ASSOCIATIVE MEMORY NETWORK

- It stores a set of patterns as memories
- When it is presented with a key pattern, it responds by producing one of the stored patterns
- The selected pattern resembles or relates to the key pattern
- The recall is through association of the key pattern
- These types of memories are called **content- addressable memories** (CAM)
- In digital computers we have **address- addressable memories**
- As in RAM/ROM it is also a matrix memory

ASSOCIATIVE MEMORY NETWORK CONTD…

- The input data is correlated with that of the stored data in CAM
- The stored patterns must be unique (That is different patterns are stored in different locations)
- Each data stored has an address
- If multiple copies are stored the data will be correct. But the address will be ambiguous
- The concept behind this search is to output any one or all stored items which matches the given search argument
- The stored data is retrieved completely or partially

ASSOCIATIVE MEMORY NETWORK CONTD…

- **Two types** are there
- **AUTO ASSOCIATIVE**
- **HETERO ASSOCIATIVE**
- If the output vector is same as the input vectors it is Auto Associative
- If the output vectors are different from the input vectors it is Hetero Associative
- To find the similarity we use the **Hamming distance**

HAMMING DISTANCE

- Suppose we have two vectors,

$$x = (x_1, x_2, \ldots x_n)^T \; and \; y = (\mathrm{y}_1, \mathrm{y}_2, \ldots \mathrm{y}_n)^T$$

- Then the Hamming Distance is defined as

$$HD(x, y) = \begin{cases} \sum_{i=1}^{n} |x_i - y_i|, & if \;\; x_i, y_i \in \{0,1\}; \\ \frac{1}{2} \sum_{i=1}^{n} |x_i - y_i|, & if \;\; x_i, y_i \in \{-1,1\}. \end{cases}$$

- The hamming distance basically denotes the number of places where the two vectors differ

TRAINING ALGORITHMS FOR PATTERN ASSOCIATION

- There are two algorithms for training of pattern association nets

  1. Hebb Rule

  2. Outer Products Rule

- **Hebb Rule**
- This is widely used for finding the weights of an associative memory neural net
- Here, the weights are updated until there is no weight change

ALGORITHMIC STEPS FOR HEBB RULE

- **STEP 0:** Set all the initial weights to zero
- ($w_{ij} = 0, i = 1, 2 \ldots n;\ j = 1, 2, \ldots \mathrm{m}$)
- **STEP 1:** For each training target input output vector pairs (s:t) perform steps 2 to 4
- **STEP 2:** Activate the input layer units to current training input
- $(x_i = s_i, i = 1, 2, \ldots n)$
- **STEP 3:** Activate the output layer units to the target output
- $(y_j = t_j,\ j = 1, 2, \ldots m)$
- **STEP 4:** Start the weight adjustments

$$w_{ij}(new) = w_{ij}(\mathrm{old}) + \mathrm{x}_i \cdot y_j, i = 1, 2, \ldots n;\ j = 1, 2, \ldots m$$

OUTER PRODUCT RULE

- Here, we have
- Input: $s = (s_1, s_2, ..., s_i, ...s_n)$
- Output: $t = (t_1, t_2, ...t_j, ...t_m)$
- The outer product is defined as: The product of the two matrices: $S = s^T \; and \; T = t$
- So, we get the weight matrix W as:

$$W = S.T = \begin{bmatrix} s_1 \\ . \\ s_i \\ . \\ s_n \end{bmatrix} . \begin{bmatrix} t_1 & . & t_j & . & t_m \end{bmatrix} = \begin{bmatrix} s_1 t_1 & . & s_1 t_j & & s_1 t_m \\ & & & & \\ s_i t_1 & & s_i t_j & & s_i t_m \\ & & & & \\ s_n t_1 & & s_n t_j & & s_n t_m \end{bmatrix}$$

OUTER PRODUCT RULE CONTD…

- In case of a set of patterns, s(p): t(p), p=1,…P; we have

$$s(p) = (s_1(p), \ldots, s_i(p), \ldots, s_n(p))$$
$$\mathrm{t}(p) = (\mathrm{t}_1(p), \ldots, \mathrm{t}_i(p), \ldots, \mathrm{t}_m(p))$$

- For the weight matrix W = (w_{ij})
-

$$w_{ij} = \sum_{p=1}^{P} s_i^T(p).t_j(p), i = 1, 2, \ldots n;\ j = 1, 2, \ldots m.$$

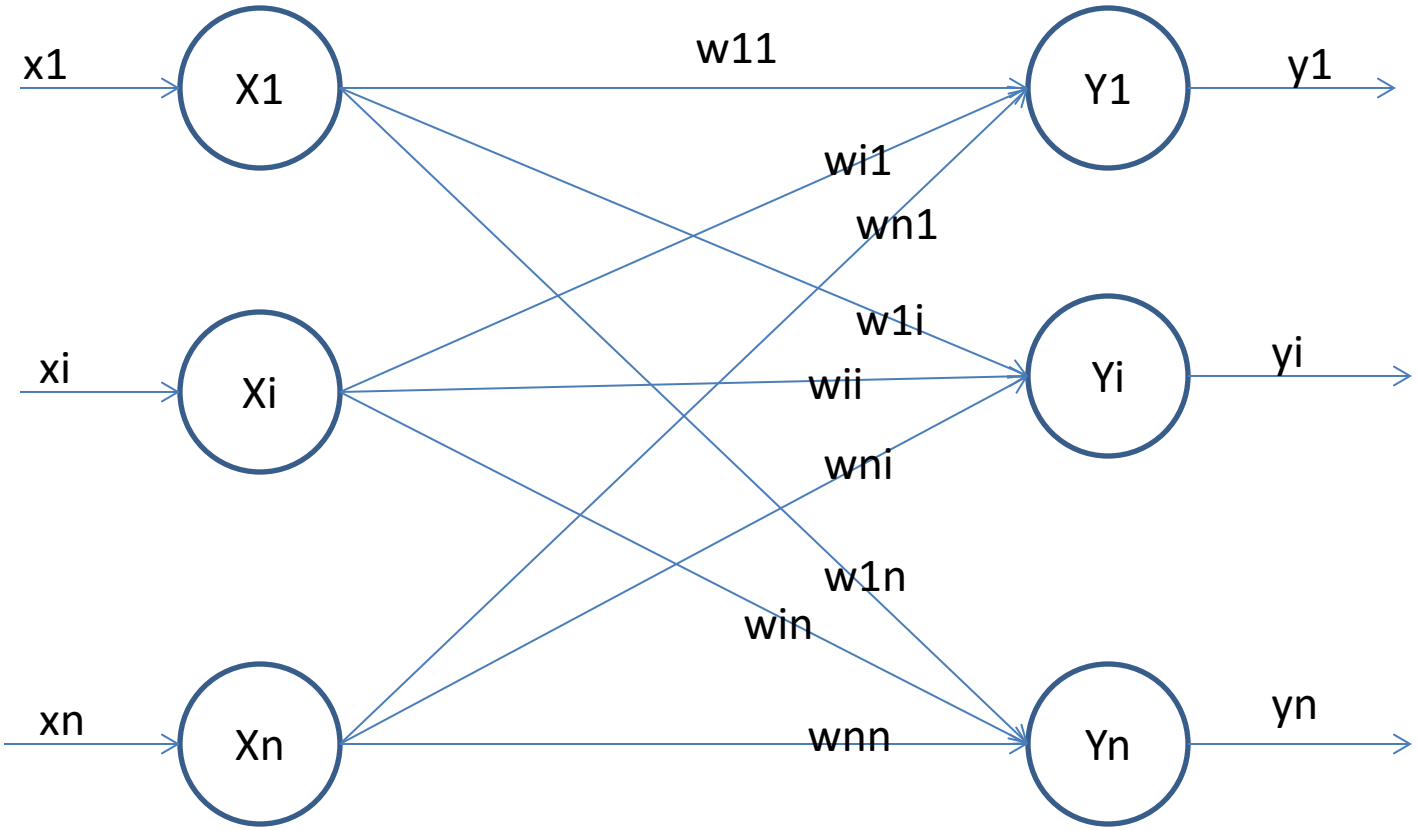
AUTOASSOCIATIVE MEMORY NETWORK

- The training input and target output vectors are the same
- The determination of weights of the association net is called **storing of vectors**
- The vectors that have been stored can be retrieved from distorted (noisy) input if the input is sufficiently similar to it
- The net's performance is based on its ability to reproduce a stored pattern from a noisy input

- **NOTE:** The weights in the diagonal can be set to '0'. These nets are called **auto-associative nets with no self-connection**
- This is more suited for iterative nets

ARCHITECTURE OF AUTOASSOCIATIVE MEMORY NETWORK

- A



TRAINING ALGORITHM

- This is same as that for the Hebb rule. Except that there are same number of output units as the number of input units
- **STEP 0:** Initialize all the weights to '0'
- ($w_{ij} = 0, i = 1,2,...n;\ j = 1,2,...n$)
- **STEP 1:** For each of the vector that has to be stored, perform
- steps 2 to 4
- **STEP 2:** Activate each of the input unit ($x_i = s_i, i = 1,2,...n$)
- **STEP 3:** Activate each of the output units ($y_j = s_j, \mathrm{j} = 1,2,...n$)
- **STEP 4:** Adjust the weights, i, j = 1,2,…n;

$$w_{ij}(new) = w_{ij}(old) + x_i.y_j = w_{ij}(old) + s_i.\mathrm{s}_j$$

TESTING ALGORITHM

- The associative memory neural network can be used to determine whether the given input vector is a “known” one or an unknown one
- The net is said to recognize a vector if the net produces a pattern of activation as output, which is the same as the one given as input
- **STEP 0:** Set the weights obtained from any of the two methods described above
- **STEP 1:** For each of the testing input vector perform steps 2-4
- **STEP 2:** Set the activations of the input unit as equal to that of the input vector

TESTING ALGORITHM CONTD…

- **STEP 3:** Calculate the net input to each of the output unit:

$$(y_{in})_j = \sum_{i=1}^{n} x_i . w_{ij}, \; j = 1, 2, \ldots n$$

- **STEP 4:** Calculate the output by applying the activation over the net input:

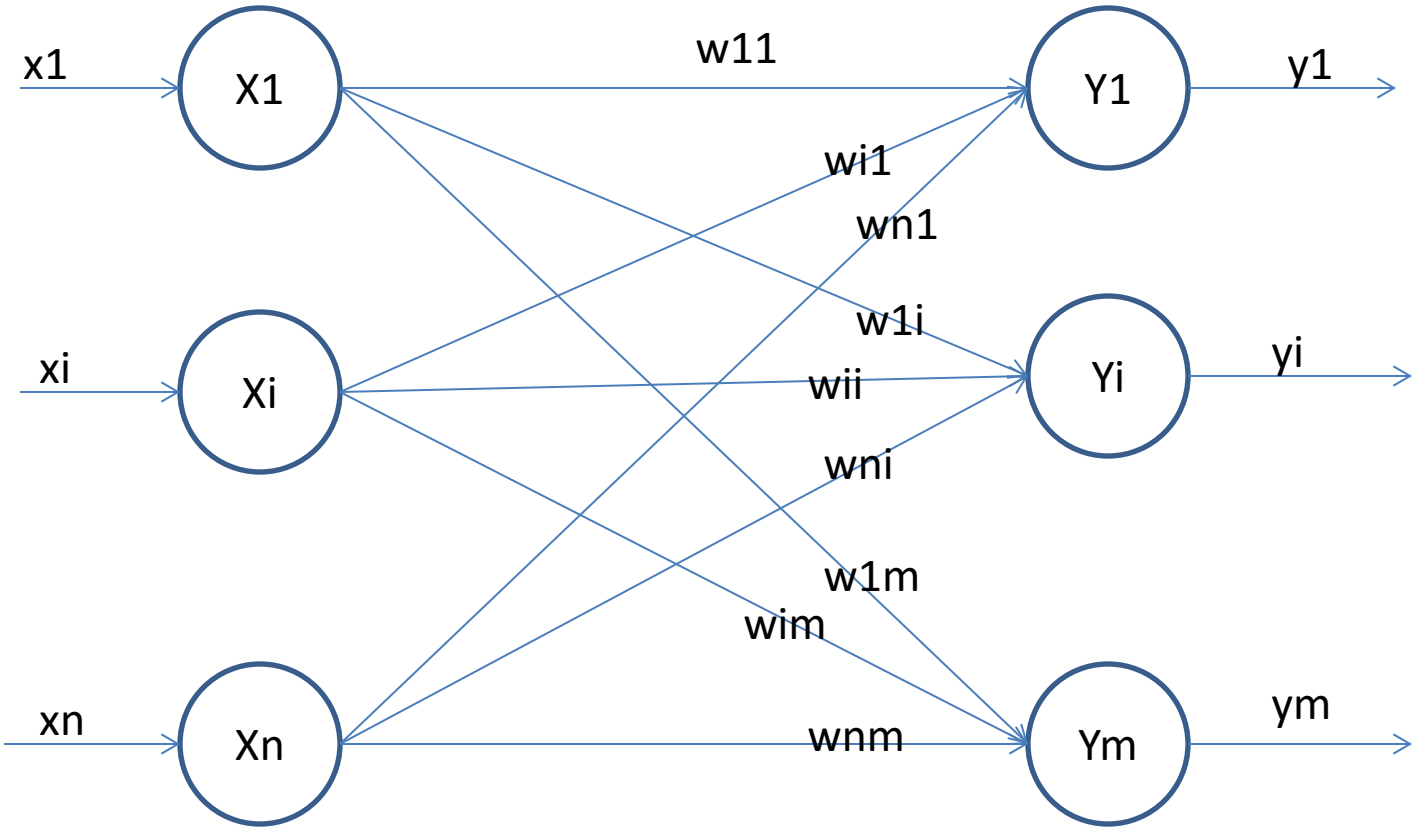
- $$y_j = f((y_{in})_j) = \begin{cases} 1, & if\ (y_{in})_j > 0; \\ -1, & if\ (y_{in})_j \leq 0. \end{cases}$$

HETEROASSOCIATIVE MEMORY NETWORK

- The training input and target output are different
- The weights are determined in such a way that the net can store a set of ‘P’ pattern associations
- Each input vector s(p) has ‘n’ components and each output vector t(p) has ‘m’ components
- The determination of weights is done by using ‘Hebb Rule’ or ‘Delta Rule’
- **THE NET FINDS AN APPROPRIATE OUTPUT VECTOR CORRESPONDING TO AN INPUT VECTOR, WHICH MAY BE ONE OF THE STORED PATTERNS OR A NEW PATTERN**

ARCHITECTURE OF HETEROASSOCIATIVE MEMORY NETWORK

- A



TESTING ALGORITHM FOR HETEROASSOCIATIVE MEMORY NETWORK

- **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 the input layer units equal to that of the current input vector given, x_i
- **STEP 3:** Calculate the net input to the output units using

$$(y_{in})_j = \sum_{i=1}^{n} x_i . w_{ij}, \; j = 1, 2, \ldots \mathrm{m}$$

- **STEP 4:** Determine the activations for the output units as:

$$y_j = \begin{cases} 1, & if\ (y_{in})_j > 0; \\ 0, & if\ (y_{in})_j = 0; \\ -1, & if\ (y_{in})_j < 0. \end{cases}$$

HETEROASSOCIATIVE MEMORY NETWORK CONTD…

- There exist weighted interconnections between the input and output layers
- The input and output layers are not correlated with each other
- We can also use the following binary activation function

$$y_j = \begin{cases} 1, & if\ (y_{in})_j > 0; \\ -1, & if\ (y_{in})_j < 0. \end{cases}$$

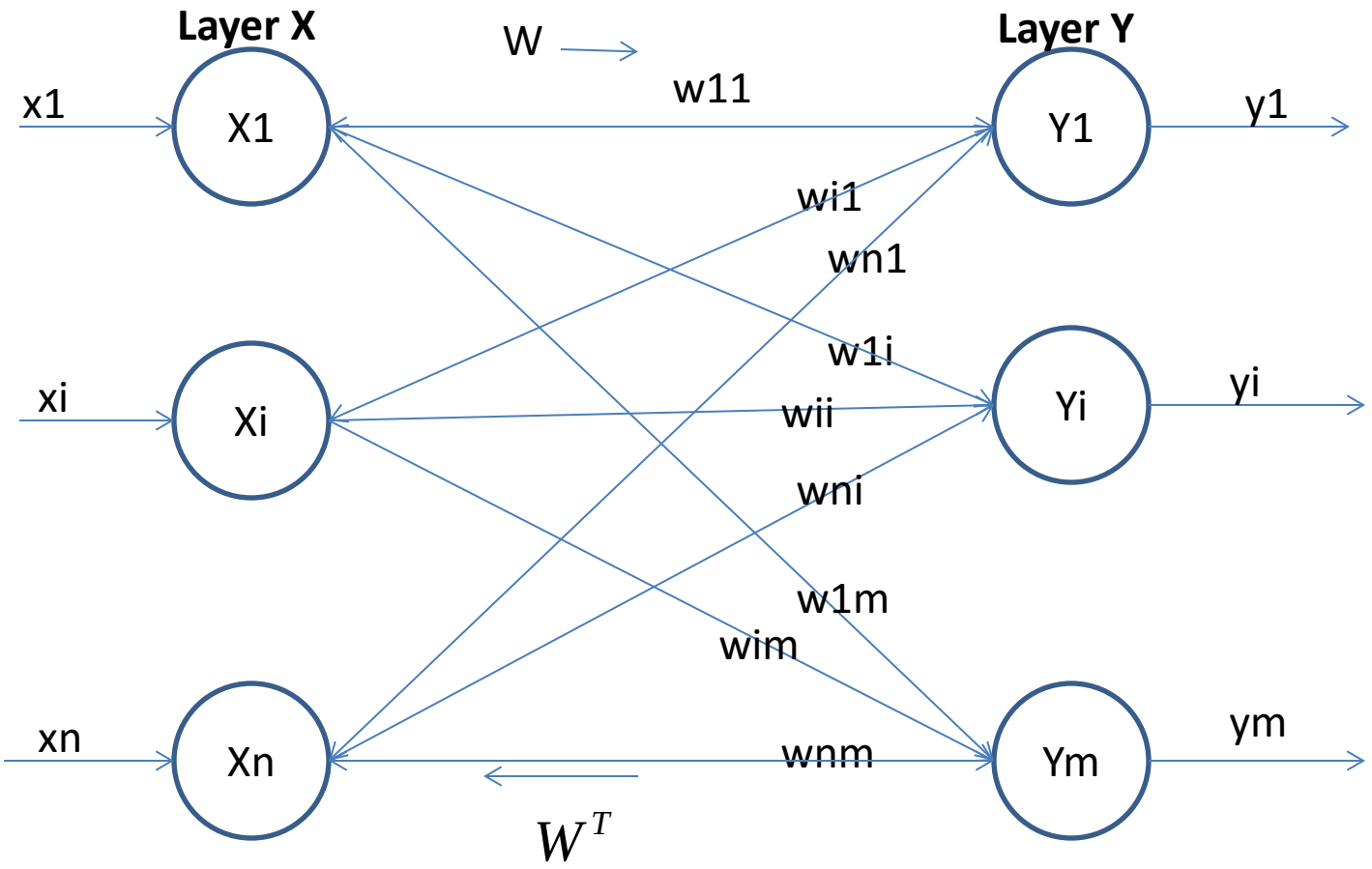
BIDIRECTIONAL ASSOCIATIVE MEMORY

- It was developed by **Kosko in 1988**
- It performs **both forward** and **backward** searches
- It **uses Hebb rule**
- It associates patterns from set A to set B and vice versa
- It responds for input from both layers

- There are **two types** of BAMs
  - Discrete
  - Continuous

THE BAM ARCHITECTURE

- A



BAM ARCHITECTURE

- Consists of **two layers** of neurons
- These layers are **connected by directed weighted path** interconnections
- The **network dynamics** involves two layers of interaction
- The **signals are sent back and forth** between the two layers until all neurons reach equilibrium
- The **weighs associated are bidirectional**
- It can respond to **inputs in either layer**
- The weight matrix from one direction (say A to B) is W
- The weight matrix from the other direction (B to A) is W^T

THE WEIGHT MATRICES

$$W = \begin{bmatrix} w_{11} & . & w_{1j} & . & w_{1m} \\ . & . & . & . & . \\ w_{i1} & . & w_{ij} & . & w_{im} \\ . & . & . & . & . \\ w_{n1} & . & w_{nj} & . & w_{nm} \end{bmatrix}$$

$$W^T = \begin{bmatrix} w_{11} & . & w_{i1} & . & w_{n1} \\ . & . & . & . & . \\ w_{1j} & . & w_{ij} & . & w_{nj} \\ . & . & . & . & . \\ w_{1m} & . & w_{im} & . & w_{nm} \end{bmatrix} = \begin{bmatrix} w_{11} & . & w_{1i} & . & w_{1n} \\ . & . & . & . & . \\ w_{j1} & . & w_{ji} & . & w_{jn} \\ . & . & . & . & . \\ w_{m1} & . & w_{mi} & . & w_{mn} \end{bmatrix}$$

DISCRETE BIDIRECTIONAL ASSOCIATIVE MEMORY

- The diagram is as shown earlier
- When the memory neurons are being activated by putting an initial vector at the input of a layer, **the network evolves a two-pattern stable state** with each pattern at the output of one layer
- The network involves two layers of interaction between each other
- The **two bivalent forms** of BAM are related to each other
- These are
- **Binary**
- **Bipolar**
- The **weights in both the cases** are found as the **sum of the outer products of the bipolar form** of the given training vector pairs

DETERMINATION OF WEIGHTS

- Let the 'P' number of input/target pairs of vectors be denoted by s(p), t(p), p = 1,2,…P
- Then the weight matrix to store a set of input/target vectors

$$s(p) = (s_1(p), \ldots, s_i(p), \ldots, s_n(p))$$

$$\mathrm{t(p)} = (\mathrm{t}_1(p), \ldots, \mathrm{t}_j(p), \ldots, \mathrm{t}_m(p))$$

- p = 1,2,…P can be determined by Hebb rule for determining weights for associative networks
- In case of binary input vectors $w_{ij} = \sum_{p=1}^{P} [2s_i(p) - 1].[2t_j(p) - 1]$
- In case of bipolar input vectors $w_{ij} = \sum_{p=1}^{P} s_i^T(p).t_j(p)$

ACTIVATION FUNCTIONS FOR BAM

- The **step activation function** with **nonzero threshold** is used as the activation function for discrete BAM network
- The activation function is **based on** whether the **input vector pairs (at the input and target vector) are binary or bipolar**
- ACTIVATION FUNCTION FOR 'Y' LAYER
- 1. If both the layers have binary input vectors

$$y_j = \begin{cases} 1, & if\ (y_{in})_j > 0; \\ y_j, & if\ (y_{in})_j = 0; \\ 0, & if\ (y_{in})_j < 0. \end{cases}$$

ACTIVATION FUNCTIONS FOR BAM

2. When the **input vector is bipolar**

$$y_j = \begin{cases} 1, & if\ (y_{in})_j > \theta_j; \\ y_j, & if\ (y_{in})_j = \theta_j; \\ -1, & if\ (y_{in})_j < \theta_j. \end{cases}$$

- **INPUT FUNCTION FOR 'X' LAYER**

1. When the **input vector is binary**

$$x_i = \begin{cases} 1, & if\ (\mathrm{x}_{in})_i > 0; \\ x_i, & if\ (\mathrm{x}_{in})_i = 0; \\ 0, & if\ (\mathrm{x}_{in})_i < 0. \end{cases}$$

ACTIVATION FUNCTIONS FOR BAM

2. When the **input vector is bipolar**

$$x_i = \begin{cases} 1, & if\ (\mathrm{x}_{in})_i > \theta_i; \\ x_i, & if\ (\mathrm{x}_{in})_i = \theta_i; \\ -1, & if\ (\mathrm{x}_{in})_i < \theta_i. \end{cases}$$

TESTING ALGORITHM FOR DISCRETE BAM

- This algorithm is **used to test the noisy patterns entering into the network**
- Basing upon the training algorithm, weights are determined
- Using the weights the net input is calculated for the given test pattern
- **Activations are applied over it to recognize the test patterns**
- ALGORITHM:
- **STEP 0:** Initialize the weights to store 'p' vectors
- **STEP 1:** Perform Steps 2 to 6 for each testing input
- **STEP 2:** (i) Set the activations of the X layer to the current input pattern, that is presenting the input pattern x to X.

TESTING ALGORITHM FOR DISCRETE BAM CONTD…

- (ii) Set the activations of the ‘Y’ layer similarly by presenting the input pattern ‘y’.
- (iii) Though it is bidirectional memory, at one time step, signals can be sent from only one layer. So, one of the input patterns may be zero vector
- **STEP 3:** Perform steps 4 to 6 when the activations are not converging
- **STEP 4:** Update the activations of units in Y layer. Calculate the net input by

$$(y_{in})_j = \sum_{i=1}^{n} x_i . w_{ij}$$

TESTING ALGORITHM FOR DISCRETE BAM CONTD…

- Applying activations, we obtain $y_j = f((y_{in})_j)$
- Send this signal to the X layer
- **STEP 5:** Update the activations of units in X layer
- Calculate the net input

$$(x_{in})_i = \sum_{j=1}^{m} y_j . w_{ij}$$

- Apply the activations over the net input $x_i = f((x_{in})_i)$
- Send this signal to the Y layer
- **STEP 6:** Test for convergence of the net. The convergence occurs if the activation vectors x and y reach equilibrium.
- If this occurs stop, otherwise continue.

CONTINUOUS BAM

- It **transforms the input smoothly and continuously** in the range 0-1 **using logistic sigmoid functions** as the **activation functions** for all units
- This activation function may be **binary sigmoidal function** or it may be **bipolar sigmoidal function**
- When a bipolar sigmoidal function with a high gain is chosen, the continuous BAM might converge to a state of vectors which will approach to the vertices of a cube
- **In such a circumstance it acts like a discrete BAM**

CONTINUOUS BAM CONTD…

- In case of binary inputs, (s(p), t(p)), p = 1,2,…P; the weight is determined by the formula

$$w_{ij} = \sum_{p=1}^{P} [2s_i(p) - 1].[2t_j(p) - 1]$$

- The activation function is the logistic sigmoidal function
- If it is **binary logistic function** then the activation function is given by

$$f((y_{in})_j) = \frac{1}{1 + e^{-(y_{in})_j}}$$

CONTINUOUS BAM CONTD…

- If the activation function used is a **bipolar function** then it is given by

$$f((y_{in})_j) = \frac{2}{1+e^{-(y_{in})_j}} - 1 = \frac{1-e^{-(y_{in})_j}}{1+e^{-(y_{in})_j}}$$

- These activations are applied over the net input to calculate the output. The net input can be calculated with a bias as

$$(y_{in})_j = b_j + \sum_{i=1}^{n} x_i . w_{ij}$$

- **All these formulae are applicable to the X layer also**

UNSUPERVISED LEARNING NETWORKS

- Unsupervised learning is the **second major learning paradigm**
- The system(environment) does not provide any feedback to indicate the desired output
- The **network itself has to discover any relationships of interest**
- Translate the discovered relationships into outputs
- These type of ANNs are also called **self-organising networks**
- **It can determine how similar a new input is to typical patterns already seen**
- **The network gradually learns what similarity is**
- **It can set up several axes along which to measure similarity to previous patterns**

UNSUPERVISED LEARNING NETWORKS CONTD…

- The axes can be any one of the following:
- Principal Component Analysis
- Clustering
- Adaptive Vector Quantization
- Feature Mapping
- The net has **added structure** by means of which the net is **forced to make a decision**
- **If there are several neurons for firing then only one of them is selected to fire (respond)**
- The process for achieving this is called a **competition**

UNSUPERVISED LEARNING NETWORKS CONTD…

- **An Example:**
- Suppose we have a set of students
- Let us classify them on the basis of their performance
- The scores will be calculated
- The one whose score is higher than all others will be the winner
- The same principle is followed for pattern classification in neural networks
- Here, there may be a tie
- Some principle is followed even when there is a tie

UNSUPERVISED LEARNING NETWORKS CONTD…

- These nets are called **competitive nets**
- The extreme form of these nets are called **winner-take-all**
- In such a case, **only one neuron in the competing group will possess a non-zero output signal** at the end of the competition
- **Several ANNs exist under this category**

1.Maxnet **2.Mexican hat**

3.Hamming net **4.Kohonen self-organizing feature map**

5.Counter propagation net **6.Learning vector quantization**

7.Adaptive Resonance Theory (ART)

UNSUPERVISED LEARNING NETWORKS CONTD…

- In case of these ANNs the net seeks to **find patterns or regularity** in the input data by **forming clusters**
- ARTs are called **clustering nets**
- **In such nets there are as many input units as an input vector possessing components**
- Since each output unit represents a cluster, **the number of output units** will limit **the number of clusters that can be formed**
- **The learning algorithm used in most of these nets is known as Kohonen learning**

KOHONEN LEARNING

- The **units update their weights** by forming a **new weight vector**, which is **a linear combination of the old weight vector and the new input vector**
- The **learning continues for the unit whose weight vector is closest to the input vector**
- The **weight updation formula** for output cluster unit 'j' is given by

$$w_j(new) = w_{\Theta j}(old) + \alpha[x - w_{\Theta j}(old)]$$

- x: Input vector
- $w_{\Theta j}$: weight vector for unit j
- α : learning rate, its value decreases monotonically as training continues

DECISION ON WINNERS

- There are **two methods** to determine the winner of the network during competition
- **Method 1:**
- The **square of the Euclidean distance** between the **input vector** and the **weight vector** is computed
- The **unit whose weight vector** is at the **smallest distance** from the **input vector** is chosen as the **winner**
- **Method 2:**
- The **dot product** of the **input vector** and the **weight vector** is computed
- This **dot product** is nothing but the **net inputs** calculated for the **corresponding cluster units**

DECISION ON WINNERS

- The weight updation is performed over it because the one with the **largest dot product corresponds to the smallest angle between the input and the weight vector**, if both are of unit length
- We know that for two vectors a and b, the dot product is given by the formula

$$a \cdot b = |a||b| \cos\theta$$

- If $|a| = |b| = 1$ then a . b = $\cos\theta$
- Also, as θ increases a . b decreases
- **Both the methods can be used if the vectors are of unit length**
- **The first method is normally preferred**

KOHONEN SELF-ORGANISING FEATURE MAPS

- Feature mapping is a process which converts the patterns of arbitrary dimensionality into a response of one or two dimensional array of neurons
- It converts a wide pattern space into atypical feature space
- Feature map is a network performing such a map
- It maintains the neighbourhood relations of input patterns
- It has to obtain a topology preserving map
- For such feature maps it is required to find a self-organizing neural array which consists of neurons arranged in a one-dimensional array or two-dimensional array

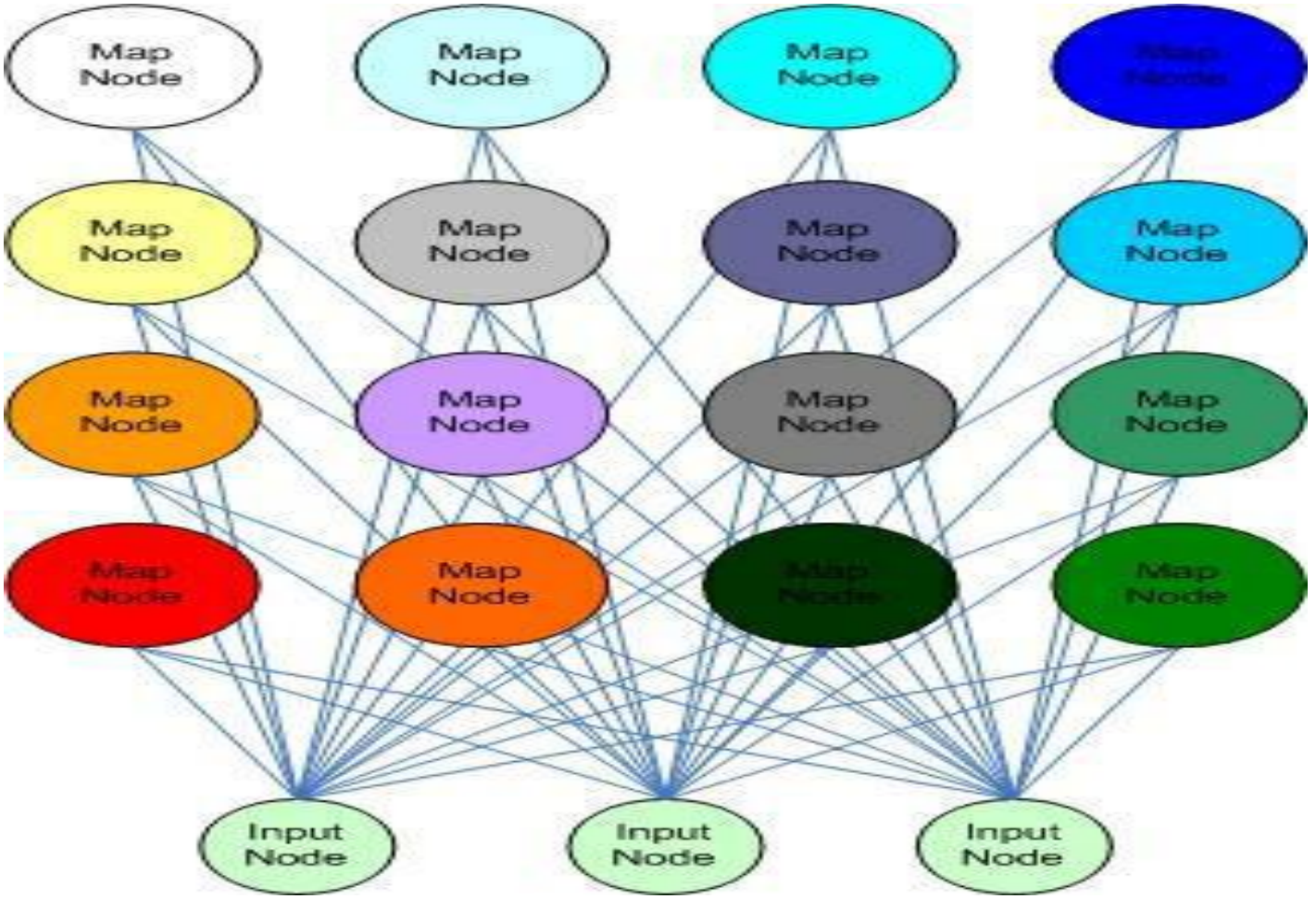
KOHONEN SELF-ORGANISING FEATURE MAPS CONTD…

- Developed in **1982**
- By **Tuevo Kohonen**, a professor emeritus of the **Academy of Finland**
- SOMs learn on their own through unsupervised competitive learning
- “Maps” is because they attempt to map their weights to conform to the given input data
- The nodes in a SOM network attempt to become like the inputs presented to them
- Retaining principle 'features’ of the input data is a fundamental principle of SOMs

KOHONEN SELF-ORGANISING FEATURE MAPS CONTD…

- **SOMs provide a way of representing multidimensional data in a much lower dimensional space** – typically one or two dimensions
- This aides in their visualization benefit, as **humans are more proficient at comprehending data in lower dimensions than higher dimensions**
- SOMs are a valuable tool in **dealing with complex or vast amounts of data**
- These are extremely **useful for the visualization and representation of complex or large quantities of data in a manner that is most easily understood by the human brain**

KOHONEN SELF-ORGANISING FEATURE MAPS CONTD…



KOHONEN SELF-ORGANISING FEATURE MAPS CONTD…

- Few key things to notice:
- **Each map node is connected to each input node**
- For this small 4x4 node network, that is 4x4x3=48 connections
- Map nodes are not connected to each other
- Nodes are organized in this manner, as a 2-D grid makes it easy to visualize the results
- **Each map node has a unique (i, j) coordinate**
- This makes it **easy to reference a node in the network**
- Also, it is **easy to calculate the distances between nodes**
- Because of the connections only to the input nodes, the map nodes are oblivious as to what values their neighbours have

KSO FEATURE MAPS: TRAINING ALGORITHM

- The steps involved in the training algorithm are:
- **STEP 0:** Initialize the weights w_{ij} (Random values are assumed)
- These can be chosen as the same as the components of the input vector
- Set the topological neighbourhood parameters (radius of the neighbourhood etc.)
- Initialize the learning rate
- **STEP 1:** Perform Steps 2 – 8 when stopping condition is false
- **STEP 2:** Perform Steps 3 – 5 for each input vector x
- **STEP 3:** Compute the square of the Euclidean distance for j = 1,2,…m.

KSO FEATURE MAPS: TRAINING ALGORITHM

- We have $D(j) = \sum_{i=1}^{m} (x_i - w_{ij})^2$

- We can use the dot product method also
- **STEP 4:** Find the winning unit J, so that D(J) is minimum.
- **STEP 5:** For all units j within a specific neighbourhood of J and for all I, calculate the new weights as:

$$w_{ij}(new) = w_{ij}(\text{old}) + \alpha.[\text{x}_i - w_{ij}(\text{old})]$$

$$\Updownarrow$$

$$w_{ij}(new) = (1-\alpha)w_{ij}(\text{old}) + \alpha.x_i$$

KSO FEATURE MAPS: TRAINING ALGORITHM

- **STEP 6:** Update the learning rate α using the formula

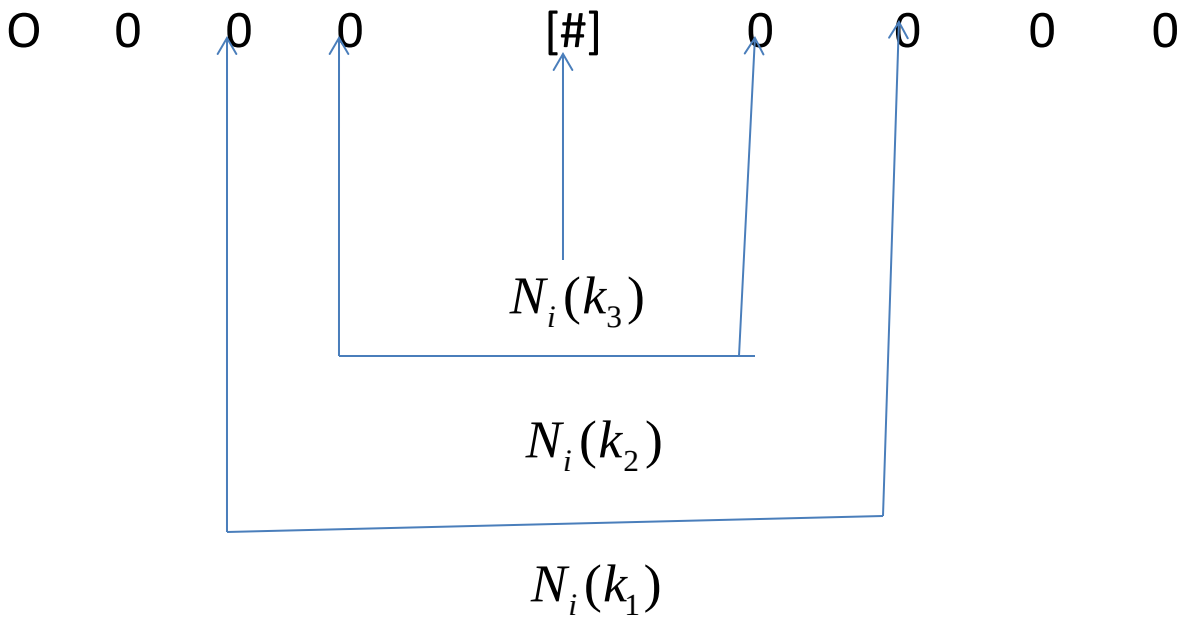
$$\alpha(t+1) = (0.5).\alpha(t)$$

- **STEP 7:** Reduce radius of topological neighbourhood at specified times
- **STEP 8:** Test for stopping condition of the network
-

KSO FEATURE MAPS: ARCHITECTURE

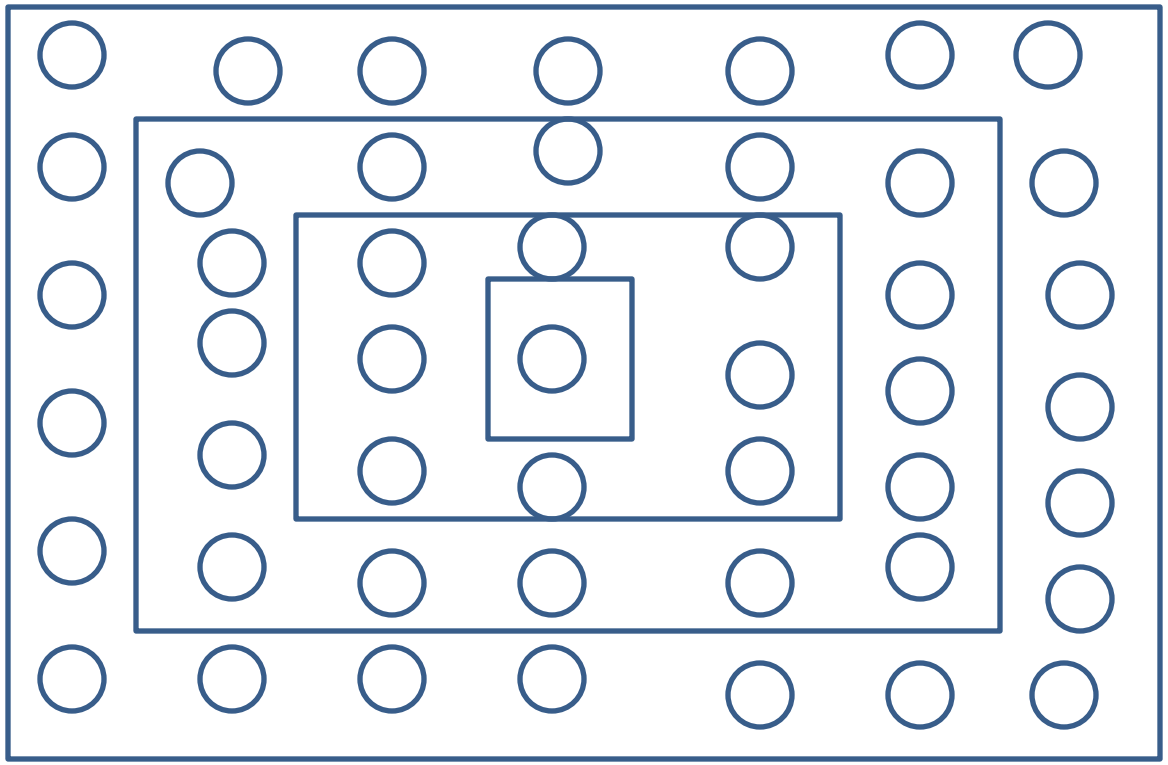
- For a linear array of cluster units, the neighbourhood units designated by ‘o’ of radii $N_i(k_1), N_i(k_2), N_i(k_3)$;

$$k_1 > k_2 > k_3; k_3 = 0, k_2 = 1, k_1 = 3$$



KSO FEATURE MAPS: ARCHITECTURE

- For a rectangular grid it looks like



KSO FEATURE MAPS: STEPS

- The self-organization process involves four major components:

- **Initialization:** All the connection weights are initialized with small random values

- **Competition:** For each input pattern, the neurons compute their respective values of a discriminant function which provides the basis for competition. *The particular neuron* with the smallest value of the discriminant function is declared the winner.

KSO FEATURE MAPS

- **Cooperation:** The winning neuron determines the spatial location of a topological neighbourhood of excited neurons, thereby providing the basis for cooperation among neighbouring neurons.

- **Adaptation:** The excited neurons decrease their individual values of the discriminant function in relation to the input pattern through suitable adjustment of the associated connection weights, such that the response of the winning neuron to the subsequent application of a similar input pattern is enhanced