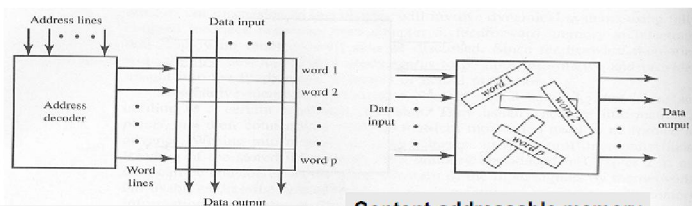
HOPFIELD NETWORK

INTRODUCTION ASSOCIATIVE MEMORY

An associative memory (AM) net may serve as a highly simplified model of human memory.

AM provide an approach of storing and retrieving data based on content rather than storage address (info. storage in a NN is distributed throughout the system in the net's weights, hence a pattern does not have a storage



Address-addressable memory

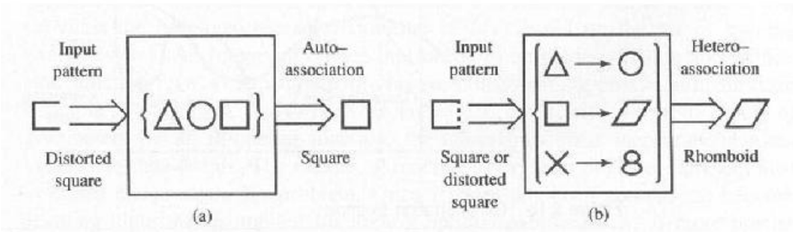
Content-addressable memory

4

Introduction

Each association is an I/P O/P vector pair , **s**:**f**.
Two types of associations. For two patterns ***s*** and ***f***

- If ***s*** = ***f*** the net is called auto-associative memory.
- If ***s*** != ***f*** the net is called hetero-associative memory



Association response: (a) autoassociation and (b) heteroassociation.

Example

HOP

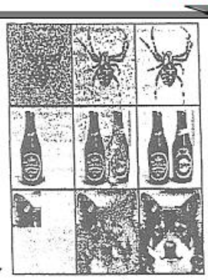


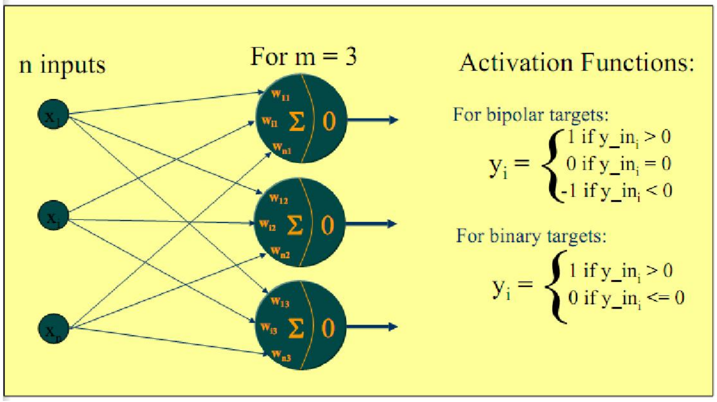
FIGURE 2.1 Example of how an associative memory can reconstruct images. These are binary images with 130 × 180 pixels. The images on the right were recalled by the memory after presentation of the corrupted images shown on the left. The middle column shows some intermediate states. A sparsely connected Hopfield network with seven stored images was used.

Corrupted image   Intermediate state of Hopfield net   Output

- States $\simeq$ Bit maps.
- Attractors $\simeq$ Prototype patterns.
- Input $\simeq$ An arbitrary pattern (e.g. picture with noise).
- Output $\simeq$ The best prototype for that pattern.

Hetero-associative

There are n input units and m output units with each input connected to each output unit.



Quantifying Hebb's Rule

Compare two nodes to calc a weight change that reflects the state correlation:

Auto-Association: $\Delta w_{jk} \propto i_{pk} i_{pj}$

* When the two components are the same (different), increase (decrease) the weight

Hetero-Association: $\Delta w_{jk} \propto i_{pk} o_{pj}$

i = input component
o = output component

Ideally, the weights will record the average correlations across all patterns:

Auto: $w_{jk} \propto \sum_{p=1}^{P} i_{pk} i_{pj}$ Hetero: $w_{jk} \propto \sum_{p=1}^{P} i_{pk} o_{pj}$

Hebbian Principle: If all the input patterns are known prior to retrieval time, then init weights as:

Auto: $w_{jk} \equiv \frac{1}{P} \sum_{p=1}^{P} i_{pk} i_{pj}$ Hetero: $w_{jk} \equiv \frac{1}{P} \sum_{p=1}^{P} i_{pk} o_{pj}$

Weights = Average Correlations

OUTER PRODUCT FOR PATTERN ASSOCIATION

Let ***s*** and ***t*** be **row** vectors.

Then for a particular training pair ***s:t***

$$\Delta W(p) = s^T(p)\cdot t(p) = \begin{bmatrix} s_1 \\ \\ \\ s_n \end{bmatrix} \begin{bmatrix} t_1,\ldots,t_m \end{bmatrix} = \begin{bmatrix} s_1t_1\ldots s_1t_m \\ s_2t_1\ldots s_2t_m \\ \\ s_nt_1\ldots s_nt_m \end{bmatrix} = \begin{bmatrix} \Delta w_{11}\ldots\Delta w_{1m} \\ \\ \\ \Delta w_{n1}\ldots\Delta w_{nm} \end{bmatrix}$$

and

$$W(p) = \sum_{p=1}^{P} s^T(p)\cdot t(p)$$

TESTING ALGORITHM(HETEROASSOCIATIVE)

Step 0: Initialize the weights from the training algorithm.
Step 1: Perform Steps 2–4 for each input vector presented.
Step 2: Set the activation for input layer units equal to that of the current input vector given, x_i.
Step 3: Calculate the net input to the output units:

$$y_{inj} = \sum_{i=1}^{n} x_i w_{ij} \quad (j = 1 \text{ to } m)$$

Step 4: Determine the activations of the output units over the calculated net input:

$$y_j = \begin{cases} 1 & \text{if} \quad y_{inj} > 0 \\ 0 & \text{if} \quad y_{inj} = 0 \\ -1 & \text{if} \quad y_{inj} < 0 \end{cases}$$

HETERO-ASSOCIATIVE MEMORY NETWORK

- Binary pattern pairs ***s:t*** with |***s***| = 4 and |***t***| = 2.
- Total weighted input to output units: $$y\_in_j = \sum_i x_i w_{ij}$$
- Activation function: threshold $$y_j = \begin{cases} 1 & if \quad y\_in_j > 0 \\ 0 & if \quad y\_in_j \le 0 \end{cases}$$
- Weights are computed by Hebbian rule (sum of outer products of all training pairs) $$W = \sum_{p=1}^{P} s_i^T(p) t_j(p)$$
- Training samples:

	s(p)	t(p)
p=1	(1 0 0 0)	(1, 0)
p=2	(1 1 0 0)	(1, 0)
p=3	(0 0 0 1)	(0, 1)
p=4	(0 0 1 1)	(0, 1)

9

COMPUTING THE WEIGHTS

$$s^T(1)\cdot t(1) = \begin{pmatrix}1\\0\\0\\0\end{pmatrix}\begin{pmatrix}1 & 0\end{pmatrix} = \begin{pmatrix}1&0\\0&0\\0&0\\0&0\end{pmatrix} \qquad s^T(2)\cdot t(2) = \begin{pmatrix}1\\1\\0\\0\end{pmatrix}\begin{pmatrix}1 & 0\end{pmatrix} = \begin{pmatrix}1&0\\1&0\\0&0\\0&0\end{pmatrix}$$

$$s^T(3)\cdot t(3) = \begin{pmatrix}0\\0\\0\\1\end{pmatrix}\begin{pmatrix}0 & 1\end{pmatrix} = \begin{pmatrix}0&0\\0&0\\0&0\\0&1\end{pmatrix} \qquad s^T(4)\cdot t(4) = \begin{pmatrix}0\\0\\1\\1\end{pmatrix}\begin{pmatrix}0 & 1\end{pmatrix} = \begin{pmatrix}0&0\\0&0\\0&1\\0&1\end{pmatrix}$$

$$W = \begin{pmatrix}2&0\\1&0\\0&1\\0&2\end{pmatrix}$$

10

TEST/ RECALL THE NETWORK

$x = [1\,0\,0\,0]$

$$\begin{pmatrix}1 & 0 & 0 & 0\end{pmatrix}\begin{pmatrix}2 & 0\\1 & 0\\0 & 1\\0 & 2\end{pmatrix} = \begin{pmatrix}2 & 0\end{pmatrix}$$

$y_1 = 1, \quad y_2 = 0$

$x = [0\,1\,1\,0]$

$$\begin{pmatrix}0 & 1 & 1 & 0\end{pmatrix}\begin{pmatrix}2 & 0\\1 & 0\\0 & 1\\0 & 2\end{pmatrix} = \begin{pmatrix}1 & 1\end{pmatrix}$$

$y_1 = 1, \quad y_2 = 1$

$x = [0\,1\,0\,0]$ similar to s(1) and s(2)

$$\begin{pmatrix}0 & 1 & 0 & 0\end{pmatrix}\begin{pmatrix}2 & 0\\1 & 0\\0 & 1\\0 & 2\end{pmatrix} = \begin{pmatrix}1 & 0\end{pmatrix}$$

$y_1 = 1, \quad y_2 = 0$

(1 0 0 0), (1 1 0 0) class (1, 0)
(0 0 0 1), (0 0 1 1) class (0, 1)
(0 1 1 0) is not sufficiently similar to any class

11

HETERO-ASSOCIATIVE

GOAL: build a neural network which will associate the following two sets of patterns using Hebb's Rule:

$s_1 = (\ 1\ \ -1\ \ -1\ \ -1)$ $\quad f_1 = (\ 1\ \ -1\ \ -1)$
$s_2 = (-1\ \ \ 1\ \ -1\ \ -1)$ $\quad f_2 = (\ 1\ \ -1\ \ \ 1)$
$s_3 = (-1\ \ -1\ \ \ 1\ \ -1)$ $\quad f_3 = (-1\ \ \ 1\ \ -1)$
$s_4 = (-1\ \ -1\ \ -1\ \ \ 1)$ $\quad f_4 = (-1\ \ \ 1\ \ \ 1)$

The process will involve 4 input neurons and 3 output neurons

The algorithm involves finding the four outer products and adding them

HETERO-ASSOCIATIVE

Pattern pair 1:

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

Pattern pair 2:

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

Pattern pair 3:

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

Pattern pair 4:

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

HETERO-ASSOCIATIVE

Add all four individual weight matrices to produce the final weight matrix:

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

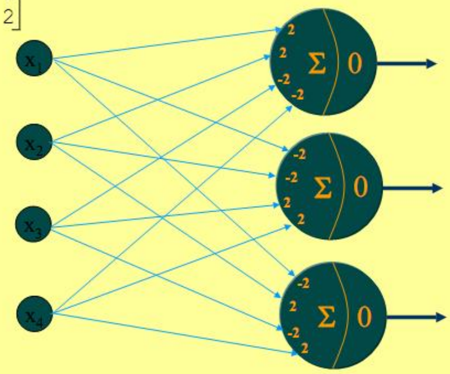
$$= \begin{bmatrix}2 & -2 & -2\\2 & -2 & 2\\-2 & 2 & -2\\-2 & 2 & 2\end{bmatrix}$$

Each column defines the weights for an output neuron

Hetero-associative

$$\begin{bmatrix} 2 & -2 & -2 \\ 2 & -2 & 2 \\ -2 & 2 & -2 \\ -2 & 2 & 2 \end{bmatrix}$$

$$y_i = \begin{cases} 1 \text{ if } y\_in_i > 0 \\ 0 \text{ if } y\_in_i = 0 \\ -1 \text{ if } y\_in_i < 0 \end{cases}$$

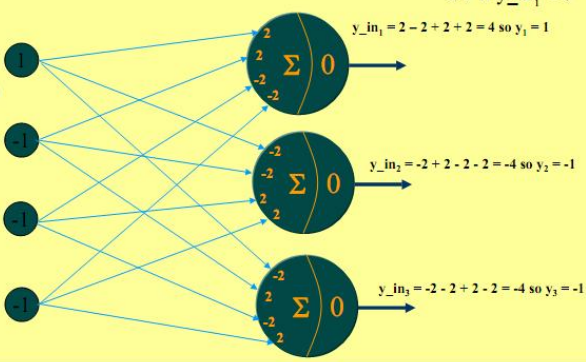


Try the first input Pattern

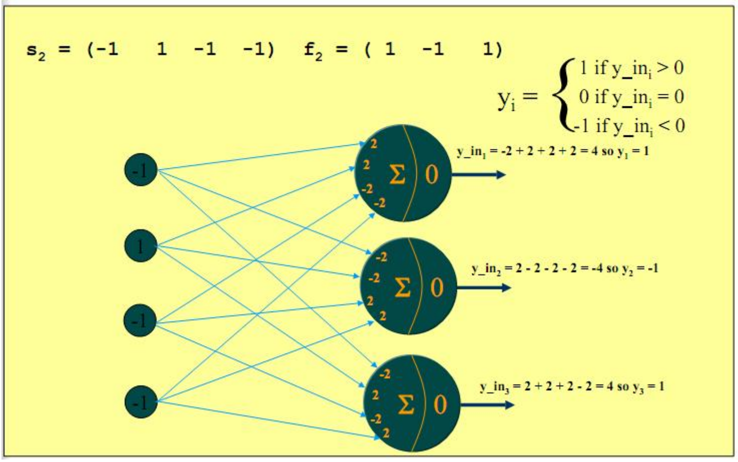
$s_1 = (1 \;\; -1 \;\; -1 \;\; -1) \quad f_1 = (1 \;\; -1 \;\; -1)$

$$y_i = \begin{cases} 1 \text{ if } y\_in_i > 0 \\ 0 \text{ if } y\_in_i = 0 \\ -1 \text{ if } y\_in_i < 0 \end{cases}$$

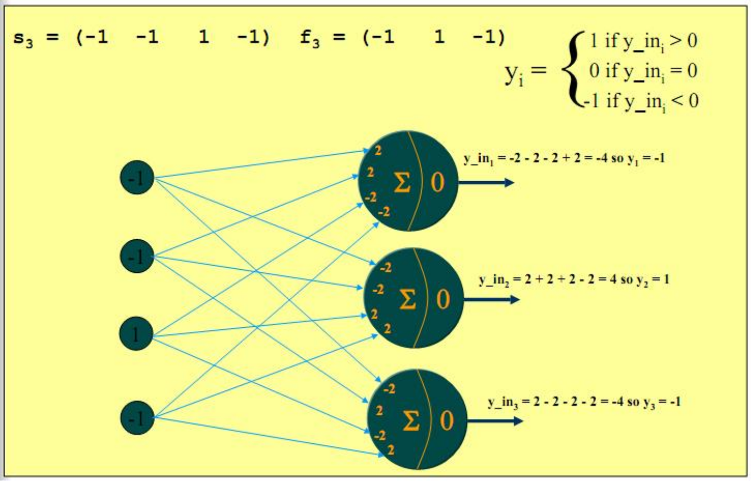
Where is this information stored?



Try the second input Pattern



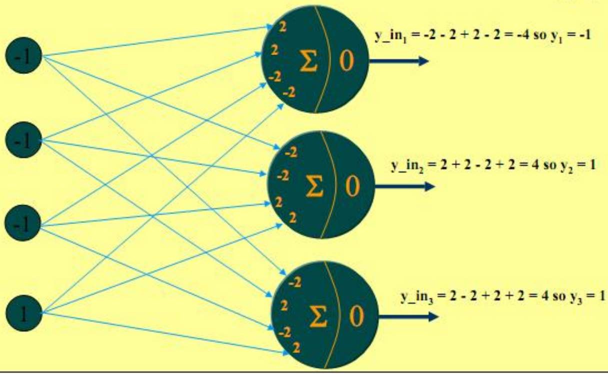
Try the third input Pattern



Try the forth input Pattern

s_4 = (-1 -1 -1 1) f_4 = (-1 1 1)

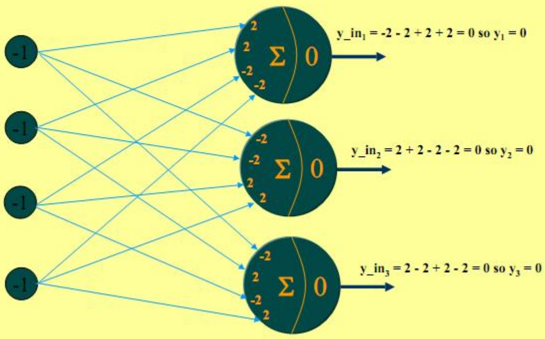
$$y_i = \begin{cases} 1 \text{ if } y\_in_i > 0 \\ 0 \text{ if } y\_in_i = 0 \\ -1 \text{ if } y\_in_i < 0 \end{cases}$$



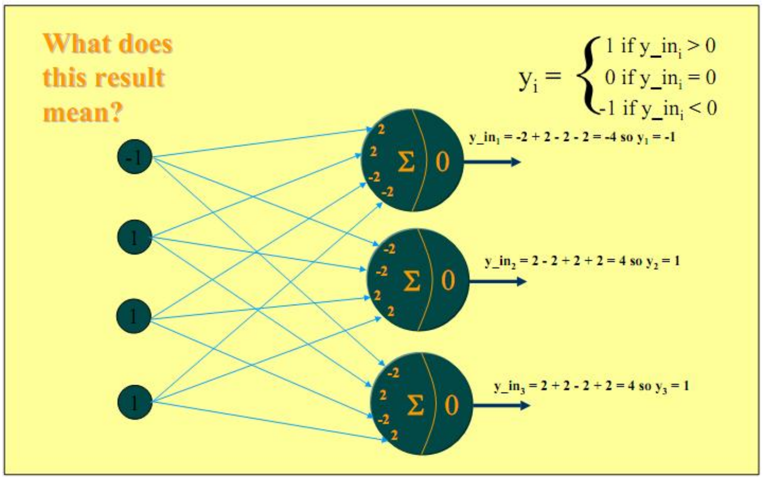
Try Non trained input Pattern(-1 ,-1 ,1 ,1)

What do the 0's mean?

$$y_i = \begin{cases} 1 \text{ if } y\_in_i > 0 \\ 0 \text{ if } y\_in_i = 0 \\ -1 \text{ if } y\_in_i < 0 \end{cases}$$



TRY NON TRAINED INPUT PATTERN(-1,1,1,1)



HETEROTYPE ASSOCIATIVE

Train a heteroassociative network to store the given bipolar input vectors $s = (s_1\ s_2\ s_3\ s_4)$ to the output vector $t = (t_1\ t_2)$. The bipolar vector pairs are as given in the table.

	s_1	s_2	s_3	s_4	t_1	t_2
1st	1	−1	−1	−1	−1	1
2nd	1	1	−1	−1	−1	1
3rd	−1	−1	−1	1	1	−1
4th	−1	−1	1	1	1	−1

Let the test vector be $x = [0\quad 1\quad 0\quad -1]$ with changes made in two components of 2nd input vector $[1\quad 1\quad -1\quad -1]$.

Let the test vector be $x = [-1\ 1\ 1 -1]$ with changes made in two components of 2nd input vector $[1\ 1\ -1\ -1]$.

AUTOASSOCIATIVE MEMORIES
TRAINING ALGORITHM

Step 0: Initialize all the weights to zero,

$$w_{ij} = 0(i = 1 \text{ to } n,\ j = 1 \text{ to } n)$$

Step 1: For each of the vector that has to be stored perform Steps 2–4.

Step 2: Activate each of the input unit,

$$x_i = s_i(i = 1 \text{ to } n)$$

Step 3: Activate each of the output unit,

$$y_j = s_j(j = 1 \text{ to } n)$$

Step 4: Adjust the weights,

$$w_{ij}(\text{new}) = w_{ij}(\text{old}) + x_i y_j$$

The weights can also be determined by the formula

$$W = \sum_{p=1}^{P} s^{\mathrm{T}}(p)s(p)$$

TESTING ALGORITHM

Step 0: Set the weights obtained for Hebb's rule or outer products.

Step 1: For each of the testing input vector presented perform Steps 2–4.

Step 2: Set the activations of the input units equal to that of input vector.

Step 3: Calculate the net input to each output unit $j = 1$ to n:

$$y_{inj} = \sum_{i=1}^{n} x_i w_{ij}$$

Step 4: Calculate the output by applying the activation over the net input:

$$y_j = f(y_{inj}) = \begin{cases} +1 & \text{if} \quad y_{inj} > 0 \\ -1 & \text{if} \quad y_{inj} \le 0 \end{cases}$$

This type of network can be used in speech processing, image processing, pattern classification etc.

AUTO-ASSOCIATIVE MEMORY NETWORK

- Same as hetero-associative nets, except ***t*(*p*) =*s* (*p*)**.
- Used to recall a pattern by a its noisy or incomplete version.
  (pattern completion/pattern recovery)
- A single pattern ***s*** = (1, 1, 1, -1) is stored (weights computed by Hebbian rule or outer product rule.

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

training pattern $(1\,1\,1\,-1)\cdot W = (4\,4\,4\,-4) \rightarrow (1\,1\,1\,-1)$

noisy pattern $(-1\,1\,1\,-1)\cdot W = (2\,2\,2\,-2) \rightarrow (1\,1\,1\,-1)$

missing info $(0\,0\,1\,-1)\cdot W = (2\,2\,2\,-2) \rightarrow (1\,1\,1\,-1)$

more noisy $(-1\,-1\,1\,-1)\cdot W = (0\,0\,0\,0)$ not recognized

25

STORAGE CAPACITY

- Number of patterns that can be correctly stored & recalled by a network.
- More patterns can be stored if they are not similar to each other (e.g., orthogonal).
- Non-orthogonal

$$\begin{matrix}(1 & -1 & -1 & 1) \\ (1 & 1 & -1 & 1)\end{matrix} \rightarrow \boldsymbol{W}_0 = \begin{bmatrix} 0 & 0 & -2 & 2 \\ 0 & 0 & 0 & 0 \\ -2 & 0 & 0 & -2 \\ 2 & 0 & -2 & 0 \end{bmatrix}$$

$(1\,-1\,-1\,1)\cdot \boldsymbol{W}_0 = (1\;\;0\;\;-1\;\;1)$

It is not stored correctly

- Orthogonal

$$\begin{matrix}(1 & 1 & -1 & -1) \\ (-1 & 1 & 1 & -1) \\ (-1 & 1 & -1 & 1)\end{matrix} \rightarrow \boldsymbol{W}_0 = \begin{bmatrix} 0 & -1 & -1 & -1 \\ -1 & 0 & -1 & -1 \\ -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}$$

All three patterns can be correctly recalled

26

Train the autoassociative network for input vector [−1 1 1 1] and also test the network for the same input vector. Test the autoassociative network with one missing, one mistake, two missing and two mistake entries in test vector.

{0 1 1 1}
{−1 1 0 0}
−1 −1 1 1

Solution: The input vector is $x = [-1\ 1\ 1\ 1]$.

The weight vector is

$$W = \sum s^{T}(p)s(p)$$

$$= \begin{bmatrix} -1 \\ 1 \\ 1 \\ 1 \end{bmatrix}_{4\times 1} \begin{bmatrix} -1 & 1 & 1 & 1 \end{bmatrix}_{1\times 4}$$

$$W = \begin{bmatrix} 1 & -1 & -1 & -1 \\ -1 & 1 & 1 & 1 \\ -1 & 1 & 1 & 1 \\ -1 & 1 & 1 & 1 \end{bmatrix}_{4\times 4}$$

HOPFIELD NETWORK

- The Hopfield network implements a so-called content addressable memory.
- A collection of patterns called fundamental memories is stored in the NN by means of weights.
- Each neuron represents a component of the input.
- The weight of the link between two neurons measures the correlation between the two corresponding components over the fundamental memories. If the weight is high then the corresponding components are often equal in the fundamental memories.

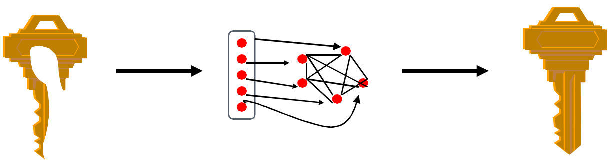
HOPFIELD NETWORK

- Input vectors values are in {-1,1} (or {0,1}).
- The number of neurons is equal to the input dimension.
- Every neuron has a link from every other neuron (recurrent architecture) except itself (no self-feedback).
- The neuron state at time n is its output value.
- The network state at time n is the vector of neurons states.
- The activation function used to update a neuron state is the *sign function* but if the input of the activation function is 0 then the new output (state) of the neuron is equal to the old one.
- Weights are symmetric: $w_{ij} = w_{ji}$

HOPFIELD NETWORKS

- Auto-Association Network
- Fully-connected (clique) with symmetric weights
- State of node = f(inputs)
- Weight values based on Hebbian principle
- Performance: Must iterate a bit to converge on a pattern, but generally much less computation than in back-propagation networks.

Input Output (after many iterations)



Discrete node update rule: $x_{pk}(t+1) = \mathrm{sgn}(\sum_{j=1}^{n} w_{kj} x_{pj}(t) + I_{pk})$

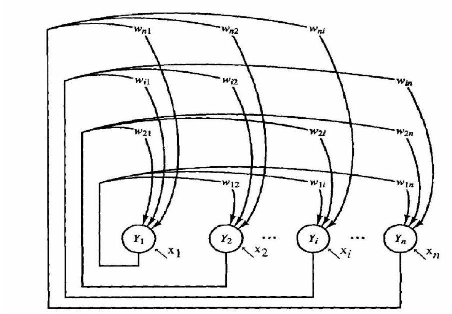
Input value

ARCHITECTURE OF DHN

- **Architecture**
  - Single-layer (units serve as both input and output):
    - ✓ nodes are threshold units (binary or bipolar).
    - ✓ weights: fully connected, symmetric, and zero diagonal.

$$w_{ij} = w_{ji}$$
$$w_{ii} = 0$$

x_i are external inputs, which may be transient or permanent.



Step 0: Initialize the weights to store patterns, i.e., weights obtained from training algorithm using Hebb rule.

Step 1: When the activations of the net are not converged, then perform Steps 2–8.

Step 2: Perform Steps 3–7 for each input vector X.

Step 3: Make the initial activations of the net equal to the external input vector X:

$$y_i = x_i \; (i = 1 \text{ to } n)$$

Step 4: Perform Steps 5–7 for each unit Y_i. (Here, the units are updated in random order.)

Step 5: Calculate the net input of the network:

$$y_{ini} = x_i + \sum_j y_j w_{ji}$$

Step 6: Apply the activations over the net input to calculate the output:

$$y_i = \begin{cases} 1 & \text{if } y_{ini} > \theta_i \\ y_i & \text{if } y_{ini} = \theta_i \\ 0 & \text{if } y_{ini} < \theta_i \end{cases}$$

where θ_i is the threshold and is normally taken as zero.

Step 7: Now feed back (transmit) the obtained output y_i to all other units. Thus, the activation vectors are updated.

Step 8: Finally, test the network for convergence.

Construct an auto associative discrete Hopfield network with input vector [1 1 1 - 1].
Test the discrete Hopfield network with missing values in first and second components of the stored vector.

Q.2 B) Draw Hopfield Neural Network with four output nodes. Also explain training and testing algorithm of Hopfield neural network. (10)

Q.3A)i) A Hopfield network made up of five neurons, which is required to store the following patterns:

$P1 = [1 \ \ 1 \ \ 1 \ \ 1 \ \ 1]^T$

$P2 = [1 \ \ -1 \ \ -1 \ \ 1 \ \ -1]^T$

$P3 = [-1 \ \ 1 \ \ -1 \ \ 1 \ \ 1]^T$

Evaluate the 5-by 5 weight matrix of the Hopfield Network (6)

REFERENCES

- Neural Networks by Fauset
- http://undergraduate.csse.uwa.edu.au/units/CITS4210/lectureNotes/Lect1-UWA.pdf
- www.idi.ntnu.no/~keithd/classes/advai/lectures/**assocnet.ppt**
- Principles of Soft Computing by Sivanandam ,Deepa