UNIT II

ASSOCIATIVE MEMORY AND UNSUPERVISED LEARNING NETWORKS

Training Algorithms for Pattern Association-Autoassociative Memory Network-Heteroassociative Memory Network-Bidirectional Associative Memory (BAM)-Hopfield Networks-Iterative Autoassociative Memory Networks-Temporal Associative Memory Network-Fixed Weight Competitive Nets-Kohonen Self-Organizing Feature Maps-Learning Vector Quantization-Counter propagation Networks-Adaptive Resonance Theory Network.

Training Algorithms for Pattern Association

These kinds of neural networks work on the basis of pattern association, which means they can store different patterns and at the time of giving an output they can produce one of the stored patterns by matching them with the given input pattern. These types of memories are also called Content-Addressable Memory. Associative memory makes a parallel search with the stored patterns as data files.

Following are the two types of associative memories we can observe –

Auto Associative Memory

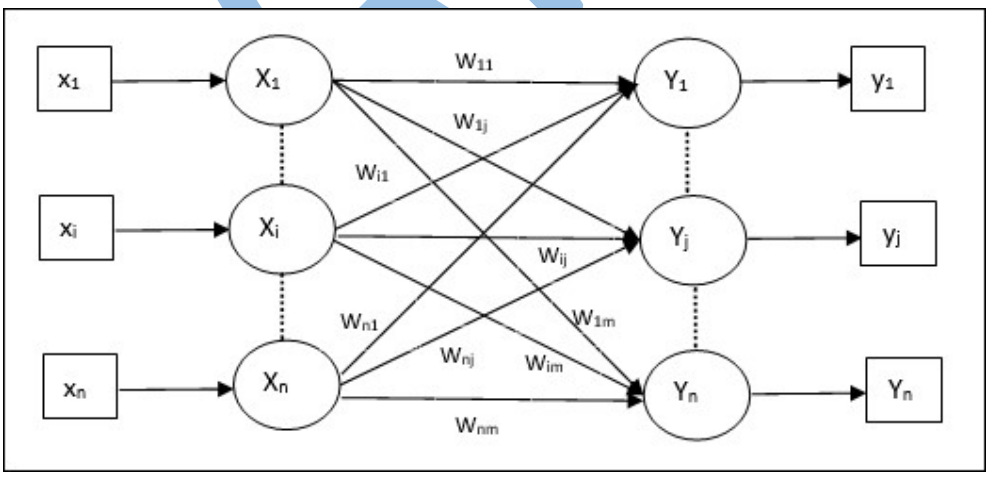
Hetero Associative memory

Auto Associative Memory

This is a single layer neural network in which the input training vector and the output target vectors are the same. The weights are determined so that the network stores a set of patterns.

Architecture

As shown in the following figure, the architecture of Auto Associative memory network has ‘n’ number of input training vectors and similar ‘n’ number of output target vectors.



APEC

Training Algorithm

For training, this network is using the Hebb or Delta learning rule.

Step 1 – Initialize all the weights to zero as $\mathbf{w_{ij}} = \mathbf{0}\ i = 1ton, j = 1ton$

Step 2 – Perform steps 3-4 for each input vector.

Step 3 – Activate each input unit as follows –

$$x_i = s_i\ (i = 1\ to\ n)$$

Step 4 – Activate each output unit as follows –

$$y_j = s_j\ (j = 1\ to\ n)$$

Step 5 – Adjust the weights as follows –

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

Testing Algorithm

Step 1 – Set the weights obtained during training for Hebb's rule.

Step 2 – Perform steps 3-5 for each input vector.

Step 3 – Set the activation of the input units equal to that of the input vector.

Step 4 – Calculate the net input to each output unit $\mathbf{j = 1\ to\ n}$

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

Step 5 – Apply the following activation function to calculate the output

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

or

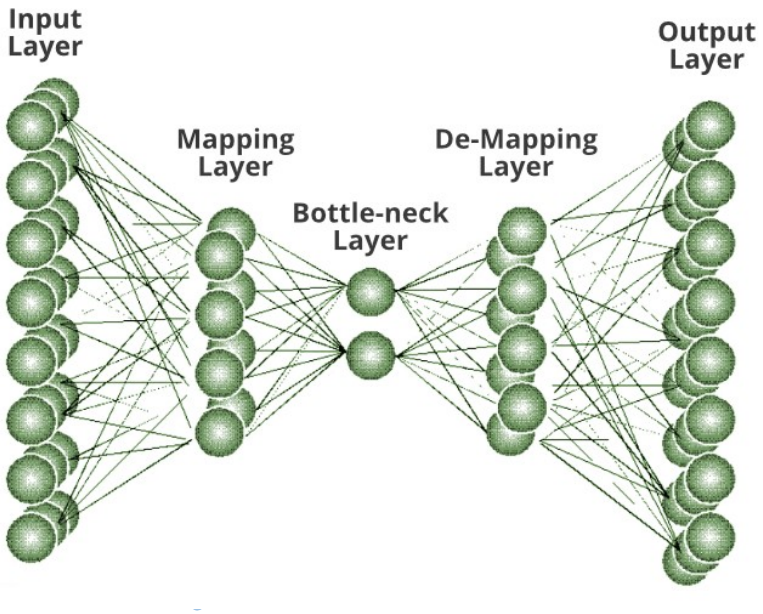
Auto-associative Neural Networks

Auto associative Neural networks are the types of neural networks whose input and output vectors are identical. These are special kinds of neural networks that are used to simulate and explore the associative process. Association in this architecture comes from the instruction of a set of simple processing elements called units which are connected through weighted connections.

In these networks, we performed training to store the vector either bipolar or binary. A stored vector can be retrieved from a distorted or noisy vector if the input is similar to it.

Architecture

AANN contains five-layer perceptron feed-forward network, that can be divided into two neural networks of 3 layers each connected in series (similar to autoencoder architecture). The network consists of an input layer followed by a hidden layer and bottleneck layer. This bottleneck layer is common between both the network and a key component of the network. It provides data compression to the input and topology with powerful feature extraction capabilities. The bottleneck layer is followed by a second non-linear hidden layer and the output layer of the second network.



Auto-Associative NN architecture

The first network compresses the information of the n-dimensional vector to smaller dimension vectors that contain a smaller number of characteristic variables and represent the whole process. The second network works opposite to the first and uses compressed information to regenerate the original n redundant measures.

Algorithm

We will be using Hebb Rule in the algorithm for setting weights because input and output vectors are perfectly correlated since the input and output both have the same number of output units and input units.

Hebb Rule:

when A and B are positively correlated, then increase the strength of the connection between them.

when A and B are negatively correlated, then decrease the strength of the connection between them.

In practice, we use following formula to set the weights:

where, W = weighted matrix

T= Learning Rate

S(p) : p-distinct n-dimensional prototype patterns

Training Algorithm

Initialize all weights for $i= 1,2,3 \ldots n$ and $j= 1,2,3 \ldots n$ such that: $\boldsymbol{w_{ij}=0}$.

For each vector to be stored repeat the following steps:

Set activation for each input unit $i= 1$ to n: $x_i = s_i$.

Set activation for each output unit j= 1 to n: $y_j = s_j$.

Update the weights for $i= 1,2,3 \ldots n$ and $j= 1,2,3 \ldots n$ such that : $\boldsymbol{w_{ij}\ (new) = w_{ij}\ (old) + x_i y_j}$

Testing / Inference Algorithm:

For testing whether the input is 'known' and 'unknown' to the model, we need to perform the following steps:

Take the weights that were generated during the training phase using Hebb's rule.

For each input vector perform the following steps:

Set activation in the input units equal to input vectors.

Set activation in output units for $j= 1,2,3 \ldots n$:

Apply activation function for j= 1, 2, 3 … n:

AANN recognizes the input vector to be *known* if the output unit after activation generated same pattern as one stored in it.

Storage Capacity

One of the important features of AANN is the number of patterns that can be stored before the network begins to forget.

The number of vectors that can be stored in the network is called the capacity of the network.

The capacity of the vectors depends upon the relationship between them i.e more vectors can be stored if they are mutually orthogonal. Generally, n-1 mutually orthogonal vectors with n components can be stored.

Applications

Auto-associative Neural Networks can be used in many fields:

Pattern Recognition

Bio-informatics

Voice Recognition

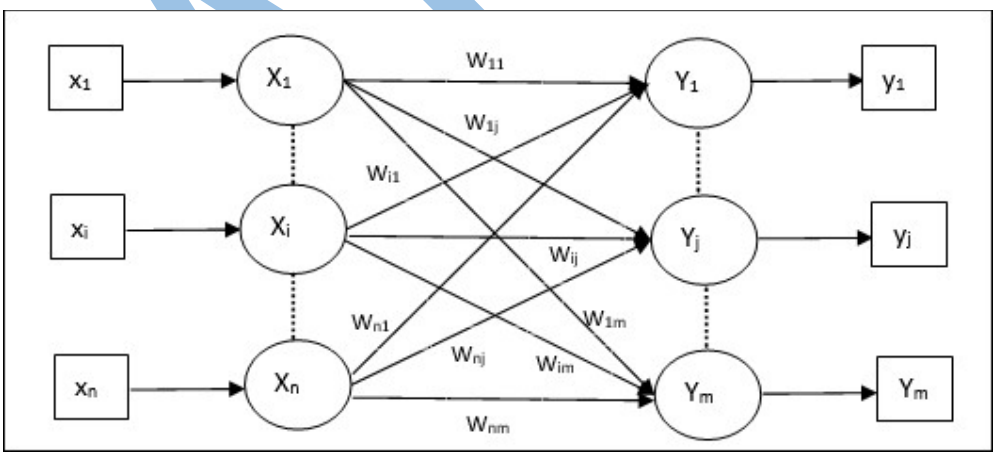
Signal Validation etc.

Hetero Associative memory

Similar to Auto Associative Memory network, this is also a single layer neural network. However, in this network the input training vector and the output target vectors are not the same. The weights are determined so that the network stores a set of patterns. Hetero associative network is static in nature, hence, there would be no non-linear and delay operations.

Architecture

As shown in the following figure, the architecture of Hetero Associative Memory network has ‘n’ number of input training vectors and ‘m’ number of output target vectors.



Training Algorithm

For training, this network is using the Hebb or Delta learning rule.

Step 1 – Initialize all the weights to zero as $\mathbf{w_{ij}} = \mathbf{0}$ $i = 1ton, j = 1tom$

Step 2 – Perform steps 3-4 for each input vector.

Step 3 – Activate each input unit as follows –

$$x_i = s_i \ (i = 1 \, to \, n)$$

Step 4 – Activate each output unit as follows –

$$y_j = s_j \ (j = 1 \, to \, m)$$

Step 5 – Adjust the weights as follows –

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

Testing Algorithm

Step 1 – Set the weights obtained during training for Hebb's rule.

Step 2 – Perform steps 3-5 for each input vector.

Step 3 – Set the activation of the input units equal to that of the input vector.

Step 4 – Calculate the net input to each output unit **j = 1 to m;**

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

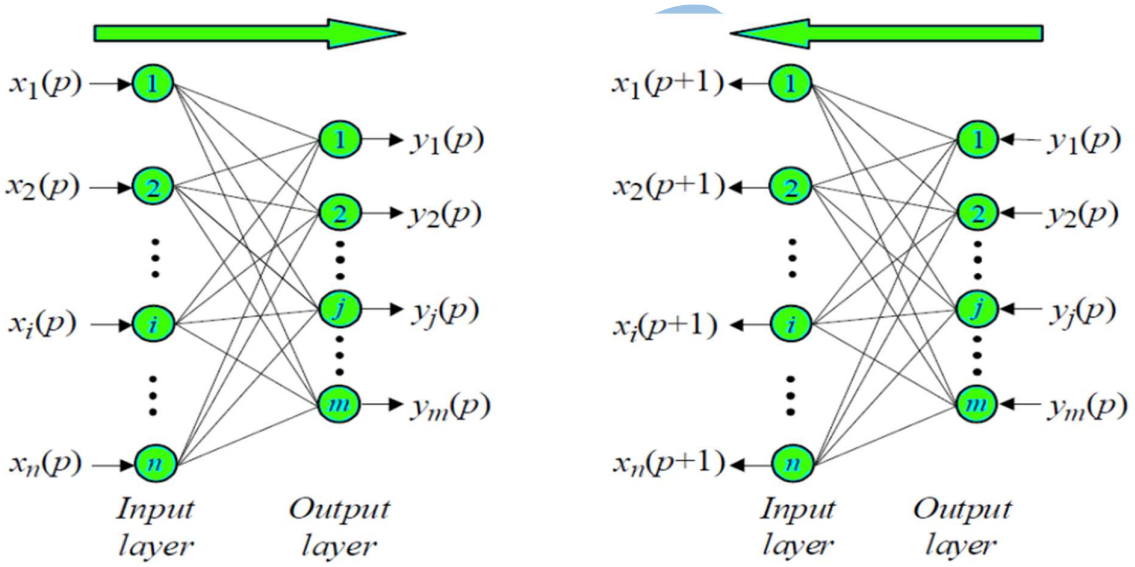
Step 5 – Apply the following activation function to calculate the output

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

ANN – Bidirectional Associative Memory (BAM)

Bidirectional Associative Memory (BAM) is a supervised learning model in Artificial Neural Network. This is ***hetero-associative memory***, for an input pattern, it returns another

pattern which is potentially of a different size. This phenomenon is very similar to the human brain. Human memory is necessarily associative. It uses a chain of mental associations to recover a lost memory like associations of faces with names, in exam questions with answers, etc. In such memory associations for one type of object with another, a Recurrent Neural Network (RNN) is needed to receive a pattern of one set of neurons as an input and generate a related, but different, output pattern of another set of neurons. **Why BAM is required?** The main objective to introduce such a network model is to store hetero-associative pattern pairs. This is used to retrieve a pattern given a noisy or incomplete pattern. **BAM Architecture:** When BAM accepts an input of ***n***-dimensional vector ***X*** from set ***A*** then the model recalls ***m***-dimensional vector ***Y*** from set ***B***. Similarly when ***Y*** is treated as input, the BAM recalls ***X***.



(*a*) Forward direction. (*b*) Backward direction.

Algorithm:

Storage (Learning): In this learning step of BAM, weight matrix is calculated between M pairs of patterns (fundamental memories) are stored in the synaptic weights of the network

following the equation

Testing: We have to check that the BAM recalls perfectly for corresponding and recalls for corresponding . Using,

$$Y_m = \text{sign}\left(W^T X_m\right), \quad m = 1.2, \ldots, M$$
$$X_m = \text{sign}\left(W Y_m\right), \quad m = 1.2, \ldots, M$$

All pairs should be recalled accordingly.

Retrieval: For an unknown vector *X* (a corrupted or incomplete version of a pattern from set *A* or *B*) to the BAM and retrieve a previously stored association:

$$X \neq X_m, \quad m = 1, 2, \ldots, M$$

- *Initialize the BAM:*

$$X(0) = X, \quad p = 0$$

- *Calculate the BAM output at iteration* p *:*

$$Y(p) = \text{sign}\,[W^T X(p)]$$

- *Update the input vector* $X(p)$ *:*

$$X(p+1) = \text{sign}[WY(p)]$$

Repeat the iteration until convergence, when input and output remain unchanged.

Limitations of BAM:

Storage capacity of the BAM: In the BAM, stored number of associations should not be exceeded the number of neurons in the smaller layer.

Incorrect convergence: Always the closest association may not be produced by BAM.

Hopfield Neural Network

The Hopfield Neural Networks, invented by Dr John J. Hopfield consists of one layer of *'n'* fully connected recurrent neurons. It is generally used in performing auto-association and optimization tasks. It is calculated using a converging interactive process and it generates a different response than our normal neural nets.

Discrete Hopfield Network

It is a fully interconnected neural network where each unit is connected to every other unit. It behaves in a discrete manner, i.e. it gives finite distinct output, generally of two types:

Binary (0/1)

Bipolar (-1/1)

The weights associated with this network are symmetric in nature and have the following properties.

1. $w_{ij} = w_{ji}$
2. $w_{ii} = 0$

Structure & Architecture of Hopfield Network

Each neuron has an inverting and a non-inverting output.

Being fully connected, the output of each neuron is an input to all other neurons but not the self.

The below figure shows a sample representation of a Discrete Hopfield Neural Network architecture having the following elements.

Discrete Hopfield Network Architecture

[x1 , x2 , ... , xn] -> Input to the n given neurons.

[y1 , y2 , ... , yn] -> Output obtained from the n given neurons

W_{ij} -> weight associated with the connection between the i^{th} and the j^{th} neuron.

Training Algorithm

For storing a set of input patterns S(p) [p = 1 to P], where **$S(p) = S_1(p) \dots S_i(p) \dots S_n(p)$**, the weight matrix is given by:

- **For binary patterns**

$$w_{ij} = \sum_{p=1}^{P}[2s_i(p)-1][2s_j(p)-1] \; (w_{ij} \; for \; all \; i \neq j)$$

- **For bipolar patterns**

$$w_{ij} = \sum_{p=1}^{P}[s_i(p)s_j(p)] \; (where \; w_{ij} = 0 \; for \; all \; i = j)$$

(i.e. weights here have no self-connection)

Steps Involved in the training of a Hopfield Network are as mapped below:

Initialize weights (w_{ij}) to store patterns (using training algorithm).

For each input vector y_i, perform steps 3-7.

Make the initial activators of the network equal to the external input vector x.

$$y_i = x_i : (for \; i = 1 \; to \; n)$$

or each vector y_i, perform steps 5-7.

Calculate the total input of the network y_{in} using the equation given below.

$$y_{in_i} = x_i + \sum_j [y_j w_{ji}]$$

Apply activation over the total input to calculate the output as per the equation given below:

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

(where θ_i (threshold) and is normally taken as 0)

Now feedback the obtained output y_i to all other units. Thus, the activation vectors are updated.

Test the network for convergence.

Consider the following problem. We are required to create a Discrete Hopfield Network with the bipolar representation of the input vector as [1 1 1 -1] or [1 1 1 0] (in case of binary representation) is stored in the network. Test the Hopfield network with missing entries in the first and second components of the stored vector (i.e. [0 0 1 0]).

Given the input vector, x = [1 1 1 -1] (bipolar) and we initialize the weight matrix (w_{ij}) as:

$$w_{ij} = \sum[s^T(p)t(p)]$$
$$= \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & -1 \end{bmatrix}$$
$$= \begin{bmatrix} 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ -1 & -1 & -1 & 1 \end{bmatrix}$$

and weight matrix with no self-connection is:

$$w_{ij} = \begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}$$

As per the question, input vector x with missing entries, x = [0 0 1 0] (**[x1 x2 x3 x4]**) (binary). Make y_i = x = [0 0 1 0] **([y1 y2 y3 y4]).** Choosing unit y_i **(order doesn't matter)** for updating its activation. Take the i^{th} column of the weight matrix for calculation.

(we will do the next steps for all values of yi and check if there is convergence or not)

$$y_{in_1} = x_1 + \sum_{j=1}^{4}[y_j w_{j1}]$$
$$= 0 + \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \\ -1 \end{bmatrix}$$
$$= 0 + 1$$
$$= 1$$

Applying activation, $y_{in_1} > 0 \implies y_1 = 1$

giving feedback to other units, we get

$$y = \begin{bmatrix} 1 & 0 & 1 & 0 \end{bmatrix}$$

which is not equal to input vector

$$x = \begin{bmatrix} 1 & 1 & 1 & 0 \end{bmatrix}$$

Hence, no covergence.'

Now for next unit, we will take updated value via feedback. (**i.e. y = [1 0 1 0]**)

$$y_{in_3} = x_3 + \sum_{j=1}^{4}[y_j w_{j3}]$$

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

$$= 1 + 1$$

$$= 2$$

$$Applying\ activation,\ y_{in_3} > 0 \implies y_3 = 1$$

$$giving\ feedback\ to\ other\ units,\ we\ get$$

$$y = \begin{bmatrix} 1 & 0 & 1 & 0 \end{bmatrix}$$

$$which\ is\ not\ equal\ to\ input\ vector$$

$$x = \begin{bmatrix} 1 & 1 & 1 & 0 \end{bmatrix}$$

$$Hence,\ no\ covergence.$$

Now for next unit, we will take updated value via feedback. **(i.e. y = [1 0 1 0])**

$$y_{in_4} = x_4 + \sum_{j=1}^{4}[y_j w_{j4}]$$

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

$$= 0 + (-1) + (-1)$$

$$= -2$$

$$Applying\ activation,\ y_{in_4} < 0 \implies y_4 = 0$$

$$giving\ feedback\ to\ other\ units,\ we\ get$$

$$y = \begin{bmatrix} 1 & 0 & 1 & 0 \end{bmatrix}$$

$$which\ is\ not\ equal\ to\ input\ vector$$

$$x = \begin{bmatrix} 1 & 1 & 1 & 0 \end{bmatrix}$$

$$Hence,\ no\ covergence.$$

Now for next unit, we will take updated value via feedback. **(i.e. y = [1 0 1 0])**

$$y_{in_2} = x_2 + \sum_{j=1}^{4}[y_j w_{j2}]$$

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

$$= 0 + 1 + 1$$

$$= 2$$

$$Applying\ activation,\ y_{in_2} > 0 \implies y_2 = 1$$

$$giving\ feedback\ to\ other\ units,\ we\ get$$

$$y = \begin{bmatrix}1 & 1 & 1 & 0\end{bmatrix}$$

$$which\ is\ equal\ to\ input\ vector$$

$$x = \begin{bmatrix}1 & 1 & 1 & 0\end{bmatrix}$$

$$Hence,\ covergence\ with\ vector\ x.$$

Continuous Hopfield Network

Unlike the discrete Hopfield networks, here the time parameter is treated as a continuous variable. So, instead of getting binary/bipolar outputs, we can obtain values that lie between 0 and 1. It can be used to solve constrained optimization and associative memory problems. The output is defined as:

$$v_i = g(u_i)$$

where,

v_i = *output from the continuous hopfield network*

u_i = *internal activity of a node in continuous hopfield network.*

Energy Function

The Hopfield networks have an energy function associated with them. It either diminishes or remains unchanged on update (feedback) after every iteration. The energy function for a continuous Hopfield network is defined as:

$$E = 0.5 \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} v_i v_j + \sum_{i=1}^{n} \theta_i v_i$$

To determine if the network will converge to a stable configuration, we see if the energy function reaches its minimum by:

$$\frac{d}{dt}E \leq 0$$

The network is bound to converge if the activity of each neuron wrt time is given by the following differential equation:

$$\frac{d}{dt}u_i = \frac{-u_i}{\tau} + \sum_{j=1}^{n} w_{ij} v_j + \theta_i$$

APEC

Iterative Autoassociative Networks

- Example:

$$x = (1, 1, 1, -1) \quad W = \begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}$$

Output units are threshold units

An incomplete recall input: $x' = (1, 0, 0, 0)$

$Wx' = (0, 1, 1, -1) = x''$

$Wx'' = (3, 2, 2, -2) \rightarrow (1, 1, 1, -1) = x$

- In general: using current output as input of the next iteration

  $x(0)$ = initial recall input

  $x(I) = S(Wx(I\text{-}1)), \quad I = 1, 2, \ldots\ldots$

  until $x(N) = x(K)$ for some $K < N$

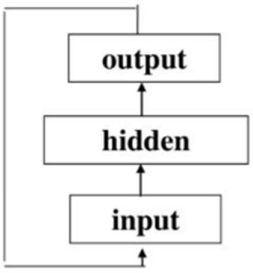
- Dynamic System: state vector x(I)
  - If K = N-1, x(N) is a stable state (fixed point)

    f(Wx(N)) = f(Wx(N-1)) = x(N)
    - If x(K) is one of the stored pattern, then x(K) is called a **genuine memory**
    - Otherwise, x(K) is a **spurious memory** (caused by cross-talk/interference between genuine memories)
    - Each fixed point (genuine or spurious memory) is an **attractor** (with different attraction basin)
  - If K != N-1, ***limit-circle***,
    - The network will repeat

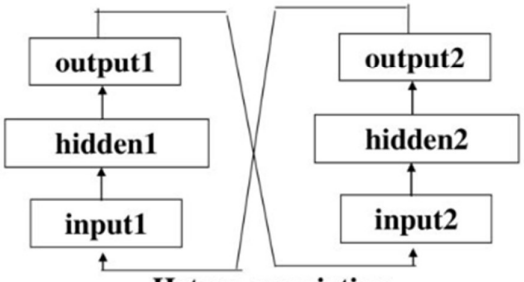
      x(K), x(K+1), …..x(N)=x(K) when iteration continues.
    - Iteration will eventually stop because the total number of distinct state is finite (3^n) if threshold units are used.
  - If patterns are continuous, the system may continue evolve forever (***chaos***) if no such K exists.

My Own Work: Turning BP net for Auto AM

- One possible reason for the small capacity of HM is that it does not have hidden nodes.
- Train feed forward network (with hidden layers) by BP to establish pattern auto-associative.
- Recall: feedback the output to input layer, making it a dynamic system.
- Shown 1) it will converge, and 2) stored patterns become genuine attractors.
- It can store many more patterns (seems ***O(2^n)***)
- Its pattern complete/recovery capability decreases when n increases (# of spurious attractors seems to increase exponentially)
- Call this model BPRR



Auto-association



Hetero-association

- Example
  - n = 10, network is (10 – 20 – 10)
  - Varying # of stored memories (8 – 128)
  - Using all 1024 patterns for recall, correct if one of the stored memories is recalled
  - Two versions in preparing training samples
    - (X, X), where X is one of the stored memory
    - Supplemented with (X', X) where X' is a noisy version of X

# stored memories	correct recall w/o relaxation		correct recall with relaxation		# spurious attractors	
8	202	(835)	861	(1024)	6	(0)
16	49	(454)	341	(980)	60	(5)
32	39	(371)	191	(928)	197	(17)
64	65	(314)	126	(662)	515	(144)
128	128	(351)	136	(561)	825	(254)

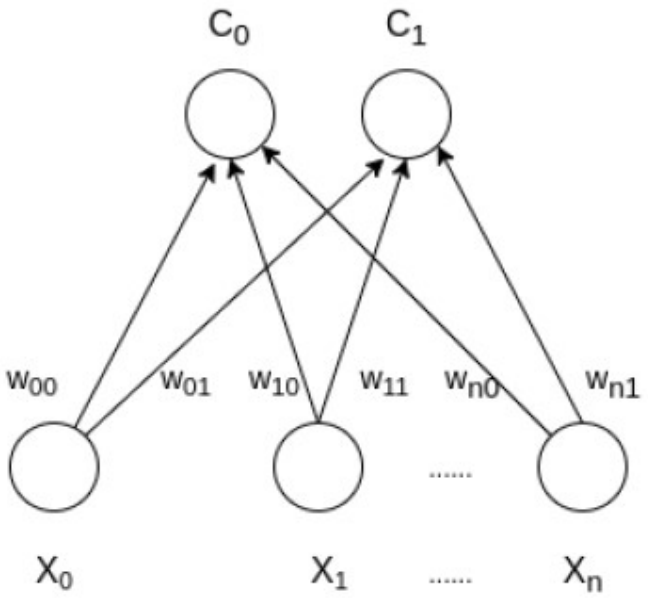
Numbers in parentheses are for learning with supplementary samples (X', X)

Kohonen Self-Organizing Feature Maps

Suppose we have some pattern of arbitrary dimensions, however, we need them in one dimension or two dimensions. Then the process of feature mapping would be very useful to convert the wide pattern space into a typical feature space. Now, the question arises why do we require self-organizing feature map? The reason is, along with the capability to convert the arbitrary dimensions into 1-D or 2-D, it must also have the ability to preserve the neighbor topology.

Self Organizing Map (or Kohonen Map or SOM) is a type of Artificial Neural Network which is also inspired by biological models of neural systems from the 1970s. It follows an unsupervised learning approach and trained its network through a competitive learning algorithm. SOM is used for clustering and mapping (or dimensionality reduction) techniques to map multidimensional data onto lower-dimensional which allows people to reduce complex problems for easy interpretation. SOM has two layers, one is the Input layer and the other one is the Output layer.

The architecture of the Self Organizing Map with two clusters and n input features of any sample is given below:



How do SOM works?

Let's say an input data of size (m, n) where m is the number of training examples and n is the number of features in each example. First, it initializes the weights of size (n, C) where C is the number of clusters. Then iterating over the input data, for each training example, it updates the winning vector (weight vector with the shortest distance (e.g Euclidean distance) from training example). Weight updation rule is given by :

$w_{ij} = w_{ij}(old) + alpha(t) * (x_i^k - w_{ij}(old))$

where alpha is a learning rate at time t, j denotes the winning vector, i denotes the i^{th} feature of training example and k denotes the k^{th} training example from the input data. After training the SOM network, trained weights are used for clustering new examples. A new example falls in the cluster of winning vectors.

Algorithm

Training:

Step 1: Initialize the weights w_{ij} random value may be assumed. Initialize the learning rate α.

Step 2: Calculate squared Euclidean distance.

$$D(j) = \Sigma (wij - xi)^2 \quad \text{where } i=1 \text{ to } n \text{ and } j=1 \text{ to } m$$

Step 3: Find index J, when D(j) is minimum that will be considered as winning index.

Step 4: For each j within a specific neighborhood of j and for all i, calculate the new weight.

$$wij(new)=wij(old) + \alpha[xi - wij(old)]$$

Step 5: Update the learning rule by using :

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

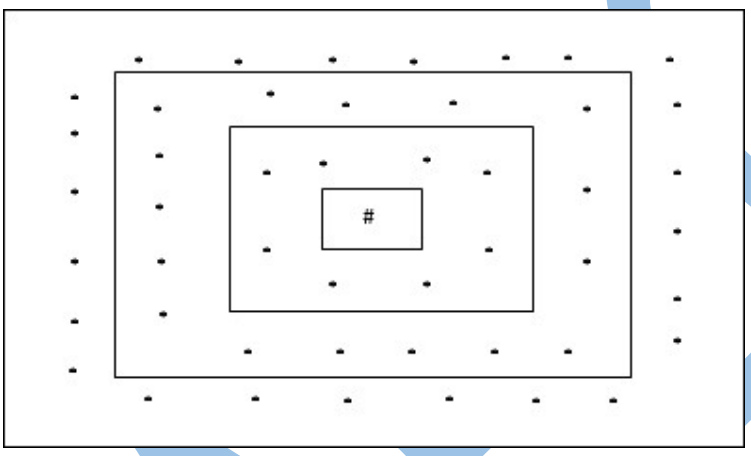
Step 6: Test the Stopping Condition.

Neighbor Topologies in Kohonen SOM

There can be various topologies, however the following two topologies are used the most −

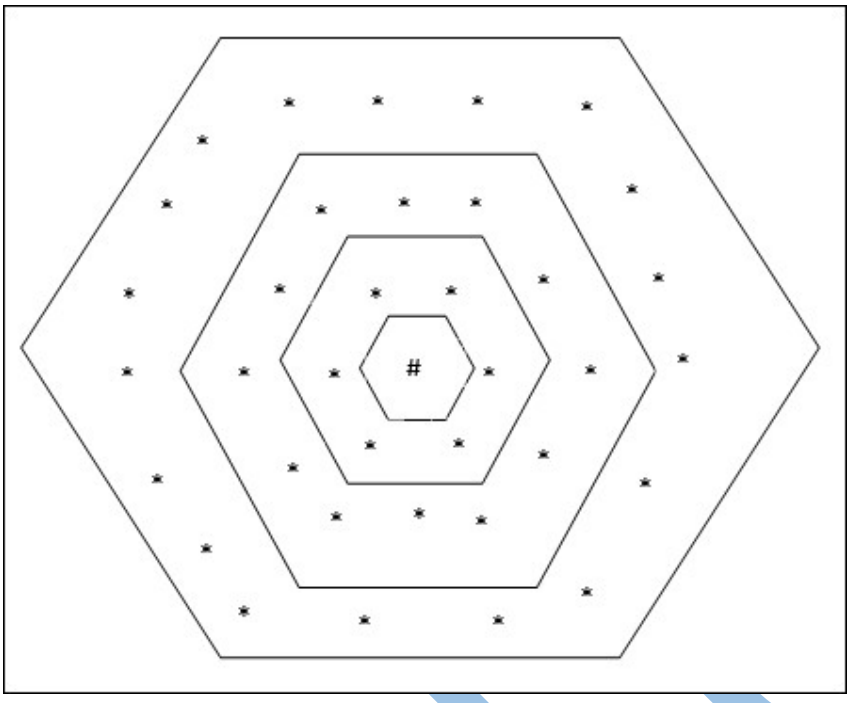
Rectangular Grid Topology

This topology has 24 nodes in the distance-2 grid, 16 nodes in the distance-1 grid, and 8 nodes in the distance-0 grid, which means the difference between each rectangular grid is 8 nodes. The winning unit is indicated by #.



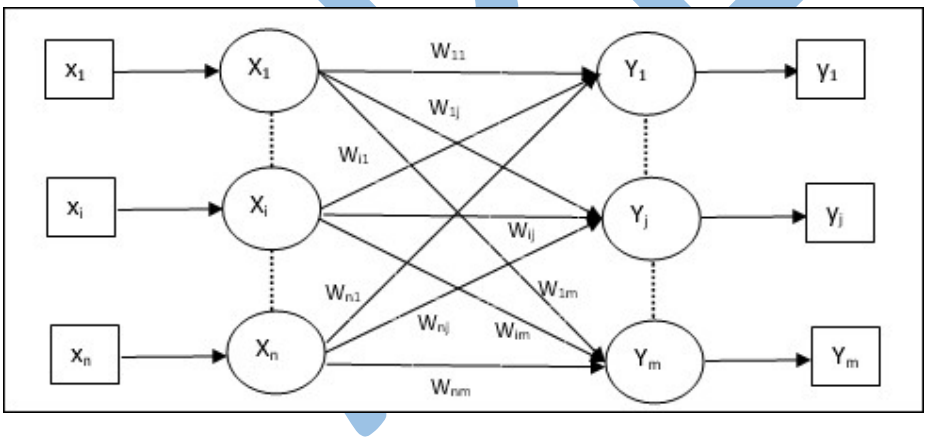
Hexagonal Grid Topology

This topology has 18 nodes in the distance-2 grid, 12 nodes in the distance-1 grid, and 6 nodes in the distance-0 grid, which means the difference between each rectangular grid is 6 nodes. The winning unit is indicated by #.



Architecture

The architecture of KSOM is similar to that of the competitive network. With the help of neighborhood schemes, discussed earlier, the training can take place over the extended region of the network.



Algorithm for training

Step 1 − Initialize the weights, the learning rate α and the neighborhood topological scheme.

Step 2 − Continue step 3-9, when the stopping condition is not true.

Step 3 − Continue step 4-6 for every input vector x.

Step 4 − Calculate Square of Euclidean Distance for j = 1 to m

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

Step 5 – Obtain the winning unit **J** where **D(j)** is minimum.

Step 6 – Calculate the new weight of the winning unit by the following relation –

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

Step 7 – Update the learning rate **α** by the following relation –

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

Step 8 – Reduce the radius of topological scheme.

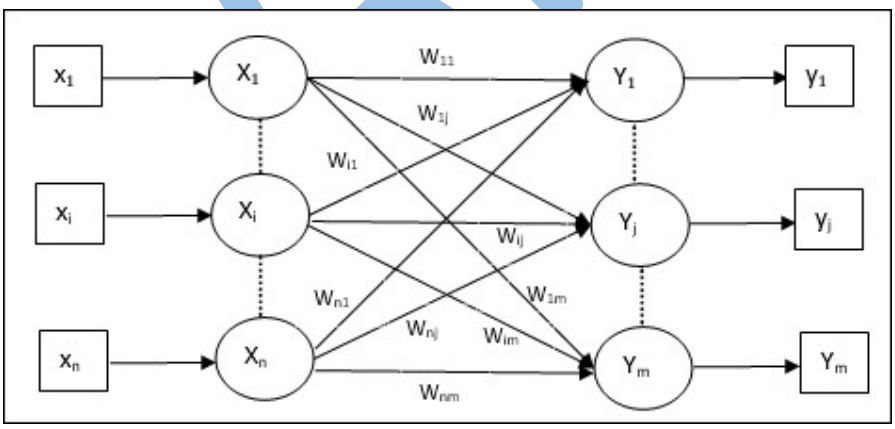
Step 9 – Check for the stopping condition for the network.

Learning Vector Quantization

Learning Vector Quantization LVQ, different from Vector quantization VQ and Kohonen Self-Organizing Maps KSOM, basically is a competitive network which uses supervised learning. We may define it as a process of classifying the patterns where each output unit represents a class. As it uses supervised learning, the network will be given a set of training patterns with known classification along with an initial distribution of the output class. After completing the training process, LVQ will classify an input vector by assigning it to the same class as that of the output unit.

Architecture

Following figure shows the architecture of LVQ which is quite similar to the architecture of KSOM. As we can see, there are "n" number of input units and "m" number of output units. The layers are fully interconnected with having weights on them.



Parameters Used

Following are the parameters used in LVQ training process as well as in the flowchart

x = training vector (x_1,...,x_i,...,x_n)

T = class for training vector x

w_j = weight vector for j^{th} output unit

C_j = class associated with the j^{th} output unit

Training Algorithm

Step 1 − Initialize reference vectors, which can be done as follows −

Step 1a − From the given set of training vectors, take the first "m" numberofclusters training vectors and use them as weight vectors. The remaining vectors can be used for training.

Step 1b − Assign the initial weight and classification randomly.

Step 1c− Apply K-means clustering method.

Step 2 − Initialize reference vector α

Step 3 − Continue with steps 4-9, if the condition for stopping this algorithm is not met.

Step 4 − Follow steps 5-6 for every training input vector x.

Step 5 − Calculate Square of Euclidean Distance for j = 1 to m and i = 1 to n

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

Step 6 − Obtain the winning unit **J** where **D**j is minimum.

Step 7 − Calculate the new weight of the winning unit by the following relation −

if **T** = $\mathbf{C_j}$ then $w_j(new) = w_j(old) + \alpha[x - w_j(old)]$

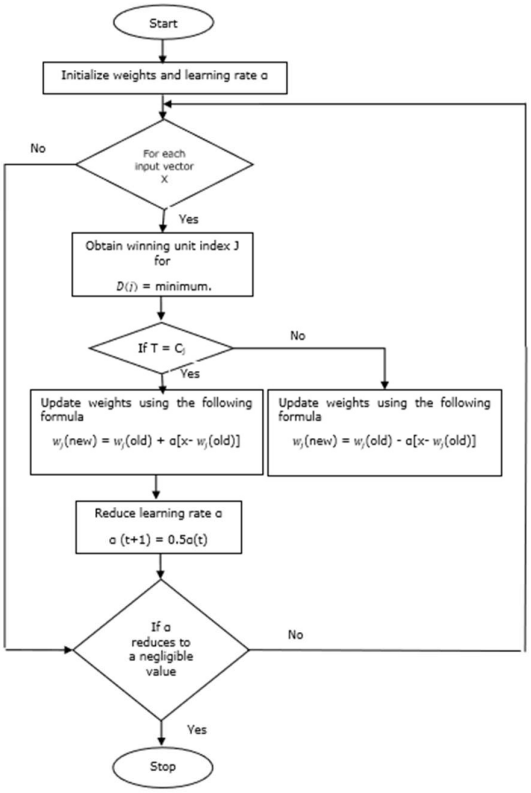
if **T** ≠ $\mathbf{C_j}$ then $w_j(new) = w_j(old) - \alpha[x - w_j(old)]$

Step 8 − Reduce the learning rate α.

Step 9 − Test for the stopping condition. It may be as follows −

- Maximum number of epochs reached.
- Learning rate reduced to a negligible value.

Flowchart



Variants

Three other variants namely LVQ2, LVQ2.1 and LVQ3 have been developed by Kohonen. Complexity in all these three variants, due to the concept that the winner as well as the runner-up unit will learn, is more than in LVQ.

LVQ2

As discussed, the concept of other variants of LVQ above, the condition of LVQ2 is formed by window. This window will be based on the following parameters −

x − the current input vector

y_c − the reference vector closest to x

y_r − the other reference vector, which is next closest to x

d_c − the distance from x to y_c

d_r − the distance from x to y_r

The input vector x falls in the window, if

$$\frac{d_c}{d_r} > 1 - \theta \; and \; \frac{d_r}{d_c} > 1 + \theta$$

Here, θ is the number of training samples.

Updating can be done with the following formula –

$$y_c(t+1) = y_c(t) + \alpha(t)[x(t) - y_c(t)]\ belongstodifferentclass$$

$$y_r(t+1) = y_r(t) + \alpha(t)[x(t) - y_r(t)]\ belongstosameclass$$

Here α is the learning rate.

LVQ2.1

In LVQ2.1, we will take the two closest vectors namely y_{c1} and y_{c2} and the condition for window is as follows

–

$$Min\left[\frac{d_{c1}}{d_{c2}}, \frac{d_{c2}}{d_{c1}}\right] > (1-\theta)$$

$$Max\left[\frac{d_{c1}}{d_{c2}}, \frac{d_{c2}}{d_{c1}}\right] < (1+\theta)$$

Updating can be done with the following formula –

$$y_{c1}(t+1) = y_{c1}(t) + \alpha(t)[x(t) - y_{c1}(t)]\ belongstodifferentclass$$

$$y_{c2}(t+1) = y_{c2}(t) + \alpha(t)[x(t) - y_{c2}(t)]\ belongstosameclass$$

Here, α is the learning rate.

LVQ3

In LVQ3, we will take the two closest vectors namely $\mathbf{y_{c1}}$ and $\mathbf{y_{c2}}$ and the condition for window is as follows –

$$Min\left[\frac{d_{c1}}{d_{c2}}, \frac{d_{c2}}{d_{c1}}\right] > (1-\theta)(1+\theta)$$

Here $\theta \approx 0.2$

Updating can be done with the following formula –

$$y_{c1}(t+1) = y_{c1}(t) + \beta(t)[x(t) - y_{c1}(t)]\ belongstodifferentclass$$

$$y_{c2}(t+1) = y_{c2}(t) + \beta(t)[x(t) - y_{c2}(t)]\ belongstosameclass$$

Here β is the multiple of the learning rate α and $\beta = m\alpha(t)$ for every **0.1 < m < 0.5**

Counter propagation

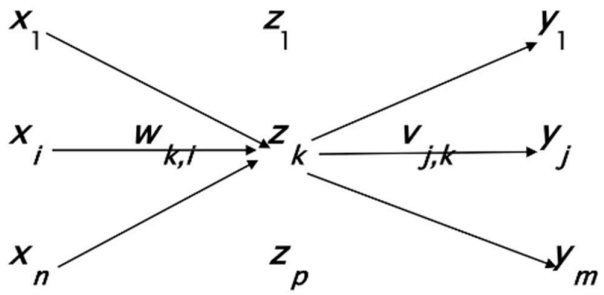
- CP algorithm consists of a **input, hidden and output layer.**
- In this case the **hidden layer** is called the **Kohonen layer** & the **output layer** is called the **Grossberg layer.**
- At the beginning of CP algorithm, output of input neurons is equal to the input vector.
- The input vector is **normalized to the length of one**. Now the progression of the Kohonen layer starts.
- This means that a **neuron with the highest net input is identified.**
- The activation of this **winner neuron is set to** 1 & the activation of **all other neurons in this layer is set to 0.**
- Now the output of all **output neurons is calculated.** There is only one neuron of the hidden layer with the activation **and the output set to 1.**
- The fact that the activation and output of neurons is the **weighted sum of output of the hidden neurons** implies that the output of the **output neurons is the weight of the link between the winner neuron and the output neurons.**

Basic idea of CN

- **Purpose:**
  - **fast and coarse approximation** of vector mapping
  - **input vectors x are divided into clusters/classes.**
  - each cluster of **x has output y**, which is (hopefully) the average of $\phi(x)$ for all x in that class.

$$y = \phi(x)$$

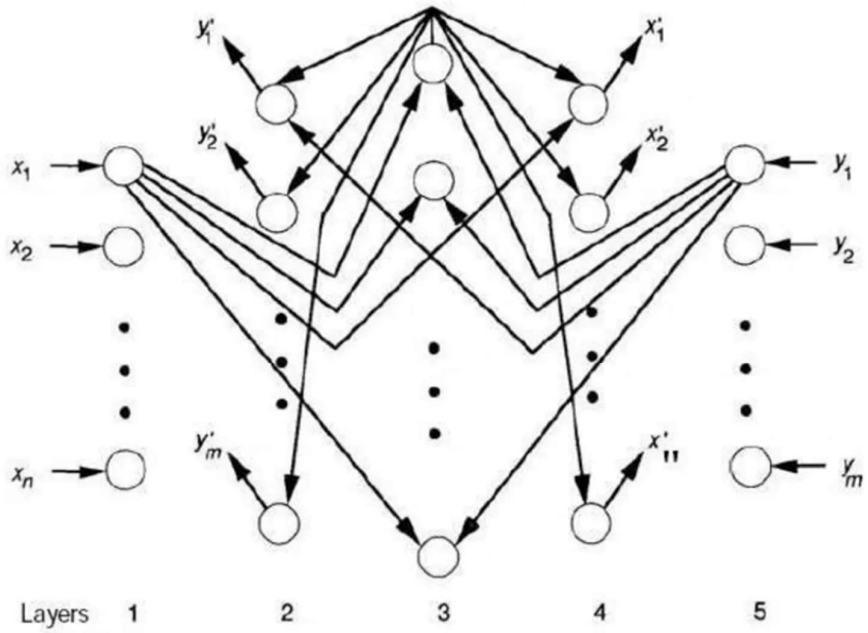
Architecture: simple forward CPN



from input to hidden (class)

from hidden (class) to output

Network architecture

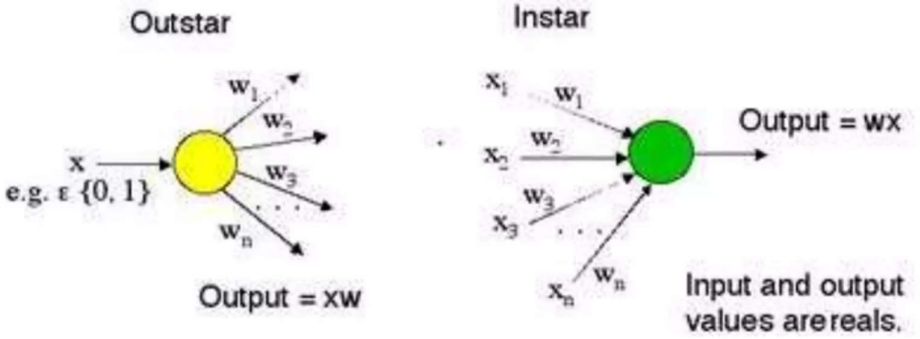


Counter propagation Network

1. Invented by **Robert Hecht-Nielson**, founder of **HNC inc.**
2. Consists of **two opposing networks**, one for **learning a function**, the other for **learning its inverse**.
3. Each network has two layers:
   - A **Kohonen first layer** that clusters inputs.
   - An **'outstar' second layer** to provide the output values for each cluster.

Outstar and Instar

1. An **instar responds to a single input.**
2. An **outstar produces a single (multi dimensional) output** d when simulated with a binary value x.



3. Biologically, **outstar would be synaptic weights**, while instar would have dendritic ones. It is common to refer to weights as '**synaptic'.**

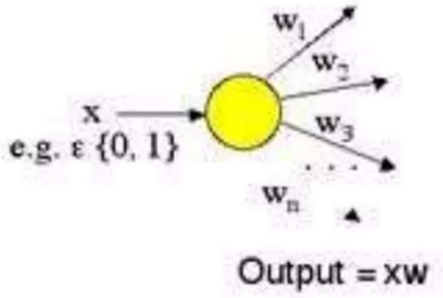
APL

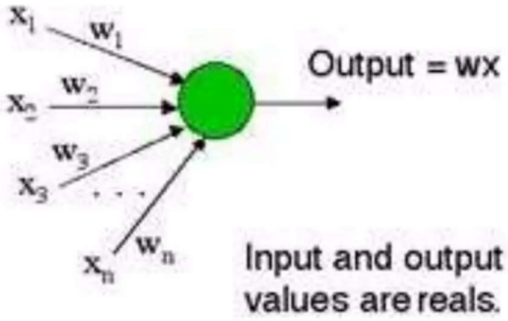
- Outstar learning rule:

$\Delta w = \alpha \times (d - w)$
(d = desired)

Instar learning rule:

$\Delta w = \alpha(x - w)\, d$
(d = desired)





Counter propagation Operation

1. An **outstar neuron is associated with each cluster** representative.
2. **Given an input**, the **winner is found.**
3. An **outstar is then stimulated** to give the output.
4. Since these **networks operate by recognizing input patterns in the first layer**, one would generally **use lots of neurons** in this layer.

Adaptive Resonance Theory Network.

What is ART ?

ART stands for "Adaptive Resonance Theory", invented by Stephen Grossberg in 1976.

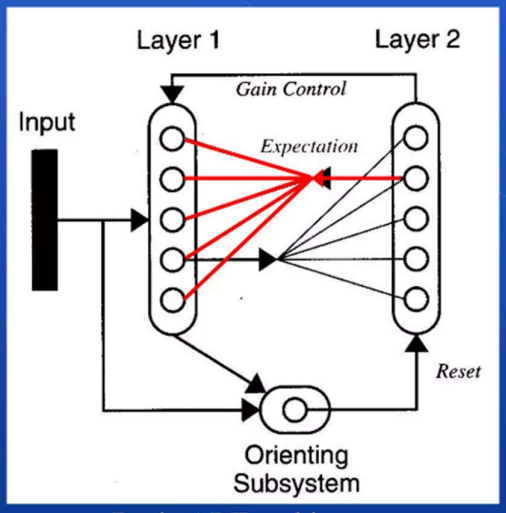
- ART represents a family of neural networks.
- The basic ART System is an unsupervised learning model.
- The term "resonance" refers to resonant state of a neural network in which a category prototype vector matches close enough to the current input vector. ART matching leads to this resonant state,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

Key Innovation

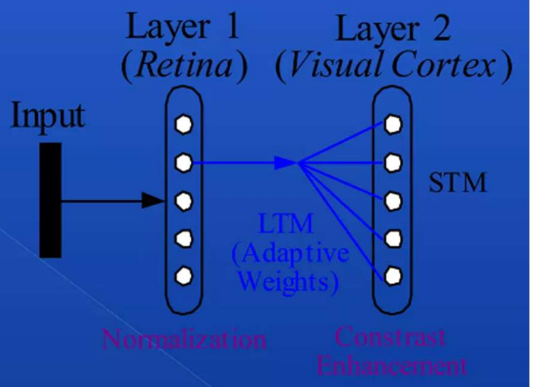
The key innovation of ART is the use of "**expectations**."

- As each **input** is presented to the network, it is compared with the **prototype vector** that is most closely **matches** (the expectation).
- 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.*

Overview



Basic ART architecture



Grossberg competitive network

Grossberg Network

- The L1-L2 connections are **instars**, which performs a **clustering** (or **categorization**) operation. When an input pattern is presented, it is multiplied (after normalization) by the L1-L2 weight matrix.
- A **competition** is performed at Layer 2 to determine which row of the weight matrix is closest to the input vector. That row is then moved toward the input vector.
- After learning is complete, ***each row of the L1-L2 weight matrix is a prototype pattern***, which represents a cluster (or a category) of input vectors.

ART Networks

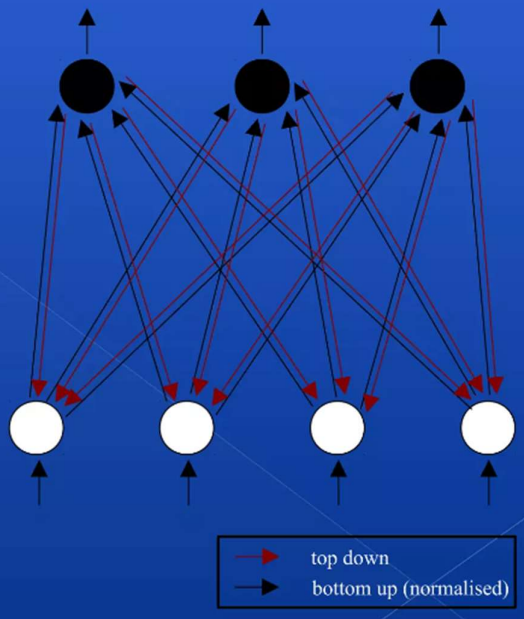
- Learning of ART networks also occurs in a set of **feedback connections** from Layer 2 to Layer 1. These connections are **outstars** which perform pattern recall.
- When a node in **Layer 2** is activated, this ***reproduces*** a prototype pattern (the expectation) at layer 1.
- **Layer 1** then performs a **comparison** between the expectation and the input pattern.
- When the expectation and the input pattern are **NOT** closely matched, the **orienting subsystem** causes a **reset** in Layer 2.

ART Networks (Contd.)

- The reset **disables** the **current** winning neuron, and the current expectation is removed.
- A **new competition** is then performed in Layer 2, *while the previous winning neuron is* ***disable***.
- The new winning neuron in Layer 2 projects a new expectation to Layer 1, through the L2-L1 connections.
- This process continues until *the L2-L1 expectation provides a close enough match to the input pattern*.

ART Architecture

- Bottom-up weights b_{ij}
- Top-down weights t_{ij}
  - Store class template
- Input nodes
  - Vigilance test
  - Input normalisation
- Output nodes
  - Forward matching
- Long-term memory
  - ANN weights
- Short-term memory
  - ANN activation pattern



ART Architecture (Contd.)

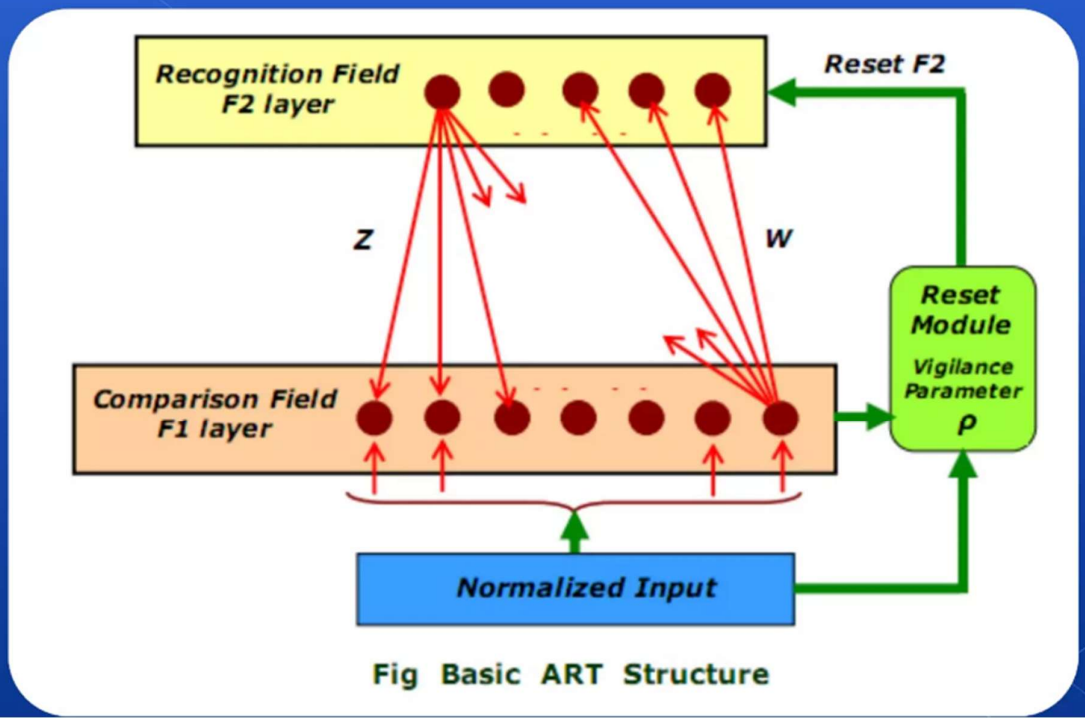


Fig Basic ART Structure

ART Architecture (Contd.)

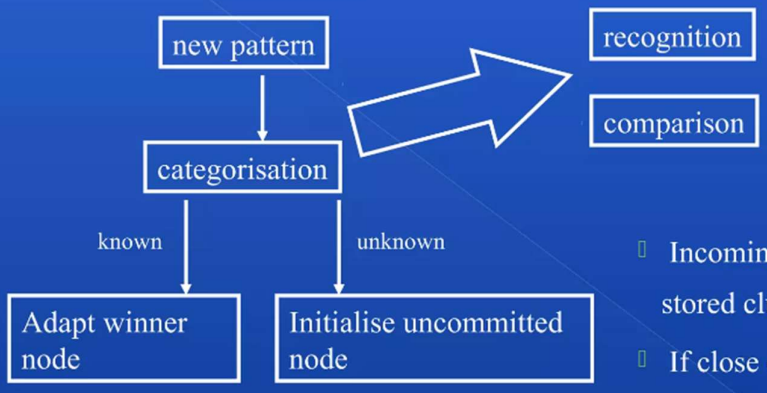
- The basic ART system is unsupervised learning model. It typically consists of
  - a comparison field and a recognition field composed of neurons,
  - a vigilance parameter, and
  - a reset module

ART Architecture (Contd.)

- Comparison field
  - The comparison field takes an input vector (a one-dimensional array of values) and transfers it to its best match in the recognition field. Its best match is the single neuron whose set of weights (weight vector) 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. In this way the recognition field exhibits lateral inhibition, allowing each neuron in it to represent a category to which input vectors are classified.

- 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.

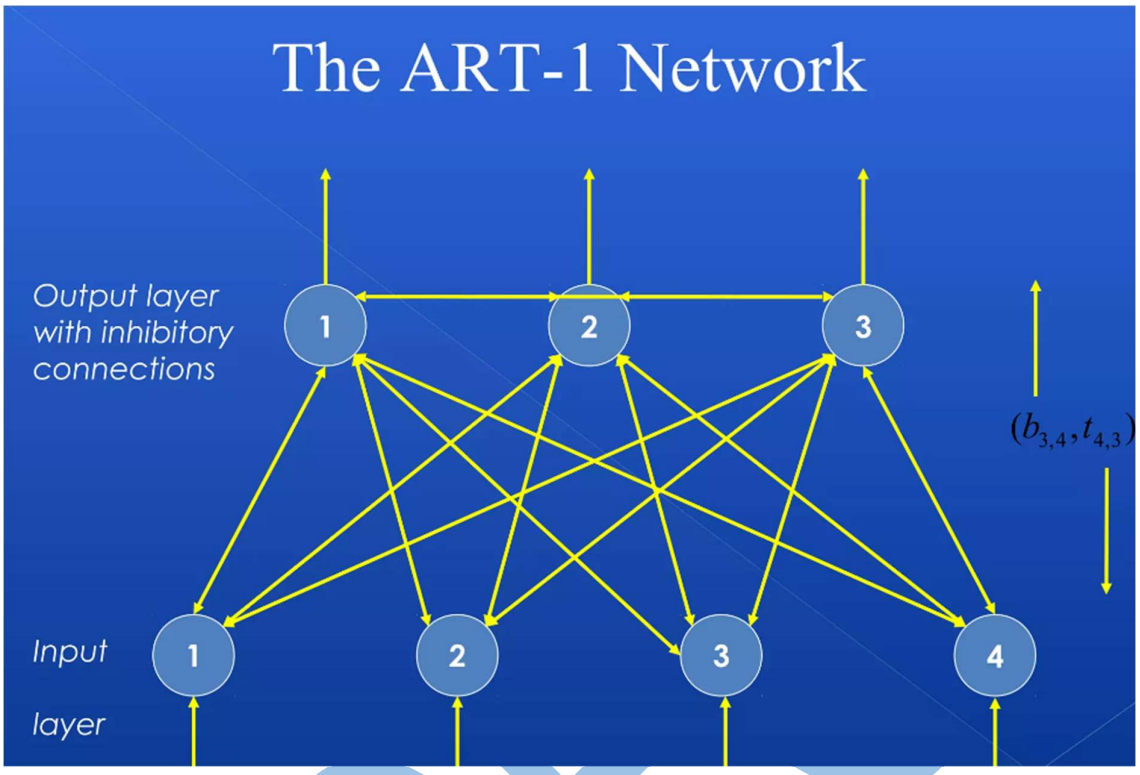
ART Algorithm

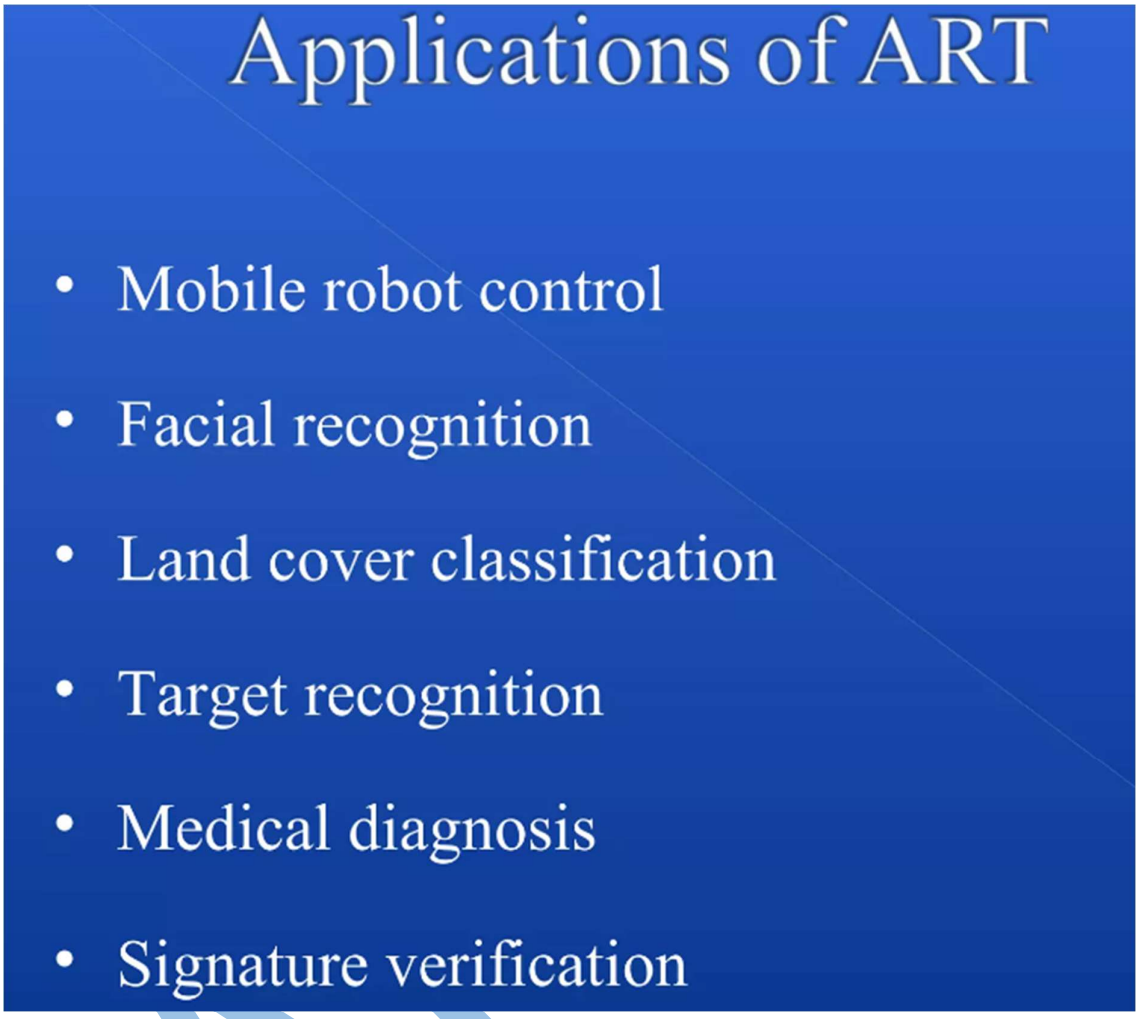


- Incoming pattern matched with stored cluster templates
- If close enough to stored template joins best matching cluster, weights adapted
- If not, a new cluster is initialised with pattern as template

ART Types

- ART-1
  - Binary input vectors
  - Unsupervised NN that can be complemented with external changes to the vigilance parameter
- ART-2
  - Real-valued input vectors
- ART-3
  - Parallel search of compressed or distributed pattern recognition codes in a NN hierarchy.
  - Search process leads to the discovery of appropriate representations of a non stationary input environment.
  - Chemical properties of the synapse emulated in the search process


The ART-1 Network
Output layer with inhibitory connections
1
2
3
$(b_{3,4}, t_{4,3})$
Input layer
1
2
3
4


Applications of ART
• Mobile robot control
• Facial recognition
• Land cover classification
• Target recognition
• Medical diagnosis
• Signature verification