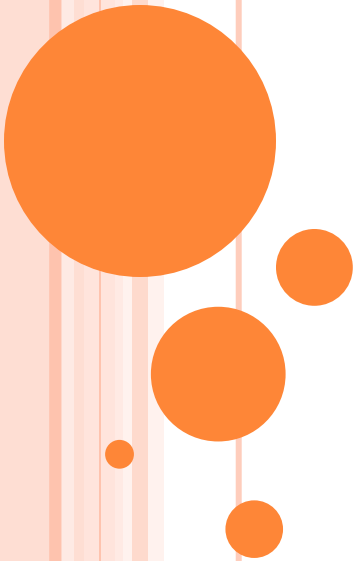


# Module 3 (upto ISE)

## Different Neural Networks

# Syllabus

- **Different Neural Networks 10 CO2, CO3**
- **3.1** Associative memory network – Basic Concepts, Types- Auto, Hetro, Bidirectional (Discrete and continuous), Testing
- **3.2** Hopfield – Discrete, continuous, Counter propagation network, ART, SOFM, Recurrent Network

# Introduction

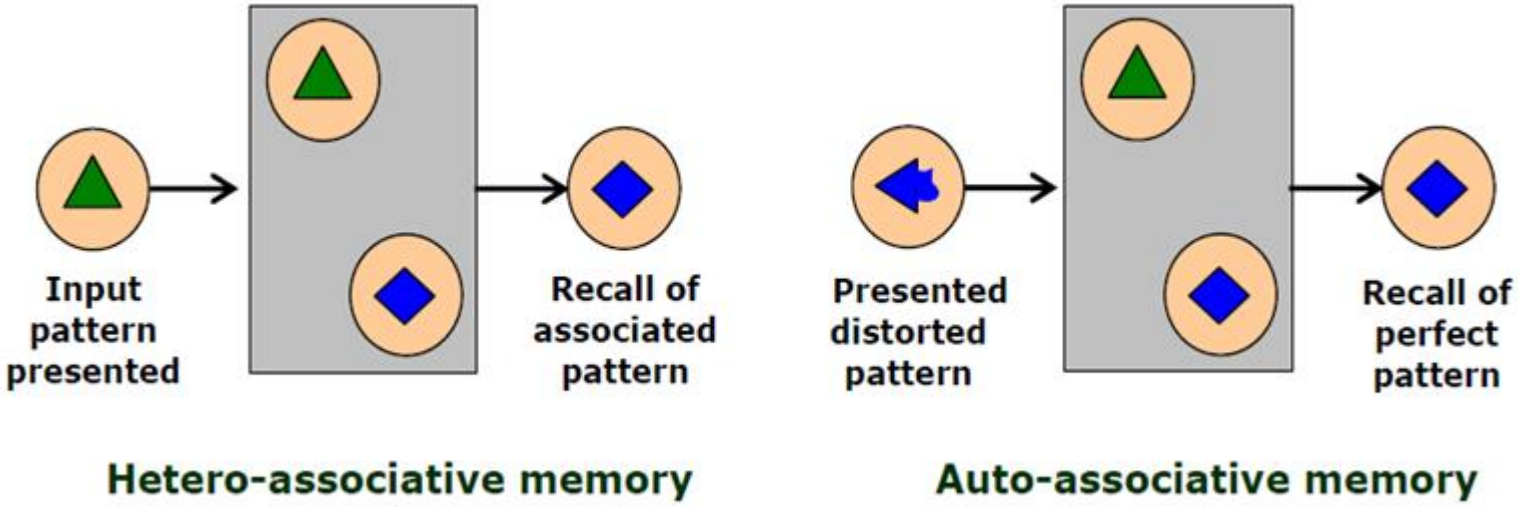
- An associative memory network can store a set of patterns as memories.
- When the associative memory is being presented with a key pattern, it responds by producing one of the stored patterns, which closely resembles or relates to the key pattern.
- Thus, the recall is through association of the key pattern, with the help of information memorized.
- These types of memories are also called as *content-addressable memories* (CAM) in contrast to that of traditional *address-addressable memories* in digital computers where stored pattern (in byres) is recalled by its address.

# Types of Associative Memory

1. *Auto-associative memory*
2. *Hetro-associative memory*

- Both these nets are single layer nets in which the weights are determined in a manner that the net stores a set of pattern associations.
- If each of the output vectors is same as the input vectors with which it is associated, then the net is a said to be auto-associative memory On the other hand if the output vectors are different from the input vectors then the net is said to be hetro-associative memory net.

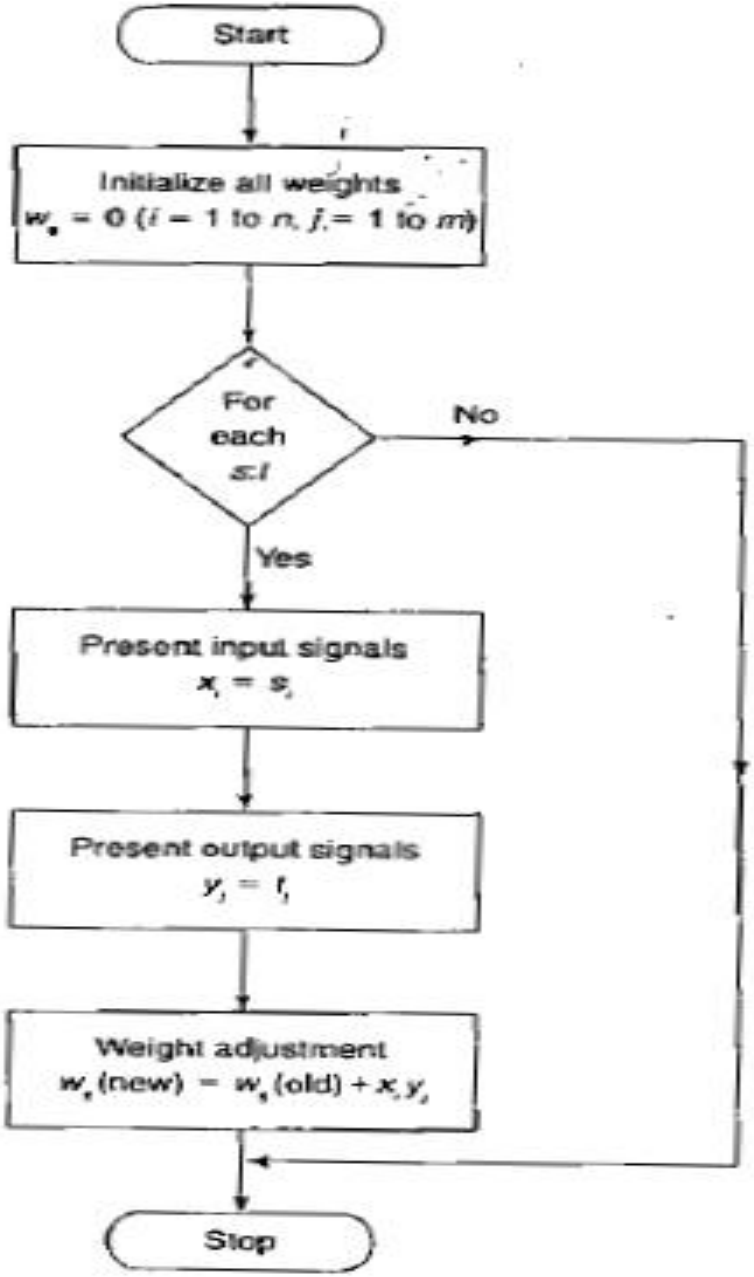
# Example


Input pattern presented
Recall of associated pattern
Hetero-associative memory
Presented distorted pattern
Recall of perfect pattern
Auto-associative memory

# Training algorithm for pattern association-Hebb Rule

- **Hebb rule**
  - Used for finding the weights of an associative memory neural net.
  - Training vector pairs are denoted by s:t
- Algorithm
  - Step 0: Initialize weights to 0
    - $w_{ij}=0$ $(i= 1 \text{ to } n, j=1 \text{ to } M)$
  - For each training target input output vectors *s:t,*
  - Activate the input layers for training input.
    - $x_i=s_i (for\ i=1 \text{ to } n)$
  - Activate the output layers for target output.
    - $y_i=t_i\ (for\ j=1 \text{ to } m)$
  - Start weight adjustment.
  - $w_{ij}(new)=w_{ij}(old)+x_iy_j\ (for\ i=1 \text{ to } n, j=1 \text{ to } m)$
  - Used with patters that can be represented as either binary or bipolar vectors.

# FLOWCHART


Start
Initialize all weights
$w_{ij} = 0$ ($i = 1$ to $n$, $j = 1$ to $m$)
For each $s:I$
No
Yes
Present input signals
$x_i = s_i$
Present output signals
$y_j = t_j$
Weight adjustment
$w_{ij}(new) = w_{ij}(old) + x_i y_j$
Stop

# Training algorithm for pattern association-Outer product Rule

- Let ***s*** and ***t*** be row vectors.
- Then for a particular training pair ***s:t***

$$\Delta W(p) = s^T(p) \cdot t(p) = \begin{bmatrix} s_1 \\ \\ \\ s_n \end{bmatrix} [t_1, \ldots, t_m] = \begin{bmatrix} s_1 t_1 \ldots s_1 t_m \\ s_2 t_1 \ldots s_2 t_m \\ \\ \\ s_n t_1 \ldots s_n t_m \end{bmatrix} = \begin{bmatrix} \Delta w_{11} \ldots \Delta w_{1m} \\ \\ \\ \Delta w_{n1} \ldots \Delta w_{nm} \end{bmatrix}$$

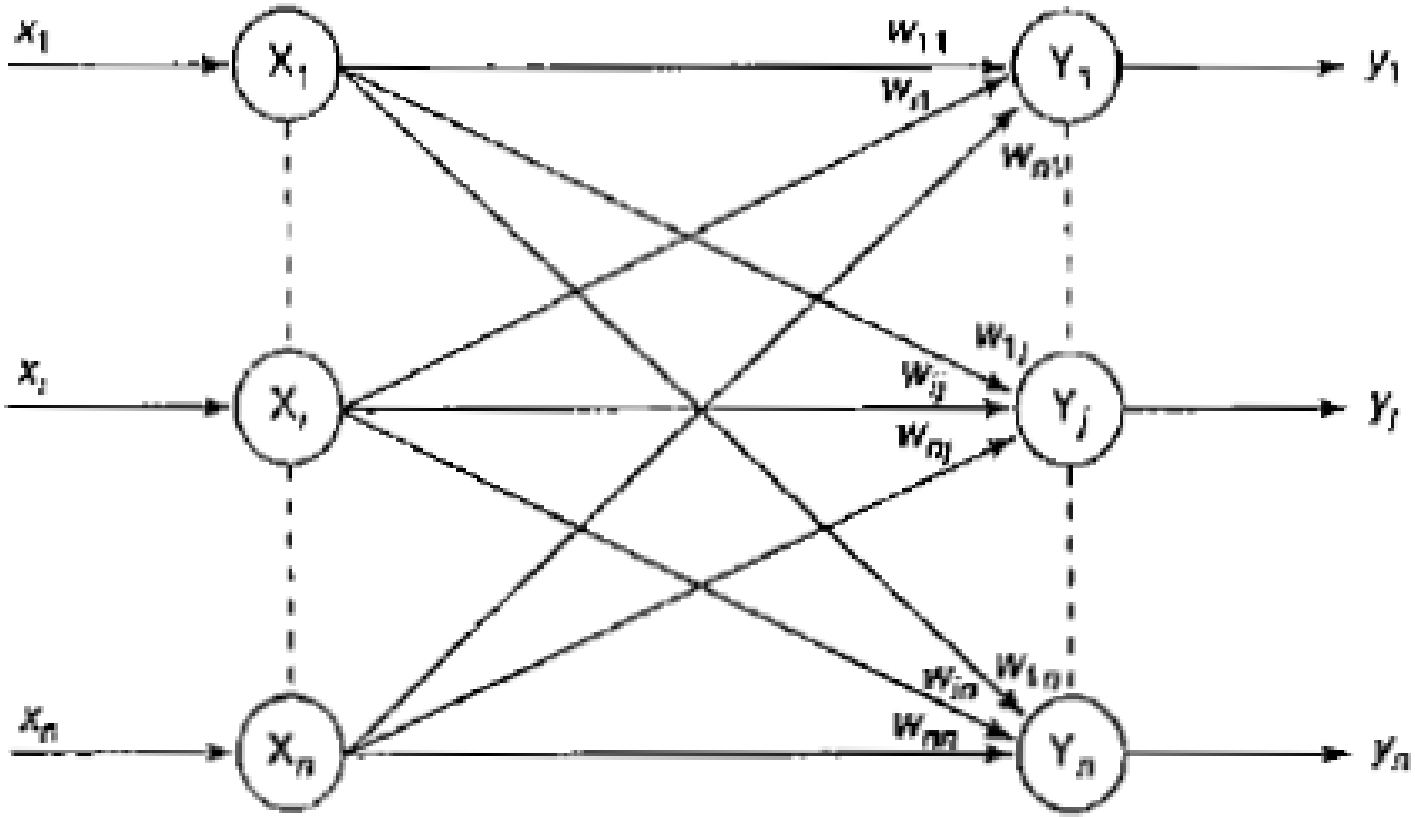
and

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

# Auto-associative Memory Network

- In the case of an auto-associative neural net, the training input and the 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.
- The weights on the diagonal can be set to zero. This can be called as auto associative net with no self connection.

# AUTO ASSOCIATIVE MEMORY NETWORK ARCHITECTURE




The input vector has n inputs and output vector has n outputs. The input and output are connected through weighted connections.

# *Training Algorithm*

- Initialize weights to 0
  - $w_{ij}$=0 ($i$= *1 to n*, $j$=*1 to n*)
- For each of the vector that has to be stored
- Activate the input layers for training input.
  - $x_i$=$s_i$(*for i=1 to n*)
- Activate the output layers for target output.
  - $y_i$=$s_i$(*for j=1 to n*)
- Adjust  weight .
  - $w_{ij}(new)=w_{ij}(old)+x_i y_j$
  - Weight can also be found by

$$W = \sum_{p=1}^{\mu} sT(\rho)s(\rho)$$

## *Testing Algorithm*

- An auto associative network can be used to determine whether the given vector is a 'known' or 'unknown vector'.
- A net is said to recognize a "known" vector if the net produces a pattern of activation on the output which is same as one stored.

*Testing procedure is as follows:*

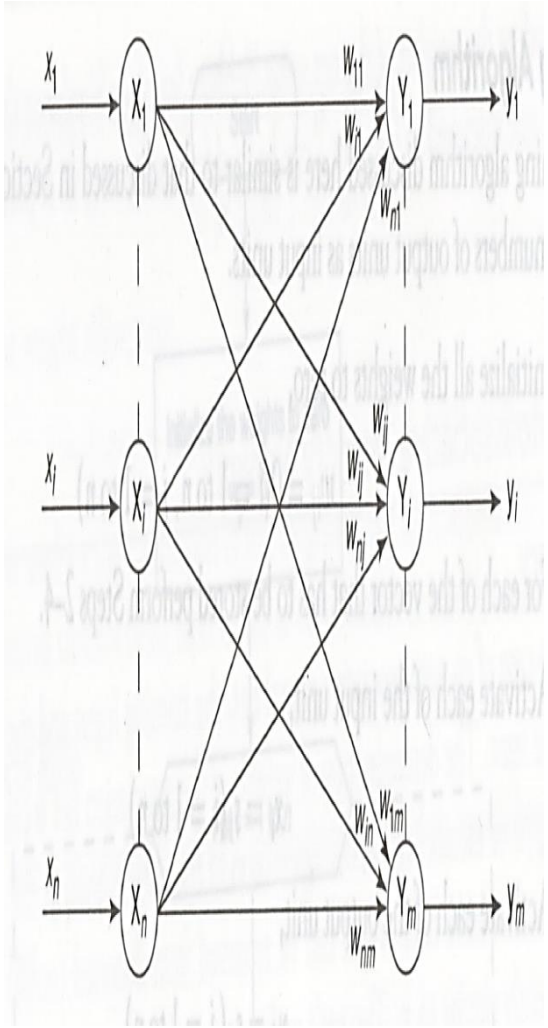
- Step 0: Set weights obtained from Hebb's rule
- Step 1: For each testing input vector perform steps 2 to 4
- Step 2: Activation of inputs is equal to input vector.
- Step 3: calculate the net input for each output unit j=1 to n:

$$Yinj = \sum_{i=1}^{n} XiWij$$

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

$$yi = f(yinj) = \begin{cases} 1 & if\ yinj > 0 \\ -1 & if\ yinj \leq 0 \end{cases}$$

# Hetero associative memory network



# HETERO ASSOCIATIVE MEMORY NETWORK

- The training input and target output vectors are different.
- Input has 'n' units and output has 'm' units and there is a weighted interconnection between input and output.

## *Testing Algorithm*

- Initialize the weights from the training algorithm.
- for each input vector presented.
- Set the activation inputs equal to current input vector

$$Yinj = \sum_{i=1}^{n} XiWij$$

- Determine the activation of the output units

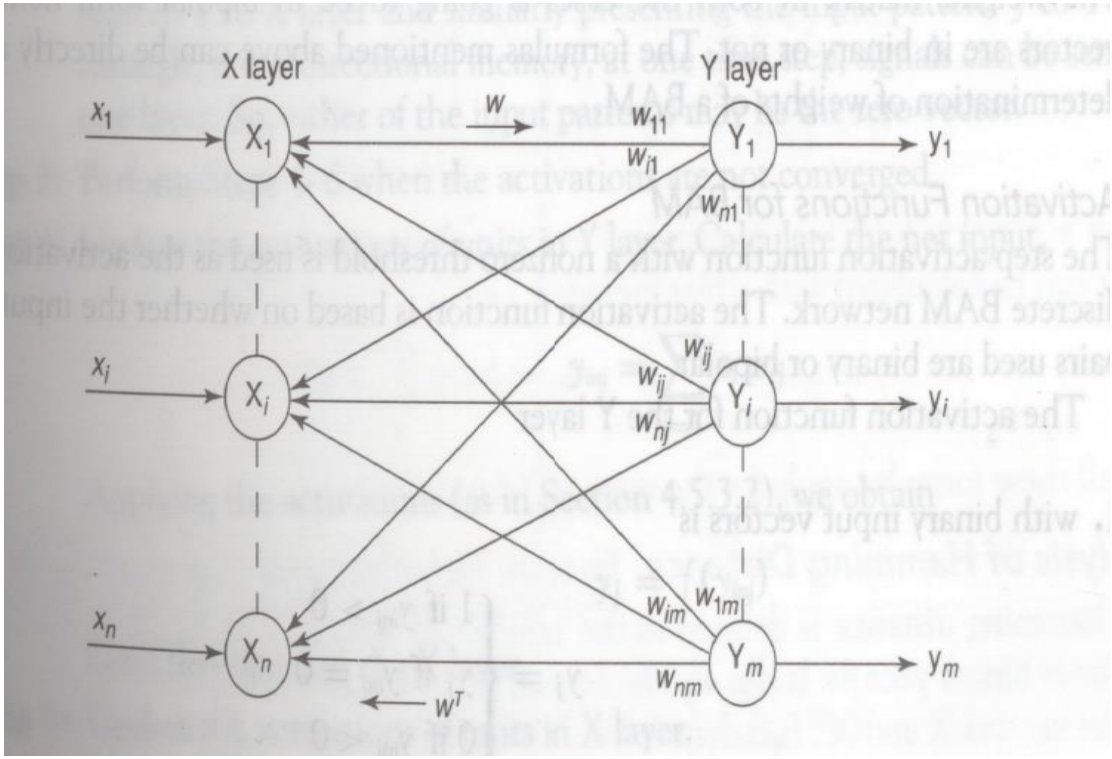
$$yj = \begin{Bmatrix} 1 \;\; if\; yinj > 0 \\ -1\; if\; yinj < 0 \\ 0\; if\; yinj = 0 \end{Bmatrix}$$

# HETERO ASSOCIATIVE MEMORY NETWORK

- The output vector y is obtained gives the pattern associated with the input vector x.
- If the responses are binary then the activation function will be as

$$yj = \begin{cases} 1 & if\ yinj > 0 \\ 0 & if\ yinj < 0 \end{cases}$$

# BIDIRECTIONAL ASSOCIATIVE MEMORY(BAM)




Weights are bidirectional

X layer has 'n' input units

Y layer has 'm' output units.

Weight matrix from X to Y is W and from Y to X is $W^T$.

# BIDIRECTIONAL ASSOCIATIVE MEMORY(BAM)

- The BAM was developed by Kosko in the ear 1988.
- The BAM network performs forward and backward associative searches for stored responses .
- The BAM is a recurrent hetero-associativve pattern-marching network that encodes binary or bipolar patterns using Hebbian learning rule.
- It associates patterns, say from set A to patterns from set B and vice versa is also performed.
- BAM neural nets can respond to input from either layers ( input layer and output layer).
- Two types : Discrete BAM, Continuous BAM

# DISCRETE BAM

- Here weight is found to be the sum of outer product of bipolar form.
- Activation function is defined with nonzero threshold.
- ***Determination of weights***
- Input vectors is denoted by s(ρ) and output vector as t(ρ).Then the weight matrix is denoted by
- $s(\rho)=(S_1(\rho),...S_i(\rho),..,S_n(\rho))$
- Output = $t(\rho) = (t_1(\rho),...t_j(\rho),..,t_m(\rho))$
- Weight matrix is determined using the Hebb Rule.
- If the input vectors is binary, then weight matrix

$$W=\{w_{ij}\}=\sum_{\rho=1}^{p}[2si(\rho)-1][2tj(\rho)-1]$$

- If the input vectors are bipolar, the weight matrix

$$W=\{w_{ij}\}=\sum_{\rho=1}^{p}[si(\rho)][tj(\rho)$$

## *Activation Function for BAM*

- The Activation Function is based on whether the input target vector pairs used are binary or bipolar.
- The Activation function for Y layer with binary input vectors is

$$yj = \begin{cases} 1 & if\ yinj > 0 \\ yj & if\ yinj = 0 \\ 0 & if\ yinj < 0 \end{cases}$$

- with bipolar input vector is

$$yj = \begin{cases} 1 & if\ yinj > \theta j \\ yj & if\ yinj = \theta j \\ -1 & if\ yinj < \theta j \end{cases}$$

- The activation function for the X layer with binary input vector is

$$xi = \begin{cases} 1 & if\ xini > 0 \\ x & if\ xini = 0 \\ -1 & if\ xini < 0 \end{cases}$$

- With bipolar input vector is

$$xi = \begin{cases} 1 & if\ xini > \theta i \\ x & if\ xini = \theta j \\ -1 & if\ xini < \theta j \end{cases}$$

- If threshold value is equal to the net input, then the previous output value is calculated is left as the activation of that unit. Signals are sent only from one layer to the other and not in both directions.

# TESTING ALGORITHM FOR DISCRETE BAM

- Test the noisy patterns entering into the network.
- Testing algorithm for the net is as follows:
- Step 0: Initialize the weights to store ρ vectors. Also initialize all the activations to zero.
- Step 1:Perform steps 2-6 for each testing input.
- Step 2: Set the Activation of X layer to current input patterns, presenting the input $x$ to X layer and presenting the input pattern $y$ to Y layer. It is bidirectional memory.
- Step 3: Perform steps 4-6 when the activations are not converged.
- Step 4: Update the activation of units in Y layer. Calculate the net input.

$$yinj = \sum_{i=1}^{n} xiwij$$

- Applying the Activation, we get yj=f(yinj).
- Send this signal to X layer.

- Step 5: Update the activation of units in X layer.
  - Calculate the net input

$$xini = \sum_{j=1}^{m} yjwij$$

  - Applying the activation over the net input
    - xi=f(xini)
    - Send this signal to 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, then stop, else continue.

# CONTINUOUS BAM

- It uses logistic Sigmoid function as the activation functions for all units.
- It may be binary sigmoid or bipolar sigmoid.
- Bipolar sigmoid function with high gain, converge to vector state and acts like DBAM.
- If the input vectors are binary, s(ρ), t(ρ), the weights are determined using the formula

$$w_{ij} = \sum_{\rho=1}^{p} [2si(\rho) - 1][2tj(\rho) - 1]$$

- If a binary logistic function is used, then the activation function is

$$f(yinj) = \frac{1}{1 + e^{-yinj}}$$

- If the activation function is bipolar logistic function then,

$$f(yinj) = \frac{2}{1 + e^{-yinj}} - 1 = \frac{1 - e^{-yinj}}{1 + e^{-yinj}}$$

- Net input calculated with bias is included $yinj = bj + \sum_{1} xiwij$

# ANALYSIS OF HAMMING DISTANCE, ENERGY FUNCTION AND STORAGE CAPACITY

- The hamming distance is defined as the number of mismatched components of two given bipolar or binary vectors.
- It is denoted as H[x , x'].
- The average hamming distance between the vectors is (1/n)H[x , x'], where 'n' is the number of components in each vector.
- Consider the vectors, x = [1 0 1 0 1 1 0], x' = [1 1 1 1 0 0 1]
- Hamming distance between these two given vectors is equal to 5. the average hamming distance is 5/7

- Stability analysis of BAM is based on the definition of Lyapunov function (energy function).
- Consider that there are p vectors association pairs to be stored in a BAM; $\{(x^1, y^1), (x^2, y^2) \ldots\ldots\ldots (x^p, y^p)\}$ .
- A Lyapunov function must be always bounded and decreasing.
- A BAM can be said to bidirectionally stable if the state converges to a stable point, i.e. $y^k \rightarrow y^{k+1} \rightarrow y^{k+2}$ *and* $y^{k+2} = y^k$
- This gives the minimum of the energy function.
- The energy function or Lyapunov function of a BAM is defined as

$$E_f(x, y) = -0.5\, x^T W^T y - 0.5\, y^T W x = -y^T W x$$

- The change in energy due to the single bit changes in both vectors y and x given as $\Delta yi$ and $\Delta xj$ can be found as
- $\Delta E_f(y_i) = \nabla_y E \Delta y_i = -Wx\Delta y_i = -\left(\sum_{j=1}^{m} xjWij\right) * \Delta y_i, \quad i = 1 \text{ to } n$
- $\Delta E_f(x_j) = \nabla_x E \Delta x_j = -W^T y\Delta x_j = -\left(\sum_{i=1}^{n} yiWij\right) * \Delta x_j, \quad j = 1 \text{ to } m$

Where $\Delta y_i$ and $\Delta x_j$ are given as

$$\Delta x_j = \begin{cases} 2 & \text{if } \sum_{i=1}^{n} yiWij > 0 \\ 0 & \text{if } \sum_{i=1}^{n} yiWij = 0 \\ -2 & \text{if } \sum_{i=1}^{n} yiWij < 0 \end{cases}$$

$$\Delta y_i = \begin{cases} 2 & \text{if } \sum_{j=1}^{m} xjWij > 0 \\ 0 & \text{if } \sum_{j=1}^{m} xjWij = 0 \\ -2 & \text{if } \sum_{j=1}^{m} xjWij < 0 \end{cases}$$

Here the energy function is bounded below by

$$E_f(x, y) \geq -\sum_{i=1}^{n} \sum_{j=1}^{m} |wij|$$

So the discrete BAM will converge to a stable state

- The memory capacity or the storage capacity of BAM may be given as

$$min\ (m,\ n)\ or\ \sqrt{\min(m\ ,n)}$$

- Where ‘n’ is the number of units in X layer and ‘m’ is the number of units in Y layer.

# Testing algorithm for Discrete Hopfield Net

*Step 0. Initialize weights to store patterns. (Use Hebb rule.)*

*Step 1.While activations of the net are not converged, do Steps 2-8.*

*Step 2. For each input vector x, do Steps 3-7.*

*Step 3. Set initial activations of net equal to the external input vector x:*

$$y_i = x_i \quad , \quad (i=1,2\ldots n)$$

*Step 4. Do Steps **5-7** for each unit* $Y_i$

*(Units should be updated in random order.)*

*Step 5. Compute net input:*

$$y\_in_i = x_i + \sum_j y_j w_{ji}$$

# Testing algorithm for Discrete Hopfield Net

*Step 6. Determine activation (output signal):*

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

*Where* $\theta_i$ *is the threshold and is normally chosen as zero.*

*Step 7. Feed back the obtained output* $y_i$ *to all other units. This updates the activation vector.)*

*Step 8. Test for convergence.*

# APPLICATIONS

- A binary Hopfield net can be used to determine whether an input vector is a "known" or an "unknown" vector.
- The net recognizes a "known" vector by producing a pattern of activation on the units of the net that is the same as the vector stored in the net.
- If the input vector is an "unknown" vector, the activation vectors produced as the net iterates will converge to an activation vector that is not one of the stored patterns.

# EXAMPLE:

- Consider an Example in which the vector (1, 1, 1,0) (or its bipolar equivalent (1, 1, 1, - 1)) was stored in a net. The binary input vector corresponding to the input vector used (with mistakes in the first and second components) is (0, 0, 1, 0). Although the Hopfield net uses binary vectors, the weight matrix is bipolar. The units update their activations in a random order. For this example the update order is

$$y_1 y_4 y_3 y_2$$

- $W = \sum s^T(p)t(p) = \begin{pmatrix} 1 \\ 1 \\ 1 \\ -1 \end{pmatrix} [1\ 1\ 1\ -1]$

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

- The weight matrix with no self connection is:

$$\mathbf{W} = \begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}.$$

Step 0. Initialize weights to store patterns:

$$\mathbf{W} = \begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}.$$

Step 1. The input vector is x = [0, 0, 1, 0].

Step 2. for this vector y = [0, 0, 1, 0].

Step 3. Choose unit $Y_i$ to update its activation:

step 4.

$$y\_in_1 = x_1 + \sum_j y_j w_{j1} \qquad = 0+[0\ 0\ 1\ 0] \begin{pmatrix} 0 \\ 1 \\ 1 \\ -1 \end{pmatrix} = 0+1 = 1$$

step 5. $y\_in_1 > 0 \rightarrow y_1 = 1$

step 6. y=(1,0,1,0) $\longrightarrow$ No Convergence

Step 3. Choose unit $y_4$ to update its activation:

step 4. $y-in_4 = x_4 + \sum_j y_j w_{j4} = 0 + (-2)$

step 5. $y - in_4 < 0 \rightarrow y_4 = 0$

step 6. y=(1,0,1,0). No Convergence

step 3. Choose unit to update its activation:

step 4. $y_3$ $y - in_3 > 0 \rightarrow y_3 = 1$

step 5. $y-in_3 = x_3 + \sum_j y_j w_{j3} = 1+1.$

Step 6. y=(1,0,1,0). No Convergence

step 3. Choose unit $y_2$ to update its activation:

step 4. $$y\_in_2 = x_2 + \sum_j y_j w_{j2} = 0 + 2$$

step 5. $$y\_in_2 > 0 = y_2 = 1$$

step 6. y=(1,1,1,0) Converges with vector x.

Step 7. Test for convergence: thus the output y has converged with vector x in this iteration itself. But, one more iteration can be done to check whether further activations are there or not.

- Weight matrix w will be same and input vector x = [1 1 1 0].
- For same sequence of updation of weights. i.e. y1, y4, y3, y2

# Analysis

***Energy Function.***

- An energy function is a function that is bounded below and is a non increasing function of the state of the system.
- For a neural net, the state of the system is the vector of activations of the units.
- Thus, if an energy function can be found for an iterative neural net, the net will converge to a stable set of activations.

$$E = -.5 \sum_{i \neq j} \sum_{j} y_i y_j w_{ij} - \sum_{i} x_i y_i + \sum_{i} \theta_i y_i.$$

- Energy function for the discrete Hopfield net is given by,

- If the network is stable, then the above energy function decreases whenever the state of any node changes.
- Assuming that node I has changed its state from $yi^{(k)}$ to $yi^{(k+1)}$. i.e. the output has changed from +1 to -1 or -1 to +1, the energy change $\Delta E_f$ is given as

$$\Delta Ef = Ef(yj^{(k+1)}) - Ef(yi^{(k)})$$

$$\Delta Ef = -\left(\sum_{j=1}^{n} yj^{(k)}\ wij + xi - \theta i\right)(yi^{(k+1)} - yi^{(k)})$$

- If the activation of the net changes by an amount $\Delta yi$, the energy changes by an amount,

$$\Delta E = -\left[\sum_j y_j w_{ij} + x_i - \theta_i\right]\Delta y_i.$$

- consider the two cases in which a change will occur in the activation of neuron $\Delta y_i$ $y_i$
  - If $y_i$ is positive, it will change to zero if,

$$x_i + \sum_j y_j w_{ji} < \theta_i$$

This gives a negative change for $y_i$ In this case, $\Delta E < 0.$

  - If $y_i$ is zero, it will change to positive if,

This gives a positive change for $y_i$ In this case, $\Delta E < 0.$

$$x_i + \sum_j y_j w_{ji} > \theta_i$$

- Hence Δyi is positive only if net input is positive and Δyi is negative only if net input is negative.
- Therefore energy cannot increase in any manner.
- As a result, because energy is bounded, the net must reach a stable state equilibrium, such that the energy does not change with further iteration.
- From this it can be concluded that the energy change depends mainly on the change in activation of one unit and on the symmetry of weight matrix with zeros existing on the diagonal.

## *STORAGE CAPACITY.*

- Hopfield found experimentally that the number of binary patterns that can be stored and recalled in a net with reasonable accuracy, is given approximately by,

$$P \approx 0.15n,$$

n = The number of neurons in the net.

# UNSUPERVISED LEARNING

- No help from the outside.
- No training data, no information available on the desired output.
- Learning by doing.
- Used to pick out structure in the input:
  - Clustering,
  - Reduction of dimensionality → compression.
- Example: Kohonen's Learning Law.

# FEW UNSUPERVISED LEARNING NETWORKS

There exists several networks under this category, such as

- Max Net,
- Mexican Hat,
- Kohonen Self-organizing Feature Maps,
- Learning Vector Quantization,
- Counterpropagation Networks,
- Hamming Network,
- Adaptive Resonance Theory.

# COMPETITIVE LEARNING

- Output units compete, so that eventually only one neuron (the one with the most input) is active in response to each output pattern.
- The total weight from the input layer to each output neuron is limited. If some connections are strengthened, others must be weakened.
- A consequence is that the winner is the output neuron whose weights best match the activation pattern.

# SELF-ORGANIZATION

- Network Organization is fundamental to the brain
  - Functional structure.
  - Layered structure.
  - Both parallel processing and serial processing require organization of the brain.

# SELF-ORGANIZING FEATURE MAP

Our brain is dominated by the cerebral cortex, a very complex structure of billions of neurons and hundreds of billions of synapses. The cortex includes areas that are responsible for different human activities (motor, visual, auditory, etc.) and associated with different sensory inputs. One can say that each sensory input is mapped into a corresponding area of the cerebral cortex. ***The cortex is a self-organizing computational map in the human brain.***

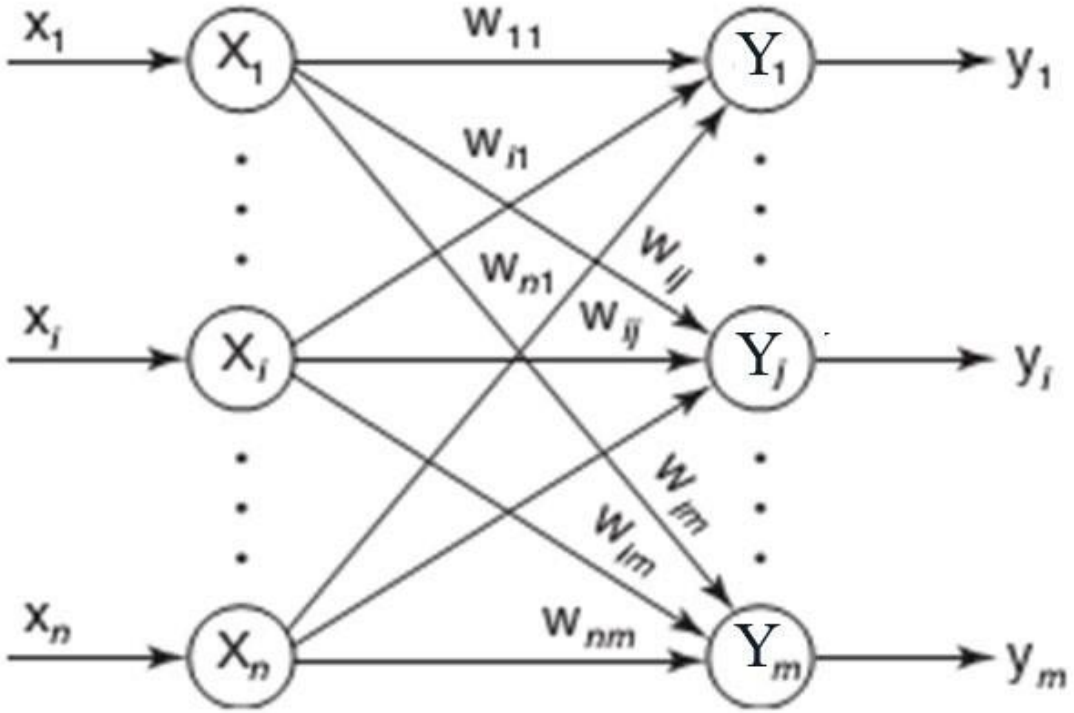
# SELF-ORGANIZING NETWORKS

- Discover significant patterns or features in the input data.
- Discovery is done without a teacher.
- Synaptic weights are changed according to local rules.
- The changes affect a neuron's immediate environment until a final configuration develops.

# KOHONEN SELF-ORGANIZING FEATURE MAP (KSOFM)

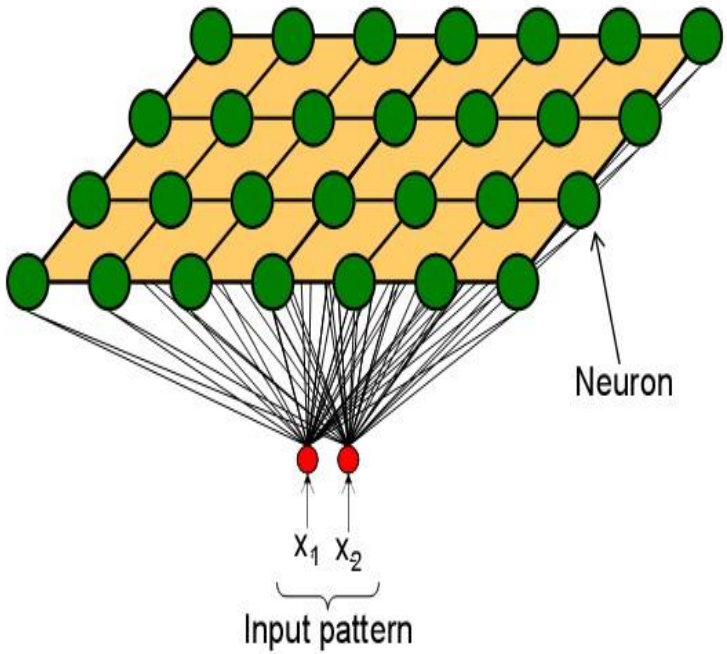
- The Kohonen model provides a topological mapping.
- It places a fixed number of input patterns from the input layer into a higher dimensional output or Kohonen layer.
- Training in the Kohonen network begins with the winner's neighborhood of a fairly large size. Then, as training proceeds, the neighborhood size gradually decreases.
- Kohonen SOMs result from the synergy of three basic processes
  - Competition,
  - Cooperation,
  - Adaptation.

# ARCHITECTURE OF KSOFM



# COMPETITION OF KSOFM

- Each neuron in an SOM is assigned a weight vector with the same dimensionality N as the input space.
- Any given input pattern is compared to the weight vector of each neuron and the closest neuron is declared the winner.
- The Euclidean norm is commonly used to measure distance.



# CO-OPERATION OF KSOFM

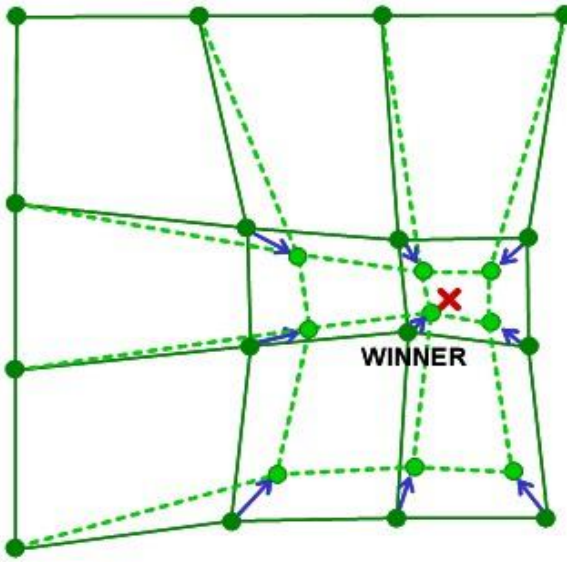
- The activation of the winning neuron is spread to neurons in its immediate neighborhood.
  - This allows topologically close neurons to become sensitive to similar patterns.
- The winner's neighborhood is determined on the lattice topology.
  - Distance in the lattice is a function of the number of lateral connections to the winner.
- The size of the neighborhood is initially large, but shrinks over time.
  - An initially large neighborhood promotes a topology-preserving mapping.
  - Smaller neighborhoods allow neurons to specialize in the latter stages of training.

# ADAPTATION OF KSOFM

During training, the winner neuron and its topological neighbors are adapted to make their weight vectors more similar to the input pattern that caused the activation.

Neurons that are closer to the winner will adapt more heavily than neurons that are further away.

The magnitude of the adaptation is controlled with a learning rate, which decays over time to ensure convergence of the SOM.



Step 0:
- Initialize the weights $w_{ij}$: Random values may be assumed. They can be chosen as the same range of values as the components of the input vector. If information related to distribution of clusters is known, the initial weights can be taken to reflect that prior knowledge.
- Set topological neighborhood parameters: As clustering progresses, the radius of the neighborhood decreases.
- Initialize the learning rate $\alpha$: It should be a slowly decreasing function of time.

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, i.e., for each $j = 1$ to $m$,

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

Step 4: Find the winning unit index J, so that D(J) is minimum. (In Steps 3 and 4, dot product method can also be used to find the winner, which is basically the calculation of net input, and the winner will be the one with the largest dot product.)

Step 5: For all units $j$ within a specific neighborhood of J and for all $i$, calculate the new weights:

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

or

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

Step 6: Update the learning rate $\alpha$ using the formula $\alpha(t + 1) = 0.5\alpha(t)$.

Step 7: Reduce radius of topological neighborhood at specified time intervals.

Step 8: Test for stopping condition of the network.

# EXAMPLE OF KSOFM

Suppose, for instance, that the 2-dimensional input vector **X** is presented to the three-neuron Kohonen network,

$$\mathbf{X} = \begin{bmatrix} 0.52 \\ 0.12 \end{bmatrix}$$

The initial weight vectors, $\mathbf{W}_j$, are given by

$$\mathbf{W}_1 = \begin{bmatrix} 0.27 \\ 0.81 \end{bmatrix} \qquad \mathbf{W}_2 = \begin{bmatrix} 0.42 \\ 0.70 \end{bmatrix} \qquad \mathbf{W}_3 = \begin{bmatrix} 0.43 \\ 0.21 \end{bmatrix}$$

Find the winning neuron using the Euclidean distance:

$$d_1 = \sqrt{(x_1 - w_{11})^2 + (x_2 - w_{21})^2} = \sqrt{(0.52 - 0.27)^2 + (0.12 - 0.81)^2} = 0.73$$

$$d_2 = \sqrt{(x_1 - w_{12})^2 + (x_2 - w_{22})^2} = \sqrt{(0.52 - 0.42)^2 + (0.12 - 0.70)^2} = 0.59$$

$$d_3 = \sqrt{(x_1 - w_{13})^2 + (x_2 - w_{23})^2} = \sqrt{(0.52 - 0.43)^2 + (0.12 - 0.21)^2} = 0.13$$

Neuron 3 is the winner and its weight vector $W_3$ is updated according to the competitive learning rule:

$$\Delta w_{13} = \alpha (x_1 - w_{13}) = 0.1 (0.52 - 0.43) = 0.01$$

$$\Delta w_{23} = \alpha (x_2 - w_{23}) = 0.1 (0.12 - 0.21) = -0.01$$

The updated weight vector W3 at iteration (p+1) is determined as:

$$\mathbf{W}_3(p+1) = \mathbf{W}_3(p) + \Delta \mathbf{W}_3(p) = \begin{bmatrix} 0.43 \\ 0.21 \end{bmatrix} + \begin{bmatrix} 0.01 \\ -0.01 \end{bmatrix} = \begin{bmatrix} 0.44 \\ 0.20 \end{bmatrix}$$

The weight vector W3 of the winning neuron 3 becomes closer to the input vector X with each iteration.

# ART-Introduction

- ART stands for "Adaptive Resonance Theory", invented by Stephen Grossberg in 1976.
- ART represents a family of neural networks.
- The basic ART System is an unsupervised learning model.
- The term "resonance" refers to resonant state of a neural network in which a category prototype vector matches close enough to the current input vector.
- ART matching leads to this resonant state, which permits learning. The network learns only in its resonant state.

# Key Innovation

- The key innovation of ART is the use of "expectations."
- As each input is presented to the network, it is compared with the prototype vector that is most closely matches.
- If the match between the prototype and the input vector is NOT adequate, a new prototype is selected. In this way, previous learned memories (prototypes) are not eroded by new learning.

# Stability Plasticity Dilemma

- Real world is faced with situation where data is continuously changing.
- In this situation every learning system faces stability-plasticity dilemma.
- **Stability:** System behaviour doesn't change after irrelevant events
- **Plasticity:** System adapts its behaviour according to significant events
- **Dilemma:**

✓ How to achieve stability without rigidity and plasticity without chaos?

✓ Ongoing learning capability

✓ Preservation of learned knowledge

- The plasticity-stability dilemma poses few questions :

✓ How can we continue to quickly learn new things about the environment and yet not forgetting what we have already learned?

✓ How can a learning system remain plastic (adaptive) in response to significant input yet stable in response to irrelevant input?

✓ How can a neural network can remain plastic enough to learn new patterns and yet be able to maintain the stability of the already learned patterns?

- How does the system know to switch between its plastic and stable modes.

- ART networks tackle the stability-plasticity dilemma.

- The ART network and algorithm maintain the plasticity required to earn new patterns while preventing the modifications of patterns that have been learned previously.

- Solves Stability – Plasticity Dilemma.

  Forms new memories or incorporates new information based on a predefined vigilance parameter.

# ART Network

- ART networks consist of an input layer and an output layer.
- Bottom-up weights are used to determine output-layer candidates that may best match the current input.
- Top-down weights represent the “prototype” for the cluster defined by each output neuron.
- A close match between input and prototype is necessary for categorizing the input.
- Finding this match can require multiple signal exchanges between the two layers in both directions until “resonance” is established or a new neuron is added.

# Basic ART network Architecture

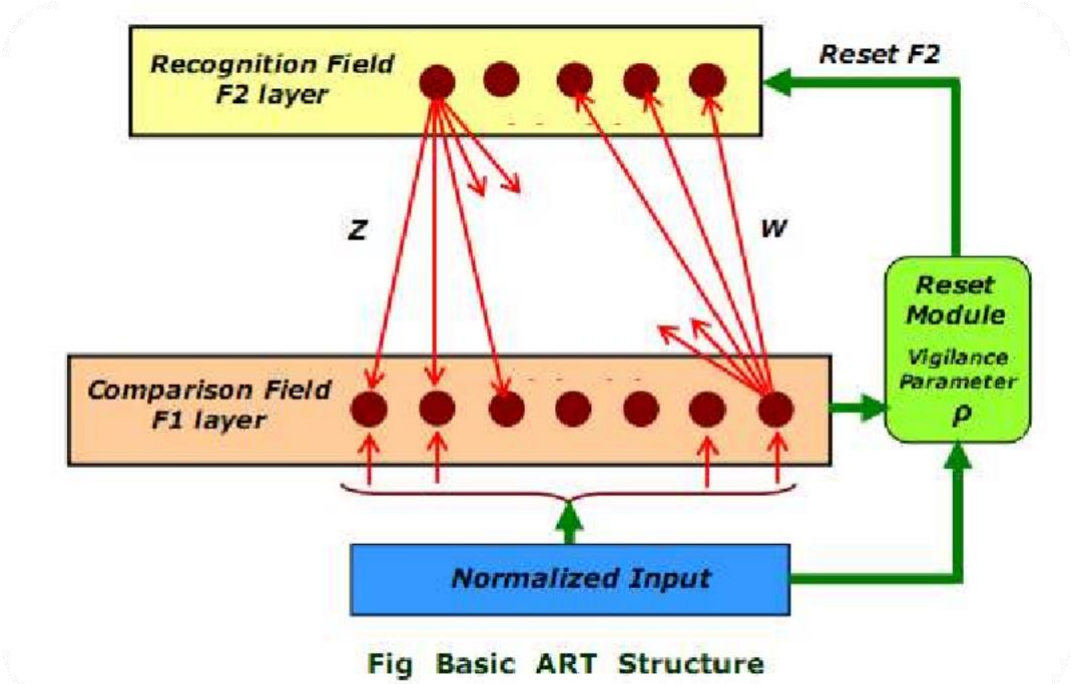



Fig Basic ART Structure

The basic ART system is unsupervised learning model.

It typically consists of

1. a comparison field
2. a recognition field composed of neurons,
3. a vigilance parameter, and
4. a reset module

- Comparison field:

  - The comparison field takes an input vector and transfers it to its best match in the recognition field.
  - Its best match is the single neuron whose set of weights most closely matches the input vector.

- Recognition field:

  - Each recognition field neuron, outputs a negative signal proportional to that neuron's quality of match to the input vector to each of the other recognition field neurons and inhibits their output accordingly.

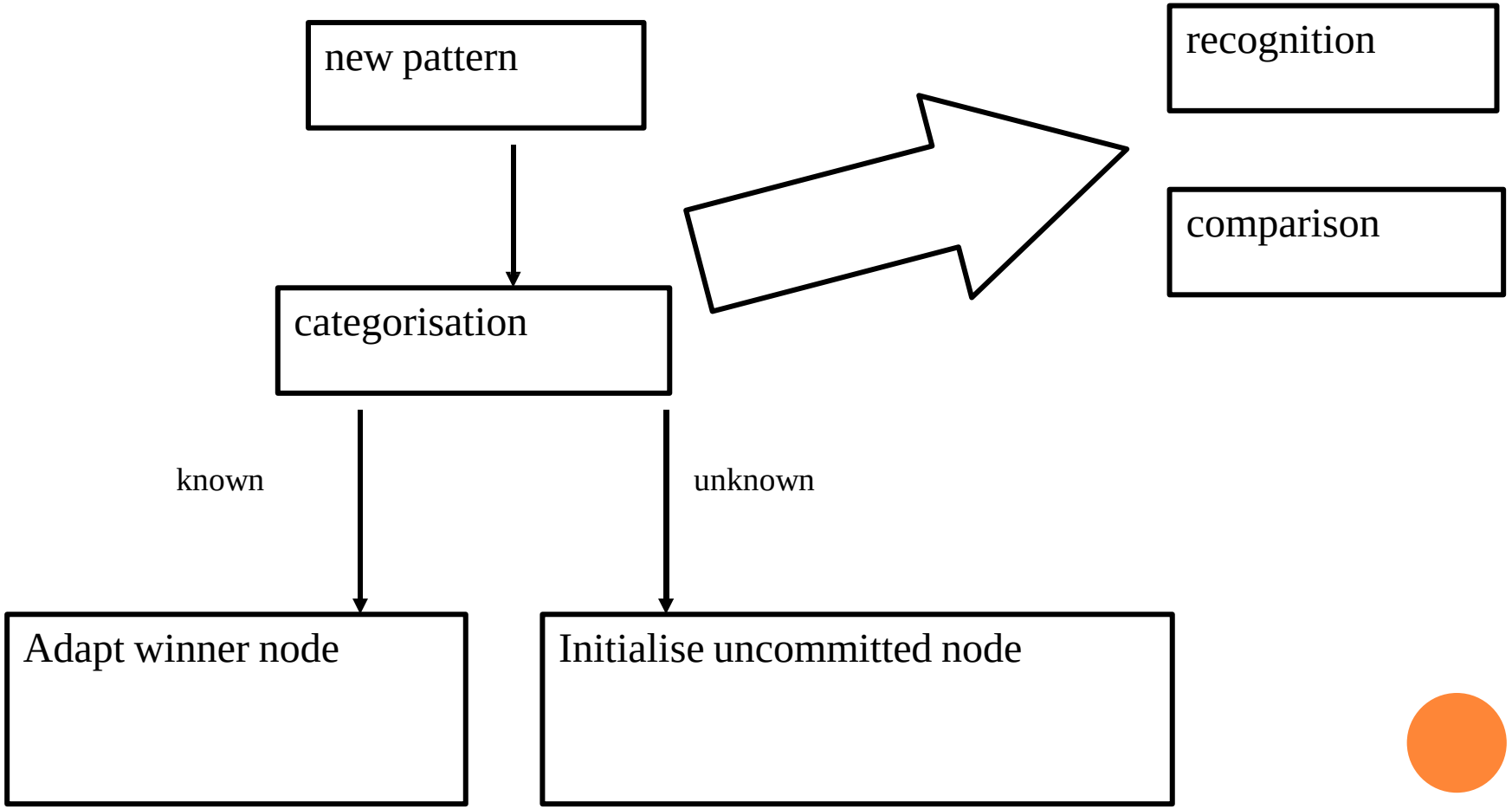
- Vigilance Parameter:

  - After the input vector is classified, a reset module compares the strength of the recognition match to a vigilance parameter.
  - The vigilance parameter has considerable influence on the system.

- Reset Module:

  - The reset module compares the strength of the recognition match to the vigilance parameter.
  - If the vigilance threshold is met, then training commences

# ART ALGORITHM



- Incoming pattern matched with stored cluster templates.
- If close enough to stored template joins best matching cluster, weights adapted.
- If not, a new cluster is initialised with pattern as template.

# ART Types

- Unsupervised ARTs
  - ART1
  - ART2
  - ART3
  - Fuzzy ART
- Supervised ARTs
  - ARTMAP
  - Fuzzy ARTMAP

- ART 1 is the simplest variety of ART networks, accepting only binary inputs.
- ART2 extends network capabilities to support continuous inputs.
- ART3 is refinement of both modal.
- Fuzzy ART implements fuzzy logic into ART's pattern recognition
- ARTMAP also known as Predictive ART, combines two slightly modified ART-1 or ART-2 units into a supervised learning structure .
- Fuzzy ARTMAP is merely ARTMAP using fuzzy ART units, resulting in a corresponding increase in efficacy.

# ART Operational Phases

1. Initialization: Before starting all weight vectors B,T and vigilance parameter must be set to initial value.

2. Recognition: Input vector is applied to initiate recognition phase.

3. Comparison: At this point, feedback signal from recognition layer cause G1 to go to Zero.

4. Search : when similarity of wining neuron is greater than vigilance than no search is required otherwise stored pattern must be searched.

# Application of ART

- Mobile robot control
- Facial recognition
- Land cover classification
- Target recognition
- Medical diagnosis
- Signature verification