: Synchronous Update

- Increment i, and repeat steps 3, 4, 5, and 6 until Invec = Outvec (note that when i reaches its maximum value, it is then next reset to 1 for the cycle to continue)

- Say input E = (1, 0, 0, 1), which is at an equal distance from A and B.

- Output is F=(0,1,1,0)

Source: https://www.youtube.com/watch?v=a7FFdo1jz6w&t=2788s

Example of an Hopfield Network

- Consider the pattern on the left as a representation of the character "plus", +, and the one on the right that of "minus", − .

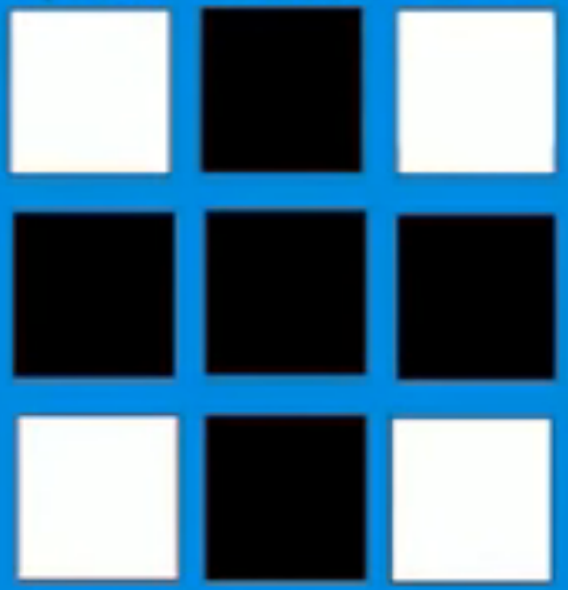


Figure 3. The "PLUS" pattern

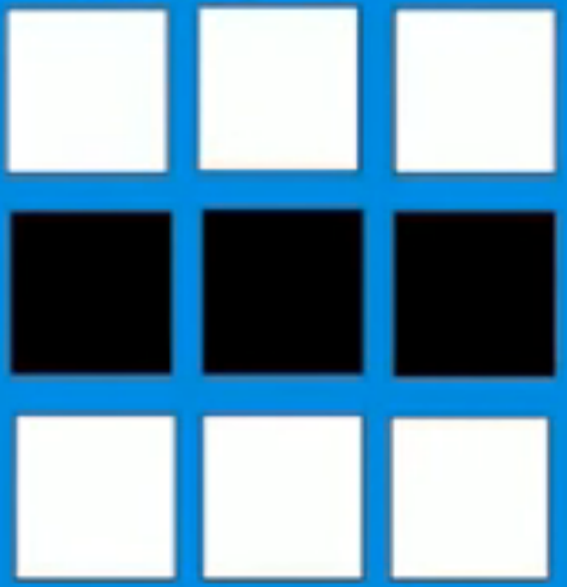


Figure 4. The "MINUS" pattern

C+ = (−1, 1, −1, 1, 1, 1, −1, 1, −1), and C− = (−1, −1, −1, 1, 1, 1, −1, −1, −1).

Source: https://www.youtube.com/watch?v=a7FFd01jz6w&t=2788s

- The weight matrix W is:

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

- The activations with input C+ are given by the vector (−12, 2, −12, 12, 12, 12, −12, 2, −12).
- With input C−, the activations vector is (−12, −2, −12, 12, 12, 12, −12, −2, −12).
- This Hopfield network uses the threshold function

$$f(x) = \begin{cases} 1 & \text{if } x > 0 \\ -1 & \text{if } x <= 0 \end{cases}$$

- Let us now input the character pattern in Figure 5 .

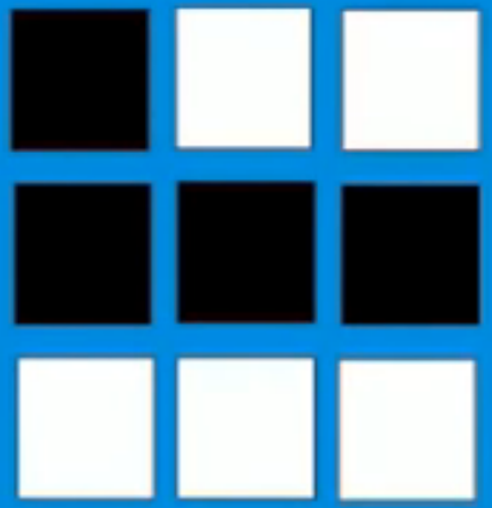


Figure 5 Corrupted "MINUS" pattern

- The corresponding bipolar vector A = (1, −1, −1, 1, 1, 1, −1, −1, −1).

- The activation vector (−12, −2, −8, 4, 4, 4, −8, −2, −8) giving the output vector, C− = (−1, −1, −1, 1, 1, 1, −1, −1, −1).
- In other words, the character −, corrupted slightly, is recalled as the character − by the Hopfield network.
- The intended pattern is recognized.

It is even able to recall some patterns (like the examples below which are neither '+' or '-')

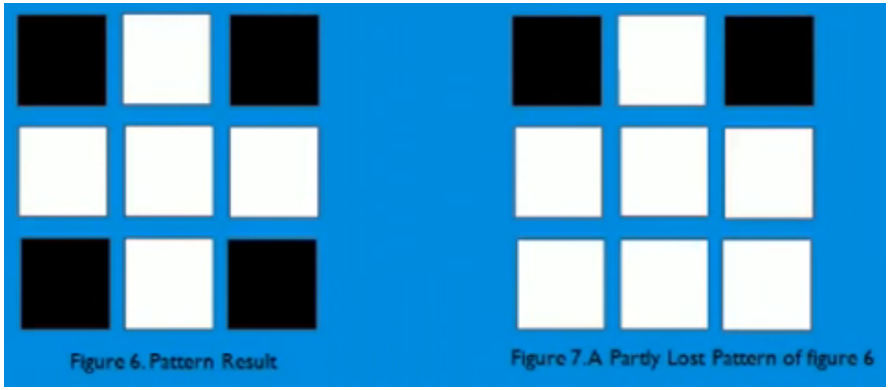


Figure 6. Pattern Result

Figure 7.A Partly Lost Pattern of figure 6

- B = {1,-1,1,-1,-1,-1,1,-1,1}.Corresponding Neuron Activations are {12,-2,12,-4,-4,-4,12,-2,12} and the output vector is {1,-1,1,-1,-1,-1,1,-1,1}
- Recalled pattern is correct
- If we omit part of the pattern in Figure 6 leaving only the top corners black, as in Figure 7, we get the bipolar vector D = (1, −1, 1, −1, −1, −1, −1, −1, −1). You can consider this also as an incomplete or corrupted version of the pattern in Figure 5. The network activations turn out to be (4, −2, 4, −4, −4, −4, 8, −2, 8) and give the output (1, −1, 1, −1, −1, −1, 1, −1, 1), which is B.

Bidirectional associative memory (BAM)
A type of Hetero Associative Memory

Learning [edit]

Imagine we wish to store two associations, A1:B1 and A2:B2.

- A1 = (1, 0, 1, 0, 1, 0), B1 = (1, 1, 0, 0)
- A2 = (1, 1, 1, 0, 0, 0), B2 = (1, 0, 1, 0)

These are then transformed into the bipolar forms:

- X1 = (1, -1, 1, -1, 1, -1), Y1 = (1, 1, -1, -1)
- X2 = (1, 1, 1, -1, -1, -1), Y2 = (1, -1, 1, -1)

From there, we calculate $M = \sum X_i^T Y_i$ where X_i^T denotes the transpose. So,

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

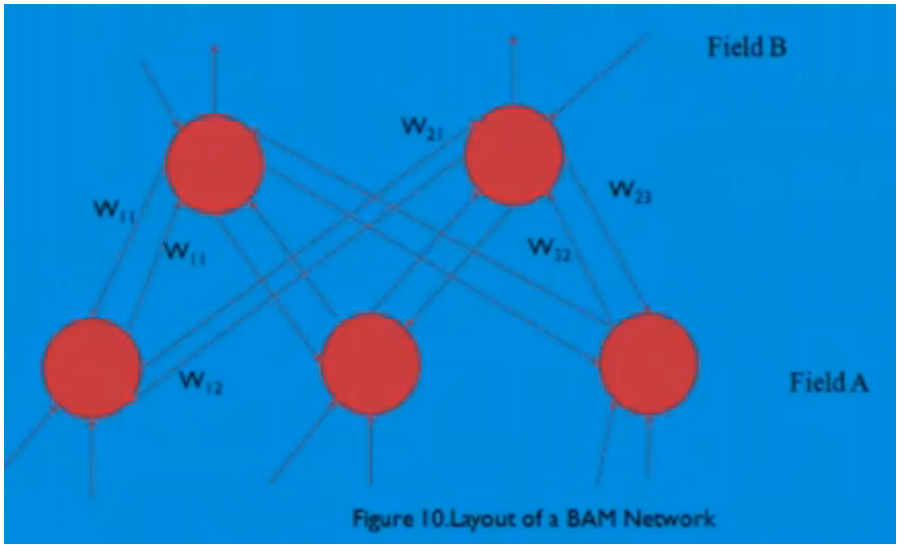


Figure 10.Layout of a BAM Network

Recall [edit]

To retrieve the association A1, we multiply it by M to get (4, 2, -2, -4), which, when run through a threshold, yields (1, 1, 0, 0), which is B1. To find the reverse association, multiply this by the transpose of M.

Bidirectional associative memory (BAM)
A type of Hetero Associative Memory

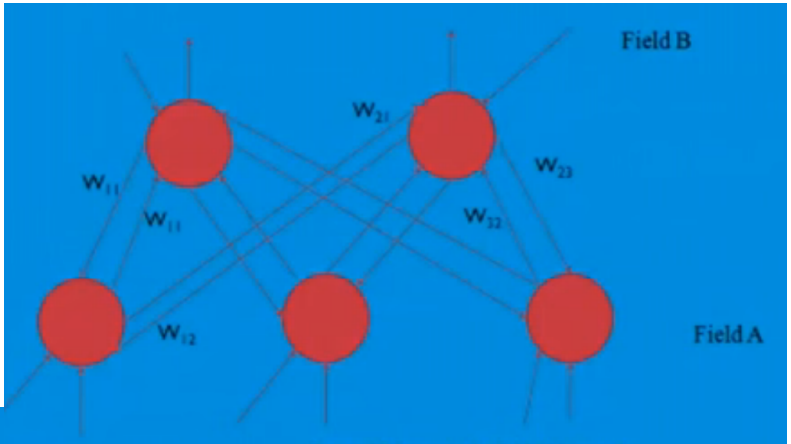


Figure 10.Layout of a BAM Network

EXAMPLE

- Suppose you choose two pairs of vectors as possible exemplars. Let them be:

  X1 = (1, 0, 0, 1), Y1= (0, 1, 1) and X2 = (0, 1, 1, 0), Y2 = (1, 0, 1)

These you change into bipolar components and get, respectively, (1, –1, –1, 1), (–1, 1, 1), (–1, 1, 1, –1), and (1, –1, 1).

$$W = \begin{matrix}1\\-1\\-1\\1\end{matrix}\begin{bmatrix}-1 & 1 & 1\end{bmatrix} + \begin{matrix}-1\\1\\1\\-1\end{matrix}\begin{bmatrix}1 & -1 & 1\end{bmatrix} = \begin{matrix}-1 & 1 & 1\\1 & -1 & -1\\1 & -1 & -1\\-1 & 1 & 1\end{matrix} + \begin{matrix}-1 & 1 & -1\\1 & -1 & 1\\1 & -1 & 1\\-1 & 1 & -1\end{matrix}$$

$$= \begin{matrix}-2 & 2 & 0\\2 & -2 & 0\\2 & -2 & 0\\-2 & 2 & 0\end{matrix}$$

And

$$W_T = \begin{matrix}-2 & 2 & 2 & -2\\2 & -2 & -2 & 2\\0 & 0 & 0 & 0\end{matrix}$$

Bidirectional associative memory (BAM)
A type of Hetero Associative Memory

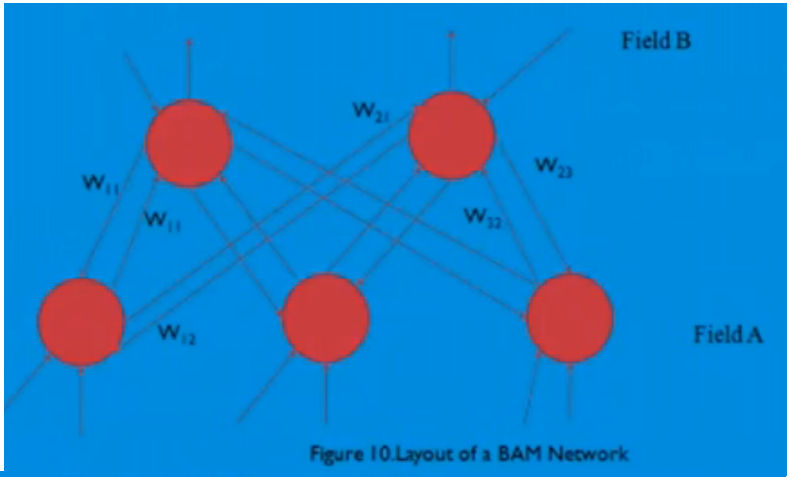


Figure 10.Layout of a BAM Network

- We now present the **thresholding** function for BAM outputs:

$$b_{i|j+1} = \begin{cases} 1 & \text{if } y_j > 0 \\ b_{i|j} & \text{if } y_j = 0 \\ 0 & \text{if } y_j < 0 \end{cases} \quad \text{and} \quad a_{i|j+1} = \begin{cases} 1 & \text{if } x_i > 0 \\ a_{i|j} & \text{if } x_i = 0 \\ 0 & \text{if } x_i < 0 \end{cases}$$

- where *xi* and *yj* are the activations of neurons *i* and *j* in the input layer and output layer, respectively, and *bj*|t refers to the output of the *j*th neuron in the output layer in the cycle *t*, while *ai*|t refers to the output of the *i*th neuron in the input layer in the cycle *t*. Note that at the start, the *ai* and *bj* values are the same as the corresponding components in the exemplar pair being used.

Bidirectional associative memory (BAM)
A type of Hetero Associative Memory

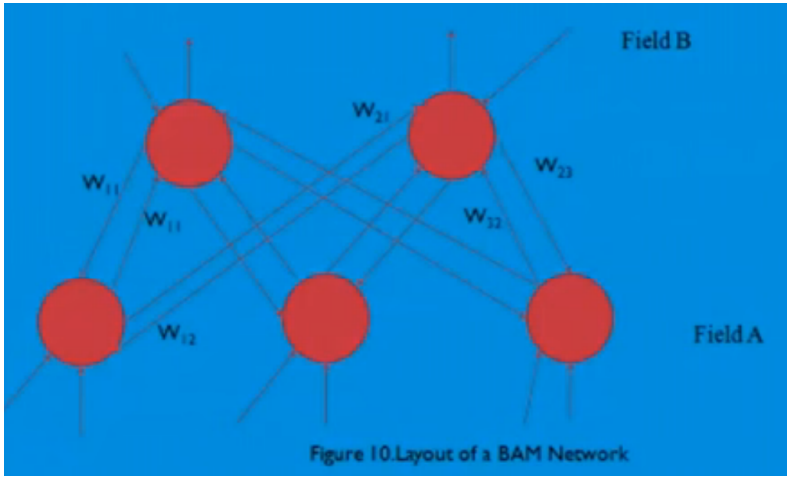


Figure 10.Layout of a BAM Network

- If **X1** = (1, 0, 0, 1) is presented to the input neurons, their activations are given by the vector (–4, 4, 0).
- The output vector, after using the **threshold** function just described is (0, 1, 1). The last component here is supposed to be the same as the output in the previous cycle, since the corresponding activation value is 0.
- If we feed **Y1** at the other end (B field) , the activations in the A field will be (2, –2, –2, 2), and the output vector will be (1, 0, 0, 1), which is **X1**.

Bidirectional associative memory (BAM)
A type of Hetero Associative Memory

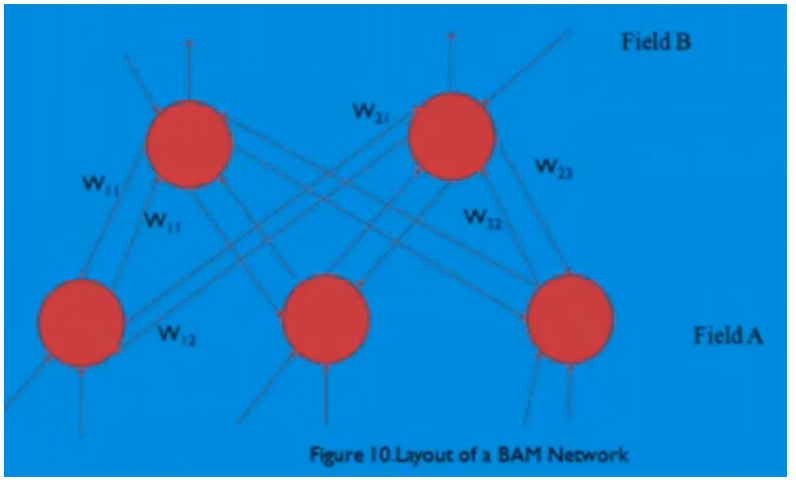


Figure 10.Layout of a BAM Network

Input vectoractivationoutput vector

$X_1 = (1, 0, 1, 1)(-4, 4, 2)(0, 1, 1) = Y_1$

$X_2 = (0, 1, 1, 0)(2, -2, 2)(1, 0, 1) = Y_2$

$Y_1 = (0, 1, 1)(2, -2, 2, 2)(1, 0, 1, 1) = X_1$

$Y_2 = (1, 0, 1)(-2, 2, 2, -2)(0, 1, 1,0) = X_2$