By
Prof. N. P. Padhy(IIT Roorkee)
And
Prof. S. P. Simon(NIT Trichy)

CHAPTER- 3

# ARTIFICIAL NEURAL NETWORKS -Second Generation

- ***Explain the Backpropagation in Neural Networks***
- ***Learn the working of Kohonnen Neural Networks***
- ***Understand the concept of Learning Vector Quantization (LVQ) Network***
- ***Get familiar with Hamming Neural Networks and Hopfield Neural Networks***
- ***Explain the fundamentals of Bidirectional Associative Memory***
- ***Elaborate on Adaptive Resonance Theory (ART) Networks***
- ***Discuss the functioning of Boltzmann Machine***
- ***Understand concepts of Radial Basis Neural Networks and Support Vector Machines***
- ***Conduct Electrical Load Forecasting using MatLab Neural Network Toolbox***

## ORGANIZATION

## INTRODUCTION TO SECOND GENERATION NEURAL NETWORK

- ***Neurons of the second generation use continuous activation function.***
- **Suitable for analog in and analog out applications.**

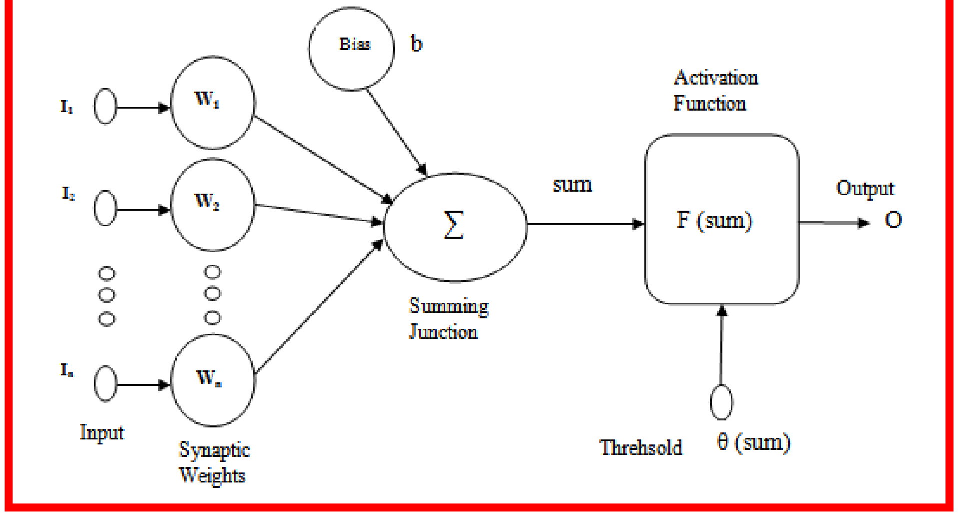
- ***<u>Example activation functions</u>***
    - ***Sigmoid .***
    - ***Hyperbolic tangent.***

- ***<u>Examples neural networks.</u>***
    - ***Feed-forward neural networks.***
    - ***Recurrent neural networks.***

- ***Requires fewer neurons than a network of the first generation***
- ***Can approximate any analog function.***

# BACKPROPAGATION NEURAL NETWORKS

- **Back propagation is a systematic method for training multiple layer ANN**
- **It is a generalization of Widrow-Hoff error correction rule.**
- **80% of ANN applications uses back propagation.**



**A simple neuron with many inputs**

**Consider a simple neuron**

- **Neuron has a summing junction and activation function.**
- **Any non linear function which differentiable every where and increases everywhere with sum can be used as activation function.**
- **<u>Examples:</u>**
  - **Logistic function.**
  - **Arc tangent function.**
  - **Hyperbolic tangent activation function.**
- **These activation function makes the multilayer network to have greater representational power than single layer network only when non-linearity is introduced.**

# BACKPROPAGATION NEURAL NETWORKS

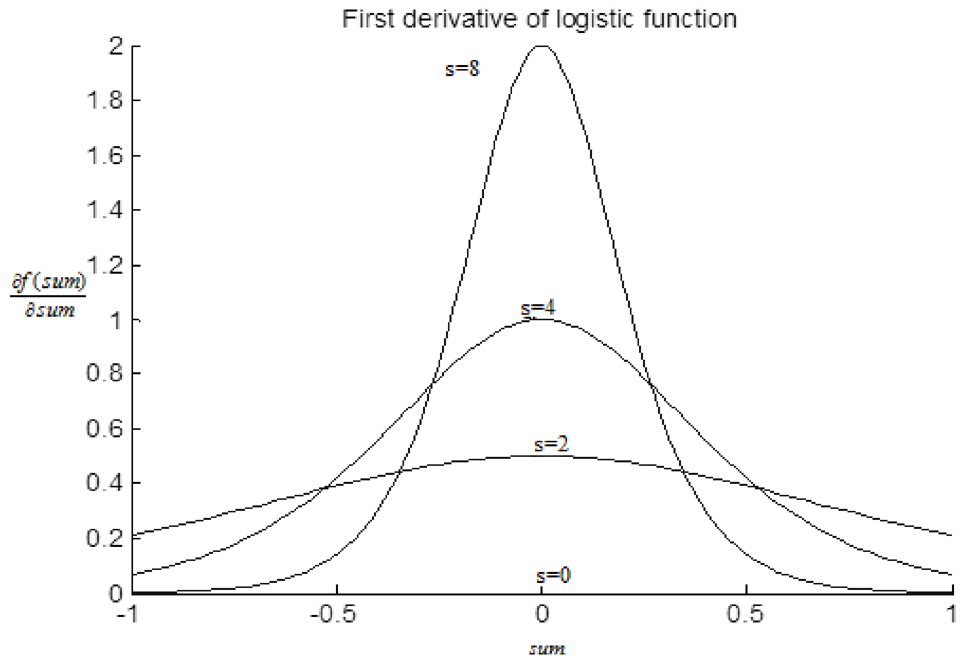
**The input to the activation function is sum which is defined by the following equation**

$$sum = I_1W_1 + I_2W_2 + ..... + I_nW_n = \sum_{j=1}^{n} I_jW_j + b$$

**Activation Function: Logistic Function**

$$f(sum) = \frac{1}{(1+e^{-s*sum})} = (1+e^{-s*sum})^{-1}$$

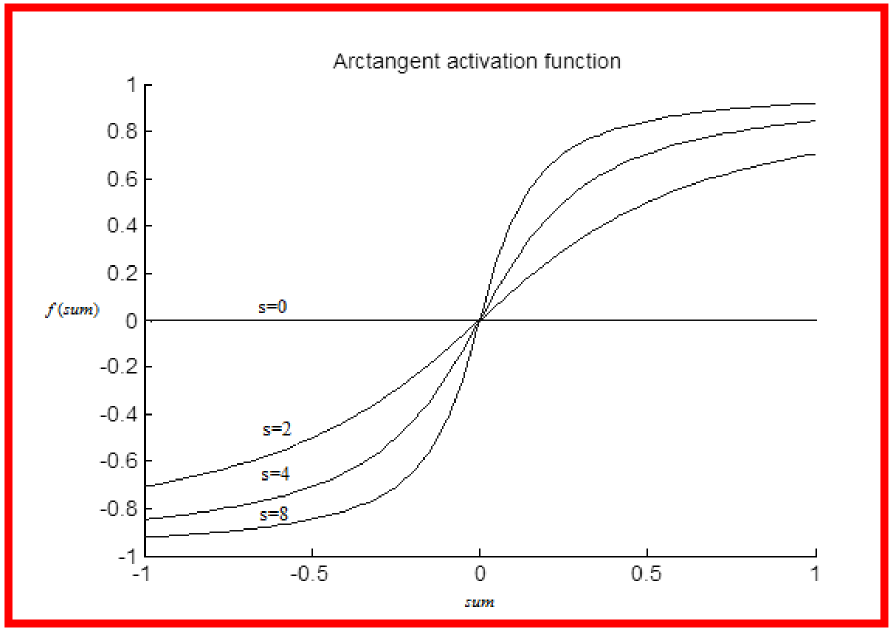
**Logistic function monotonically increases from a lower limit (0 or -1) to an upper limit (+1) as *sum* increases. In which values vary between 0 and 1, with a value of 0.5 when I is zero**



## BACKPROPAGATION NEURAL NETWORKS

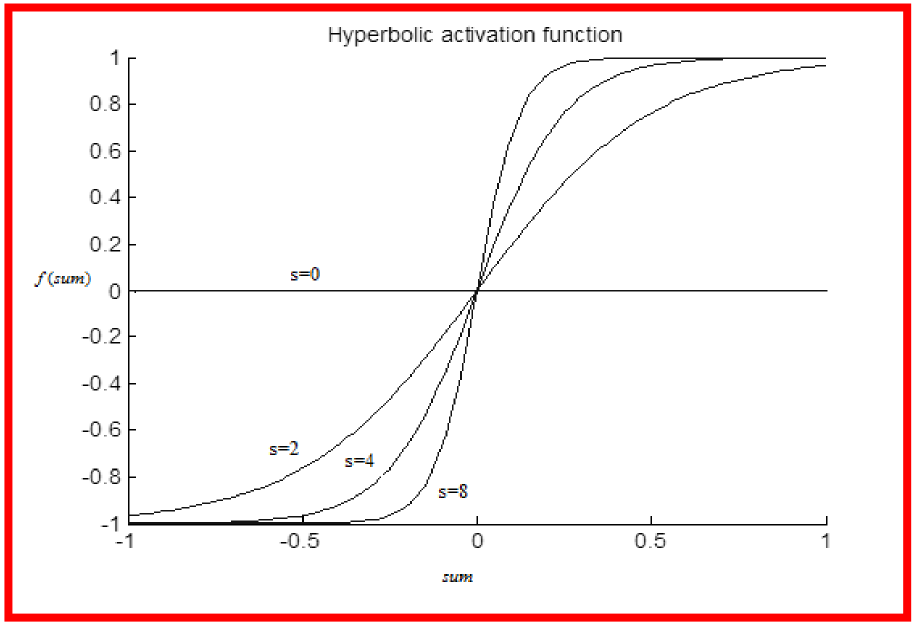
**Activation Function: Arc Tangent**

$$f(sum) = \frac{2}{\pi}\tan^{-1}(s * sum)$$



**Activation Function: Hyperbolic Tangent**

$$f(sum) = \tanh(s * I) = \frac{e^{s*sum} - e^{-s*sum}}{e^{s*sum} + e^{-s*sum}}$$



# BACKPROPAGATION NEURAL NETWORKS

## Need of Hidden layers

- ***A network with only two layers (input and output) can only represent the input with whatever representation already exists in the input data.***
- ***If the data's are discontinuous or non-linearly separable, the innate representation is inconsistent, and the mapping cannot be learned using two layers(Input & Output).***
- ***Therefore, hidden layer(s) are used between input and output layers.***

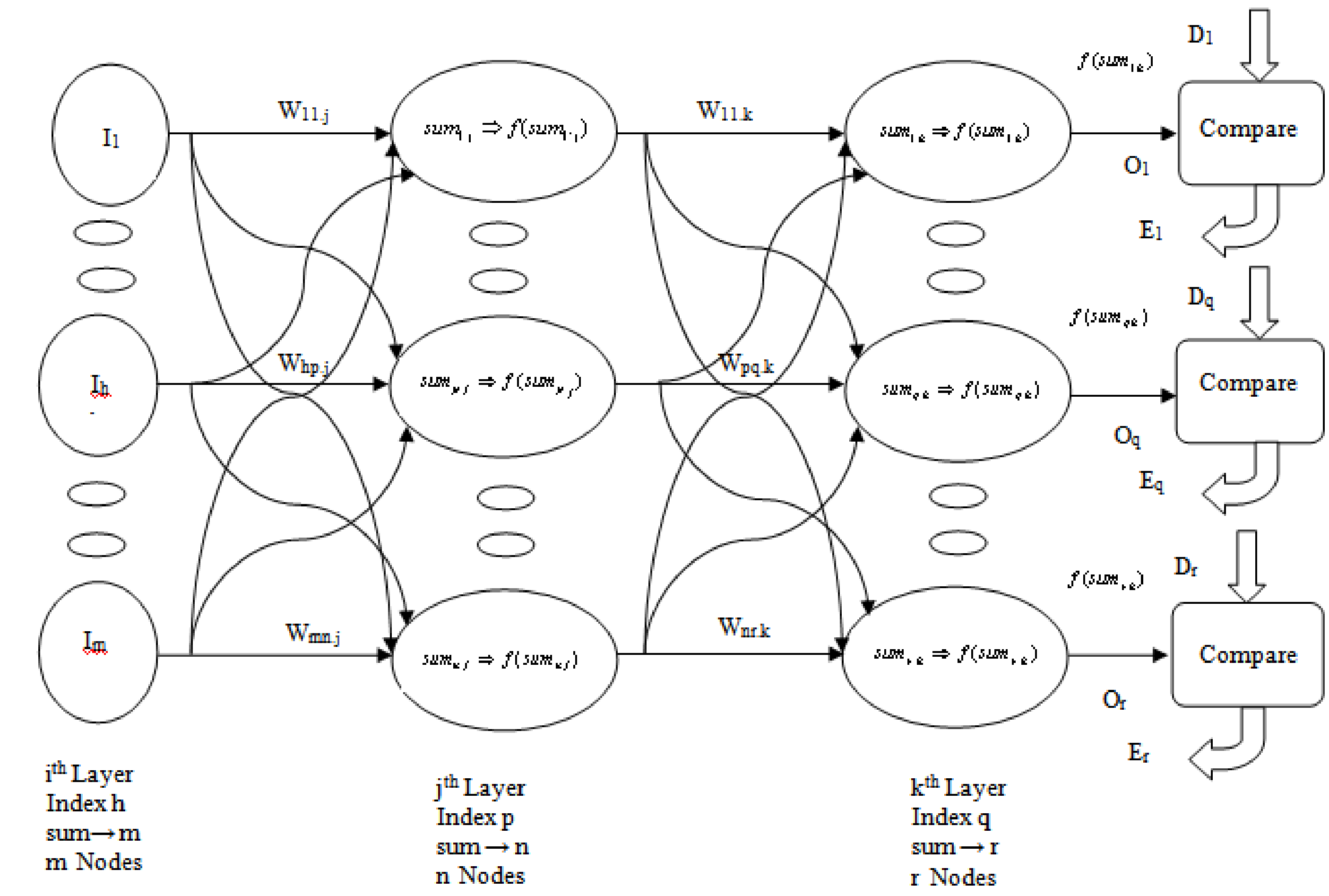
***INPUT*** ***HIDDEN*** ***OUTPUT***

## WEIGHTS

- ***Weights connects unit(neuron) in one layer only to those in the next higher layer.***
- ***The output of the unit is scaled by the value of the connecting weight, and it is fed forward to provide a portion of the activation for the units in the next higher layer***

# BACKPROPAGATION NEURAL NETWORKS

- ***Consider a three-layer network where all activation functions are logistic functions***
- ***Back propagation can be applied to an artificial neural network with any number of hidden layers.***
- ***The training objective is to adjust the weights so that the application of a set of inputs produces the desired outputs.***


I1
Ih
Im
W11j
Whpj
Wmnj
$sum_{\cdot 1} \Rightarrow f(sum_{\cdot 1})$
$sum_{p j} \Rightarrow f(sum_{p j})$
$sum_{n j} \Rightarrow f(sum_{n j})$
W11k
Wpqk
Wnrk
$sum_{1k} \Rightarrow f(sum_{1k})$
$sum_{qk} \Rightarrow f(sum_{qk})$
$sum_{rk} \Rightarrow f(sum_{rk})$
$f(sum_{1k})$
$f(sum_{qk})$
$f(sum_{rk})$
D1
Dq
Dr
Compare
Compare
Compare
O1
Oq
Or
E1
Eq
Er
ith Layer
Index h
sum→m
m Nodes
jth Layer
Index p
sum→n
n Nodes
kth Layer
Index q
sum→r
r Nodes

# BACKPROPAGATION NEURAL NETWORKS

**Training procedure.**

- ***The network is usually trained with a large number of input-output pairs.***
  1. ***Generate weights randomly to small random values (both positive and negative) to ensure that the network is not saturated by large values of weights -(if all weights start at equal values, and the desired performance requires unequal weights, the network would not train at all).***
  2. ***Choose a training pair from the training set.***
  3. ***Apply the input vector to network input.***
  4. ***Calculate the network output.***
  5. ***Calculate the error, the difference between the network output and the desired output.***
  6. ***Step 6: Adjust the weights of the network in a way that minimizes this error.***
  7. ***Repeat steps 2-6 for each pair of input-output in the training set until the error for the entire system is acceptably low.***

## BACKPROPAGATION NEURAL NETWORKS

**Forward pass  and backward pass.**

- ***Back propagation neural network training involves two passes.***
- ***In the forward pass, the input signals moves forward from the network input to the output.***
- ***In the backward pass, the calculated error signals propagate backward through the network, where they are used to adjust the weights.***
- ***In the forward pass, the calculation of the output is carried out, layer by layer, in the forward direction. -The output of one layer is the input to the next layer.***
- ***In the reverse pass,***
  - ***The weights of the output neuron layer are adjusted first since the target value of each output neuron is available to guide the adjustment of the associated weights, using the delta rule.***
  - ***Next, we adjust the weights of the middle layers. As the middle layer neurons have no target values, it makes the problem complex***

# BACKPROPAGATION NEURAL NETWORKS

## Selection of number of hidden units.

❖ ***The number of hidden units depends on the number of input units.***

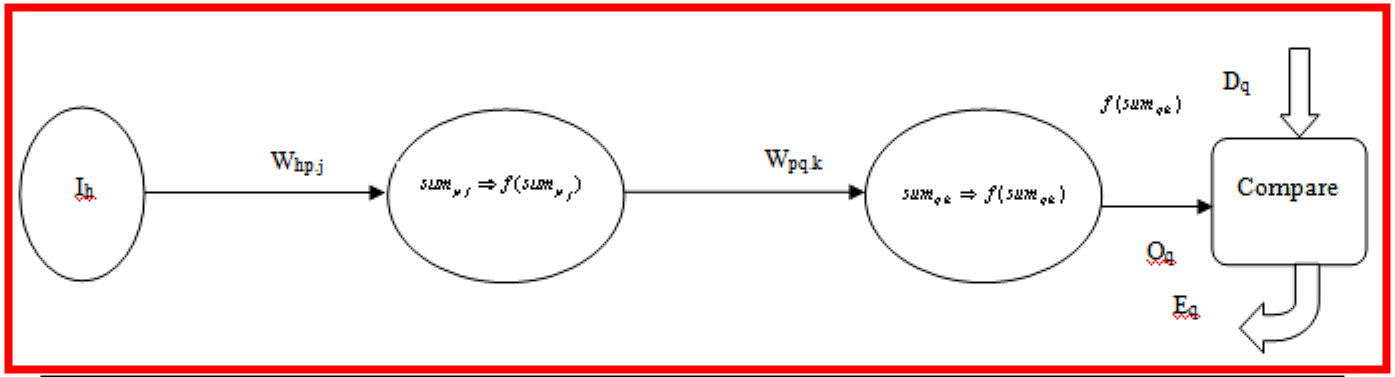
**"Any function of n variables may be represented by the superposition of a set of 2n+1 univariate functions to derive the upper bound for the required number of hidden units as one greater than twice the number of input units".**

***Kolomogorov's theorem***

1. ***Never choose h to be more than twice the number of input units.***
2. ***You can load p patterns of I elements into $\log_2 p$ hidden units. So never use more. If we need good generalization, use considerably less.***
3. ***Ensure that we must have at least 1/e times as many training examples as we have weights in our network.***
4. ***Feature extraction requires fewer hidden units than inputs.***
5. ***Learning many examples of disjointed inputs requires more hidden units than inputs.***
6. ***The number of hidden units required for a classification task increases with the number of classes in the task. Large networks require longer training times.***

# BACKPROPAGATION NEURAL NETWORKS

## Calculation of Weights for Output Layer Neurons


Ih
Whp.j
$sum_{pj} \Rightarrow f(sum_{pj})$
Wpq.k
$sum_{qk} \Rightarrow f(sum_{qk})$
$f(sum_{qk})$
Dq
Compare
Oq
Eq

***Representation of neurons for output layer neurons weight***

- ***i, j, k→ Input layer, Hidden layer output layer.***
- ***h, p, q→ Input neuron, Hidden neuron, Output neuron.***
- ***$F_{pj}$→ Output of neuron 'p' in hidden layer 'j'.***
- ***$F_{qk}$→ Output of neuron 'q' in hidden layer 'k'.***
- ***$W_{hp,j}$→ Weight connecting input neuron 'h' and hidden neuron 'p' in the hidden layer 'j'.***
- ***$W_{pq,k}$→ Weight connecting hidden neuron 'p' and output neuron 'q' in the output layer 'k'.***
- ***$D_p$→ Target output value of neuron 'q'.***

# BACKPROPAGATION NEURAL NETWORKS

## Calculation of Weights for Output Layer Neurons

❖ ***The squared error signal 'E' is produced by calculating the the difference between $D_q$ and $O_q$ ($f_{q,k}$).***

$$E^2 = E_q^2 = [D_q - f_{q.k}]^2$$

❖***By delta rule, the change in a weight is proportional to the rate of change of the square error with respect to that weight.***

$$\Delta W_{pq.k} = -\eta_{p.q} \frac{\partial E_q^2}{\partial W_{pq.k}}$$

❖***Where $\eta_{p,q}$ is the constant of proportionality called 'learning rate'***

# BACKPROPAGATION NEURAL NETWORKS

## Calculation of Weights for Hidden Layer Neurons

- ***In the hidden layer of the network (say neuron 'p'), there is no specified desired response for the neuron.***
- ***The error signal for a hidden neuron would have to be determined recursively in terms of the error signals of all the neurons to which the hidden neuron is directly connected.***
- ***Since the hidden layers have no target vectors, the problem of adjusting the weights of the hidden layers is a major issue.***
- ***Back propagation trains hidden layers by propagating the adjusted error back through the network, layer by layer, adjusting the weight of each layer as it goes.***
- ***The equations for the hidden layer are the same as for the output layer except that the error term  must be generated without a target vector.***
- ***Weight of neuron in the middle layer includes the contributions from the errors in each neuron in the output layer to which it is connected.***

## BACKPROPAGATION NEURAL NETWORKS

**Calculation of Weights for Hidden Layer Neurons**

- *The procedure for calculating $W_{hp,j}$ is substantially the same as calculating $W_{pq,k}$. Consider, the neuron (p) at layer 'j' is connected to 'r' number of neurons in output layer 'k'.*
- *Then the weight at iteration 't=t+1' is given by*

$$W_{hp.j}(t+1) = W_{hp.j}(t) + \eta_{h.p} I_h \sum_{q=1}^{r} \chi_{hp}$$

$$\chi_{hp.j} \equiv \chi_{pq.k} W_{pq.k} \frac{\partial f_{p.j}}{\partial sum_{p.j}}$$

$$\Delta W_{hp.j} = -\eta_{h.p} \frac{\partial E^2}{\partial W_{hp.j}} = \eta_{h.p} \sum_{q=1}^{r} \chi_{pq.k} W_{pq.k} \frac{\partial f_{p.j}}{\partial I_{p.j}} I_h$$

# BACKPROPAGATION NEURAL NETWORKS

## Factors influencing Back Propagation Training

***The training time can be reduced by using***

***Bias:*** *Networks with biases can represent relationships between inputs and outputs more easily than networks without biases. Adding a bias (a + 1 input with a training weight, which can be either positive or negative) to each neuron is usually desirable to offset the origin of the activation function. The weight of the bias is trainable similar to weight except that the input is always +1.*

***Momentum:*** *The use of momentum enhances the stability of the training process. Momentum is used to keep the training process going in the same general direction analogous to the way that momentum of a moving object behaves.*
*In back propagation with momentum, the weight change is a combination of the current gradient and the previous gradient.*

$$W_{pq.k}(t+1) = W_{pq.k}(t) + \Delta W_{pq.k}(t+1)$$

$$\Delta W_{pq.k}(t+1) = -\eta_{pq}\chi_{pq.k} f_{p.j} + \mu \Delta W_{pq.k}(t)$$

# BACKPROPAGATION NEURAL NETWORKS

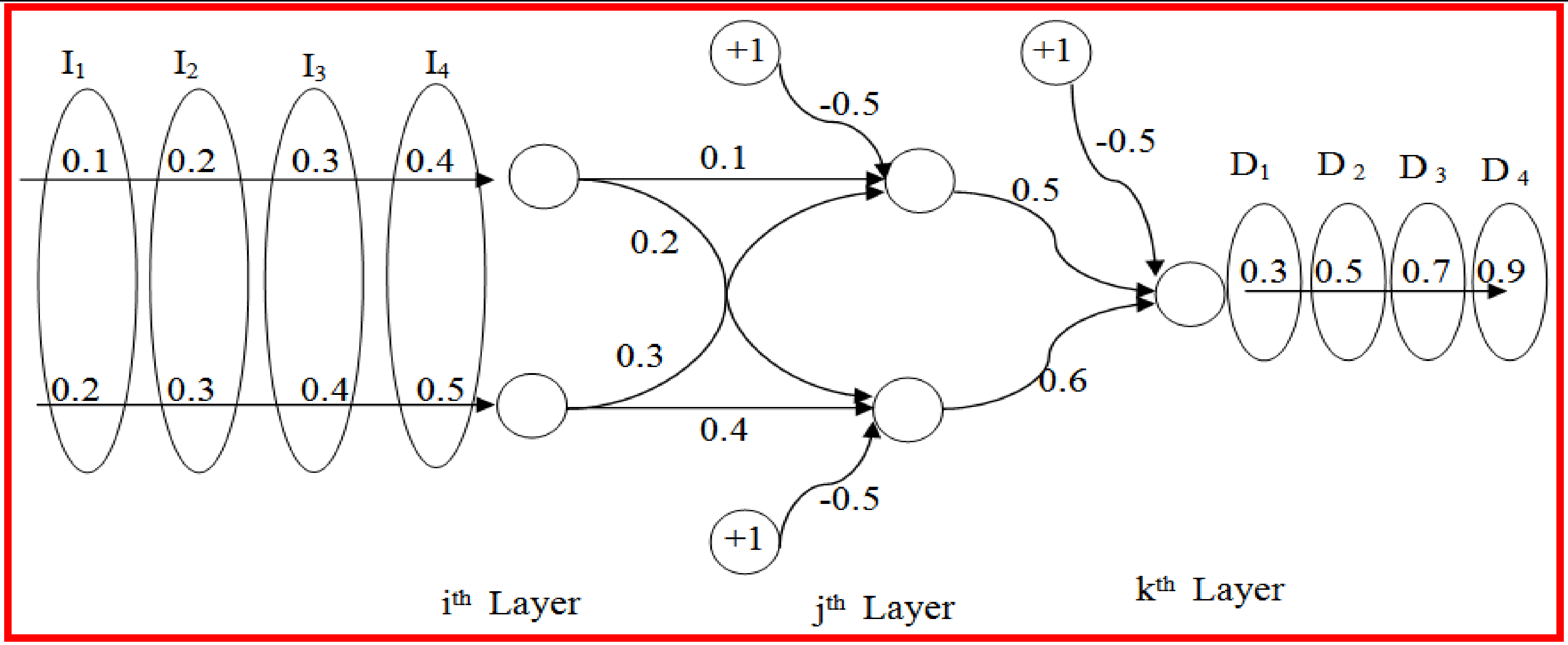
**Numerical Example 1:**

***Consider a neural network where,***

***All neurons have same logistic function with s=1;***

***Learning rate of all neurons are 1.***

***The weight and bias updation are as follows,***



***Weight updation for a simple back propagation***

# BACKPROPAGATION NEURAL NETWORKS

**Numerical Example 1:**

*The change in weights and bias values from output layer to hidden layer is given by the following equations*

$$\Delta W_{pq.k} = -\eta_{p.q}\left(-2 * s * \left(D_q - f_{q.k}\right) f_{q.k}\left(1 - f_{q.k}\right) f_{p.j}\right)$$
$$\Delta b_{q.k} = -\eta_{p.q}\left(-2 * s * \left(D_q - f_{q.k}\right) f_{q.k}\left(1 - f_{q.k}\right) * 1\right)$$

*The change in weights and bias values from hidden layer to input layer is given by the following equations*

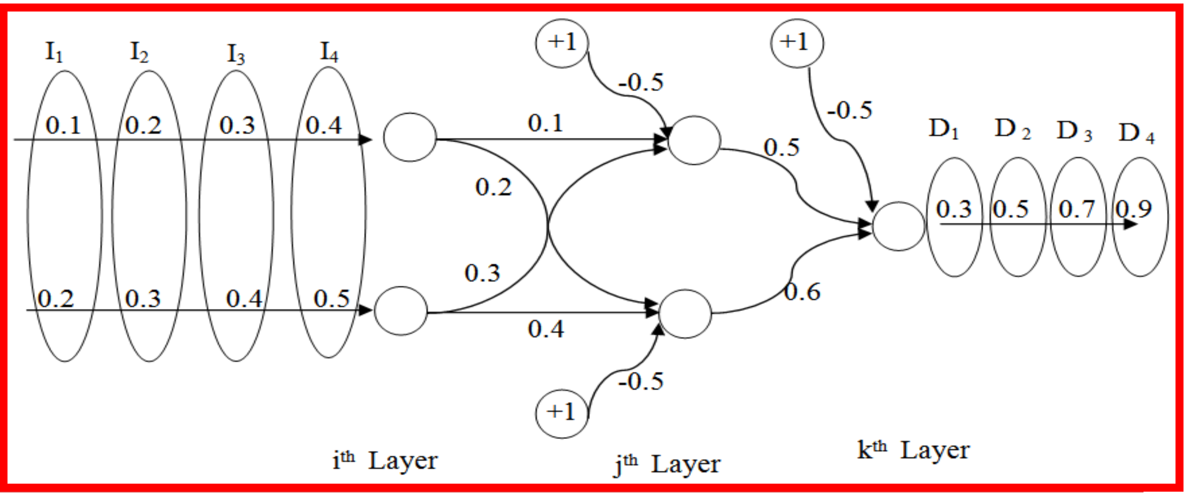
$$\Delta W_{hp.j} = -\eta_{h.p}\left[\sum_{q=1}^{r} (-2) * s * (D_q - f_{q.k})\left[f_{q.k}(1 - f_{q.k})\right] W_{pq.k} * s * \left[f_{p.j}(1 - f_{p.j})\right] I_h\right]$$
$$\Delta b_{p.j} = -\eta_{h.p}\left[\sum_{q=1}^{r} (-2) * s * (D_q - f_{q.k})\left[f_{q.k}(1 - f_{q.k})\right] W_{pq.k} * s * \left[f_{p.j}(1 - f_{p.j})\right] * 1\right]$$

# BACKPROPAGATION NEURAL NETWORKS

**Numerical Example 1:**

*The subscripts h, p, q are the index of input, hidden and output layer.*
*m, n, and r are number of neurons in input, hidden and output layer.*
*Here, m = 2, n = 2 , r=1; Assume μ =0;*

| Training pattern | |
|---|---|
| $I_1$ | $I_2$ |
| 0.10 | 0.20 |
| 0.20 | 0.30 |
| 0.30 | 0.40 |
| 0.40 | 0.50 |



# BACKPROPAGATION NEURAL NETWORKS

## Numerical Example 1:

Epoch Number 1 / Pattern Number 1

$I_1 = 0.1; I_2 = 0.2$

**Hidden layer units weighted sum and output**

$$sum_{1.j} = (0.1\times 0.1 + 0.2\times 0.3) + 1\times -0.5 = -0.43$$

$$f(sum_{1.j}) = \frac{1}{(1+e^{(-1\times sum_{1.j})})} = 0.3941$$

$$sum_{2.j} = (0.1\times 0.2 + 0.2\times 0.4) + 1\times -0.5 = -0.40$$

$$f(sum_{2.j}) = \frac{1}{(1+e^{(-1\times sum_{2.j})})} = 0.4013$$

***Output layer unit weighted sum and output***

$$sum_{1.k} = (0.3941\times 0.5 + 0.4013\times 0.6) + 1\times -0.5 = -0.0621$$

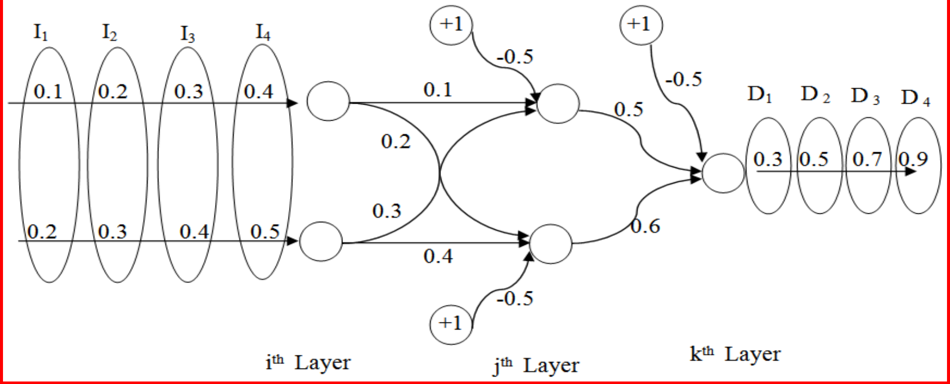
$$f(sum_{1.k}) = \frac{1}{(1+e^{(-1\times sum_{1.j})})} = 0.4845$$

$$sum_{q.k} = \sum_{p=1}^{n} W_{pq.k} f_{p.j}$$

$$f(sum) = \frac{1}{(1+e^{-s*sum})} = (1+e^{-s*sum})^{-1}$$

$$sum_{p.j} = \sum_{h=1}^{m} W_{hp.j} I_h$$

$$f(sum) = \frac{1}{(1+e^{-s*sum})} = (1+e^{-s*sum})^{-1}$$



# BACKPROPAGATION NEURAL NETWORKS

**Numerical Example 1:**

Epoch Number 1 / Pattern Number 1

$I_1 = 0.1, \ I_2 = 0.2$

***Change in weights in the output layer***

$$\Delta W_{11.k} = -0.5 \times \left(-2 \times 1 \times (0.3 - 0.4845) \times 0.4845 \times (1 - 0.4845) \times 0.3941\right) = -0.0182$$

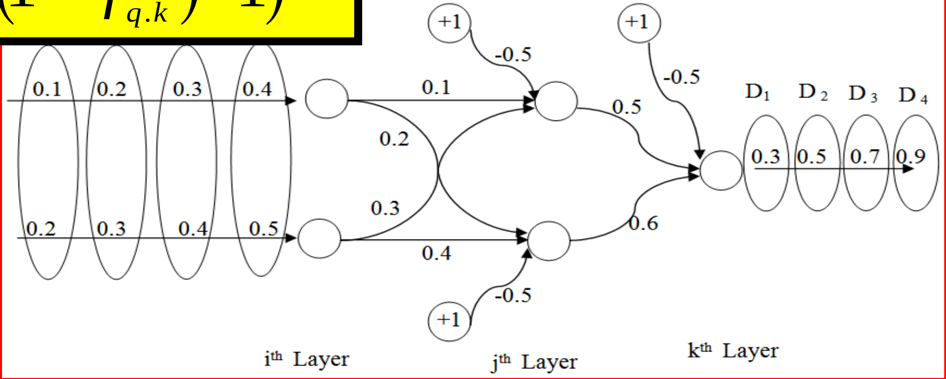
$$\Delta W_{21.k} = -0.5 \times \left(-2 \times 1 \times (0.3 - 0.4845) \times 0.4845 \times (1 - 0.4845) \times 0.4013\right) = -0.0185$$

***Change in bias in the output layer***

$$\Delta b_{1.k} = -0.5 \times \left(-2 \times 1 \times (0.3 - 0.4845) \times 0.4845 \times (1 - 0.4845) \times 1\right) = -0.0461$$

$$\Delta W_{pq.k} = -\eta_{p.q}\left(-2 * s * \left(D_q - f_{q.k}\right) f_{q.k}\left(1 - f_{q.k}\right) f_{p.j}\right)$$

$$\Delta b_{q.k} = -\eta_{p.q}\left(-2 * s * \left(D_q - f_{q.k}\right) f_{q.k}\left(1 - f_{q.k}\right) * 1\right)$$



# BACKPROPAGATION NEURAL NETWORKS

**Numerical Example 1:**

Epoch Number 1 / Pattern Number 1

$I_1 = 0.1; I_2 = 0.2$

***Change in weights in the input layer***

$$\Delta W_{11.j} = -0.5\sum_{q=1}^{r=1}\left[\begin{array}{l}(-2)\times 1\times(0.3-0.4845)\times[0.4845\times(1-0.4845)]\times 0.5\times 1\times \\ [0.3941\times(1-0.3941)]\times 0.1\end{array}\right] = \text{-0.0006}$$

$$\Delta W_{12.j} = -0.5\sum_{q=1}^{r=1}\left[\begin{array}{l}(-2)\times 1\times(0.3-0.4845)\times[0.4845\times(1-0.4845)]\times 0.6\times 1\times \\ [0.3941\times(1-0.3941)]\times 0.1\end{array}\right] = \text{-0.0007}$$

$$\Delta W_{21.j} = -0.5\sum_{q=1}^{r=1}\left[\begin{array}{l}(-2)\times 1\times(0.3-0.4845)\times[0.4845\times(1-0.4845)]\times 0.5\times 1\times \\ [0.4013\times(1-0.4013)]\times 0.2\end{array}\right] = \text{-0.0011}$$

$$\Delta W_{22.j} = -0.5\sum_{q=1}^{r=1}\left[\begin{array}{l}(-2)\times 1\times(0.3-0.4845)\times[0.4845\times(1-0.4845)]\times 0.6\times 1\times \\ [0.4013\times(1-0.4013)]\times 0.2\end{array}\right] = \text{-0.0013}$$

$$\Delta W_{hp.j} = -\eta_{h.p}\left[\sum_{q=1}^{r}(-2)*s*(D_q - f_{q.k})[f_{q.k}(1-f_{q.k})]W_{pq.k}*s*[f_{p.j}(1-f_{p.j})]I_h\right]$$

# BACKPROPAGATION NEURAL NETWORKS

**Numerical Example 1:**

Epoch Number 1 / Pattern Number 1

$I_1 = 0.1; I_2 = 0.2$

***Change in bias in the input layer***

$$\Delta b_{1.j} = -0.5\sum_{q=1}^{r=1}\begin{bmatrix}(-2)\times 1\times(0.3-0.4845)\times[0.4845\times(1-0.4845)]\times 0.5\times 1\times \\ [0.3941\times(1-0.3941)]\times 1\end{bmatrix} = \text{-}0.0055$$

$$\Delta b_{2.j} = -0.5\sum_{q=1}^{r=1}\begin{bmatrix}(-2)\times 1\times(0.3-0.4845)\times[0.4845\times(1-0.4845)]\times 0.6\times 1\times \\ [0.4013\times(1-0.4013)]\times 1\end{bmatrix} = \text{-}0.0066$$

***New weights in the output layer***

$$W_{11.k}(t+1) = 0.5 - 0.0182 = 0.4818$$

$$W_{21.k}(t+1) = 0.6 - 0.0185 = 0.5815$$

***New bias in the output layer***

$$b_{1.k} = -0.5 - 0.0461 = \text{-}0.5461$$

***New bias in the input layer***

$$b_{1.j} = -0.5 - 0.0055 = \text{-}0.5055$$

$$\text{b}_{2.j} = -0.5 - 0.0066 = \text{-}0.5066$$

***New weights in the input layer***

$$W_{11.j} = 0.1 - 0.0006 = 0.0994 \,;\, W_{12.j} = 0.2 - 0.0007 = 0.1993$$

$$W_{21.j} = 0.3 - 0.0011 = 0.2989; W_{22.j} = 0.4 - 0.0013 = 0.3987$$

$$\Delta W_{hp.j} = -\eta_{h.p}\left[\sum_{q=1}^{r}(-2)*s*(D_q - f_{q.k})[f_{q.k}(1-f_{q.k})]W_{pq.k}*s*[f_{p.j}(1-f_{p.j})]I_h\right]$$

# BACKPROPAGATION NEURAL NETWORKS

## Numerical Example 1:

Epoch Number 1 / Pattern Number 2

$I_1 = 0.2; I_2 = 0.3$

### Hidden layer units weighted sum and output

$$sum_{1.j} = (0.2 \times 0.0994 + 0.3 \times 0.2989) + 1 \times -0.5055 = -0.3959$$

$$f(sum_{1j}) = \frac{1}{\left(1+e^{(-1 \times sum_{1.j})}\right)} = 0.4023$$

$$sum_{2.j} = (0.2 \times 0.1993 + 0.3 \times 0.3987) + 1 \times -0.5066 = -0.3472$$

$$f(sum_{2j}) = \frac{1}{\left(1+e^{(-1 \times sum_{2.j})}\right)} = 0.4141$$

### Output layer unit weighted sum and output

$$sum_{1.k} = (0.4023 \times 0.4818 + 0.4141 \times 0.5815) + 1 \times -0.5461 = -0.1114$$

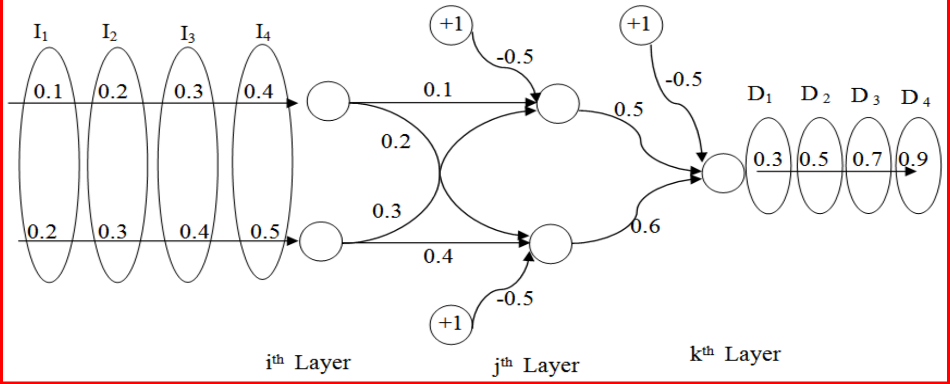
$$f(sum_{1.k}) = \frac{1}{\left(1+e^{(-1 \times sum_{1.j})}\right)} = 0.4722$$

$$sum_{q.k} = \sum_{p=1}^{n} W_{pq.k} f_{p.j}$$

$$f(sum) = \frac{1}{(1+e^{-s*sum})} = (1+e^{-s*sum})^{-1}$$

$$sum_{p.j} = \sum_{h=1}^{m} W_{hp.j} I_h$$

$$f(sum) = \frac{1}{(1+e^{-s*sum})} = (1+e^{-s*sum})^{-1}$$



# BACKPROPAGATION NEURAL NETWORKS

**Numerical Example 1:**

Epoch Number 1 / Pattern Number 2

$I_1 = 0.2,\ I_2 = 0.3$

***Change in weights in the output layer***

$$\Delta W_{11.k} = -0.5 \times \left(-2 \times 1 \times (0.5 - 0.4722) \times 0.4722 \times (1 - 0.4722) \times 0.4023\right) = 0.0028$$

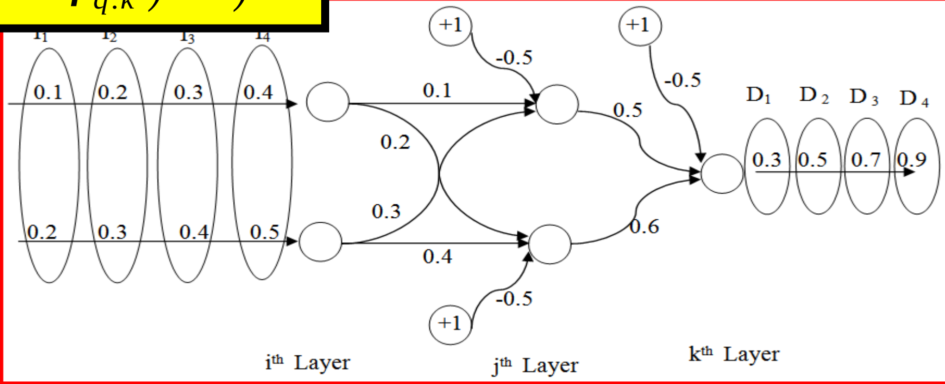
$$\Delta W_{21.k} = -0.5 \times \left(-2 \times 1 \times (0.5 - 0.4722) \times 0.4722 \times (1 - 0.4722) \times 0.4141\right) = 0.0029$$

***Change in bias in the output layer***

$$\Delta b_{1.k} = -0.5 \times \left(-2 \times 1 \times (0.5 - 0.4722) \times 0.4722 \times (1 - 0.4722) \times 1\right) = 0.0069$$

$$\Delta W_{pq.k} = -\eta_{p.q}\left(-2 * s * \left(D_q - f_{q.k}\right) f_{q.k}\left(1 - f_{q.k}\right) f_{p.j}\right)$$

$$\Delta b_{q.k} = -\eta_{p.q}\left(-2 * s * \left(D_q - f_{q.k}\right) f_{q.k}\left(1 - f_{q.k}\right) * 1\right)$$



# BACKPROPAGATION NEURAL NETWORKS

**Numerical Example 1:**

Epoch Number 1 / Pattern Number 2

$I_1 = 0.2; I_2 = 0.3$

***Change in weights in the input layer***

$$\Delta W_{11.j} = -0.5\sum_{q=1}^{r=1}\left[\begin{matrix}(-2)\times 1\times(05-0.4722)\times[0.4722\times(1-0.4722)]\times 0.4818\times 1\times \\ [0.4023\times(1-0.4023)]\times 0.2\end{matrix}\right] = \text{-}0.0001607$$

$$\Delta W_{12.j} = -0.5\sum_{q=1}^{r=1}\left[\begin{matrix}(-2)\times 1\times(0.5-0.4722)\times[0.4722\times(1-0.4722)]\times 0.5815\times 1\times \\ [0.4141\times(1-0.4141)]\times 0.2\end{matrix}\right] = \text{-}0.0001957$$

$$\Delta W_{21.j} = -0.5\sum_{q=1}^{r=1}\left[\begin{matrix}(-2)\times 1\times(0.5-0.4722)\times[0.4722\times(1-0.4722)]\times 0.4818\times 1\times \\ [0.4023\times(1-0.4023)]\times 0.3\end{matrix}\right] = 0.0002411$$

$$\Delta W_{22.j} = -0.5\sum_{q=1}^{r=1}\left[\begin{matrix}(-2)\times 1\times(0.5-0.4722)\times[0.4722\times(1-0.4722)]\times 0.5815\times 1\times \\ [0.4141\times(1-0.4141)]\times 0.3\end{matrix}\right] = 0.000293$$

$$\Delta W_{hp.j} = -\eta_{h.p}\left[\sum_{q=1}^{r}(-2)*s*(D_q - f_{q.k})[f_{q.k}(1-f_{q.k})]W_{pq.k}*s*[f_{p.j}(1-f_{p.j})]I_h\right]$$

# BACKPROPAGATION NEURAL NETWORKS

**Numerical Example 1:**

Epoch Number 1 / Pattern Number 2

$I_1 = 0.2; I_2 = 0.3$

***Change in bias in the input layer***

$$\Delta b_{1.j} = -0.5\sum_{q=1}^{r=1}\begin{bmatrix}(-2)\times 1\times(0.5-0.4722)\times[0.4722\times(1-0.4722)]\times 0.4818\times 1\times \\ [0.4023\times(1-0.4023)]\times 1\end{bmatrix} = 0.0008037$$

$$\Delta b_{2.j} = -0.5\sum_{q=1}^{r=1}\begin{bmatrix}(-2)\times 1\times(0.5-0.4722)\times[0.4722\times(1-0.4722)]\times 0.5818\times 1\times \\ [0.4141\times(1-0.4141)]\times 1\end{bmatrix} = 0.0009787$$

***New weights in the output layer***

$W_{11.k}(t+1) = 0.4818 + 0.0028 = 0.4846$

$W_{21.k}(t+1) = 0.5815 + 0.0029 = 0.5844$

***New bias in the output layer***

$b_{1.k} = -0.5461 + 0.0069 = -0.5391$

***New bias in the input layer***

$b_{1.j} = -0.5055 - 0.0008037 = -0.5047$

$b_{2.j} = -0.5 - 0.0066 = -0.5066$

***New weights in the input layer***

$W_{11.j} = 0.0994 - 0.0001607 = 0.0996$ ; $W_{12.j} = 0.1993 - 0.0001957 = 0.1995$

$W_{21.j} = 0.2989 - 0.0002411 = 0.2991$; $W_{22.j} = 0.3987 - 0.0008037 = 0.3990$

$$\Delta W_{hp.j} = -\eta_{h.p}\left[\sum_{q=1}^{r}(-2)*s*(D_q - f_{q.k})[f_{q.k}(1-f_{q.k})]W_{pq.k}*s*[f_{p.j}(1-f_{p.j})]I_h\right]$$

# BACKPROPAGATION NEURAL NETWORKS

## Numerical Example 1:

Epoch Number 1 / Pattern Number 3

$$I_1 = 0.3; I_2 = 0.4$$

### Hidden layer units weighted sum and output

$$sum_{1.j} = (0.3\times 0.0996 + 0.4\times 0.2991) + 1\times -0.5047 = \text{-}0.3552$$

$$\mathrm{f}(\mathrm{sum}_{1.j}) = \frac{1}{\left(1+e^{(-1\times sum_{1.j})}\right)} = 0.4121$$

$$sum_{2.j} = (0.3\times 0.1995 + 0.4\times 0.3990) + 1\times -0.5066 = \text{-}0.2862$$

$$\mathrm{f}(\mathrm{sum}_{2.j}) = \frac{1}{\left(1+e^{(-1\times sum_{2.j})}\right)} = 0.4289$$

### Output layer unit weighted sum and output

$$sum_{1.k} = (0.4121\times 0.4846 + 0.4289\times 0.5844) + 1\times -0.5391 = \text{-}0.0887$$

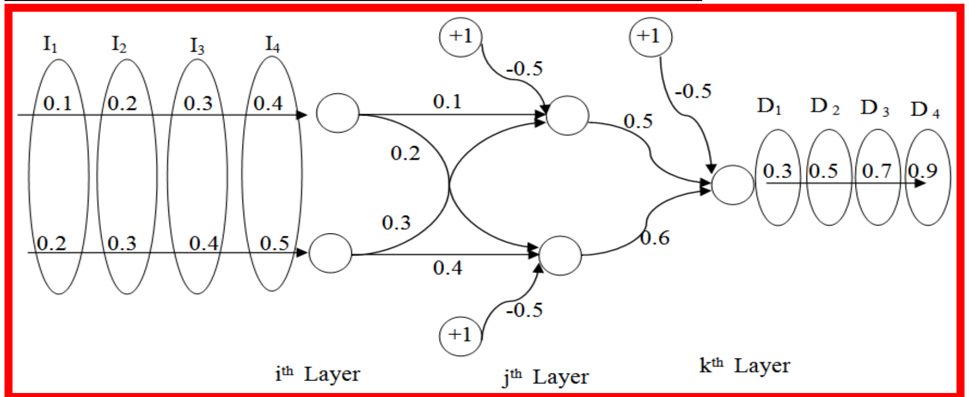
$$\mathrm{f}(\mathrm{sum}_{1.k}) = \frac{1}{\left(1+e^{(-1\times sum_{1.j})}\right)} = 0.4778$$

$$sum_{q.k} = \sum_{p=1}^{n} W_{pq.k}\, f_{p.j}$$

$$f(sum) = \frac{1}{(1+e^{-s*sum})} = (1+e^{-s*sum})^{-1}$$

$$sum_{p.j} = \sum_{h=1}^{m} W_{hp.j} I_h$$

$$f(sum) = \frac{1}{(1+e^{-s*sum})} = (1+e^{-s*sum})^{-1}$$



# BACKPROPAGATION NEURAL NETWORKS

**Numerical Example 1:**

Epoch Number 1 / Pattern Number 3

$I_1 = 0.3; I_2 = 0.4$

***Change in weights in the output layer***

$$\Delta W_{11.k} = -0.5\times\left(-2\times 1\times(0.7-0.4778)\times 0.4778\times(1-0.4778)\times 0.4121\right) = 0.0228$$

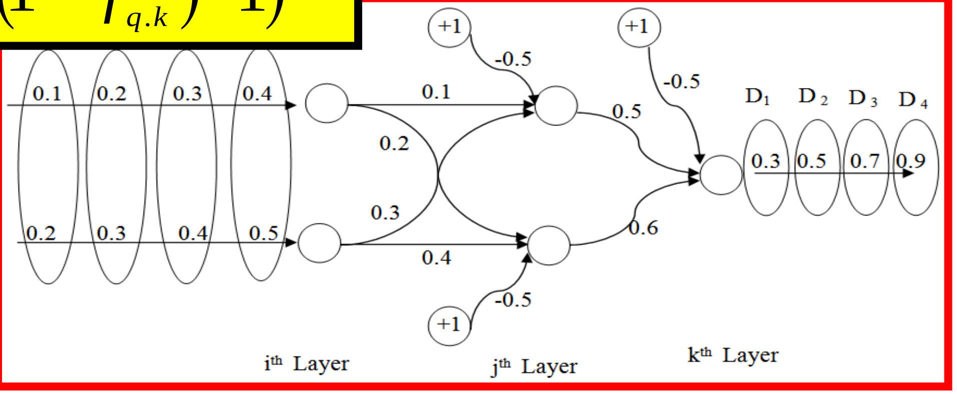
$$\Delta W_{21.k} = -0.5\times\left(-2\times 1\times(0.7-0.4778)\times 0.4778\times(1-0.4778)\times 0.4289\right) = 0.0238$$

***Change in bias in the output layer***

$$\Delta b_{1.k} = -0.5\times\left(-2\times 1\times(0.7-0.4778)\times 0.4778\times(1-0.4778)\times 1\right) = 0.0554$$

$$\Delta W_{pq.k} = -\eta_{p.q}\left(-2*s*\left(D_q - f_{q.k}\right)f_{q.k}\left(1-f_{q.k}\right)f_{p.j}\right)$$

$$\Delta b_{q.k} = -\eta_{p.q}\left(-2*s*\left(D_q - f_{q.k}\right)f_{q.k}\left(1-f_{q.k}\right)*1\right)$$



# BACKPROPAGATION NEURAL NETWORKS

**Numerical Example 1:**

Epoch Number 1 / Pattern Number 3

$I_1 = 0.3; I_2 = 0.4$

***Change in weights in the input layer***

$$\Delta W_{11.j} = -0.5\sum_{q=1}^{r=1}\left[\begin{array}{l}(-2)\times 1\times(0.7-0.4778)\times[0.4778\times(1-0.4778)]\times 0.4846\times 1\times\\ [0.4121\times(1-0.4121)]\times 0.3\end{array}\right] = 0.0020$$

$$\Delta W_{12.j} = -0.5\sum_{q=1}^{r=1}\left[\begin{array}{l}(-2)\times 1\times(0.7-0.4778)\times[0.4778\times(1-0.4778)]\times 0.5844\times 1\times\\ [0.4289\times(1-0.4289)]\times 0.3\end{array}\right] = 0.0024$$

$$\Delta W_{21.j} = -0.5\sum_{q=1}^{r=1}\left[\begin{array}{l}(-2)\times 1\times(0.7-0.4778)\times[0.4778\times(1-0.4778)]\times 0.4846\times 1\times\\ [0.4121\times(1-0.4121)]\times 0.4\end{array}\right] = 0.0026$$

$$\Delta W_{22.j} = -0.5\sum_{q=1}^{r=1}\left[\begin{array}{l}(-2)\times 1\times(0.7-0.4778)\times[0.4778\times(1-0.4778)]\times 0.5844\times 1\times\\ [0.4289\times(1-0.4289)]\times 0.4\end{array}\right] = 0.0032$$

$$\Delta W_{hp.j} = -\eta_{h.p}\left[\sum_{q=1}^{r}(-2)*s*(D_q - f_{q.k})[f_{q.k}(1-f_{q.k})]W_{pq.k}*s*[f_{p.j}(1-f_{p.j})]I_h\right]$$

# BACKPROPAGATION NEURAL NETWORKS

**Numerical Example 1:**

Epoch Number 1 / Pattern Number 3

$I_1 = 0.3; I_2 = 0.4$

**Change in bias in the input layer**

$$\Delta b_{1.j} = -0.5\sum_{q=1}^{r=1}\begin{bmatrix}(-2)\times 1\times(0.7-0.4778)\times\left[0.4778\times(1-0.4778)\right]\times 0.4846\times 1\times \\ \left[0.4121\times(1-0.4121)\right]\times 1\end{bmatrix} = 0.0065$$

$$\Delta b_{2.j} = -0.5\sum_{q=1}^{r=1}\begin{bmatrix}(-2)\times 1\times(0.7-0.4778)\times\left[0.4778\times(1-0.4778)\right]\times 0.5844\times 1\times \\ \left[0.4289\times(1-0.4289)\right]\times 1\end{bmatrix} = 0.0079$$

**New weights in the output layer**

$$W_{11.k}(t+1) = 0.4846 + 0.0228 = 0.5075$$

$$W_{21.k}(t+1) = 0.5844 + 0.0238 = 0.6082$$

**New bias in the input layer**

$$b_{1.j} = -0.5047 + 0.0065 = -0.4021$$

$$b_{2.j} = -0.5066 + 0.0079 = -0.4977$$

**New bias in the output layer**

$$b_{1.k} = -0.5391 - 0.0554 = -0.4837$$

**New weights in the input layer**

$$W_{11.j} = 0.0996 + 0.0020 = 0.1016 \; ; \; W_{12.j} = 0.1995 - 0.0024 = 0.2019;$$

$$W_{21.j} = 0.2991 + 0.0026 = 0.3017; \; W_{22.j} = 0.3990 + 0.0032 = 0.4021$$

$$\Delta W_{hp.j} = -\eta_{h.p}\left[\sum_{q=1}^{r}(-2)*s*(D_q - f_{q.k})\left[f_{q.k}(1-f_{q.k})\right]W_{pq.k}*s*\left[f_{p.j}(1-f_{p.j})\right]I_h\right]$$

# BACKPROPAGATION NEURAL NETWORKS

**Numerical Example 1:**

Epoch Number 1 / Pattern Number 4

$I_1 = 0.4; I_2 = 0.5$

**Hidden layer units weighted sum and output**

$$sum_{1.j} = (0.4\times 0.1016 + 0.5\times 0.3017) + 1\times -0.4021 = \text{-}0.3067$$

$$\text{f}(\text{sum}_{1\text{j}}) = \frac{1}{\left(1+e^{(-1\times sum_{1.j})}\right)} = 0.4239$$

$$sum_{2.j} = (0.4\times 0.2019 + 0.5\times 0.4021) + 1\times -0.4977 = \text{-}0.215$$

$$\text{f}(\text{sum}_{2\text{j}}) = \frac{1}{\left(1+e^{(-1\times sum_{2.j})}\right)} = 0.4462$$

**Output layer unit weighted sum and output**

$$sum_{1.k} = (0.4239\times 0.5075 + 0.4462\times 0.6082) + 1\times -0.4837 = 0.0028$$

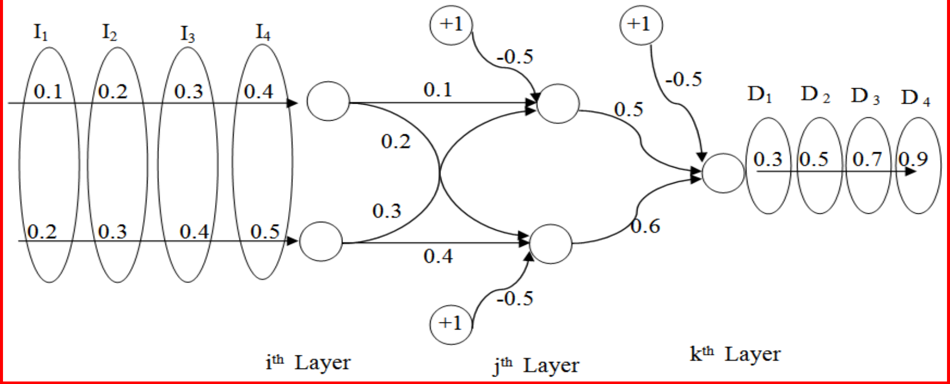
$$\text{f}(\text{sum}_{1\text{.k}}) = \frac{1}{\left(1+e^{(-1\times sum_{1.j})}\right)} = 0.5007$$

$$sum_{q.k} = \sum_{p=1}^{n} W_{pq.k}\, f_{p.j}$$

$$f(sum) = \frac{1}{(1+e^{-s*sum})} = (1+e^{-s*sum})^{-1}$$

$$sum_{p.j} = \sum_{h=1}^{m} W_{hp.j}\, I_h$$

$$f(sum) = \frac{1}{(1+e^{-s*sum})} = (1+e^{-s*sum})^{-1}$$



# BACKPROPAGATION NEURAL NETWORKS

**Numerical Example 1:**

Epoch Number 1 / Pattern Number 4

$I_1 = 0.4; I_2 = 0.5$

***Change in weights in the output layer***

$$\Delta W_{11.k} = -0.5 \times \left(-2 \times 1 \times (0.9 - 0.5007) \times 0.5007 \times (1 - 0.5007) \times 0.4239\right) = 0.0423$$

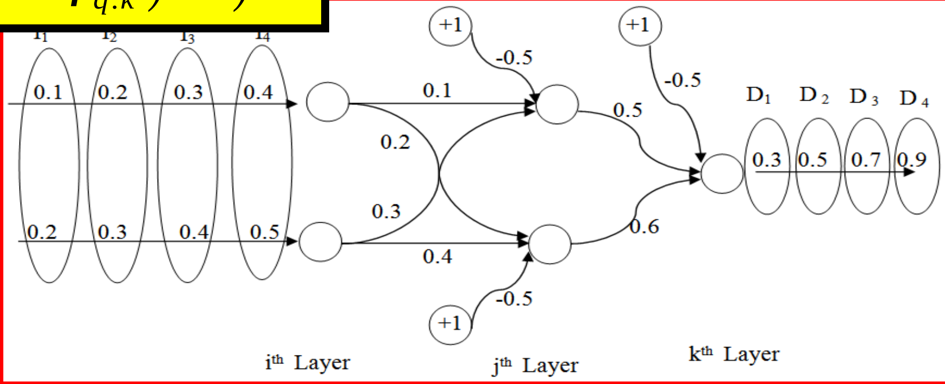
$$\Delta W_{21.k} = -0.5 \times \left(-2 \times 1 \times (0.9 - 0.5007) \times 0.5007 \times (1 - 0.5007) \times 0.4462\right) = 0.0445$$

***Change in bias in the output layer***

$$\Delta b_{1.k} = -0.5 \times \left(-2 \times 1 \times (0.9 - 0.5007) \times 0.5007 \times (1 - 0.5007) \times 1\right) = 0.0998$$

$$\Delta W_{pq.k} = -\eta_{p.q}\left(-2 * s * \left(D_q - f_{q.k}\right) f_{q.k}\left(1 - f_{q.k}\right) f_{p.j}\right)$$

$$\Delta b_{q.k} = -\eta_{p.q}\left(-2 * s * \left(D_q - f_{q.k}\right) f_{q.k}\left(1 - f_{q.k}\right) * 1\right)$$



# BACKPROPAGATION NEURAL NETWORKS

**Numerical Example 1:**

Epoch Number 1 /
Pattern Number 4

$I_1 = 0.4; I_2 = 0.5$

***Change in weights in the input layer***

$$\Delta W_{11.j} = -0.5\sum_{q=1}^{r=1}\left[\begin{array}{l}(-2)\times 1\times(0.9-0.5007)\times[0.5007\times(1-0.5007)]\times 0.5075\times 1\times\\ [0.4239\times(1-0.4239)]\times 0.4\end{array}\right] = 0.0049$$

$$\Delta W_{12.j} = -0.5\sum_{q=1}^{r=1}\left[\begin{array}{l}(-2)\times 1\times(0.9-0.5007)\times[0.5007\times(1-0.5007)]\times 0.6082\times 1\times\\ [0.4462\times(1-0.4462)]\times 0.4\end{array}\right] = 0.0060$$

$$\Delta W_{21.j} = -0.5\sum_{q=1}^{r=1}\left[\begin{array}{l}(-2)\times 1\times(0.9-0.5007)\times[0.5007\times(1-0.5007)]\times 0.5075\times 1\times\\ [0.4239\times(1-0.4239)]\times 0.5\end{array}\right] = 0.0062$$

$$\Delta W_{22.j} = -0.5\sum_{q=1}^{r=1}\left[\begin{array}{l}(-2)\times 1\times(0.9-0.5007)\times[0.5007\times(1-0.5007)]\times 0.6082\times 1\times\\ [0.4462\times(1-0.4462)]\times 0.5\end{array}\right] = 0.0075$$

$$\Delta W_{hp.j} = -\eta_{h.p}\left[\sum_{q=1}^{r}(-2)*s*(D_q - f_{q.k})[f_{q.k}(1-f_{q.k})]W_{pq.k}*s*[f_{p.j}(1-f_{p.j})]I_h\right]$$

# BACKPROPAGATION NEURAL NETWORKS

**Numerical Example 1:**

Epoch Number 1 / Pattern Number 4

$I_1 = 0.4; I_2 = 0.5$

***Change in bias in the input layer***

$$\Delta b_{1.j} = -0.5\sum_{q=1}^{r=1}\begin{bmatrix}(-2)\times 1\times(0.9-0.5007)\times\left[0.5007\times(1-0.5007)\right]\times 0.5075\times 1\times \\ \left[0.4239\times(1-0.4239)\right]\times 1\end{bmatrix} = 0.0124$$

$$\Delta b_{2.j} = -0.5\sum_{q=1}^{r=1}\begin{bmatrix}(-2)\times 1\times(0.9-0.5007)\times\left[0.5007\times(1-0.5007)\right]\times 0.6082\times 1\times \\ \left[0.4462\times(1-0.4462)\right]\times 1\end{bmatrix} = 0.0150$$

***New weights in the output layer***

$W_{11.k}(t+1) = 0.5075 + 0.0423 = 0.5498$

$W_{21.k}(t+1) = 0.6082 + 0.0445 = 0.6527$

***New bias in the output layer***

$b_{1.k} = -0.4837 + 0.0998 = -0.3839$

***New bias in the input layer***

$b_{1.j} = -0.4021 + 0.0124 = -0.4858$

$b_{2.j} = -0.4977 + 0.0150 = -0.4827$

***New weights in the input layer***

$W_{11.j} = 0.1016 + 0.0049 = 0.1065\ ; W_{12.j} = 0.2019 + 0.0060 = 0.2079$

$W_{21.j} = 0.3017 + 0.0062 = 0.3079; W_{22.j} = 0.4021 - 0.0075 = 0.4096$

$$\Delta W_{hp.j} = -\eta_{h.p}\left[\sum_{q=1}^{r}(-2)*s*(D_q - f_{q.k})\left[f_{q.k}(1-f_{q.k})\right]W_{pq.k}*s*\left[f_{p.j}(1-f_{p.j})\right]I_h\right]$$

# BACKPROPAGATION NEURAL NETWORKS

## Numerical Example 1:

***Sum Squared Error (Epoch Number 1)***

$$sse = \left( \sum_{Pattern \quad Number(PN)=1}^{4} \left( \sum_{q=1}^{r=1} \left(D_q - f_{q.k}\right)^2 \right) \right)$$

$$= (0.3 - 0.4845)^2 + (0.5 - 0.4722)^2 + (0.7 - 0.4778)^2 + (0.9 - 0.5007)^2$$

$$= 0.2436$$

***Sum Squared Error (Epoch Number 1)***

***The training should be carried out for more number of epochs to reduce the sum squared error, and thereby the accuracy of the test output will be improved.***

***After training for 10,000 epochs with a sum squared error of 7.6902e-004. The final weights and bias values that are obtained after training is given below.***

| Output layer weights and bias | | | Input layer weights and biases | | | | | |
|---|---|---|---|---|---|---|---|---|
| $W_{11}$ | $W_{12\cdot1}$ | $b_{1.k}$ | $W_{11\cdot j}$ | $W_{12\cdot j}$ | $W_{21\cdot j}$ | $W_{22\cdot j}$ | $b_{1.j}$ | $b_{2.j}$ |
| 3.6357 | 4.9436 | -2.3061 | 2.3589 | 3.1077 | 2.4296 | 3.0606 | -1.7930 | -2.9714 |

# BACKPROPAGATION NEURAL NETWORKS

## Numerical Example 1:

***After the training is over, the following are the results***

| Test input | | Desired output | Actual/Predicted output |
|---|---|---|---|
| | | | |
| 0.1 | 0.2 | 0.3000 | 0.3071 |
| 0.2 | 0.3 | 0.5000 | 0.4855 |
| 0.3 | 0.4 | 0.7000 | 0.7162 |
| 0.4 | 0.5 | 0.9000 | 0.8878 |
| 0.4 | 0.3 | 0.7000 | 0.7159 |
| 0.2 | 0.5 | 0.7000 | 0.7165 |
| 0.18 | 0.32 | 0.5000 | 0.4856 |
| 0.367 | 0.438 | 0.8050 | 0.8196 |
| 0.463 | 0.333 | 0.7960 | 0.8117 |
| 0.345 | 0.543 | 0.8880 | 0.8806 |

- ***During the testing mode, the test inputs are given from the input patterns already present in the training set, and the output results obtained are closer to the desired one.***
- ***After training, Even if we give a typical or similar kind of input pattern not present in the training set, the neural network is capable of giving an output which is closer to the desired target pattern.***
- ***This shows the adaptability of the neural network for similar kind of input patterns that are not present in the training process.***

## BACKPROPAGATION NEURAL NETWORKS

***Character Recognition using Back propagation Neural Network***

***Character recognition is a trivial task for humans, however to make a computer program that does character recognition is extremely difficult.***

***The main reason may be the many sources of variability and high level of abstraction.***

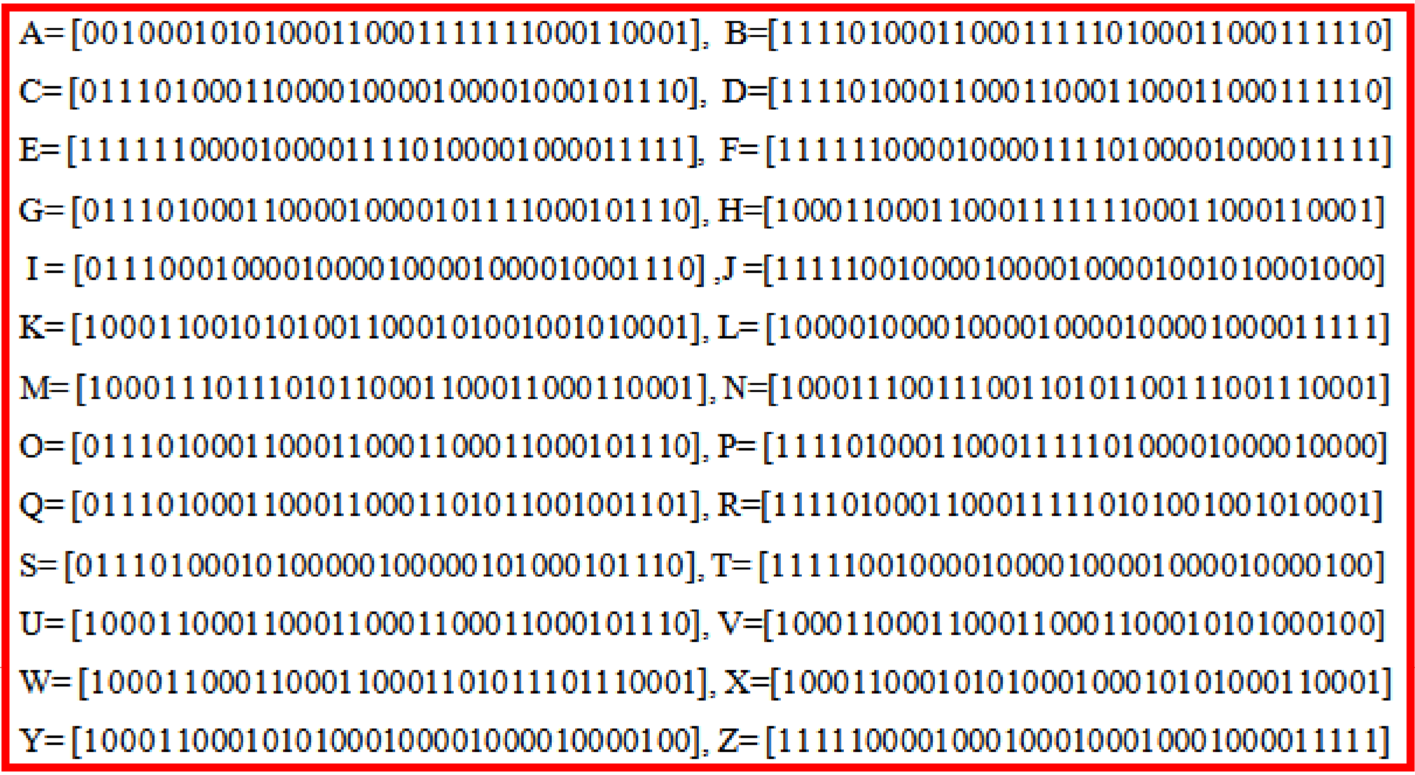
***Variability***
***Noise for example, consists of random changes to a pattern, particularly near the edges and a character with much noise may be interpreted as a completely different character by a computer program.***

***High level of abstraction***
***There are thousands styles of type in common use and a character recognition program must recognize most of these to be of any use.***

# BACKPROPAGATION NEURAL NETWORKS

***Character Recognition using Back propagation Neural Network***

***Alphabets from A to Z are used for training, and have been tested with error incorporated in the test pattern. The alphabet is represented using a 7 X 5 matrix of 35 binary bits as shown below.***



A= [00100010101000110001111111000110001], B=[11110100011000111110100011000111110]

C= [01110100011000010000100001000101110], D=[11110100011000110001100011000111110]

E= [11111100001000011110100001000011111], F= [11111100001000011110100001000011111]

G= [01110100011000010000101111000101110], H=[10001100011000111111100011000110001]

I = [01110001000010000100001000010001110] ,J =[11111001000010000100001001010001000]

K= [10001100101010011000101001001010001], L= [10000100001000010000100001000011111]

M= [10001110111010110001100011000110001], N=[10001110011100110101100111001110001]

O= [01110100011000110001100011000101110], P= [11110100011000111110100001000010000]

Q= [01110100011000110001101011001001101], R=[11110100011000111110101001001010001]

S= [01110100010100000100000101000101110], T= [11111001000010000100001000010000100]

U= [10001100011000110001100011000101110], V=[10001100011000110001100010101000100]

W= [10001100011000110001101011101110001], X=[10001100010101000100010101000110001]

Y= [10001100010101000100001000010000100], Z= [11111000010001000100010001000011111]

# BACKPROPAGATION NEURAL NETWORKS

## *Character Recognition using Back propagation Neural Network*

A

```
0 0 1 0 0
0 1 0 1 0
1 0 0 0 1
1 0 0 0 1
1 1 1 1 1
1 0 0 0 1
1 0 0 0 1
```

Z

```
1 1 1 1 1
0 0 0 0 1
0 0 0 1 0
0 0 1 0 0
0 1 0 0 0
1 0 0 0 0
1 1 1 1 1
```

- ***The forward back propagation neural network is designed with 35 input and output units. The training set consists of 26 patterns.***
- ***Sigmoidal logistic function is used for all neurons, during the testing mode, the actual output of the neural network is passed through a binary logic to get binary output.***
- ***Once the training is carried of successful with less sum squared error, the network is tested to recognize the patterns with and without error.***
- ***The accuracy of the predicted output depends upon the increased epochs of training and selection of suitable network parameters. However, there are limitations that correspond to back propagation neural network and the limitations of the learning rules that used for training.***
- ***The accuracy of the output can be increased by increasing the number of elements in the training patterns since they should have at least a minimal difference between individual training patterns. This will enable the network to generalize and train effectively.***

# KOHONEN NEURAL NETWORK (KNN)

- ***In 1989, Finnish professor Teuvo Kohonen had developed a topological structure analogous to a typical neural network with competitive units or cluster units in network layers. This topology uses an unsupervised learning procedure to produce a 2-dimensional discretized representation of the input space of the training samples, called a map. Therefore, this network is called 'self-organizing map' or simply a 'Kohonen neural network'.***

- ***Kohonen neural network creates a competition among cluster units similar to a property observed in the brain but not in other artificial neural networks.***

- ***Clustering progresses by checking the closeness of the input patterns with the weight vector associated with each of the cluster units. A cluster unit is considered as a winner, if the Euclidean distance between the weight vector associated with it and the given input pattern is the minimum when compared among the other neighbour-hood cluster units.***

- ***The weights associated with the winner cluster unit and neighbour cluster units are updated. The neighbours are the cluster units nearer to the winner cluster unit and can be considered based on a measure of geometrical boundary.***

## KOHONEN NEURAL NETWORK (KNN)

***Step1: Initialize the random weight values. Give the topological parameter R (geometric measure of the neighborhood boundary), and set the learning rate within .***

***Step2: For each input pattern $I_h$ training pair compute the Euculidean distance for each output cluster unit k and Get the winner cluster unit index K for which the ED is minimum.***

$$\boxed{ED(k) = \sum_{h=1:n} \left(w_{hk} - I_h\right)^2}$$

***Step3: Update the weights for all the k units within the neighbourhood boundary of the winner K. Then, update the learning rate. Decrease the topological parameter R at specified times after the completion of an epoch.***

$$\boxed{w_{hk}(t+1) = w_{hk}(t) + \eta * \left(x_h - w_{hk}(t)\right)}$$

***Repeat Steps 2,3 till the maximum number of epochs are reached.***

# KOHONEN NEURAL NETWORK (KNN)

***Step1: Initialize the random weight values. Give the topological parameter R (geometric measure of the neighborhood boundary), and set the learning rate within .***

***Step2: For each input pattern $I_h$ training pair compute the Euculidean distance for each output cluster unit k and Get the winner cluster unit index K for which the ED is minimum.***

$$\boxed{ED(k) = \sum_{h=1:n} \left(w_{hk} - I_h\right)^2}$$

***Step3: Update the weights for all the k units within the neighbourhood boundary of the winner K. Then, update the learning rate. Decrease the topological parameter R at specified times after the completion of an epoch.***

$$\boxed{w_{hk}(t+1) = w_{hk}(t) + \eta * \left(x_h - w_{hk}(t)\right)}$$

***Repeat Steps 2,3 till the maximum number of epochs are reached.***

# KOHONEN NEURAL NETWORK (KNN)

## *Illustration on Clustering of Bipolar Input Patterns*

***<u>Cluster 4 bipolar patterns into 2 clusters</u>***

***Consider***
***n_=_4***
***m_=_2***
***Topological parameter R_=_0,***
***learning rate (n)=0.9***
***geometrically decrease 0.5 times for every epoch.***

***The input pattern(I),***
***Initial Weights(W) .***

$$I = \begin{array}{c} \left[\begin{array}{cccc} 1 & 1 & 1 & -1 \\ -1 & -1 & -1 & 1 \\ 1 & -1 & -1 & -1 \\ -1 & -1 & 1 & 1 \end{array}\right] \end{array} \begin{array}{c} (I_1) \\ (I_2) \\ (I_3) \\ (I_3) \end{array}$$

$$W = \begin{bmatrix} 0.2 & 0.8 \\ 0.6 & 0.4 \\ 0.5 & 0.7 \\ 0.9 & 0.3 \end{bmatrix}_{n\ X\ m}$$

# KOHONEN NEURAL NETWORK (KNN)

## *Illustration on Clustering of Bipolar Input Patterns*

Epoch Number 1 / Pattern Number 1

### ***Calculation of Euclidean Distance***

$$ED(1) = \sum_{h=1:n} (w_{h1} - I_h)^2$$

(Cluster k=1)

$$ED(1) = (0.2-1)^2 + (0.6-1)^2 + (0.5-(-1))^2 + (0.9-(-1))^2 = 4.6600$$

(Cluster k=2)

$$ED(2) = (0.8-1)^2 + (0.4-1)^2 + (0.7-(-1))^2 + (0.3-(-1))^2 = 2.1800$$

***The winner cluster unit is K = 2 because ED is minimum. Therefore, the weights connected to the winner cluster unit 2 should be updated***

### **Weight Updation**

$$w_{hK-2}(new) = w_{hk-2}(old) + 0.9 * (I_1 - w_{hk-2}(old))$$

$$w_{12} = 0.8 + 0.9 \times (1 - 0.8) = 0.9800$$

$$w_{22} = 0.4 + 0.9 \times (1 - 0.4) = 0.9400$$

$$w_{32} = 0.7 + 0.9 \times (1 - 0.7) = 0.9700$$

$$w_{42} = 0.3 + 0.9 \times (-1 - 0.3) = -0.8700$$

$$W = \begin{bmatrix} 0.2 & 0.98 \\ 0.6 & 0.94 \\ 0.5 & 0.97 \\ 0.9 & -0.87 \end{bmatrix}$$

# KOHONEN NEURAL NETWORK (KNN)

## *Illustration on Clustering of Bipolar Input Patterns*

Epoch Number 1 / Pattern Number 2

### *Calculation of Euclidean Distance*

$$ED(1) = \sum_{h=1:n} (w_{h1} - I_h)^2$$

(Cluster k=1)

$$ED(1) = (0.2-(-1))^2 + (0.6-(-1))^2 + (0.5-(-1))^2 + (0.9-1)^2 = 6.2600$$

(Cluster k=2)

$$ED(2) = (0.98-(-1))^2 + (0.94-(-1))^2 + (0.97-(-1))^2 + (-0.87-1)^2 = 15.0618$$

***The winner cluster unit is K = 1 because ED is minimum. Therefore, the weights connected to the winner cluster unit 1 should be updated***

### Weight Updation

$$w_{hK-1}(new) = w_{hk-1}(old) + 0.9*(I_2 - w_{hk-1}(old))$$

$$w_{11} = 0.2 + 0.9\times(-1-0.2) = -0.8800$$

$$w_{21} = 0.6 + 0.9\times(-1-0.6) = -0.8400$$

$$w_{31} = 0.5 + 0.9\times(-1-0.5) = -0.8500$$

$$w_{41} = 0.0 + 0.9\times(-1-0.9) = 0.9900$$

$$W = \begin{bmatrix} -0.88 & 0.98 \\ -0.84 & 0.94 \\ -0.85 & 0.97 \\ 0.99 & -0.87 \end{bmatrix}$$

# KOHONEN NEURAL NETWORK (KNN)

***Illustration on Clustering of Bipolar Input Patterns***

Epoch Number 1 / Pattern Number 3

***Calculation of Euclidean Distance***

$$ED(1) = \sum_{h=1:n} (w_{h1} - I_h)^2$$

(Cluster k=1)

$$ED(1) = (-0.88-1)^2 + (-0.84-(-1))^2 + (-0.85-(-1))^2 + (0.99-(-1))^2 = 7.5426$$

(Cluster k=2)

$$ED(2) = (0.98-1)^2 + (0.94-(-1))^2 + (0.97-(-1))^2 + (-0.87-(-1))^2 = 7.6618$$

***The winner cluster unit is K = 1 because ED is minimum. Therefore, the weights connected to the winner cluster unit 1 should be updated***

**Weight Updation**

$$w_{hK-1}(new) = w_{hk-1}(old) + 0.9 * (I_3 - w_{hk-1}(old))$$

$$w_{11} = -0.88 + 0.9 \times (1 - (-0.88)) = 0.8120$$

$$w_{21} = -0.84 + 0.9 \times (-1 - (-0.84)) = -0.9840$$

$$w_{31} = -0.85 + 0.9 \times (-1 - (-0.85)) = -0.9850$$

$$w_{41} = 0.99 + 0.9 \times (-1 - 0.99) = -0.8010$$

$$W = \begin{bmatrix} 0.8120 & 0.98 \\ -0.984 & 0.94 \\ -0.985 & 0.97 \\ -0.801 & -0.87 \end{bmatrix}$$

# KOHONEN NEURAL NETWORK (KNN)

## *Illustration on Clustering of Bipolar Input Patterns*

Epoch Number 1 / Pattern Number 4

### ***Calculation of Euclidean Distance***

$$ED(1) = \sum_{h=1:n} (w_{h1} - I_h)^2$$

(Cluster k=1)

$$ED(1) = (0.81-(-1))^2 + (-0.984-(-1))^2 + (-0.985-1)^2 + (-0.801-1)^2 = 10.4674$$

(Cluster k=2)

$$ED(2) = (0.98-(-1))^2 + (0.94-(-1))^2 + (0.97-1)^2 + (-0.87-1)^2 = 11.1818$$

***The winner cluster unit is K = 1 because ED is minimum. Therefore, the weights connected to the winner cluster unit 1 should be updated***

### **Weight Updation**

$$w_{hK-1}(new) = w_{hk-1}(old) + 0.9*(I_4 - w_{hk-1}(old))$$

$$w_{11} = 0.812 + 0.9\times(-1-0.812)) = -0.8188$$

$$w_{21} = -0.984 + 0.9\times(-1-(-0.984)) = -0.9984$$

$$w_{31} = -0.985 + 0.9\times(1-(-0.985)) = 0.8015$$

$$w_{41} = -0.801 + 0.9\times(1-(-0.801)) = 0.8199$$

$$W = \begin{bmatrix} 0.8120 & 0.98 \\ -0.984 & 0.94 \\ -0.985 & 0.97 \\ -0.801 & -0.87 \end{bmatrix}$$

## KOHONEN NEURAL NETWORK (KNN)

***Clustering of Numerical Characters***

- ***This application aims to cluster 25 binary patterns representing numerals from 1 to 9. They are represented by 9 X 7 matrix format.***

- ***The objective is to cluster the 25 patterns into 9 groups. Each of the input patterns are represented as binary input vectors .***

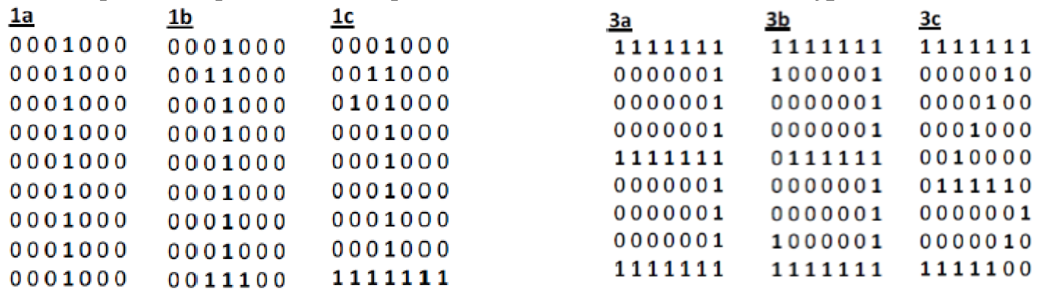
- ***Let, n = 25, m = 2, initial topological parameter R = 4 . Let the learning rate be 0.9 and will geometrically decrease 0.5 times for every epoch.***

- ***The topological parameter R (geometrical radius) will be decreased by subtracting a small value of 0.2 for every epoch. The value of R should be rounded off to obtain an integer value.***

# KOHONEN NEURAL NETWORK (KNN)

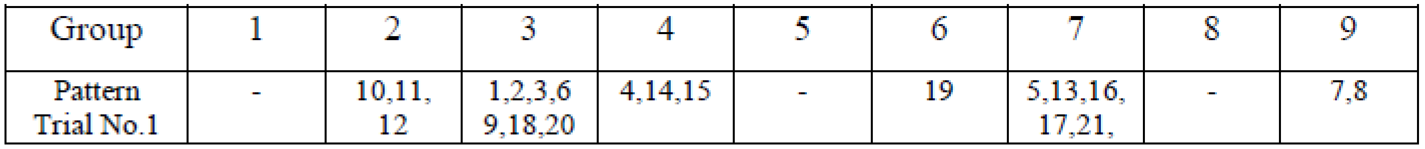
## *Clustering of Numerical Characters*

❖ ***Sample input test pattern for clustering***



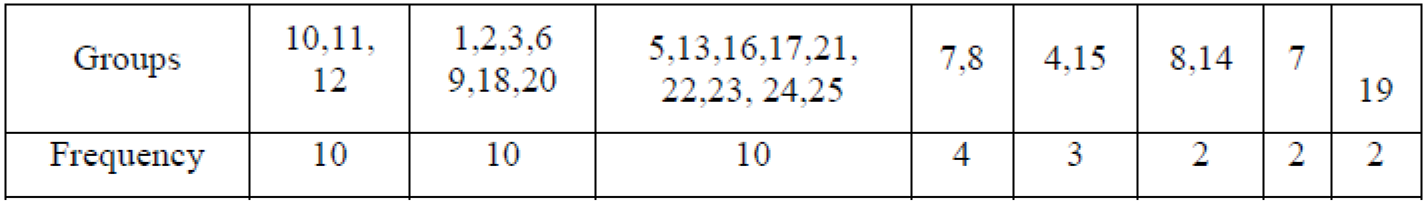
```
1a        1b        1c                3a        3b        3c
0001000   0001000   0001000           1111111   1111111   1111111
0001000   0011000   0011000           0000001   1000001   0000010
0001000   0001000   0101000           0000001   0000001   0000100
0001000   0001000   0001000           0000001   0000001   0001000
0001000   0001000   0001000           1111111   0111111   0010000
0001000   0001000   0001000           0000001   0000001   0111110
0001000   0001000   0001000           0000001   0000001   0000001
0001000   0001000   0001000           0000001   1000001   0000010
0001000   0011100   1111111           1111111   1111111   1111100
```

❖ ***Results: Clusters***



| Group | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| Pattern Trial No.1 | - | 10,11, 12 | 1,2,3,6 9,18,20 | 4,14,15 | - | 19 | 5,13,16, 17,21, | - | 7,8 |

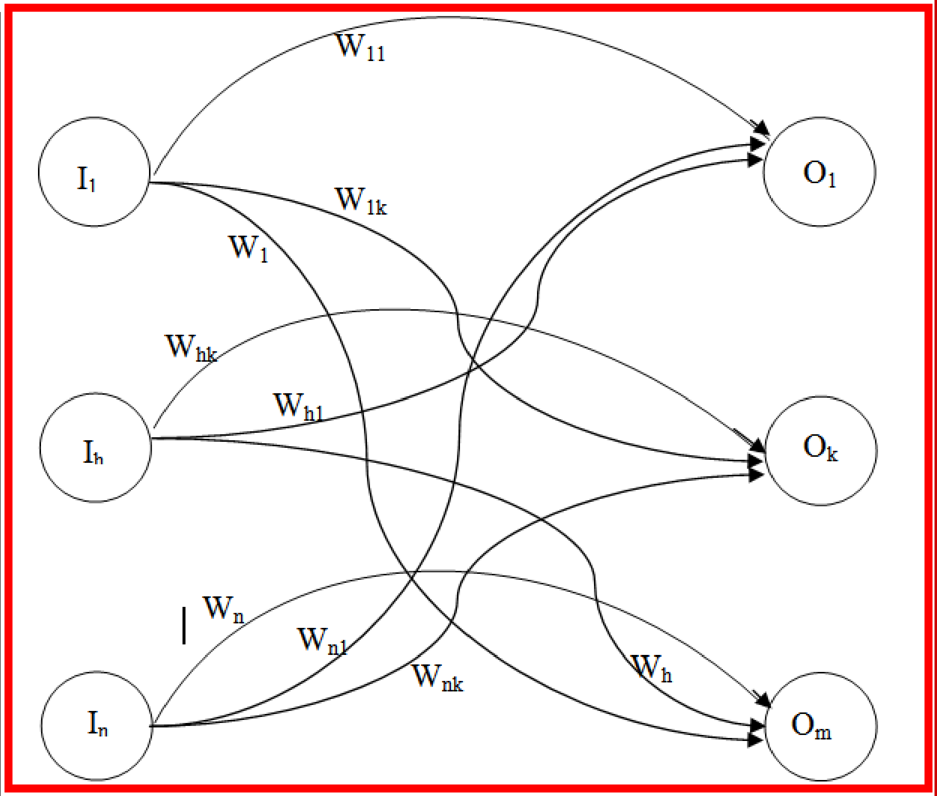
❖ ***The simulation is conducted for 10 trial runs and frequency of occurrence of clustered groups .***



| Groups | 10,11, 12 | 1,2,3,6 9,18,20 | 5,13,16,17,21, 22,23, 24,25 | 7,8 | 4,15 | 8,14 | 7 | 19 |
|---|---|---|---|---|---|---|---|---|
| Frequency | 10 | 10 | 10 | 4 | 3 | 2 | 2 | 2 |

# LEARNING VECTOR QUANTIZATION (LVQ)

- ***Learning Vector Quantization (LVQ) network is a supervised neural network where the input vectors are trained for a specific class or group already mapped in the training set.***
- ***The architecture of the LVQ is similar to the Kohonen neural network where the number of output units is equal to the number of available classes, but without a topological structure which is assumed for the output units.***
- ***The weight updation is carried out only for the weight vector for which the input vector corresponds to the output unit. A reference input vector is selected for a specific class.***



***Architecture of LVQ***

## LEARNING VECTOR QUANTIZATION (LVQ)

**Steps**

1. *Initialize the input vector as reference vector i.e., initial weight values and set the learning rate(η) within 0.1 ≤n* η ≤1.*
2. *For each input pattern $I_h$ training pair compute the Euclidean distance between input vector and Weight vector for each output cluster unit k. Find the unit index K for which the ED is minimum.*

$$\boxed{ED(k) = \sum_{h=1:n} (w_{hk} - I_h)^2}$$

3. *Update the weights for kth output unit*

*If $T=O_k$,* $$\boxed{w_{hk}(t+1) = w_{hk}(t) + \eta * (x_h - w_{hk}(t))}$$

*If $T\neq O_k$,* $$\boxed{w_{hk}(t+1) = w_{hk}(t) - \eta * (x_h - w_{hk}(t))}$$

4. *Reduce the learning rate*

*Repeat steps 2,3 & 4 till the maximum number of epochs is reached*

# LEARNING VECTOR QUANTIZATION (LVQ)

***Clustering of Bipolar Input Patterns in LVQ***
***<u>Hand worked Example: Cluster 6 bipolar patterns into 2 clusters</u>***

***Let, n = 4 and m = 2 and learning rate be η=0.9 and will geometrically decrease 0.5 times for every epoch.***

***Initialize the input vector as reference vector i.e., initial weight values and set the learning rate(η) within 0.1 ≤n* η ≤1.***

***Arbitrarily select, the reference vector.***

***Here, $I_1$ is selected as reference vector for the first cluster and $I_2$ is selected as reference vector for the second cluster.***

$$I = \left[\begin{array}{cccc} 1 & 1 & 1 & -1 \\ -1 & -1 & -1 & 1 \\ 1 & -1 & -1 & -1 \\ -1 & -1 & 1 & 1 \\ -1 & 1 & 1 & 1 \\ -1 & 1 & -1 & -1 \end{array}\right]\begin{array}{c} \mathrm{I1} \\ \mathrm{I2} \\ \mathrm{I3} \\ \mathrm{I4} \\ \mathrm{I5} \\ \mathrm{I6} \end{array}$$

$$T = \begin{bmatrix} 1 \\ 2 \\ 2 \\ 1 \\ 1 \\ 2 \end{bmatrix}_{n\ X\ m} \qquad W = \begin{bmatrix} 1 & -1 \\ 1 & -1 \\ 1 & -1 \\ -1 & 1 \end{bmatrix}_{n\ X\ m}$$

$$\eta = 0.9$$

# LEARNING VECTOR QUANTIZATION (LVQ)

***Clustering of Bipolar Input Patterns in LVQ***
***<u>Hand worked Example: Cluster 6 bipolar patterns into 2 clusters</u>***

**Epoch Number 1**
**Pattern Number 1**

## <u>*Calculation of Euclidean Distance*</u>

$$ED(1) = \sum_{h=1:n} (w_{h1} - I_h)^2$$

*(Cluster k=1)*

$$ED(1) = (1-1)^2 + (1-1)^2 + (1-1)^2 + ((-1)-(-1))^2 = 0$$

*(Cluster k=2)*

$$ED(2) = ((-1)-1)^2 + ((-1)-1)^2 + ((-1)-1)^2 + (1-(-1))^2 = 16$$

*Output unit Class is 1. i.e. $O_2$=1 ED (1) is minimum.*
*Target Class is 1, i.e. T =1.*

*Since T = $O_2$, the weights connected to cluster unit 1 should be updated.*

## <u>*WEIGHT UPDATION*</u>

$$w_{hK=1}(new) = w_{hk=1}(old) + 0.9 * (I_1 - w_{hk=1}(old))$$

$$w_{11} = 1 + 0.9 \times (1-1) = 1$$

$$w_{21} = 1 + 0.9 \times (1-1) = 1$$

$$w_{31} = 1 + 0.9 \times (1-1) = 1$$

$$w_{41} = (-1) + 0.9 \times (-1-(-1)) = -1$$

$$W = \begin{bmatrix} 1 & -1 \\ 1 & -1 \\ 1 & -1 \\ -1 & 1 \end{bmatrix}_{n \ X \ m}$$

# LEARNING VECTOR QUANTIZATION (LVQ)

***Clustering of Bipolar Input Patterns in LVQ***
***Hand worked Example: Cluster 6 bipolar patterns into 2 clusters***

**Epoch Number 1**
**Pattern Number 2**

## Calculation of Euclidean Distance

$$ED(1) = \sum_{h=1:n} (w_{h1} - I_h)^2$$

*(Cluster k=1)*

$$ED(1) = (1-(-1))^2 + (1-(-1))^2 + (1-(-1))^2 + ((-1)-1)^2 = 16$$

*(Cluster k=2)*

$$ED(2) = (-1-(-1))^2 + (-1-(-1))^2 + (-1-(-1))^2 + (1-1)^2 = 0$$

*Output unit Class 2. i.e. $O_2$=2 ED (1) is minimum.*
*Target Class is 2, i.e. T =2.*

*Since T = $O_2$, the weights connected to cluster unit 1 should be updated.*

## WEIGHT UPDATION

$$w_{hK=2}(new) = w_{hk=2}(old) + 0.9*(I_2 - w_{hk=2}(old))$$

$$w_{12} = -1 + 0.9 \times (-1-(-1)) = -1$$

$$w_{22} = -1 + 0.9 \times (-1-(-1)) = -1$$

$$w_{32} = -1 + 0.9 \times (-1-(-1)) = -1$$

$$w_{42} = 1 + 0.9 \times (-1-1) = 1$$

$$W = \begin{bmatrix} 1 & -1 \\ 1 & -1 \\ 1 & -1 \\ -1 & 1 \end{bmatrix}_{n\ X\ m}$$

# LEARNING VECTOR QUANTIZATION (LVQ)

***Clustering of Bipolar Input Patterns in LVQ***
***Hand worked Example: Cluster 6 bipolar patterns into 2 clusters***

**Epoch Number 1**
**Pattern Number 3**

## Calculation of Euclidean Distance

$$ED(1) = \sum_{h=1:n} (w_{h1} - I_h)^2$$

*(Cluster k=1)*

$$ED(1) = (1-1)^2 + (1-(-1))^2 + (1-(-1))^2 + (-1-(-1))^2 = 8$$

*(Cluster k=2)*

$$ED(2) = (-1-1)^2 + (-1-(-1))^2 + (-1-(-1))^2 + (1-(-1))^2 = 8$$

*ED (1)=ED(2); T=2;*

*Weights corresponding to O1 and O2 are updated.*

## WEIGHT UPDATION

$$w_{hK=1}(new) = w_{hk=1}(old) - 0.9 * (I_3 - w_{hk=1}(old))$$

$$w_{11} = 1 - 0.9 \times (1-1) = 1$$

$$w_{21} = 1 - 0.9 \times (-1-1) = 2.8$$

$$w_{31} = 1 - 0.9 \times (-1-1) = 2.8$$

$$w_{41} = -1 - 0.9 \times (-1-(-1)) = -1$$

$$w_{hK=2}(new) = w_{hk=2}(old) + 0.9 * (I_3 - w_{hk=2}(old))$$

$$w_{12} = -1 + 0.9 \times (1-(-1)) = 0.8$$

$$w_{22} = -1 + 0.9 \times (-1-(-1)) = -1$$

$$w_{32} = -1 + 0.9 \times (-1-(-1)) = -1$$

$$w_{42} = 1 + 0.9 \times (-1-1) = -0.8$$

$$W = \begin{bmatrix} 1 & 0.8 \\ 2.8 & -1 \\ 2.8 & -1 \\ -1 & -0.8 \end{bmatrix}_{n \times m}$$

# LEARNING VECTOR QUANTIZATION (LVQ)

***Clustering of Bipolar Input Patterns in LVQ***
***<u>Hand worked Example: Cluster 6 bipolar patterns into 2 clusters</u>***

**Epoch Number 1**
**Pattern Number 4**

## ***Calculation of Euclidean Distance***

$$ED(1) = \sum_{h=1:n} (w_{h1} - I_h)^2$$

*(Cluster k=1)*

$$ED(1) = (1-(-1))^2 + (2.8-(-1))^2 + (2.8-1)^2 + (-1-1)^2 = 25.68$$

*(Cluster k=2)*

$$ED(2) = (0.8-(-1))^2 + (-1-(-1))^2 + (-1-1)^2 + (-0.8-1)^2 = 10.48$$

*ED(2) is minimum O=2; T=1;*

*T ≠ $O_2$, Weights corresponding to O2 are updated.*

## ***WEIGHT UPDATION***

$$w_{hK=2}(new) = w2(old) - 0.9 * (I_4 - w_{hk=2}(old))$$

$$w_{12} = 0.8 - 0.9 \times (-1 - 0.8) = 2.42$$

$$w_{22} = -1 - 0.9 \times (-1 - (-1)) = -1$$

$$w_{32} = -1 - 0.9 \times (1 - (-1)) = -2.8$$

$$w2 = -0.8 - 0.9 \times (1 - (-0.8)) = -2.42$$

$$W = \begin{bmatrix} 1 & 2.42 \\ 2.8 & -1 \\ 2.8 & -2.8 \\ -1 & -2.42 \end{bmatrix}_{n \ X \ m}$$

# LEARNING VECTOR QUANTIZATION (LVQ)

***Clustering of Bipolar Input Patterns in LVQ***
***Hand worked Example: Cluster 6 bipolar patterns into 2 clusters***

**Epoch Number 1**
**Pattern Number 5**

## Calculation of Euclidean Distance

$$ED(1) = \sum_{h=1:n} (w_{h1} - I_h)^2$$

*(Cluster k=1)*

$$ED(1) = (1-(-1))^2 + (2.8-1)^2 + (2.8-1)^2 + (-1-1)^2 = 14.48$$

*(Cluster k=2)*

$$ED(2) = (2.42-(-1))^2 + (-1-1)^2 + (-2.8-1)^2 + (-2.42-1)^2 = 41.8328$$

*ED(1) is minimum O=1; T=1;*

*$T = O_1$ ,*

*The weights connected to cluster unit 1 should be updated*

## WEIGHT UPDATION

$$w_{hK=1}(new) = w_{hk=1}(old) + 0.9 * (I_5 - w_{hk=1}(old))$$

$$w_{11} = 1 + 0.9 \times (-1-1) = -0.8$$

$$w_{21} = 2.8 + 0.9 \times (1-2.8) = 1.18$$

$$w_{31} = 2.8 + 0.9 \times (1-2.8) = 1.18$$

$$w_{41} = -1 + 0.9 \times (1-(-1)) = 0.8$$

$$W = \begin{bmatrix} -0.8 & 2.42 \\ 1.18 & -1 \\ 1.18 & -2.8 \\ 0.8 & -2.42 \end{bmatrix}_{n \ X \ m}$$

# LEARNING VECTOR QUANTIZATION (LVQ)

***Clustering of Bipolar Input Patterns in LVQ***
***Hand worked Example: Cluster 6 bipolar patterns into 2 clusters***

**Epoch Number 1**
**Pattern Number 6**

***Calculation of Euclidean Distance***

$$ED(1) = \sum_{h=1:n} (w_{h1} - I_h)^2$$

*(Cluster k=1)*

$$ED(1) = (-0.8-(-1))^2 + (1.18-1)^2 + (1.18-(-1))^2 + (0.8-(-1))^2 = 8.0648$$

*(Cluster k=2)*

$$ED(2) = (2.42-(-1))^2 + (-1-1)^2 + (-2.8-(-1))^2 + (-2.42-(-1))^2 = 20.95$$

*ED(1) is minimum O=1; T=2;*

*$T \neq O_1$;*
*The weights connected to cluster unit 1 should be updated*

***WEIGHT UPDATION***

$$w_{hK=1}(new) = w_{hk=1}(old) - 0.9 * (I_5 - w_{hk=1}(old))$$

$$w_{11} = -0.8 - 0.9 \times (-1 - (-0.8)) = -0.62$$

$$w_{21} = 1.18 - 0.9 \times (1 - 1.18) = 1.342$$

$$w_{31} = 1.18 - 0.9 \times (-1 - 1.18) = 3.142$$

$$w1 = 0.8 - 0.9 \times (-1 - 0.8)) = 2.42$$

$$W = \begin{bmatrix} -0.62 & 2.42 \\ 1.342 & -1 \\ 3.142 & -2.8 \\ 2.42 & -2.42 \end{bmatrix}_{n \text{ X } m}$$

# LEARNING VECTOR QUANTIZATION (LVQ)

***Clustering of Bipolar Input Patterns in LVQ***
***Hand worked Example: Cluster 6 bipolar patterns into 2 clusters***

RESULTS

***At the second epoch, the learning rate will geometrically decrease 0.5 times for every epoch. Therefore, the learning rate will be 0.45 for the starting of the second epoch.***

***After 1000 epochs, the learning rate and weights are found to be*** $\eta = 8.3994\text{e-}302$

$$W = \begin{bmatrix} -0.4760 & -0.3775 \\ 0.3906 & -0.2086 \\ 1.0822 & -1.0691 \\ 0.5451 & -0.4139 \end{bmatrix}$$

***Pattern 1- (1 1 1 -1), Pattern 4- (-1 -1 1 1) and Pattern 5- (-1 1 1 1) belongs to the first output unit.***

***Pattern 2- (-1 -1 -1 1), Pattern 3- (1 -1 -1 -1) and Pattern 6- (-1 1 -1 -1) belongs to the second output unit.***

# LEARNING VECTOR QUANTIZATION (LVQ)

## *Classification of Numerical Characters*

***This application aims cluster 25 binary patterns representing numerals from 1 to 9 using Matlab.***
***Numerals are represented by 9 X 7 matrix format. Some sample numeral patterns are given below.***

| 1a | 1b | 1c | 8b | 9a | 9b |
|---|---|---|---|---|---|
| 0001000 | 0001000 | 0001000 | 0111110 | 1111111 | 1111111 |
| 0001000 | 0011000 | 0011000 | 1000001 | 1000001 | 1000001 |
| 0001000 | 0001000 | 0101000 | 1000001 | 1000001 | 1000001 |
| 0001000 | 0001000 | 0001000 | 1000001 | 1000001 | 1000001 |
| 0001000 | 0001000 | 0001000 | 0111110 | 1111111 | 1111111 |
| 0001000 | 0001000 | 0001000 | 1000001 | 0000001 | 0000001 |
| 0001000 | 0001000 | 0001000 | 1000001 | 0000001 | 0000001 |
| 0001000 | 0001000 | 0001000 | 1000001 | 0000001 | 1000001 |
| 0001000 | 0011100 | 1111111 | 0111110 | 1111111 | 1111111 |

**Pattern No. 1 (1a) ,2 (1b), 3(1c) representing numeral 1**

**PatternNo.22(8b) representing numeral 8**
**Pattern No. 23(9a), 24(9b) representing numeral 9**

**Similar to patterns at left, various patterns representing numerals are used and the pattern number of the numerals are given in Table .**

| Numeral | Pattern No |
|---|---|
| 1 | 1,2,3 |
| 2 | 4,5,6 |
| 3 | 7,8,9 |
| 4 | 10,11,12 |
| 5 | 13,14 |
| 6 | 15,16,17 |
| 7 | 18,19,20 |
| 8 | 21,22 |
| 9 | 23,24,25 |

***Here, n = 25. Let, m = 2, η=0.9 and decreases 0.5 times at each epoch. The simulation of 1,000 epochs are carried out and the results are***

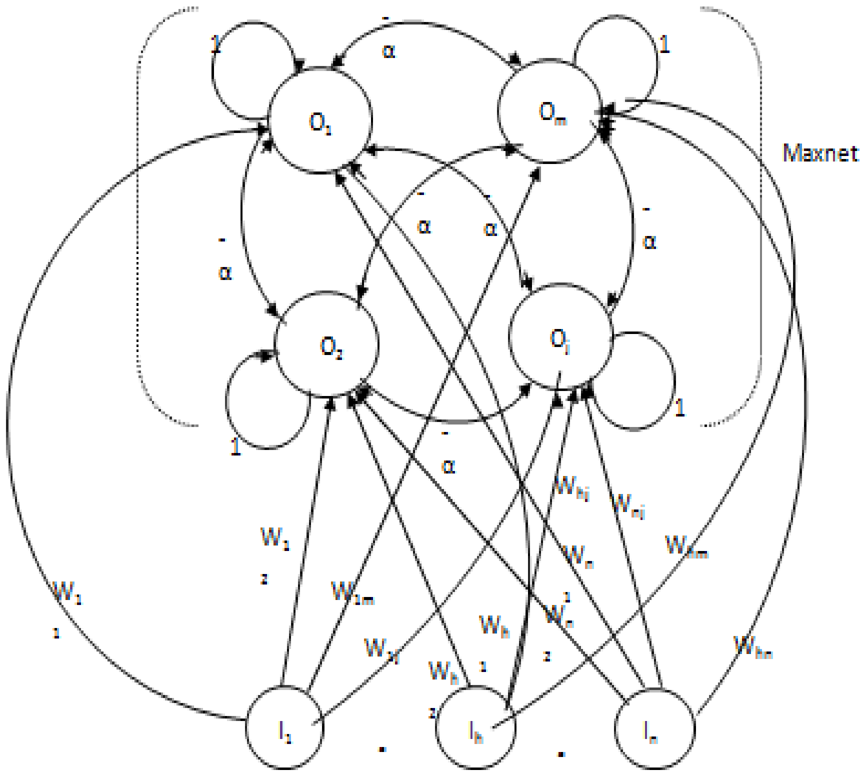
| Class | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| Patterns | 1,2,3 | 6 | 5,7,8 | 10,11, 12 | 9,13,14, 16 | 15 | 18, 19,20 | 17,22 | 4,21, 23, 24, 25 |

# HAMMING NEURAL NETWORK (HNN)

***Lippmann (1987)  modelled a two layer bipolar network called Hamming neural network. The first layer is the Hamming net and the second layer is the MAXNET.***

***The first layer is a feed forward type network which classifies the input patterns based on minimum Hamming distance. The Hamming distance (HD) between any two vectors is the number of components in which the vectors differ.***

***The Hamming net uses MAXNET in the second layer as a subnet to find the unit with the largest net input. The second layer operates as recurrent recall network which suppresses all the outputs except the initially obtained maximum output of the first layer.***



***Architecture of HNN***

# HAMMING NEURAL NETWORK (HNN)

***Let I (1-11111) and S (11-1-111) be the two fixed length bipolar vectors .***
***Hamming distance HD (I, S) is equal to 3.***
***The scalar product of A and B is***

$$I^tS= [n-HD (I, S)]-HD (I, S)$$

***If n is the number of components in the vectors, then [n-HD (I, S)] are the number of components in which the vectors agree.***

$$I^tS = n-2HD (I, S)$$

***Let I be the input vector and S be the vector that represents the patterns placed on a cluster. For a two layer classifier of bipolar vector, the strongest response of a neuron indicates that the minimum HD exists between the two vectors I and S. For setting up the weights and bias, the above equation is written as:***

$$HD (I, S) = I^t \cdot S/2 +n/2$$

***If the weights are fixed to one half of the standard vector S/2 and bias to n/2, then the network will be able to find the input vector I, closest to the standard vector S. This is done by finding the output unit with the largest net input.***

# HAMMING NEURAL NETWORK (HNN)

***Illustration on Finding the Best Match with Standard Vector:***
*Hand worked example:*
*Cluster 4 bipolar patterns and find the patterns closest to 2 standard bipolar patterns*

*The two standard bipolar patterns are S(1) = (1 1 -1 -1 1 1) and S(2) = (-1 -1 1 -1 1 1).*
*Here n= 6, m= 2 and I= 4.*
*The 4 bipolar input patterns (I), initial weights (W) and bias (B) are*

$$I = \begin{bmatrix} 1 & 1 & 1 & -1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & -1 \\ 1 & -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 & 1 & -1 \end{bmatrix}_{I\times n} \begin{matrix} (I_1) \\ (I_2) \\ (I_3) \\ (I_4) \end{matrix}$$

$$W = \begin{bmatrix} 0.5 & -0.5 \\ 0.5 & -0.5 \\ -0.5 & 0.5 \\ -0.5 & -0.5 \\ 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}_{n\ X\ m} \qquad B = [3\ \ 3]$$

*The net input to each output unit of the first layer for all the 4 input patterns is calculated from*

$$O_j = B_j + \sum_{h=1}^{n} I_{ih} W_{hj},\ j = 1,..m$$

# HAMMING NEURAL NETWORK (HNN)

***Illustration on Finding the Best Match with Standard Vector:***
*Hand worked example:*
*Cluster 4 bipolar patterns and find the patterns closest to 2 standard bipolar patterns*

**Pattern 1**
**(1 1 1 -1 1 1)**

***Computation in the first layer***

$$O_1(0) = 3 + 1\times 0.5 + 1\times 0.5 + 1\times(-0.5) + (-1)\times(-0.5) + 1\times 0.5 + 1\times 0.5 = 5$$

$$O_2(0) = 3 + 1\times -0.5 + 1\times -0.5 + 1\times 0.5 + (-1)\times(-0.5) + 1\times 0.5 + 1\times 0.5 = 4$$

***Computation in the second layer (MAXNET)***

$$w_{ij} = \begin{bmatrix} 1 & 0.25 \\ 0.25 & 1 \end{bmatrix}$$

$\boxed{t=0}$

$O_1(0+1) = f(O_1(0) - 0.25\times(4)) = 4\,;\, O_2(0+1) = f(O_2(0) - 0.25\times(5)) = 2.75$
$O_1(0) = 4; O_2(0) = 2.75$

$\boxed{t=1}$

$O_1(1+1) = f(O_1(1) - 0.25\times(2.75)) = 3.3125\,;\; O_2(1+1) = f(O_2(1) - 0.25\times(4)) = 1.75$
$O_1(1) = 3.2125; O_2(1) = 1.75$

$\boxed{t=2}$

$O_1(2+1) = f(O_1(2) - 0.25\times(1.75)) = 2.8750; O_2(2+1) = f(O_2(2) - 0.25\times(3.3125)) = 0.9219$
$O_1(2) = 2.8750; O_2(2) = 0.9219$

$\boxed{t=3}$

$O_1(3+1) = f(O_1(3) - 0.25\times(0.9219)) = 2.6445; O_2(3+1) = f(O_2(3) - 0.25\times(2.8750)) = 0.2031$
$O_1(3) = 2.6445; O_2(3) = 0.2031$

$\boxed{t=4}$

$O_1(4+1) = f(O_1(4) - 0.25\times(0.2031)) = 2.5938\,;\, O_2(4+1) = f(O_2(4) - 0.25\times(2.6445)) = 0$

$$sum_2 = -0.4580 < 0,\quad f(sum_2) = 0$$

*The non-zero input unit j=1 of the MAXNET is the winner*

*(1 1 1 -1 1 1) is closer to S(1) = (1 1 -1 -1 1 1).*

# HAMMING NEURAL NETWORK (HNN)

***Illustration on Finding the Best Match with Standard Vector:***
*Hand worked example:*
*Cluster 4 bipolar patterns and find the patterns closest to 2 standard bipolar patterns*

**Pattern 2**
***(-1 -1 -1 1 1 -1)***

Computation in the first layer

$$O_1(0) = 3 + (-1)\times 0.5 + (-1)\times 0.5 + (-1)\times(-0.5) + 1\times(-0.5) + 1\times 0.5 + (-1)\times 0.5 = 2$$

$$O_2(0) = 3 + (-1)\times -0.5 + (-1)\times -0.5 + (-1)\times 0.5 + 1\times(-0.5) + 1\times 0.5 + (-1)\times 0.5 = 3$$

Computation in the second layer (MAXNET)

$\boxed{t=0}$

$$O_1(0+1) = f(O_1(0) - 0.25\times(3)) = 1.25;\ O_2(0+1) = f(O_2(0) - 0.25\times(2)) = 2.5$$
$$O_1(0) = 1.25; O_2(0) = 2.5$$

$\boxed{t=1}$

$$O_1(1+1) = f(O_1(1) - 0.25\times(2.5)) = 0.625;\ O_2(1+1) = f(O_2(1) - 0.25\times(1.25)) = 2.1875$$
$$O_1(1) = 0.625; O_2(1) = 2.1875$$

$\boxed{t=2}$

$$O_1(2+1) = f(O_1(2) - 0.25\times(2.1875)) = 0.0781; O_2(2+1) = f(O_2(2) - 0.25\times(0.625)) = 2.0312$$
$$O_1(2) = 0.0781; O_2(2) = 2.0312$$

$\boxed{t=3}$

$$O_1(3+1) = f(O_1(3) - 0.25\times(2.0312)) = 0;O_2(3+1) = f(O_2(3) - 0.25\times(0.0781)) = 2.0117$$

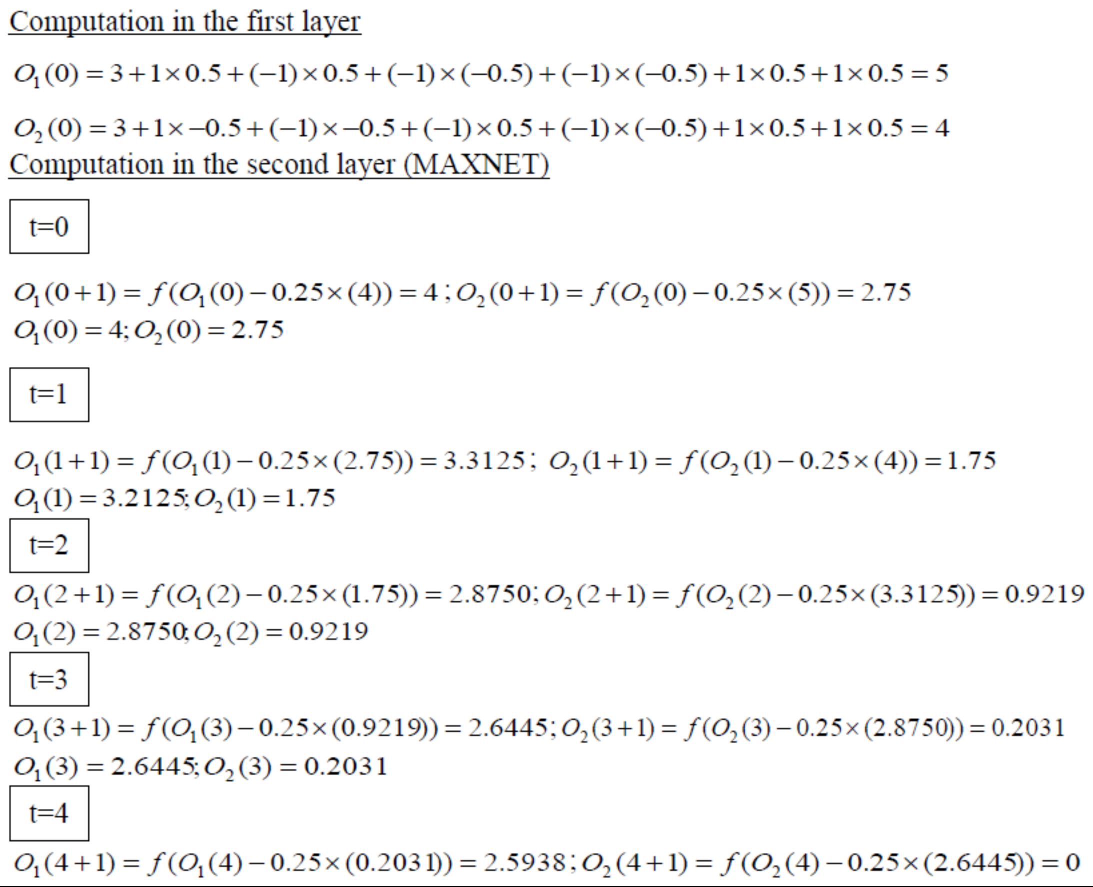
$$sum_1 = -0.4297 < 0,$$
$$f(sum_1) = 0$$

*The non-zero input unit j=2 of the MAXNET is the winner*

*$I_2$=(-1 -1 -1 1 1 -1)*
*is closer to*
*S(2) = (-1 -1 1 -1 1 1).*

# HAMMING NEURAL NETWORK (HNN)

***Illustration on Finding the Best Match with Standard Vector:***
*Hand worked example:*
*Cluster 4 bipolar patterns and find the patterns closest to 2 standard bipolar patterns*



Computation in the first layer

$O_1(0) = 3 + 1\times 0.5 + (-1)\times 0.5 + (-1)\times(-0.5) + (-1)\times(-0.5) + 1\times 0.5 + 1\times 0.5 = 5$

$O_2(0) = 3 + 1\times -0.5 + (-1)\times -0.5 + (-1)\times 0.5 + (-1)\times(-0.5) + 1\times 0.5 + 1\times 0.5 = 4$

Computation in the second layer (MAXNET)

t=0

$O_1(0+1) = f(O_1(0) - 0.25\times(4)) = 4\,;\, O_2(0+1) = f(O_2(0) - 0.25\times(5)) = 2.75$

$O_1(0) = 4; O_2(0) = 2.75$

t=1

$O_1(1+1) = f(O_1(1) - 0.25\times(2.75)) = 3.3125;\; O_2(1+1) = f(O_2(1) - 0.25\times(4)) = 1.75$

$O_1(1) = 3.2125; O_2(1) = 1.75$

t=2

$O_1(2+1) = f(O_1(2) - 0.25\times(1.75)) = 2.8750; O_2(2+1) = f(O_2(2) - 0.25\times(3.3125)) = 0.9219$

$O_1(2) = 2.8750; O_2(2) = 0.9219$

t=3

$O_1(3+1) = f(O_1(3) - 0.25\times(0.9219)) = 2.6445; O_2(3+1) = f(O_2(3) - 0.25\times(2.8750)) = 0.2031$

$O_1(3) = 2.6445; O_2(3) = 0.2031$

t=4

$O_1(4+1) = f(O_1(4) - 0.25\times(0.2031)) = 2.5938\,;\, O_2(4+1) = f(O_2(4) - 0.25\times(2.6445)) = 0$

**Pattern 3**
***(1 -1 -1 -1 1 1)***

$sum_2 = -0.4580 < 0,$
$f(sum_2) = 0$

*The non-zero input unit j=1 of the MAXNET is the winner.*
*$I_3$(1 -1 -1 -1 1 1)*
*is closer to*
*S(1) = (1 1 -1 -1 1 1).*

# HAMMING NEURAL NETWORK (HNN)

***Illustration on Finding the Best Match with Standard Vector:***
*Hand worked example:*
*Cluster 4 bipolar patterns and find the patterns closest to 2 standard bipolar patterns*

**Pattern 4**
***(-1 -1 1 1 1 -1)***

Computation in the first layer

$O_1(0) = 3 + (-1)\times 0.5 + (-1)\times 0.5 + 1\times(-0.5) + 1\times(-0.5) + 1\times 0.5 + (-1)\times 0.5 = 1$

$O_2(0) = 3 + (-1)\times -0.5 + (-1)\times -0.5 + 1\times 0.5 + 1\times(-0.5) + 1\times 0.5 + (-1)\times 0.5 = 4$

Computation in the second layer (MAXNET)

$\boxed{t=0}$

$O_1(0+1) = f(O_1(0) - 0.25\times(4)) = 0$; $O_2(0+1) = f(O_2(0) - 0.25\times(1)) = 3.75$

Since $sum_1 = 0 \le 0, f(sum_1) = 0$

$sum_1 = 0 \le 0,$
$f(sum_1) = 0$

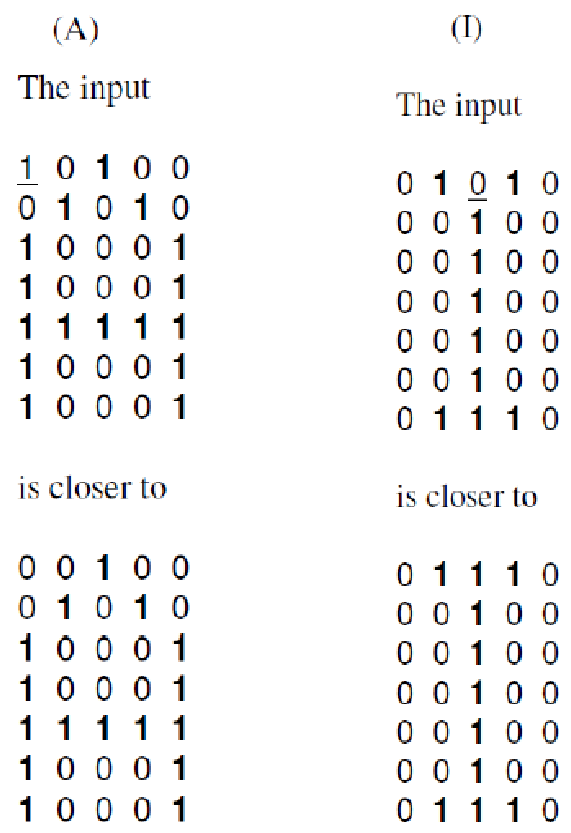
*The non-zero input unit j=2 of the MAXNET is the winner.*
*$I_4$ (-1 -1 1 1 1 -1) icloser to*
*S(2) = (-1 -1 1 -1 1 1).*

# HAMMING NEURAL NETWORK (HNN)

***Character Recognition through Clustering of Numerical Characters***

***This application aims to recognize the closest match of the input test pattern of an alphabet with an error. Here, the representation of an alphabet is by a matrix of 7_X_5 bipolar elements.***

***Though Hamming neural network can be used for clustering of patterns, this application tries to recognize the input patterns with an error and finds the closest match. Here, n = 35, m = 26 (Number of cluster units), and T = 26 (Number of input patterns).***



(A)

The input

$\underline{1}$ 0 1 0 0
0 1 0 1 0
1 0 0 0 1
1 0 0 0 1
1 1 1 1 1
1 0 0 0 1
1 0 0 0 1

is closer to

0 0 1 0 0
0 1 0 1 0
1 0 0 0 1
1 0 0 0 1
1 1 1 1 1
1 0 0 0 1
1 0 0 0 1

(I)

The input

0 1 $\underline{0}$ 1 0
0 0 1 0 0
0 0 1 0 0
0 0 1 0 0
0 0 1 0 0
0 0 1 0 0
0 1 1 1 0

is closer to

0 1 1 1 0
0 0 1 0 0
0 0 1 0 0
0 0 1 0 0
0 0 1 0 0
0 0 1 0 0
0 1 1 1 0

***Simulation Results of HNN for Character Recognition***

# HOPFIELD NEURAL NETWORK (HNN)

***John Hopfield -1982- recurrent artificial neural network***

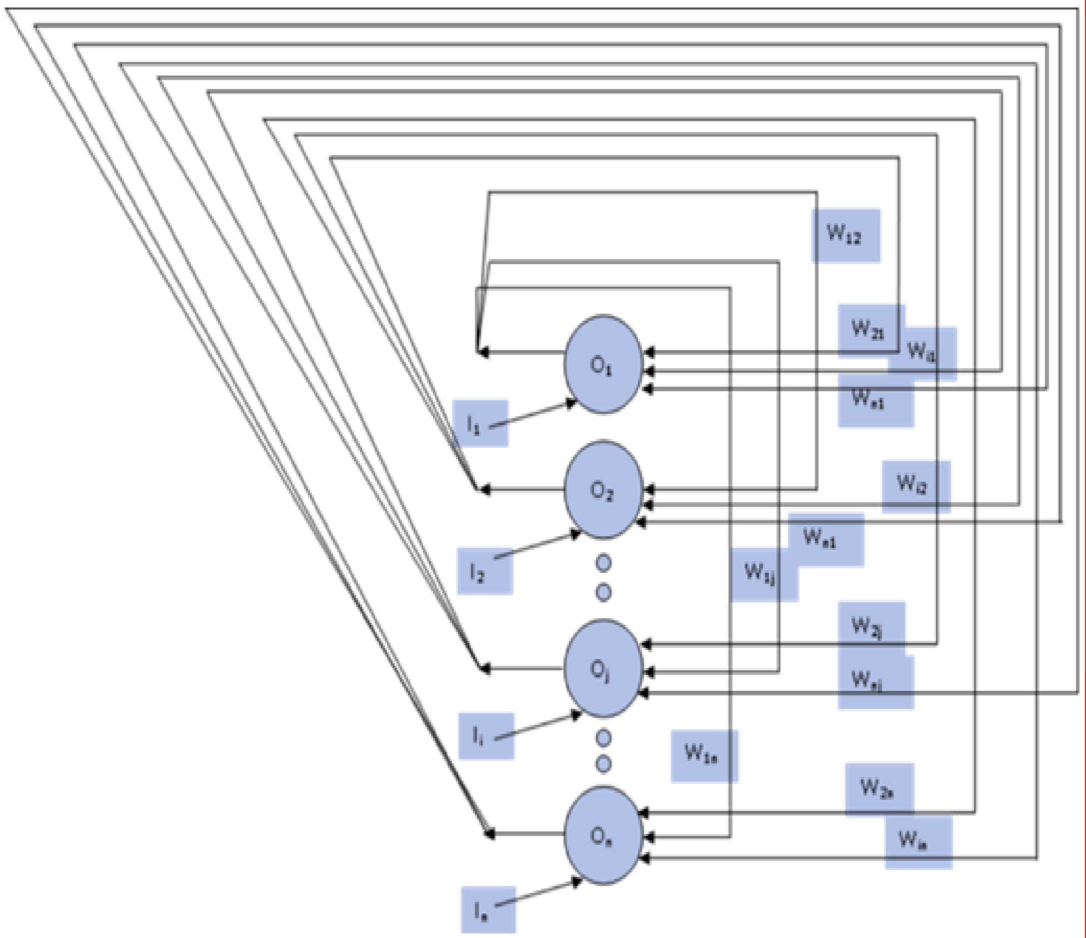
***It is used as a content-addressable memory systems with binary threshold units.***

***A content-addressable memory systems allows the recall of data on the degree of similarity between the input patterns and the patterns stored in memory.***

***Hopfield neural networks is an example of Associative memory neural networks (AMNNs).***

***AMNNs are single-layer nets in which the weights are determined for the network to store a set of pattern associations.***

***In the Hopfield network, only one unit updates it activations at a time based on the signals it receives from each other unit. Also, each unit continues to receive an external signal in addition to the signal from other units in the net.***



***Hopfield Neural Network***

# HOPFIELD NEURAL NETWORK (HNN)

## *Illustration of Settlement of Stable Input Patterns: Hand worked example*

*Test 3 binary input patterns and find the patterns that settles or converges to any one of 2 standard binary patterns S(1) = (1 1 1 1 0 0) and S(2) = (0 1 1 1 1 1).*

*Here, P = 2, T = 3 and n = 6*

$$I = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}_{T\times n} \begin{matrix} (I_1) \\ (I_2) \\ (I_3) \\ (I_4) \end{matrix}$$

$$\begin{aligned} &when\ h = 1 \\ &W_{11} = 0 \\ &W_{12} = ((2\times 1)-1)\times((2\times 1)-1) = 1 \\ &\vdots \\ &W_{65} = ((2\times 0)-1)\times((2\times 0)-1) = 1 \\ &W_{66} = 0 \end{aligned}$$

$$\begin{aligned} &when\ h = 2 \\ &W_{11} = 0 \\ &W_{12} = ((2\times 0)-1)\times((2\times 1)-1) = -1 \\ &\vdots \\ &W_{65} = ((2\times 1)-1)\times((2\times 1)-1) = 1 \\ &W_{66} = 0 \end{aligned}$$

***<u>Initialization of weights as per Hebb rule for binary numbers</u>***

*The weight of the binary patterns are*

$$W_{ij} = \sum_{h=1}^{P} (2S_i(h)-1)\times(2S_j(h)-1), where\ i = 1,\ldots n,\ j = 1,\ldots n\ \ and\ i \neq j$$

$$W_{ij} = 0\ \ for\ \ i = j$$

*If the patterns to be handled are bipolar, then the weights are,*

$$W_{ij} = \sum_{h=1}^{P} S_i(h)\times S_j(h), where\ i = 1,\ldots n,\ j = 1,\ldots n\ \ and\ i \neq j$$

$$W_{ij} = 0\ \ for\ \ i = j$$

*Here the weights are initialized as,*

$$W_{ij} = \sum_{h=1}^{2} (2S_i(h)-1)\times(2S_j(h)-1), where\ i = 1,\ldots 6,\ j = 1,\ldots 6\ \ and\ i \neq j$$

$$W_{ij} = 0\ \ for\ \ i = j$$

# HOPFIELD NEURAL NETWORK (HNN)

## *Illustration of Settlement of Stable Input Patterns: Hand worked example*

*Test 3 binary input patterns and find the patterns that settles or converges to any one of 2 standard binary patterns S(1) = (1 1 1 1 0 0) and S(2) = (0 1 1 1 1 1).*

$$W_{ij} = \begin{bmatrix} 0 & 1 & 1 & 1 & -1 & -1 \\ 1 & 0 & 1 & 1 & -1 & -1 \\ 1 & 1 & 0 & 1 & -1 & -1 \\ 1 & 1 & 1 & 0 & -1 & -1 \\ -1 & -1 & -1 & -1 & 0 & 1 \\ -1 & -1 & -1 & -1 & 1 & 0 \end{bmatrix}_{h=1} + \begin{bmatrix} 0 & -1 & -1 & -1 & -1 & -1 \\ -1 & 0 & 1 & 1 & 1 & 1 \\ -1 & 1 & 0 & 1 & 1 & 1 \\ -1 & 1 & 1 & 0 & 1 & 1 \\ -1 & 1 & 1 & 1 & 0 & 1 \\ -1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}_{h=2} = \begin{bmatrix} 0 & 0 & 0 & 0 & -2 & -2 \\ 0 & 0 & 2 & 2 & 0 & 0 \\ 0 & 2 & 0 & 2 & 0 & 0 \\ 0 & 2 & 2 & 0 & 0 & 0 \\ -2 & 0 & 0 & 0 & 0 & 2 \\ -2 & 0 & 0 & 0 & 2 & 0 \end{bmatrix}_{n\times n}$$

# HOPFIELD NEURAL NETWORK (HNN)

***Illustration of Settlement of Stable Input Patterns: Hand worked example***

*Test 3 binary input patterns and find the patterns that settles or converges to any one of 2 standard binary patterns S(1) = (1 1 1 1 0 0) and S(2) = (0 1 1 1 1 1).*

**Pattern 1**
***(1 1 1 0 1 0)***

*Let the order of the asynchronous updation of units be [4 3 6 5 1 2].*
*Computing the net input to the units (k=1)*

$$O_{1,net\_4} = 0 + (1\times 0) + (1\times 2) + (1\times 2) + (0\times 0) + (1\times 0) + (0\times 0) = 4$$

$$O_{14} = 1, \because O_{1,net\_4} > 0 \,,\, O_1 = (1 \quad 1 \quad 1 \quad \underline{1} \quad 1 \quad 0)$$

$$O_{1,net\_3} = 1 + (1\times 0) + (1\times 2) + (1\times 0) + (1\times 2) + (1\times 0) + (0\times 0) = 5$$

$$O_{13} = 1, \because O_{1,net\_3} > 0 \,,\, O_1 = (1 \quad 1 \quad 1 \quad \underline{1} \quad 1 \quad 0)$$

$$O_{1,net\_6} = 0 + (1\times -2) + (1\times 0) + (1\times 0) + (1\times 0) + (1\times 2) + (0\times 0) = 0$$

$$O_{16} = 0, \because O_{1,net\_6} = 0 \,,\, O_1 = (1 \quad 1 \quad 1 \quad \underline{1} \quad 1 \quad 0)$$

$$O_{1,net\_5} = 1 + (1\times -2) + (1\times 0) + (1\times 0) + (1\times 0) + (1\times 0) + (0\times 2) = -1$$

$$O_{15} = 0, \because O_{1,net\_5} < 0 \,,\, O_1 = (1 \quad 1 \quad 1 \quad \underline{1} \quad \underline{0} \quad 0)$$

$$O_{1,net\_1} = 1 + (1\times 0) + (1\times 0) + (1\times 0) + (1\times 0) + (0\times -2) + (0\times -2) = 1$$

$$O_{11} = 1, \because O_{1,net\_1} > 0 \,,\, O_1 = (1 \quad 1 \quad 1 \quad \underline{1} \quad \underline{0} \quad 0)$$

$$O_{1,net\_2} = 1 + (1\times 0) + (1\times 0) + (1\times 2) + (1\times 2) + (0\times 0) + (0\times 0) = 5$$

$$O_{12} = 1, \because O_{1,net\_2} > 0 \,,\, O_1 = (1 \quad 1 \quad 1 \quad \underline{1} \quad \underline{0} \quad 0) \rightarrow S(1) = (1 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0)\text{. Converged}$$

# HOPFIELD NEURAL NETWORK (HNN)

## *Illustration of Settlement of Stable Input Patterns: Hand worked example*

*Test 3 binary input patterns and find the patterns that settles or converges to any one of 2 standard binary patterns S(1) = (1 1 1 1 0 0) and S(2) = (0 1 1 1 1 1).*

**Pattern 2**
***(0 1 0 1 1 1)***

*Let the order of the asynchronous updation of units be [3 1 6 4 2 5].*
*Computing the net input to the units (k=2)*

$$O_{2,net\_3} = 0+(0\times 0)+(1\times 2)+(0\times 2)+(1\times 2)+(1\times 0)+(1\times 0)=4$$

$$O_{23}=1, \because O_{2,net\_3}>0, O_2=(0 \quad 1 \quad \underline{1} \quad 1 \quad 1 \quad 1)$$

$$O_{2,net\_1} = 0+(0\times 0)+(1\times 0)+(1\times 0)+(1\times 0)+(1\times -2)+(1\times -2)=-4$$

$$O_{21}=-1, \because O_{2,net\_1}<0, O_2=(0 \quad 1 \quad \underline{1} \quad 1 \quad 1 \quad 1)$$

$$O_{2,net\_6} = 1+(0\times -2)+(1\times 0)+(1\times 0)+(1\times 0)+(1\times 2)+(1\times 0)=3$$

$$O_{26}=1, \because O_{2,net\_6}>0, O_2=(0 \quad 1 \quad \underline{1} \quad 1 \quad 1 \quad 1)$$

$$O_{2,net\_4} = 1+(0\times 0)+(1\times 2)+(1\times 2)+(1\times 0)+(1\times 0)+(1\times 0)=5$$

$$O_{24}=1, \because O_{2,net\_4}>0, O_2=(0 \quad 1 \quad \underline{1} \quad 1 \quad 1 \quad 1)$$

$$O_{2,net\_2} = 1+(0\times 0)+(1\times 0)+(1\times 2)+(1\times 2)+(1\times 0)+(1\times 0)=5$$

$$O_{22}=1, \because O_{2,net\_2}>0, O_2=(0 \quad 1 \quad \underline{1} \quad 1 \quad 1 \quad 1)$$

$$O_{2,net\_5} = 0+(0\times -2)+(1\times 0)+(1\times 0)+(1\times 0)+(1\times 0)+(1\times 2)=3$$

$$O_{25}=1, \because O_{2,net\_5}>0, O_2=(0 \quad 1 \quad \underline{1} \quad 1 \quad 1 \quad 1)\rightarrow S(2)=(0 \; 1 \; 1 \; 1 \; 1 \; 1), \text{Converged}$$

# HOPFIELD NEURAL NETWORK (HNN)

## *Illustration of Settlement of Stable Input Patterns: Hand worked example*

*Test 3 binary input patterns and find the patterns that settles or converges to any one of 2 standard binary patterns S(1) = (1 1 1 1 0 0) and S(2) = (0 1 1 1 1 1).*

**Pattern 3**
***(0 0 1 1 1 1)***

*Let the order of the asynchronous updation of units be [4 2 1 6 5 3].*
*Computing the net input to the units (k=3)*

$$O_{3,net\_4} = 1 + (0\times0) + (0\times2) + (1\times2) + (1\times0) + (1\times0) + (1\times0) = 3$$

$$O_{34} = 1, \because O_{3,net\_4} > 0, O_3 = (0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1)$$

$$O_{3,net\_2} = 0 + (0\times0) + (0\times0) + (1\times2) + (1\times2) + (1\times0) + (1\times0) = 4$$

$$O_{32} = 1, \because O_{3,net\_2} > 0, O_3 = (0 \quad \underline{1} \quad 1 \quad 1 \quad 1 \quad 1)$$

$$O_{3,net\_1} = 0 + (0\times0) + (1\times0) + (1\times0) + (1\times0) + (1\times-2) + (1\times-2) = -4$$

$$O_{31} = 0, \because O_{3,net\_1} < 0, O_3 = (0 \quad \underline{1} \quad 1 \quad 1 \quad 1 \quad 1)$$

$$O_{3,net\_6} = 1 + (0\times-2) + (1\times0) + (1\times0) + (1\times0) + (1\times2) + (1\times0) = 3$$

$$O_{36} = 1, \because O_{3,net\_6} > 0, O_3 = (0 \quad \underline{1} \quad 1 \quad 1 \quad 1 \quad 1)$$

$$O_{3,net\_5} = 1 + (0\times-2) + (1\times0) + (1\times0) + (1\times0) + (1\times0) + (1\times2) = 3$$

$$O_{35} = 1, \because O_{3,net\_5} > 0, O_3 = (0 \quad \underline{1} \quad 1 \quad 1 \quad 1 \quad 1)$$

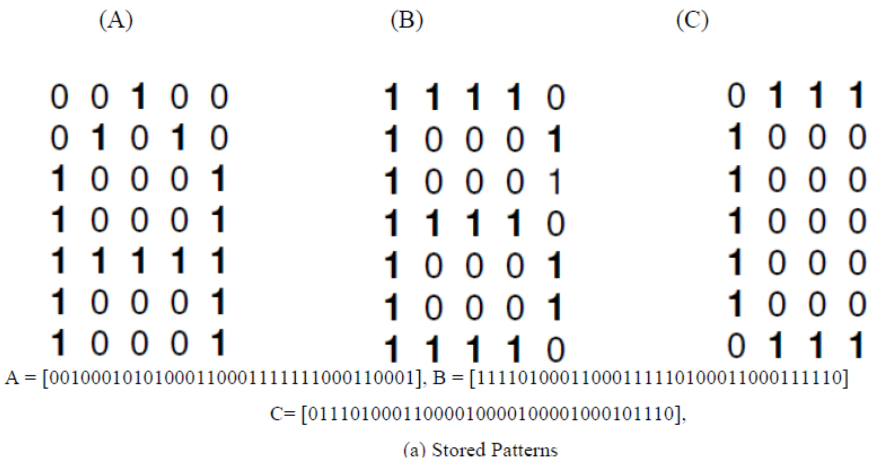
$$O_{3,net\_3} = 1 + (0\times0) + (1\times2) + (1\times0) + (1\times2) + (1\times0) + (1\times0) = 5$$

$$O_{33} = 1, \because O_{3,net\_3} > 0, O_3 = (0 \quad \underline{1} \quad 1 \quad 1 \quad 1 \quad 1) \rightarrow S(2) = (0 \; 1 \; 1 \; 1 \; 1 \; 1), \text{Converged}$$

# HOPFIELD NEURAL NETWORK (HNN)

## *Character Recognition through Stabilization of Input Test Patterns*

- ***The Hopfield network can be used for pattern recognition to identify the standard pattern associated with the input test pattern.***
- ***Here, 3 alphabets (A, B & C) are the standard patterns.***
- ***The representation of an alphabet is by a matrix of 7X5 binary elements.***



(A)

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 |

(B)

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |

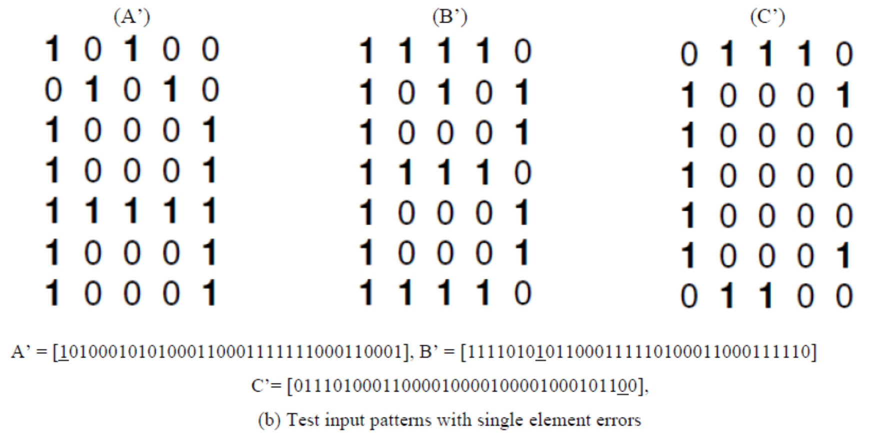
(C)

| | | | | |
|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |

A = [00100010101000110001111111000110001], B = [11110100011000111110100011000111110]

C= [01110100011000010000100001000101110],

(a) Stored Patterns



(A')

| | | | | |
|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 |

(B')

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |

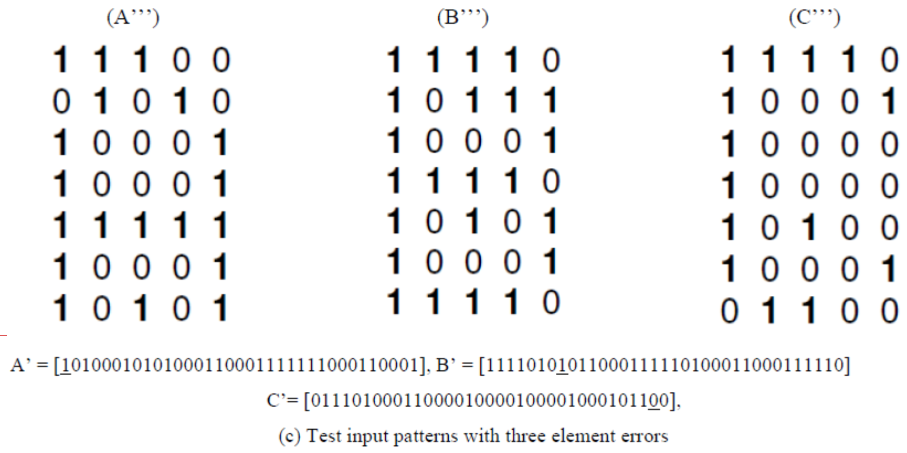
(C')

| | | | | |
|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 |

A' = [10100010101000110001111111000110001], B' = [11110101011000111110100011000111110]

C'= [01110100011000010000100001000101100],

(b) Test input patterns with single element errors



(A''')

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 |

(B''')

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |

(C''')

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 |

A' = [10100010101000110001111111000110001], B' = [11110101011000111110100011000111110]

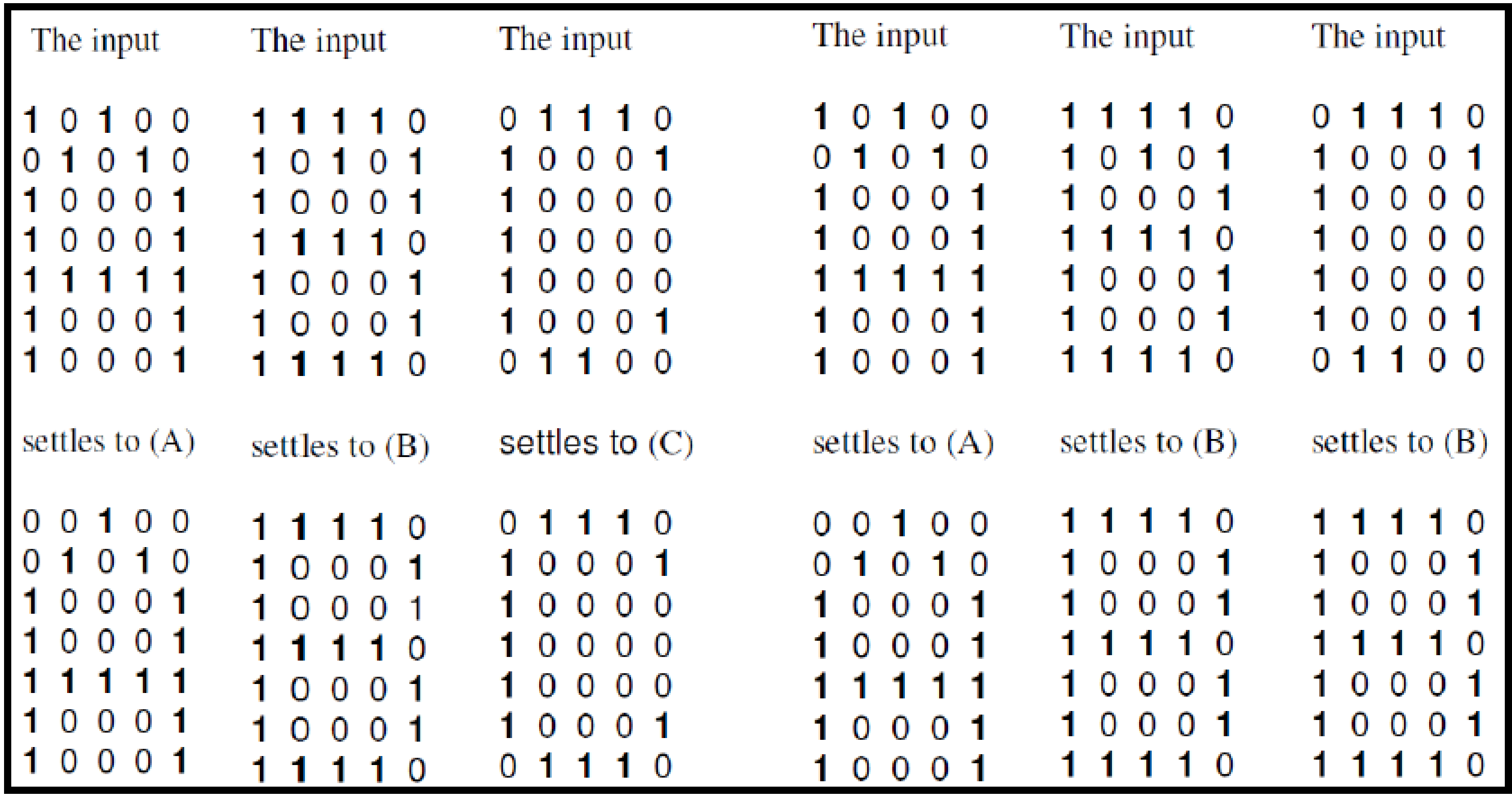
C'= [01110100011000010000100001000101100],

(c) Test input patterns with three element errors

# HOPFIELD NEURAL NETWORK (HNN)

***Character Recognition through Stabilization of Input Test Patterns***

***Simulation Results for Test Input Patterns with Single Element Errors***

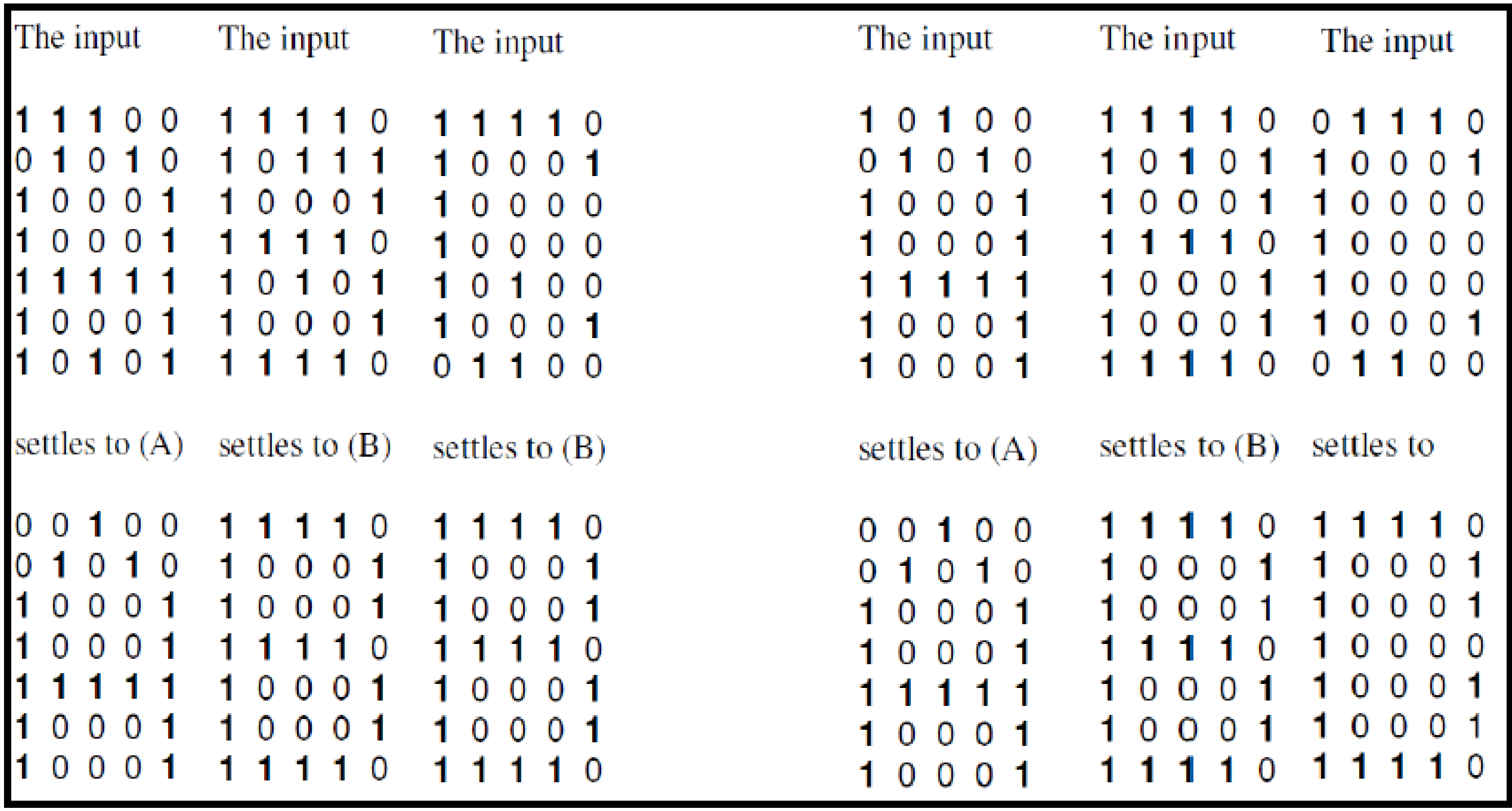


| | | | | | |
|---|---|---|---|---|---|
| The input | The input | The input | The input | The input | The input |
| 1 0 1 0 0 | 1 1 1 1 0 | 0 1 1 1 0 | 1 0 1 0 0 | 1 1 1 1 0 | 0 1 1 1 0 |
| 0 1 0 1 0 | 1 0 1 0 1 | 1 0 0 0 1 | 0 1 0 1 0 | 1 0 1 0 1 | 1 0 0 0 1 |
| 1 0 0 0 1 | 1 0 0 0 1 | 1 0 0 0 0 | 1 0 0 0 1 | 1 0 0 0 1 | 1 0 0 0 0 |
| 1 0 0 0 1 | 1 1 1 1 0 | 1 0 0 0 0 | 1 0 0 0 1 | 1 1 1 1 0 | 1 0 0 0 0 |
| 1 1 1 1 1 | 1 0 0 0 1 | 1 0 0 0 0 | 1 1 1 1 1 | 1 0 0 0 1 | 1 0 0 0 0 |
| 1 0 0 0 1 | 1 0 0 0 1 | 1 0 0 0 1 | 1 0 0 0 1 | 1 0 0 0 1 | 1 0 0 0 1 |
| 1 0 0 0 1 | 1 1 1 1 0 | 0 1 1 0 0 | 1 0 0 0 1 | 1 1 1 1 0 | 0 1 1 0 0 |
| settles to (A) | settles to (B) | settles to (C) | settles to (A) | settles to (B) | settles to (B) |
| 0 0 1 0 0 | 1 1 1 1 0 | 0 1 1 1 0 | 0 0 1 0 0 | 1 1 1 1 0 | 1 1 1 1 0 |
| 0 1 0 1 0 | 1 0 0 0 1 | 1 0 0 0 1 | 0 1 0 1 0 | 1 0 0 0 1 | 1 0 0 0 1 |
| 1 0 0 0 1 | 1 0 0 0 1 | 1 0 0 0 0 | 1 0 0 0 1 | 1 0 0 0 1 | 1 0 0 0 1 |
| 1 0 0 0 1 | 1 1 1 1 0 | 1 0 0 0 0 | 1 0 0 0 1 | 1 1 1 1 0 | 1 1 1 1 0 |
| 1 1 1 1 1 | 1 0 0 0 1 | 1 0 0 0 0 | 1 1 1 1 1 | 1 0 0 0 1 | 1 0 0 0 1 |
| 1 0 0 0 1 | 1 0 0 0 1 | 1 0 0 0 1 | 1 0 0 0 1 | 1 0 0 0 1 | 1 0 0 0 1 |
| 1 0 0 0 1 | 1 1 1 1 0 | 0 1 1 1 0 | 1 0 0 0 1 | 1 1 1 1 0 | 1 1 1 1 0 |

# HOPFIELD NEURAL NETWORK (HNN)

***Character Recognition through Stabilization of Input Test Patterns***

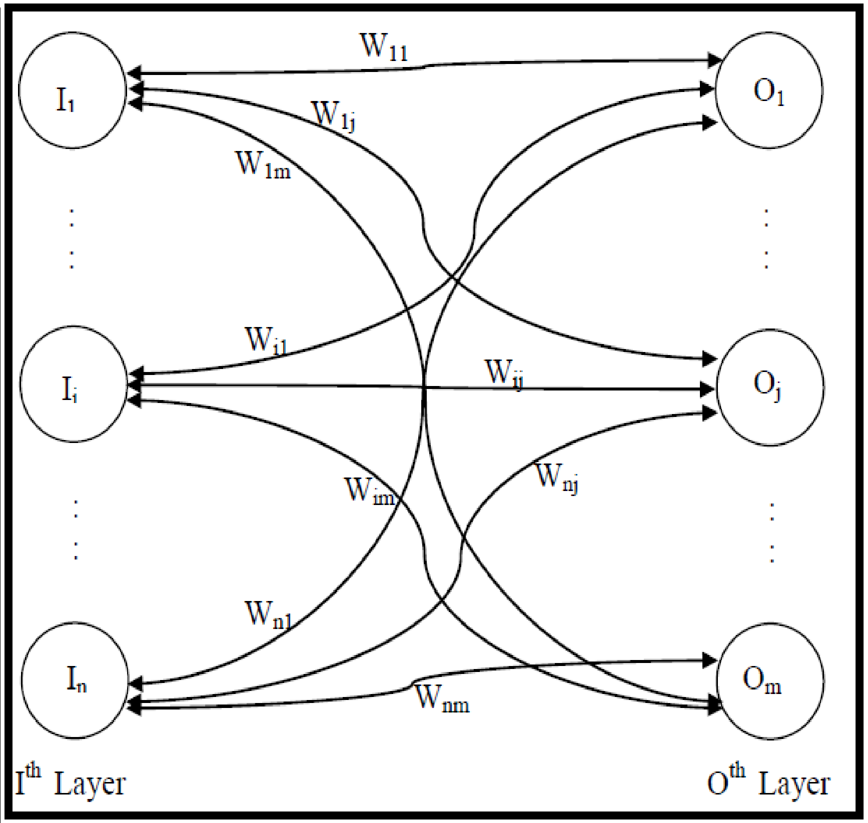
***Simulation Results***



| The input | The input | The input | The input | The input | The input |
|---|---|---|---|---|---|
| 1 1 1 0 0 | 1 1 1 1 0 | 1 1 1 1 0 | 1 0 1 0 0 | 1 1 1 1 0 | 0 1 1 1 0 |
| 0 1 0 1 0 | 1 0 1 1 1 | 1 0 0 0 1 | 0 1 0 1 0 | 1 0 1 0 1 | 1 0 0 0 1 |
| 1 0 0 0 1 | 1 0 0 0 1 | 1 0 0 0 0 | 1 0 0 0 1 | 1 0 0 0 1 | 1 0 0 0 0 |
| 1 0 0 0 1 | 1 1 1 1 0 | 1 0 0 0 0 | 1 0 0 0 1 | 1 1 1 1 0 | 1 0 0 0 0 |
| 1 1 1 1 1 | 1 0 1 0 1 | 1 0 1 0 0 | 1 1 1 1 1 | 1 0 0 0 1 | 1 0 0 0 0 |
| 1 0 0 0 1 | 1 0 0 0 1 | 1 0 0 0 1 | 1 0 0 0 1 | 1 0 0 0 1 | 1 0 0 0 1 |
| 1 0 1 0 1 | 1 1 1 1 0 | 0 1 1 0 0 | 1 0 0 0 1 | 1 1 1 1 0 | 0 1 1 0 0 |
| settles to (A) | settles to (B) | settles to (B) | settles to (A) | settles to (B) | settles to |
| 0 0 1 0 0 | 1 1 1 1 0 | 1 1 1 1 0 | 0 0 1 0 0 | 1 1 1 1 0 | 1 1 1 1 0 |
| 0 1 0 1 0 | 1 0 0 0 1 | 1 0 0 0 1 | 0 1 0 1 0 | 1 0 0 0 1 | 1 0 0 0 1 |
| 1 0 0 0 1 | 1 0 0 0 1 | 1 0 0 0 1 | 1 0 0 0 1 | 1 0 0 0 1 | 1 0 0 0 1 |
| 1 0 0 0 1 | 1 1 1 1 0 | 1 1 1 1 0 | 1 0 0 0 1 | 1 1 1 1 0 | 1 0 0 0 0 |
| 1 1 1 1 1 | 1 0 0 0 1 | 1 0 0 0 1 | 1 1 1 1 1 | 1 0 0 0 1 | 1 0 0 0 1 |
| 1 0 0 0 1 | 1 0 0 0 1 | 1 0 0 0 1 | 1 0 0 0 1 | 1 0 0 0 1 | 1 0 0 0 1 |
| 1 0 0 0 1 | 1 1 1 1 0 | 1 1 1 1 0 | 1 0 0 0 1 | 1 1 1 1 0 | 1 1 1 1 0 |

# BIDIRECTIONAL ASSOCIATIVE MEMORY (BAM)

- ***Bart Kosko -1988***
- ***BAM has the properties of two-layer non-linear feedback neural networks***
- ***Heteroassociative information is encoded in a BAM by summing correlation weights matrices obtained from the associative pairs of the binary or bipolar patterns. The architecture of the BAM consists of two layers of neurons, connected by bi-directional weights***
- ***The weights of the BAM are initialized based on the Hebb rule.***



***BAM Architecture***

## BIDIRECTIONAL ASSOCIATIVE MEMORY (BAM)

- ***Suppose s(h) and T(k) are associated P paired patterns.***
- ***The bidirectional weights for the P paired binary patterns are***

$$W_{ij} = \sum_{p=1}^{P} (2S_i(k)-1)\times(2T_j(k)-1), where\ i=1,\dots n,\ j=1,\dots m$$

- ***The bidirectional weights of for the P paired bipolar patterns are***

$$W_{ij} = \sum_{p=1}^{P} S_i(k)\times T_j(k), where\ i=1,\dots n,\ j=1,\dots m$$

- ***The activation functions of the and layers for the binary vectors are I^th and O^th***

$$I_{tj} = \begin{Bmatrix} 1 & if & I_{t,net\_i} > \theta_i \\ I_{ti} & if & I_{t,net\_i} = \theta_i \\ 0 & if & I_{t,net\_i} < \theta_i \end{Bmatrix}$$

$$O_{tj} = \begin{Bmatrix} 1 & if & O_{t,net\_j} > \theta_j \\ O_{tj} & if & O_{t,net\_j} = \theta_j \\ 0 & if & O_{tk,net\_j} < \theta_j \end{Bmatrix}$$

- ***For the bipolar vectors,***

$$I_{tj} = \begin{Bmatrix} 1 & if & I_{t,net\_i} > \theta_i \\ I_{ti} & if & I_{t,net\_i} = \theta_i \\ -1 & if & I_{t,net\_i} < \theta_i \end{Bmatrix}$$

$$O_{tj} = \begin{Bmatrix} 1 & if & O_{t,net\_j} > \theta_j \\ O_{tj} & if & O_{t,net\_j} = \theta_j \\ -1 & if & O_{tk,net\_j} < \theta_j \end{Bmatrix}$$

## BIDIRECTIONAL ASSOCIATIVE MEMORY (BAM)

***Illustration of Settlement of Stable Input Patterns***

- ***Store 2 paired binary standard patterns in BAM and test 4 binary test patterns.***

- ***Standard Binary Patterns***

  A= [00100010101000110001111111000110001] is paired with 1= [01]

  B= [11110100011000111110100011000111110] is paired with 2= [10]

- ***Binary test patterns***

  A = [00100010101000110001111111000110001]

  B = [11110100011000111110100011000111110]

  A’ = [10100010101000110001111111000110001]

  B’ = [11110101011000111110100011000111110]

- ***Here, P = 2, N = 4, n = 35 and m = 2.***

# BIDIRECTIONAL ASSOCIATIVE MEMORY (BAM)

## *Illustration of Settlement of Stable Input Patterns: Hand worked example*

- ***Store 2 paired binary standard patterns in BAM and test 4 binary test patterns.***

- ***Activation of the and layer of BAM I<sup>th</sup> O<sup>th</sup>***

$$I = \begin{bmatrix} 00100010101000110001111111000110001 \\ 11110100011000111110100011000111110 \\ 10100010101000110001111111000110001 \\ 11110101011000111110100011000111110 \end{bmatrix}_{N\times n} \begin{matrix} (I_1) \\ (I_2) \\ (I_3) \\ (I_4) \end{matrix} \qquad O = \begin{bmatrix} 0 \ 0 \\ 0 \ 0 \\ 0 \ 0 \\ 0 \ 0 \end{bmatrix}_{N\times m} \begin{matrix} (O_1) \\ (O_2) \\ (O_3) \\ (O_4) \end{matrix}$$

- ***The P paired binary patterns are stored as weights using Hebb rule***

$$S = \begin{bmatrix} 00100010101000110001111111000110001 \\ 11110100011000111110100011000111110 \end{bmatrix}_{P\times n} \qquad T = \begin{bmatrix} 01 \\ 00 \end{bmatrix}_{P\times m}$$

- ***The weights are initialized***

$$W_{ij} = \sum_{p=1}^{2} S_i(h) \times T_j(h), \text{where } i = 1,\ldots 35,\ j = 1,..2$$

$$W_{ij}^{Transpose} = \begin{bmatrix} 2 & 2 & 0 & 2 & 0 & 2 & -2 & 0 & -2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 & 2 & -2 & 0 & -2 & -2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 & 2 & -2 \\ -2 & -2 & 0 & -2 & 0 & -2 & 2 & 0 & 2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & -2 & -2 & 2 & 0 & 2 & 2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & -2 & -2 & 2 \end{bmatrix}$$

# BIDIRECTIONAL ASSOCIATIVE MEMORY (BAM)

## *Illustration of Settlement of Stable Input Patterns : Hand worked example*

- ***Store 2 paired binary standard patterns in BAM and test 4 binary test patterns.***

***Pattern 1- [00100010101000110001111111000110001]***

***Pattern 1- [00100010101000110001111111000110001]***
***Computing the net input to the output units (t=1)***

$$O_{t,net\_1} = \bar{I}_{1^{st}\ row\times 35} \times \bar{W}_{35\times 1^{st}\ column} = -1\bar{4}$$

$$O_{1,1} = 0, \because O_{1,net\_1} < 0,$$

$$O_{t,net\_2} = I_{1^{st}\ row\times 35} \times W_{35\times 2^{nd}\ column} = 14$$

$$O_{1,2} = 1, \because O_{1,net\_2} > 0,$$

$$O = \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}_{N\times m} \begin{matrix} (O_1) \\ (O_2) \\ (O_3) \\ (O_4) \end{matrix}$$

- ***Computing the net input to the input units (t=1)***

$$I_{t,net\_1} = O_{1^{st}\ row\times 2} \times W_{2\times 1^{st}\ column} = -2$$

$$I_{1,1} = 0, \because I_{1,net\_1} < 0,$$

$$I_{t,net\_2} = O_{1^{st}\ row\times 2} \times W_{2\times 2^{nd}\ column} = -2$$

$$I_{1,2} = 0, \because I_{1,net\_2} < 0,$$

$$\vdots$$

$$I_{t,net\_35} = O_{1^{st}\ row\times 2} \times W_{2\times 35^{th}\ column} = 2$$

$$I_{1,35} = 1, \because I_{1,net\_35} > 0,$$

$$I_{t=1,net\_n} = [-2\ -2\ 0\ -2\ 0\ -2\ 2\ 0\ 2\ -2\ 0\ 0\ 0\ 0\ 0\ 0\ -2\ -2\ -2\ 2\ 0\ 2\ 2\ 2\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ -2\ -2\ -2\ 2]_{1\times n}$$

$$I_{t=1,n} = [0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1]_{1\times n}$$

***checking the equilibrium state, the activations of the output unit $O_{1,m}$=[0 1] has already become equal to $T_{1,m}$= [ 0 1] the , A = [00100010101000110001111111000110001], the input test pattern ( t=1) has converged and A is associated with 1= [01] ]***

# BIDIRECTIONAL ASSOCIATIVE MEMORY (BAM)

## *Illustration of Settlement of Stable Input Patterns : Hand worked example*

***Store 2 paired binary standard patterns in BAM and test 4 binary test patterns.***

***Computing the net input to the output units (t=2)***

$$O_{t,net\_1} = I_{2^{nd}\ row \times 35} \times W_{35\times 1^{st}\ column} = 22$$

$$O_{2,1} = 1, \because O_{2,net\_1} > 0,$$

$$O_{t,net\_2} = I_{2^{nd}\ row \times 35} \times W_{35\times 2^{nd}\ column} = -22$$

$$O_{2,2} = 0, \because O_{1,net\_2} < 0,$$

$$O = \begin{bmatrix} 0\ 1 \\ 1\ 0 \\ 0\ 0 \\ 0\ 0 \end{bmatrix}_{N\times m} \begin{matrix} (O_1) \\ (O_2) \\ (O_3) \\ (O_4) \end{matrix}$$

***Pattern 2- [11110100011000111110100011000111110]***

***Computing the net input to the input units (t=2)***

$$I_{t,net\_1} = O_{2^{nd}\ row \times 2} \times W_{2\times 1^{st}\ column} = 2$$

$$I_{2,1} = 1, \because I_{2,net\_1} > 0,$$

$$I_{t,net\_2} = O_{2^{nd}\ row \times 2} \times W_{2\times 2^{nd}\ column} = 2$$

$$I_{2,2} = 1, \because I_{2,net\_2} > 0,$$

$$\vdots$$

$$I_{t,net\_35} = O_{2^{nd}\ row \times 2} \times W_{2\times 35^{th}\ column} = -2$$

$$I_{2,35} = 0, \because I_{2,net\_35} < 0,$$

$$I_{t=2,net\_n} = [2\ 2\ 0\ 2\ 0\ 2\ {-2}\ 0\ {-2}\ 2\ 0\ 0\ 0\ 0\ 0\ 0\ 2\ 2\ 2\ {-2}\ 0\ {-2}\ {-2}\ {-2}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 2\ 2\ 2\ {-2}]_{1\times n}$$

$$I_{t=2,n} = [1\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 0]_{1\times n}$$

***checking the equilibrium state, the activations of the output unit $O_{2,m}$=[1 0] has already become equal to $T_{2,m}$ =[1 0] the , B = [11110100011000111110100011000111110], the input test pattern ( t=2) has converged and B is associated with 2= [1 0]***

# BIDIRECTIONAL ASSOCIATIVE MEMORY (BAM)

## *Illustration of Settlement of Stable Input Patterns : Hand worked example*

***Store 2 paired binary standard patterns in BAM and test 4 binary test patterns.***

***Computing the net input to the output units (t=3)***

$$O_{t,net\_1} = I_{3^{rd}\ row \times 35} \times W_{35 \times 1^{st}\ column} = -12$$

$$O_{3,1} = 0, \because O_{3,net\_1} < 0,$$

$$O_{t,net\_2} = I_{3^{rd}\ row \times 35} \times W_{35 \times 2^{nd}\ column} = 12$$

$$O_{3,2} = 1, \because O_{3,net\_2} > 0,$$

$$O = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}_{N \times m} \begin{matrix} (O_1) \\ (O_2) \\ (O_3) \\ (O_4) \end{matrix}$$

***Pattern 3- [10100010101000110001111111000110001]***

***Computing the net input to the input units (t=3)***

$$I_{t,net\_1} = O_{3^{rd}\ row \times 2} \times W_{2 \times 1^{st}\ column} = -2$$

$$I_{3,1} = 0, \because I_{3,net\_1} < 0,$$

$$I_{t,net\_2} = O_{3^{rd}\ row \times 2} \times W_{2 \times 2^{nd}\ column} = -2$$

$$I_{3,2} = 0, \because I_{3,net\_2} < 0,$$

$$\vdots$$

$$I_{t,net\_35} = O_{3^{rd}\ row \times 2} \times W_{2 \times 35^{th}\ column} = 2$$

$$I_{3,35} = 1, \because I_{3,net\_35} > 0,$$

$$I_{t=3,net\_n} = [-2\ -2\ 0\ -2\ 0\ -2\ 2\ 0\ 2\ -2\ 0\ 0\ 0\ 0\ 0\ 0\ -2\ -2\ -2\ 2\ 0\ 2\ 2\ 2\ 0\ 0\ 0\ 0\ 0\ 0\ -2\ -2\ -2\ 2]_{1 \times n}$$

$$I_{t=3,n} = [0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1]_{1 \times n}$$

***checking the equilibrium state, the activations of the output unit $O_{3,m}$=[0 1] has already become equal to $T_{3,m}$ =[0 1] the , A′ = [10100010101000110001111111000110001], the input test pattern ( t=3) has converged and $A'$ is associated with 1= [0 1]***

# BIDIRECTIONAL ASSOCIATIVE MEMORY (BAM)

***Illustration of Settlement of Stable Input Patterns : Hand worked example***

***Store 2 paired binary standard patterns in BAM and test 4 binary test patterns.***

***Computing the net input to the output units (t=4)***

$$O_{t,net\_1} = I_{4^{th}\ row \times 35} \times W_{35\times 1^{st}\ column} = 22$$

$$O_{4,1} = 1, \because O_{4,net\_1} > 0,$$

$$O_{t,net\_2} = I_{4^{th}\ row \times 35} \times W_{35\times 2^{nd}\ column} = -22$$

$$O_{4,2} = 0, \because O_{4,net\_2} < 0,$$

$$O = \begin{bmatrix} 0\ 1 \\ 1\ 0 \\ 0\ 1 \\ 1\ 0 \end{bmatrix}_{N\times m} \begin{matrix} (O_1) \\ (O_2) \\ (O_3) \\ (O_4) \end{matrix}$$

***Pattern 4- [11110101011000111110100011000111110]***

***Computing the net input to the input units (t=4)***

$$I_{t,net\_1} = O_{4^{th}\ row \times 2} \times W_{2\times 1^{st}\ column} = 2$$

$$I_{4,1} = 1, \because I_{4,net\_1} > 0,$$

$$I_{t,net\_2} = O_{4^{th}\ row \times 2} \times W_{2\times 2^{nd}\ column} = 2$$

$$I_{4,2} = 1, \because I_{4,net\_2} > 0,$$

$$\vdots$$

$$I_{t,net\_35} = O_{4^{th}\ row \times 2} \times W_{2\times 35^{th}\ column} = -2$$

$$I_{4,35} = 0, \because I_{4,net\_35} < 0,$$

$$I_{t=4,net\_n} = [2\ 2\ 0\ 2\ 0\ 2\ {-2}\ 0\ {-2}\ 2\ 0\ 0\ 0\ 0\ 0\ 0\ 2\ 2\ 2\ {-2}\ 0\ {-2}\ {-2}\ {-2}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 2\ 2\ 2\ {-2}]_{1\times n}$$

$$I_{t=4,n} = [1\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 0]_{1\times n}$$

***checking the equilibrium state, the activations of the output unit $O_{4,m}$=[1 0] has already become equal to $T_{4,m}$ =[0 1] the , B′ = [11110101011000111110100011000111110], the input test pattern ( t=4) has converged and B′ is associated with 2= [1 0]***

# BIDIRECTIONAL ASSOCIATIVE MEMORY (BAM)

***Illustration of Settlement of Stable Input Patterns : Hand worked example***

***Store 2 paired binary standard patterns in BAM and test 4 binary test patterns.***

***Computing the net input to the output units (t=4)***

$$O_{t,net\_1} = I_{4^{th}\ row \times 35} \times W_{35\times 1^{st}\ column} = 22$$

$$O_{4,1} = 1, \because O_{4,net\_1} > 0,$$

$$O_{t,net\_2} = I_{4^{th}\ row \times 35} \times W_{35\times 2^{nd}\ column} = -22$$

$$O_{4,2} = 0, \because O_{4,net\_2} < 0,$$

$$O = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}_{N\times m} \begin{matrix} (O_1) \\ (O_2) \\ (O_3) \\ (O_4) \end{matrix}$$

***Since the all the input test patterns had settled to any one of the stored binary pattern, the iteration process of the algorithm has reached its stoppage criteria.***

***Pattern 4- [11110101011000111110100011000111110]***

***Computing the net input to the input units (t=4)***

$$I_{t,net\_1} = O_{4^{th}\ row \times 2} \times W_{2\times 1^{st}\ column} = 2$$

$$I_{4,1} = 1, \because I_{4,net\_1} > 0,$$

$$I_{t,net\_2} = O_{4^{th}\ row \times 2} \times W_{2\times 2^{nd}\ column} = 2$$

$$I_{4,2} = 1, \because I_{4,net\_2} > 0,$$

$$\vdots$$

$$I_{t,net\_35} = O_{4^{th}\ row \times 2} \times W_{2\times 35^{th}\ column} = -2$$

$$I_{4,35} = 0, \because I_{4,net\_35} < 0,$$

$$I_{t=4,net\_n} = [2\ 2\ 0\ 2\ 0\ 2\ -2\ 0\ -2\ 2\ 0\ 0\ 0\ 0\ 0\ 0\ 2\ 2\ 2\ -2\ 0\ -2\ -2\ -2\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 2\ 2\ 2\ -2]_{1\times n}$$

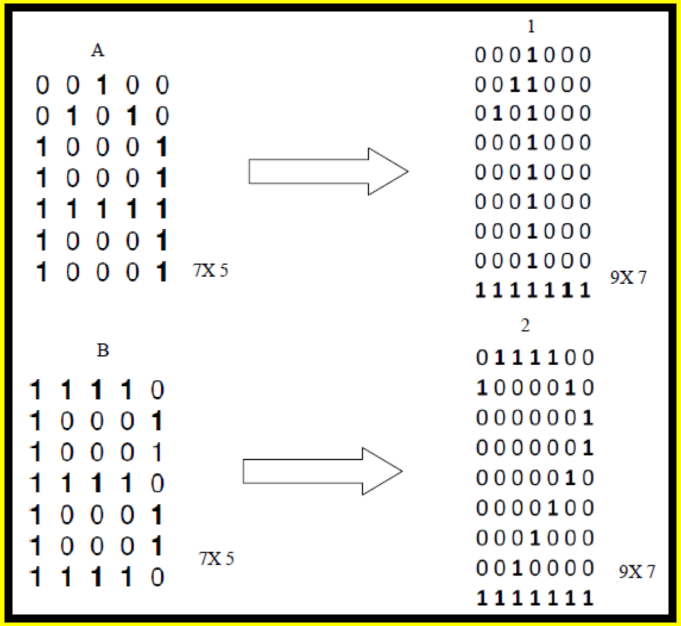
$$I_{t=4,n} = [1\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 0]_{1\times n}$$

***checking the equilibrium state, the activations of the output unit $O_{4,m}$=[1 0] has already become equal to $T_{4,m}$ =[0 1] the , B′ = [11110101011000111110100011000111110], the input test pattern ( t=4) has converged and B′ is associated with 2= [1 0]***

# BIDIRECTIONAL ASSOCIATIVE MEMORY (BAM)

***BAM for Character Mapping***
***BAM can be used for mapping two unrelated patterns through hetero association.***

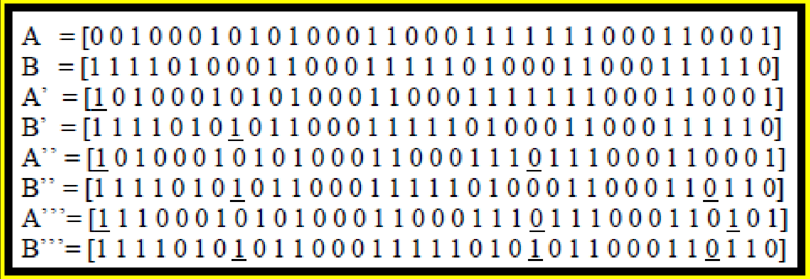
***consider two pairs of binary patterns***



```
     A                            1
0 0 1 0 0                      0001000
0 1 0 1 0                      0011000
1 0 0 0 1                      0101000
1 0 0 0 1          ⇒           0001000
1 1 1 1 1                      0001000
1 0 0 0 1                      0001000
1 0 0 0 1   7X 5               0001000
                               0001000
                               1111111   9X 7

     B                            2
1 1 1 1 0                      0111100
1 0 0 0 1                      1000010
1 0 0 0 1                      0000001
1 1 1 1 0          ⇒           0000001
1 0 0 0 1                      0000010
1 0 0 0 1                      0000100
1 1 1 1 0   7X 5               0001000
                               0010000   9X 7
                               1111111
```

***Eights binary test patterns are given to BAM. Here A′, B′; A′′, B′′ and A′′′, B′′′ are input test patterns with single, double and three element errors***



$A = [0 0 1 0 0 0 1 0 1 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 1 1 0 0 0 1 1 0 0 0 1]$

$B = [1 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 0]$

$A' = [\underline{1} 0 1 0 0 0 1 0 1 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 1 1 0 0 0 1 1 0 0 0 1]$

$B' = [1 1 1 1 0 1 0 \underline{1} 0 1 1 0 0 0 1 1 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 0]$

$A'' = [\underline{1} 0 1 0 0 0 1 0 1 0 1 0 0 0 1 1 0 0 0 1 1 1 \underline{0} 1 1 1 0 0 0 1 1 0 0 0 1]$

$B'' = [1 1 1 1 0 1 0 \underline{1} 0 1 1 0 0 0 1 1 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 \underline{0} 1 1 0]$

$A''' = [\underline{1} 1 1 0 0 0 1 0 1 0 1 0 0 0 1 1 0 0 0 1 1 1 \underline{0} 1 1 1 0 0 0 1 1 0 \underline{1} 0 1]$

$B''' = [1 1 1 1 0 1 0 \underline{1} 0 1 1 0 0 0 1 1 1 1 1 0 1 0 \underline{1} 0 1 1 0 0 0 1 1 \underline{0} 1 1 0]$

***The result shows that all the input test patterns are correctly mapped to its associated counter part, i.e, (A,1); (B,2); (A′,1); (B′,2); (A′′,1); (B′′,2); (A′′′,1); and (B′′′,2).***

# ADAPTIVE RESONANCE THEORY (ART) NEURAL NETWORKS

***Carpenter and Stephen Grosberg (1986)***

- ***The problems with competitive neural networks are***
- ***They always form stable clusters.***
- ***They are oscillatory when more input patterns are presented.***
- ***There is no guarantee that, as more inputs are applied to a neural network used for clustering purpose, the weight matrix will eventually converge and be stable.***
- ***The learning instability occurs because of the network's adaptability (or plasticity), which causes prior learning to be eroded by more recent learning.***

***ART is designed to overcome the problems occurring in learning stability by a modified type of competitive learning called adaptive resonance theory.***

***Types of ART networks:***

- ***ART-1 (1986) that can cluster only binary inputs;***
- ***ART-2 (1987) that can handle gray-scale inputs;***
- ***ART-3 (1989) that can handle analog inputs better;***

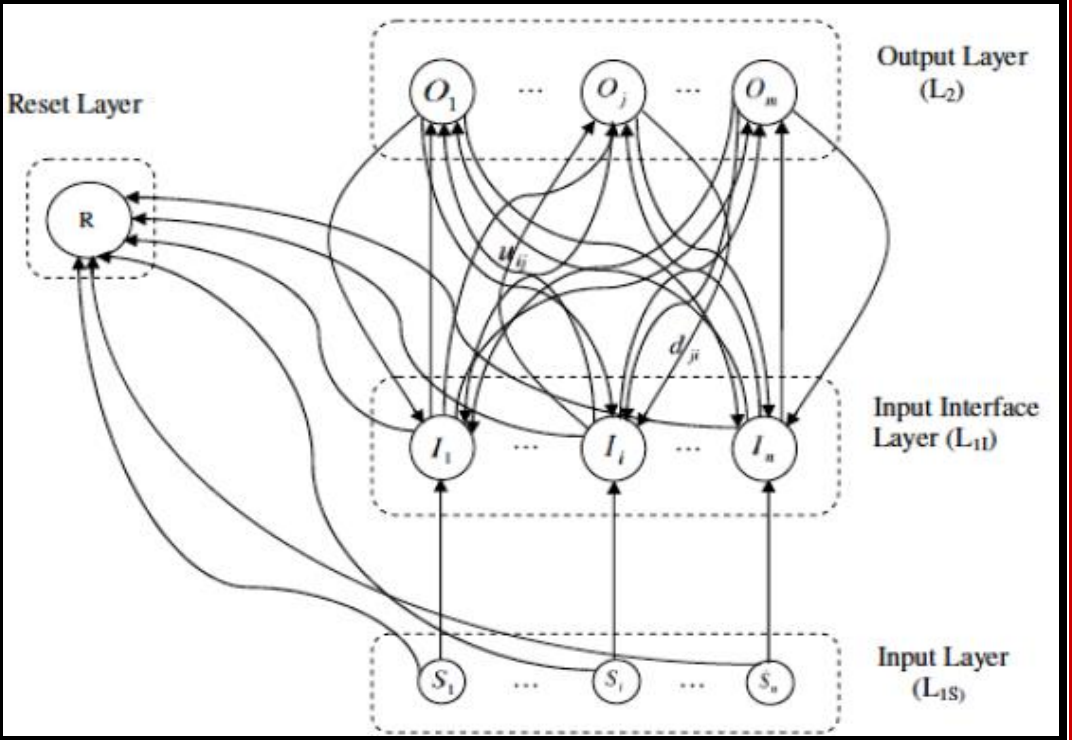
# ADAPTIVE RESONANCE THEORY (ART) NEURAL NETWORKS

***ART uses a degree of expectations called vigilance parameter.***

***Vigilance parameter is the user specified value to decide the degree of similarity essential for the input patterns to be assigned to a cluster unit.***

***Each input it is compared with the prototype vector for a match.***

***If the match between the prototype and the input vector is not adequate, a new prototype or a cluster unit is selected. In this way, previous learned memories (prototypes) are not eroded by new learning.***



***The basic ART learning is an unsupervised one. The term 'resonance' in ART is the state of the network, when a class of a prototype vector very closely matches to the current input vector, leads to a state which permits learning. During this resonant state, the weight updation takes place.***

# ART NEURAL NETWORKS

***Layers in ART***

- ***Input processing layer ($L_1$)- Process the inputs***
- ***Output layer ($L_2$) with the cluster units***
- ***Reset layer (R) - decides the degree of similarity of patterns placed on the same cluster by reset mechanism.***
- ***Input processing layer***
  - ***Input layer($L_{1s}$)***
  - ***Input Interface layer($L_{1I}$)***
- ***Bottom-up weights connect input interface layer and the output layer($u_{ij}$).***
- ***Top-down weights connect the output layer and the interface layer($d_{ij}$).***

***The output layer is a competitive layer or a recognition region where the cluster units participates to check the closeness of the input patterns.***

***The interface layer is usually called the 'comparison region', where it gets an input vector and transfers it to its best match in the recognition region.***

***The best match is the single neuron in the competitive layer whose set of weights closely matches the input vector.***

***The reset layer compares the strength of the recognition match to the vigilance parameter.***

***If the vigilance threshold is met, then the training or the updation of weights takes place, else the firing of the recognition neuron is inhibited until a new input vector is applied***

# ART NEURAL NETWORKS

*<u>Operation of the ART-1</u>*

*A binary input vector is presented to the input layer $L_{1S}$*
*The information is passed to its corresponding units in the input interface layer $L_{1I}$.*
*The interface units transmit the information to the output layer $L_2$ cluster units through the bottom-up weights .*

*The output units compete to become a winner.*
*The largest net input to the output unit usually becomes the winner and the activation becomes 1. All the other output units will have an activation of 0. Let the winning cluster unit's index is 'J'.*

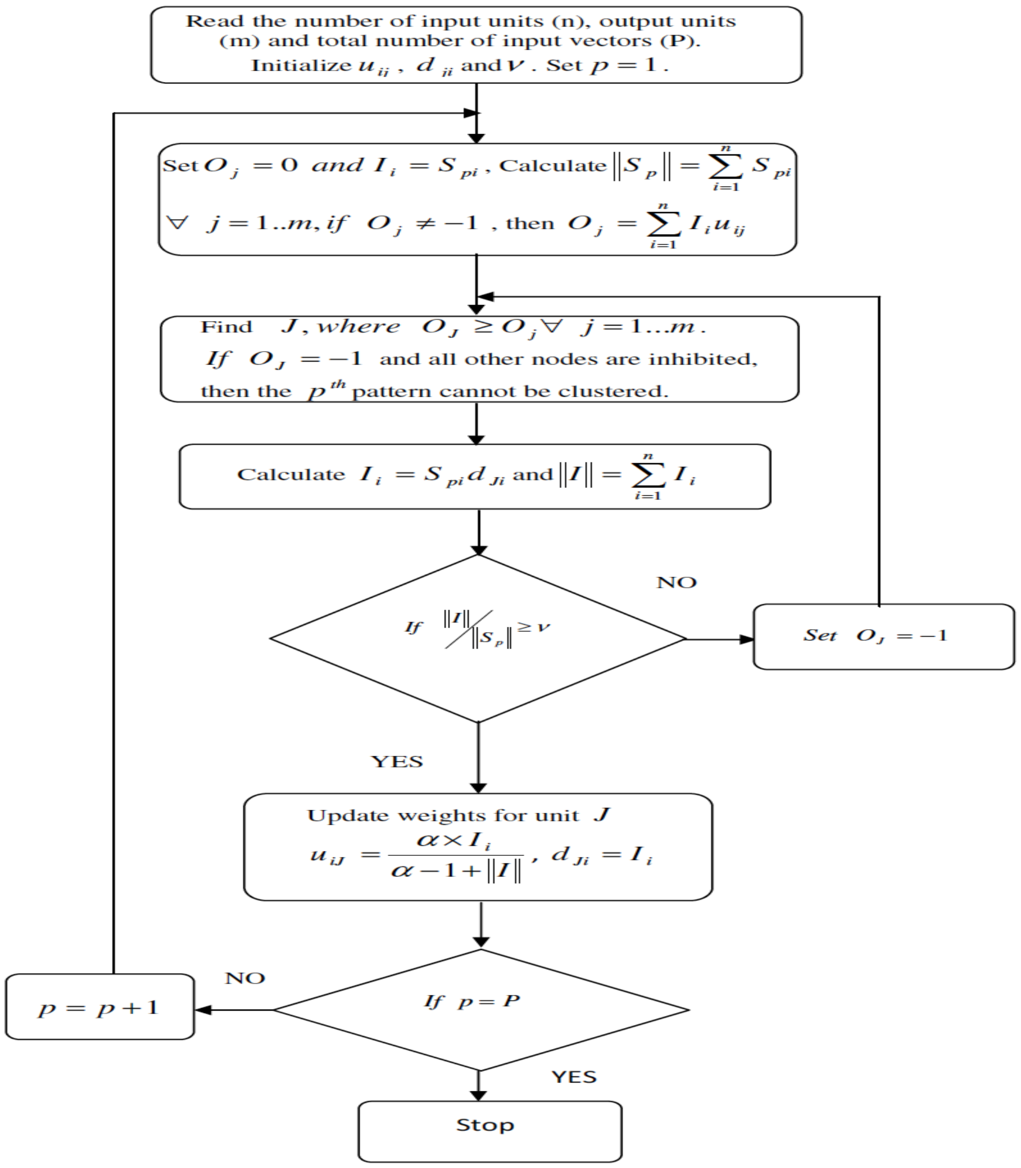
*The information about the winner is sent from the output layer $L_2$ to the interface layer $L_{1S}$ through the top-down weights $d_{ji}$ .*

*The interface unit's activations is 1; if a unit receives a non-zero signal simultaneously from the input layer L1S and the output layer L2.*

*Then, the norm of the vector I of the comparison region gives the number of components for which the top-down weight vector $d_{Ji}$ for the winning unit J and the input vector Sp  are same as 1.*

*The value of I gives a evaluate the degree of the match. The learning will occur only if the match is acceptable to the vigilance parameter.*
*The updation of the weights is carried out if*

$$\|I\| \big/ \|S_p\| \geq \nu$$


Read the number of input units (n), output units (m) and total number of input vectors (P). Initialize $u_{ij}$ , $d_{ji}$ and $\nu$ . Set $p=1$ .
Set $O_j = 0$ and $I_i = S_{pi}$ , Calculate $\|S_p\| = \sum_{i=1}^{n} S_{pi}$
$\forall$ $j=1..m$, if $O_j \neq -1$ , then $O_j = \sum_{i=1}^{n} I_i u_{ij}$
Find $J$, where $O_J \geq O_j \forall$ $j=1...m$.
If $O_J = -1$ and all other nodes are inhibited, then the $p^{th}$ pattern cannot be clustered.
Calculate $I_i = S_{pi} d_{Ji}$ and $\|I\| = \sum_{i=1}^{n} I_i$
If $\|I\| / \|S_p\| \geq \nu$
NO
Set $O_J = -1$
YES
Update weights for unit $J$
$u_{iJ} = \frac{\alpha \times I_i}{\alpha - 1 + \|I\|}$ , $d_{Ji} = I_i$
NO
$p = p+1$
If $p = P$
YES
Stop

# BOLTZMAN MACHINE NEURAL NETWORKS (BMNN)

***1983, Geoffrey Hinton and Terry Sejnowski***

***stochastic recurrent neural network .***

***BMNN is a network of symmetrically connected, neuron-like units that make stochastic decisions about whether to be on or off. Boltzmann machines have a simple learning algorithm that allows them to discover interesting features that represent complex regularities in the training data.***

**Differences & Similarities with Hopfield NN**

| **Hopfield** | **BMNN** |
|---|---|
| Local updation and Hebbian learning | powerful stochastic learning scheme |
| Deterministic updation of activations | Stochastic updation of activations |
| Hidden layer is absent | Hidden layers is present |
| Symmetric Weights<br>Random asynchronous activation updation.<br>Units have no self-feedback | |

# BOLTZMAN MACHINE NEURAL NETWORKS (BMNN)

***BMNN for Learning input and output patterns***

***Every individual unit in BMNN will have any one of the two states namely, ON or OFF (1 or 0 in binary representation) or (1 or -1 in bipolar representation). This state of the unit is a function of probabilistic function of the states of its neighbouring units and the weights on its links to them.***

***ON or OFF can be considered as the acceptance or rejection of a hypothesis of the problem.***

***The energy of any global configuration of a BMNN***

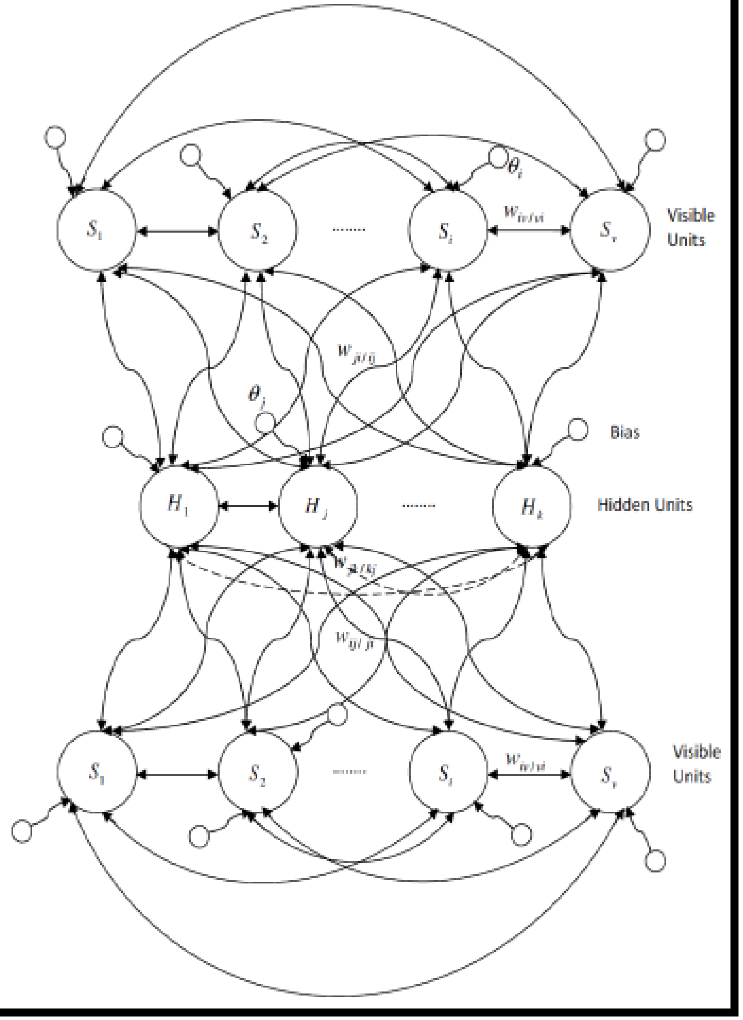
$$E = -\sum_{r<s} w_{rs} S_r S_s + \sum_r \theta_r S_r$$

***$w_{rs}$ → Strength of connection between units r and s ;***
***$S_r$ → State of the unit (0 or 1) ; $\theta_r$ → Threshold of $r_{th}$ unit.***

***Rejection or acceptance of a hypothesis for the is determined by an Energy gap.***

$$\Delta E_r = \sum_r w_{rs} S_r - \theta_r$$

***An Unit can be ON if its total net input obtained by summing up of the signals from the neighbouring units of the system exceeds its threshold.***

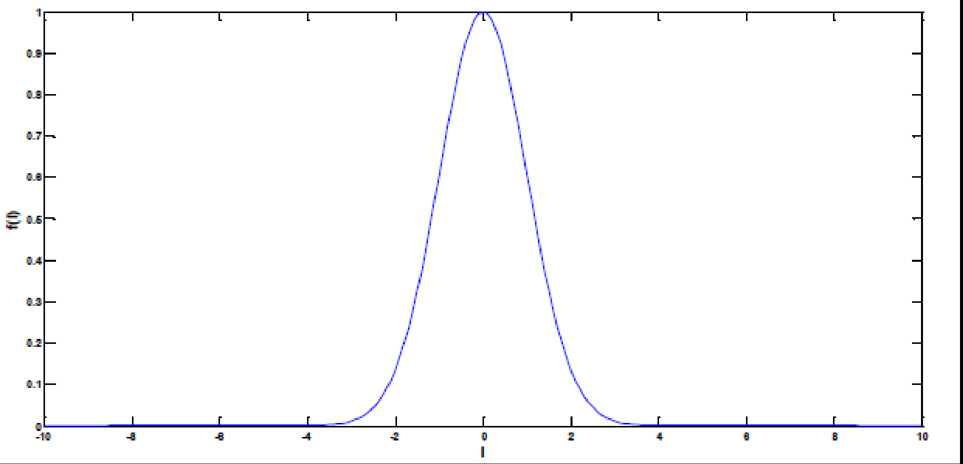


***BMNN Architecture***

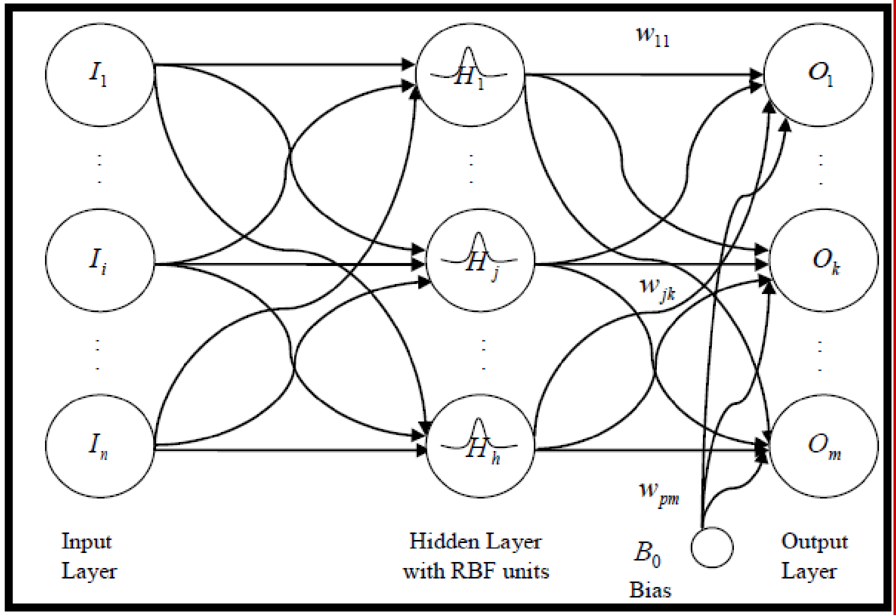
## RADIAL BASIS FUNCTION NEURAL NETWORKS (RBF)

***Moody and Darken, 1989;***
***Hush and Horne, 1993;***
***Wassermann, 1993***

***The hidden layer units incorporates the specialised activation function called radial basis functions. These functions produce localized, bounded, and radially symmetric activations that decreases the distance from the function's centres.***



***Gaussian Radial Basis function***



***RBFNN Architecture***

## SUPPORT VECTOR MACHINES (SVM)

***SVM is a learning algorithm typically used for classification problems.***

- ***Text categorization***
- ***Character recognition***
- ***Image classification***

***Derived from statistical learning theory by Vapnik and Chervonenkis***

***Classification tasks based on drawing separating lines to distinguish between objects of different class memberships are known as 'hyper plane classifiers'.***

***SVM tries to minimize the upper bound of the generalization error and maximizes the margin between a separating hyper plane and the training data.***

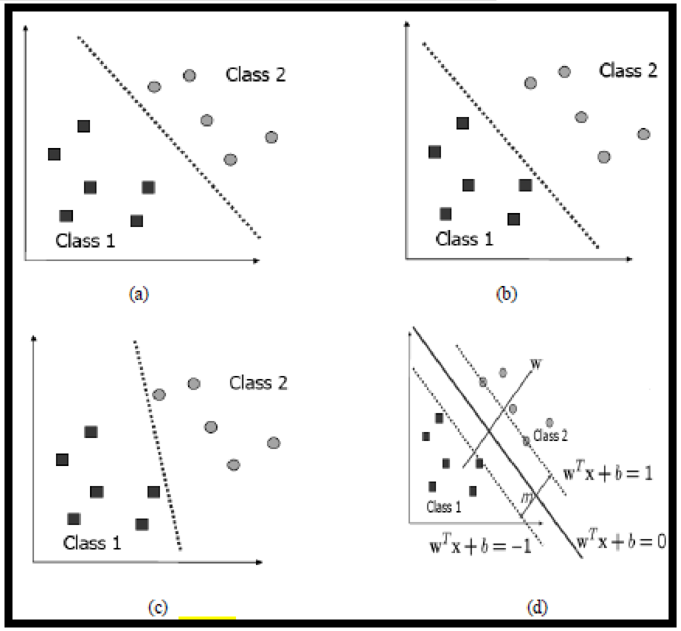
***The goal of the SVM is to optimize "generalization", the ability to correctly classify unseen data.***

***It determines a linear decision boundary in the feature space by constructing the "optimal separating hyperplane" distinguishing the classes***

# SUPPORT VECTOR MACHINES (SVM)

***Illustration : Linearly separable two class problem***

- ***The two classes can be separated by many decision boundaries as shown in Fig (a, b, c).***
- ***Ambiguity to choose the one that is the best.***
- ***The decision boundary should be as far way from the data of both classes as possible. Therefore, the margin 'm' as shown in Fig (d) between the two classes has to be maximized by an optimization problem***
- ***$X= \{x_1,x_2,\ldots,x_n\}$ → Points to be clasified;***
- ***yi ∈ {1,1}---→ Class lable of $x_i$***
- ***The decision boundary should classify all points correctly as*** $y_i\left(w^T x_i + b\right) \ge 1, \forall i$ .
- ***w and b are the weights and biases or the coefficients of a decision boundary***



***Decision Boundaries between Two Classes***

Minimize $\frac{1}{2}\|w\|^2$

Subject to $y_i\left(w^T x_i + b\right) \ge 1, \forall i$

Maximize $w(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j-1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j$

Subject to $y_i\left(w^T x_i + b\right) \ge 1, \forall i$

$w = \sum_{i=1}^{n} \alpha_i y_i x_i$

$w = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} x_{t_j}$

$w^T z + b = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} \left(x_{t_j}^T z\right) + b$

# SUPPORT VECTOR MACHINES (SVM)

## *Illustration : Linearly separable two class problem*

*1. finding the solution to the constrained optimization problem as in equation is the training part of the SVM.*

$$\textit{Minimize } \frac{1}{2}\|w\|^2$$
$$\text{Subject to } y_i\left(w^T x_i + b\right) \geq 1, \forall i$$

*2. The minimization problem can be transformed into it dual as*

$$\textit{Maximize } w(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j-1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j$$
$$\text{Subject to } y_i\left(w^T x_i + b\right) \geq 1, \forall i$$

*3. This is quadratic programming (QP) problem, where the optimal value of $\alpha_i$ can be recovered and w can be recovered by*

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

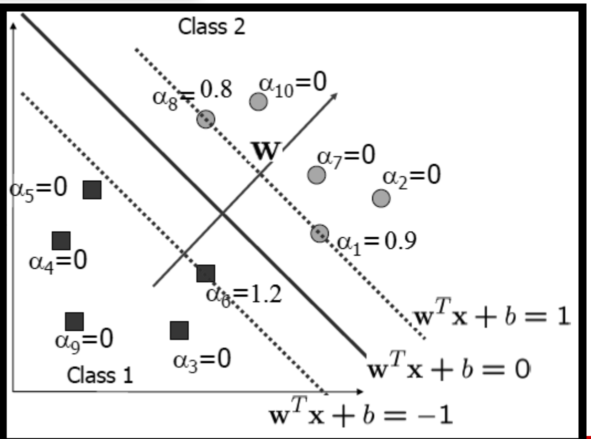
*4. Many of the $\alpha_i$ zeros. The weights w will be a linear combination of a small number of data. $x_i$ with non-zero $\alpha_i$ can be called support vectors (SV). $t_j$ → indices of the 's' SVs then w is*

$$w = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} x_{t_j}$$

*5. Once the training is over, a new set of data can be tested by computing the equation z*

$$w^T z + b = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} \left(x_{t_j}^T z\right) + b$$

***Decision Boundaries with $\alpha_i$ coefficients***



# SUPPORT VECTOR MACHINES (SVM)

## *Linearly inseparable two class problem*

*If the set of points is inseparable by a straight line, then an error **$\varepsilon_i$** can be incorporated during classification which belongs to a field of soft margin hyperplane decision boundaries.*

*1. The equation for boundaries incorporating **$\varepsilon_i$** is*

$$\begin{aligned} w^T z + b &\geq 1 - \varepsilon_i & y_i &= 1 \\ w^T z + b &\leq 1 - \varepsilon_i & y_i &= -1 \\ \varepsilon_i &\geq 0 & \forall i \end{aligned}$$

*2. The optimization problem can be formulated*

$$\text{Minimize } \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\varepsilon_i$$

$$\text{Subject to } y_i(w^T x_i + b) \geq 1 - \varepsilon_i, \quad \varepsilon_i \geq 0$$

*3. The minimization problem can be transformed into it dual as*

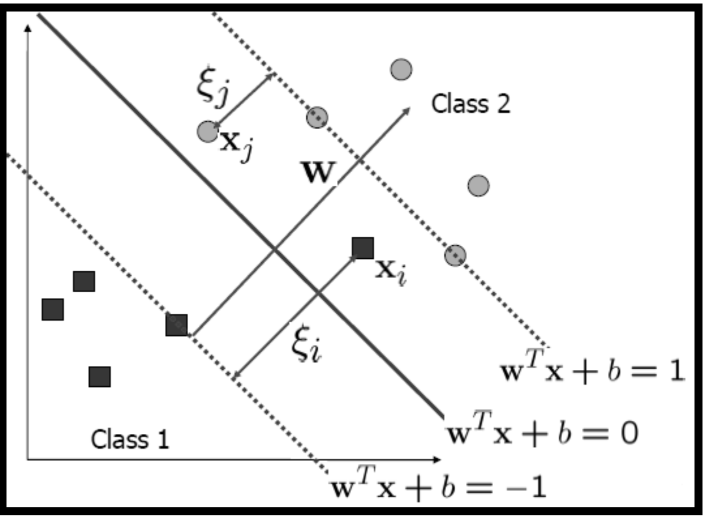
$$\text{Maximize } w(\alpha) = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1,j-1}^{n}\alpha_i\alpha_j y_i y_j x_i^T x_j$$

$$\text{Subject to } C \geq \alpha_i \geq 0$$

$$\sum_{i=1}^{n}\alpha_i y_i = 0$$

*4. w can be recovered by*

$$w = \sum_{j=1}^{s}\alpha_{t_j} y_{t_j} x_{t_j}$$


$\xi_j$
$\mathbf{x}_j$
Class 2
$\mathbf{w}$
$\mathbf{x}_i$
$\xi_i$
$\mathbf{w}^T\mathbf{x} + b = 1$
$\mathbf{w}^T\mathbf{x} + b = 0$
$\mathbf{w}^T\mathbf{x} + b = -1$
Class 1

***Decision Boundaries with $\varepsilon_i$ for linearly inseparable classes***

***Alternate method:***
***Input space → feature space.***
***High Computation burden.***
***Kernel mapping.***

# ELECTRICAL LOAD FORECASTING USING MATLAB NEURAL NETWORK TOOLBOX

***Electrical load forecasting is the process by which the electrical load at a future time is predicted based on past values of load as well as weather, economic and demographic factors.***

***Types of Load forecasting:***
*Short-term → 1 hour to several days*
*Medium term → 1 week to several months*
*Long term → 1 year to several years*

***Factors influencing electrical load:***
*Time Weather End user Connected loads and demographic and economic conditions*

***Methods to forecast electrical load***
*Similar day approach, Regression models, Time series, Expert systems, Fuzzy logic and neural networks.*

***Types NN for Load forecasting***
*Hopfield, Back propagation, Boltzmann machine.*

***Most Commonly used:*** *back propagation neural network with continuous valued functions and supervised learning*

# ELECTRICAL LOAD FORECASTING USING MATLAB NEURAL NETWORK TOOLBOX

***Data Sources:***
*Electrical load data → PJM (Electricity Power Market)*
*http://www.pjm.com/markets-and-operations/energy/real-time/loadhryr.aspx*

*Weather data → NOAA*
*http://www7.ncdc.noaa.gov/CDO/georegion*

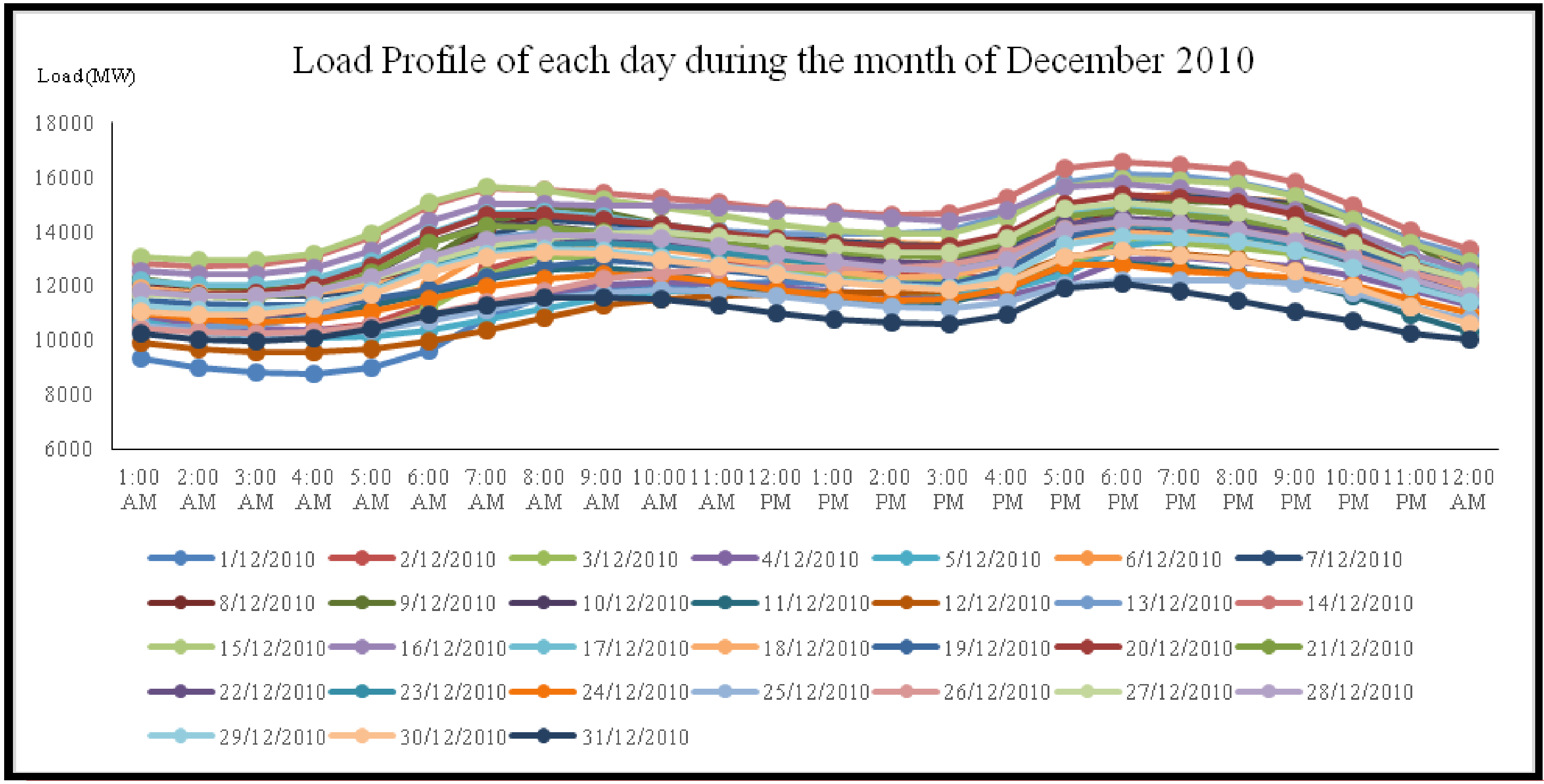
*The training data is split into seven groups each containing data for a particular weekday.*

*7 different neural networks are trained, 1 for each weekday, with the 7 different training data sets.*

***Training period:***
*December 1 2010 to*
*December 20 2010*

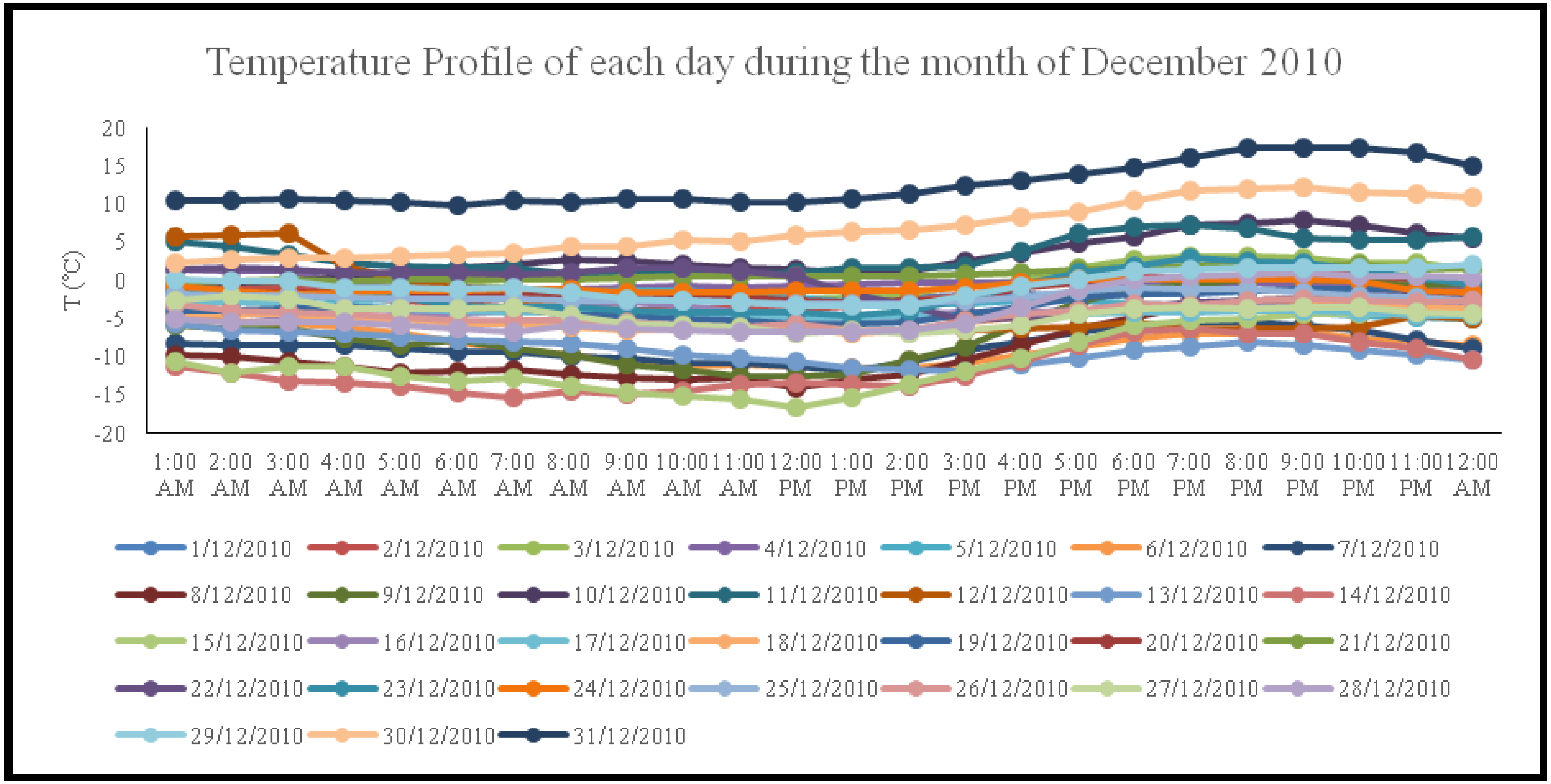
***Testing period:***
*December 25 2010 to*
*December 31 2010*

# ELECTRICAL LOAD FORECASTING USING MATLAB NEURAL NETWORK TOOLBOX



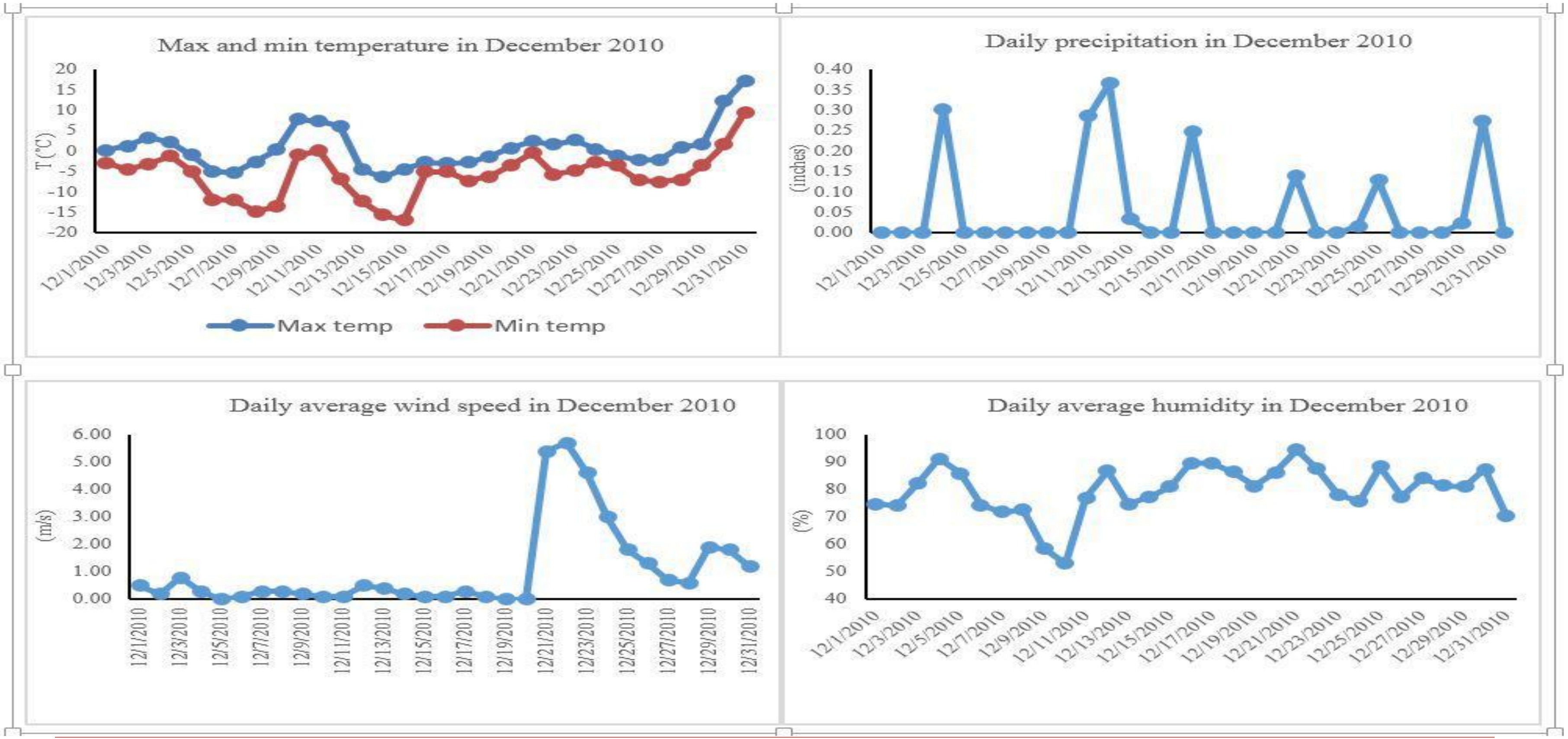
***Historical load data***

# ELECTRICAL LOAD FORECASTING USING MATLAB NEURAL NETWORK TOOLBOX



***Historical temperature data***

# ELECTRICAL LOAD FORECASTING USING MATLAB NEURAL NETWORK TOOLBOX



***Historical weather data***

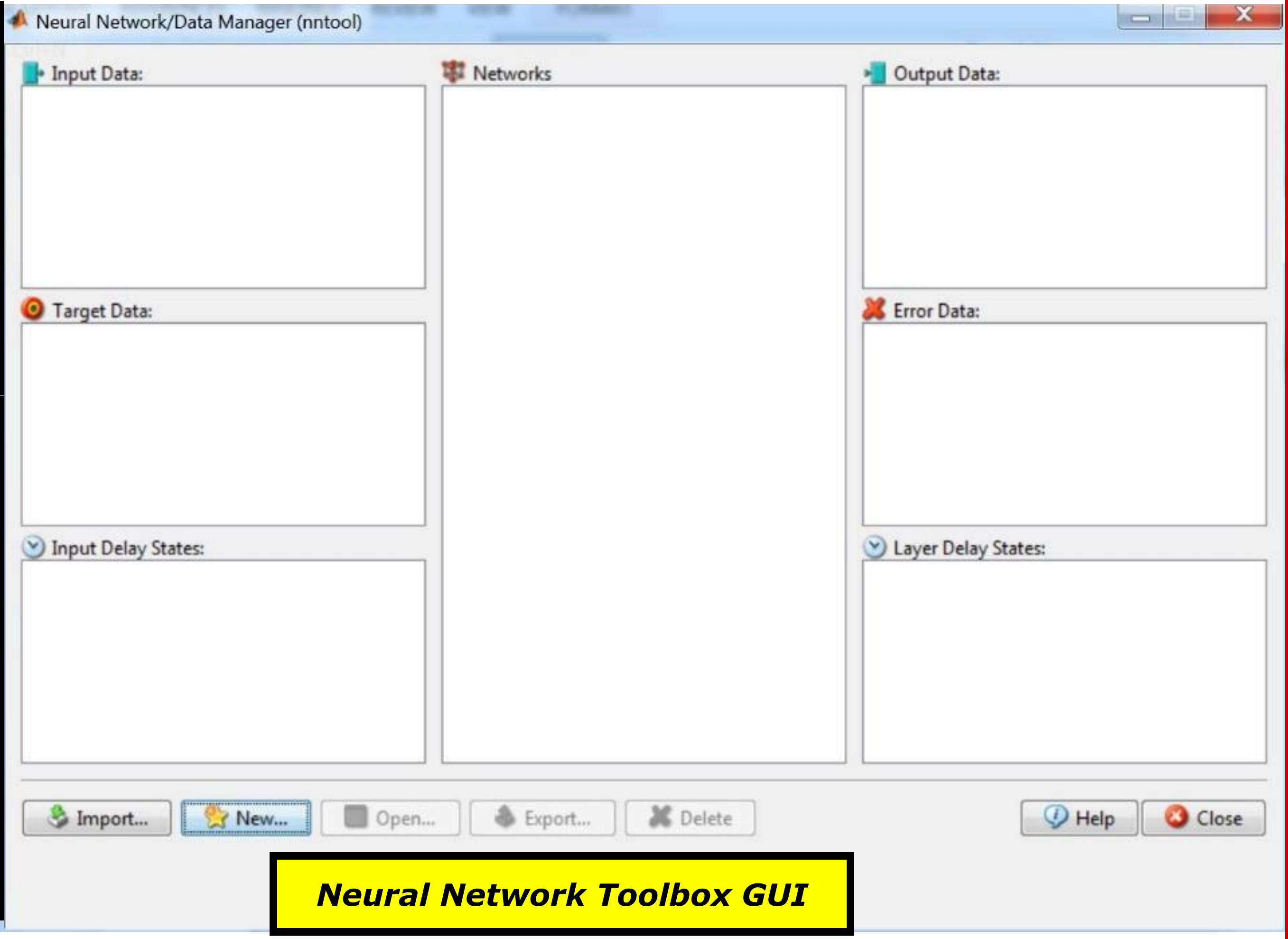
# ELECTRICAL LOAD FORECASTING USING MATLAB NEURAL NETWORK TOOLBOX

***MATLAB NN toolbox***

***Inbuilt Libraries to implement various types of NN such as*** *perceptron, feed forward back propagation, Hopfield, radial basis and self-organizing map.*

***<u>GUI:</u>*** Graphical User Interface helps user to specify following parameters Input and target vectors, type of network, the transfer function of each layer, the learning rate etc,.

In command window type ***nntool***



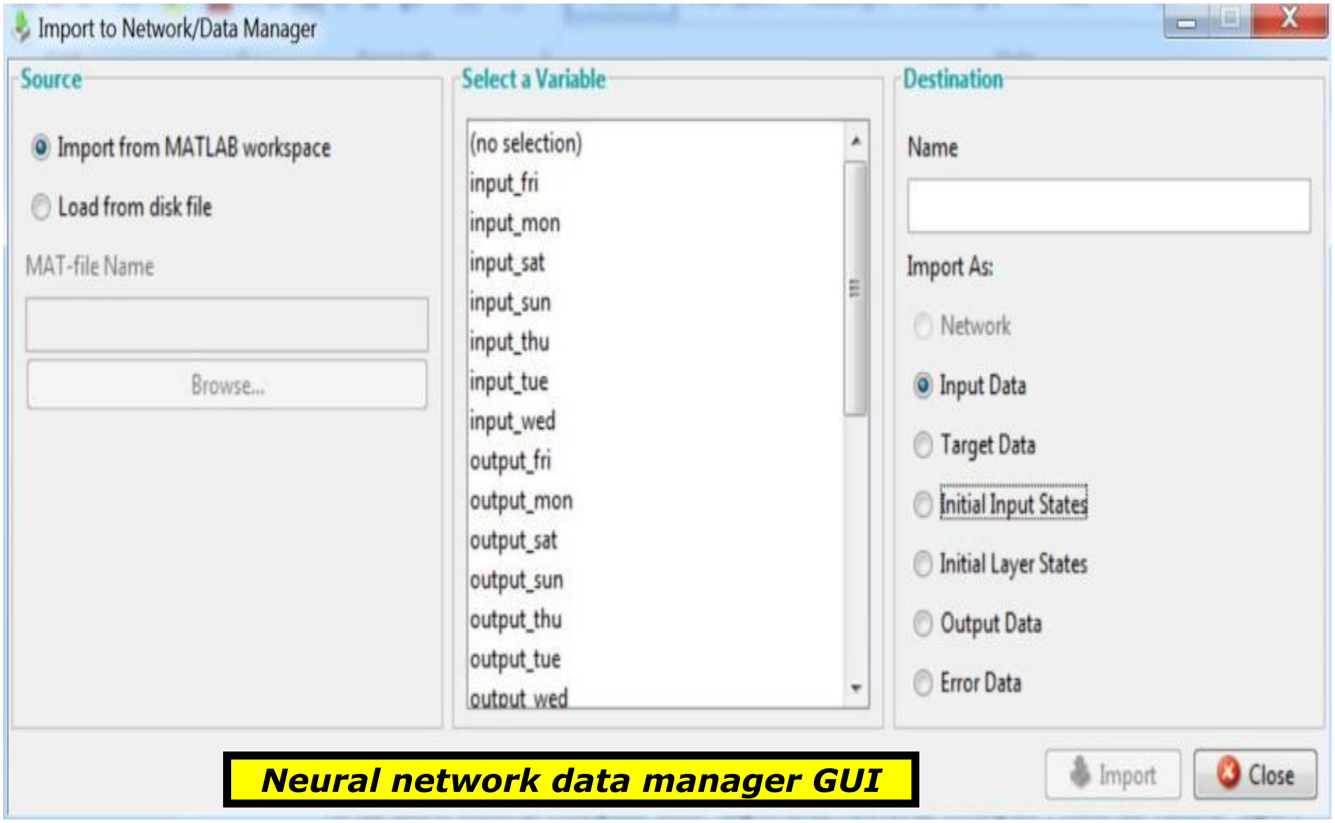
***Neural Network Toolbox GUI***

# ELECTRICAL LOAD FORECASTING USING MATLAB NEURAL NETWORK TOOLBOX

***MATLAB NN toolbox***

***Select data(from workspace or file) and import.***

***In this example, the data is stored in workspace. So, select 'import from MATLAB workspace', select the variable 'input_sun', and in destination, click 'Input Data'. Similarly import Target data***



Import to Network/Data Manager

Source

[✓ Import from MATLAB workspace]

Load from disk file

MAT-file Name

Select a Variable

(no selection)
input_fri
input_mon
input_sat
input_sun
input_thu
input_tue
input_wed
output_fri
output_mon
output_sat
output_sun
output_thu
output_tue
output_wed

Destination

Name

Import As:

Network
[✓ Input Data]
Target Data
Initial Input States
Initial Layer States
Output Data
Error Data

***Neural network data manager GUI***

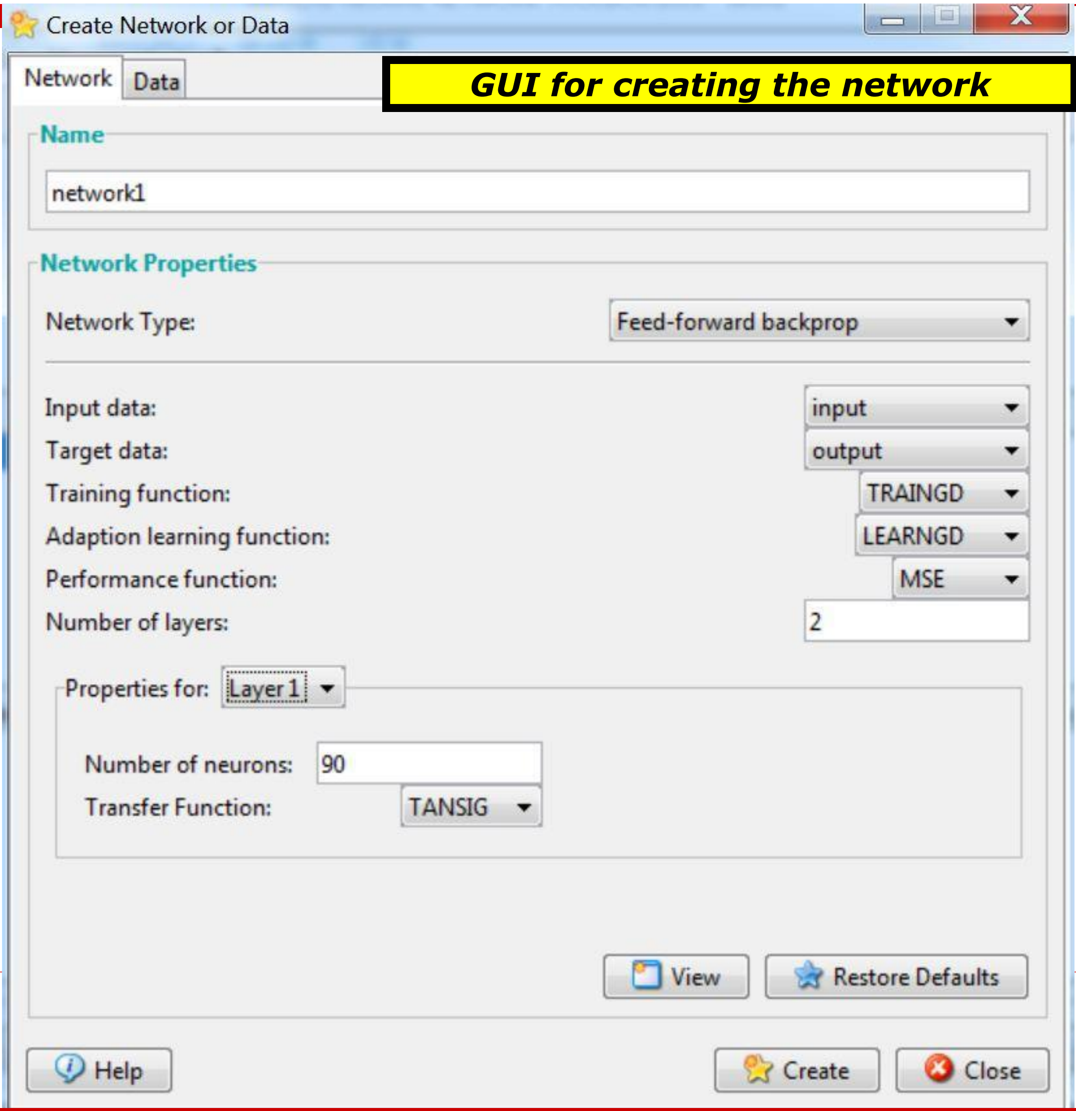
# ELECTRICAL LOAD FORECASTING USING MATLAB NN TOOLBOX

***MATLAB NN toolbox***

***create the neural network architecture as well as specify the training input and output data***

*In this example, Select*
***Network type***
*'Feed forward backprop' for feed forward back propagation network.*
***Training function***
*'TRAINGD' for gradient descent algorithm*
***Performance function***
*MSE*
***No of Layers:****2*
***Number of neurons:*** *90*
*&*
***Press Create***

***GUI for creating the network***


Create Network or Data

Network | Data

Name
network1

Network Properties
Network Type: Feed-forward backprop

Input data: input
Target data: output
Training function: TRAINGD
Adaption learning function: LEARNGD
Performance function: MSE
Number of layers: 2

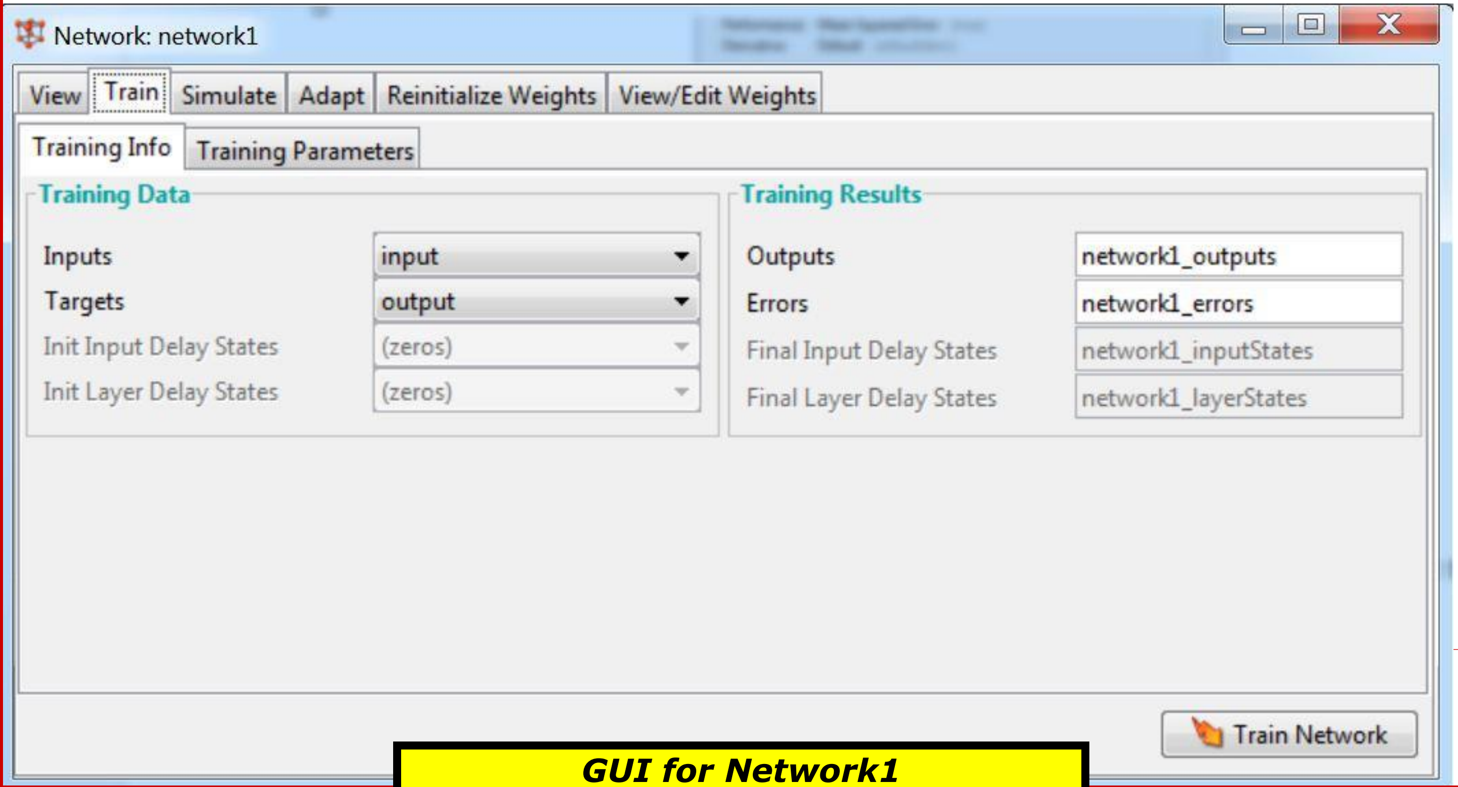
Properties for: Layer 1
Number of neurons: 90
Transfer Function: TANSIG

# ELECTRICAL LOAD FORECASTING USING MATLAB NN TOOLBOX

***MATLAB NN toolbox***

*In nntool GUI (Fig. 3.47), click the created network and then click Open*

*In **Training Info tab** select the Inputs as 'input_sun' and Targets as 'output_sun*
*In **training parameters** tab specify the number of epochs, learning rate (lr) and minimum gradient*


Network: network1

View | Train | Simulate | Adapt | Reinitialize Weights | View/Edit Weights

Training Info | Training Parameters

Training Data

| | |
|---|---|
| Inputs | input |
| Targets | output |
| Init Input Delay States | (zeros) |
| Init Layer Delay States | (zeros) |

Training Results

| | |
|---|---|
| Outputs | network1_outputs |
| Errors | network1_errors |
| Final Input Delay States | network1_inputStates |
| Final Layer Delay States | network1_layerStates |

***GUI for Network1***

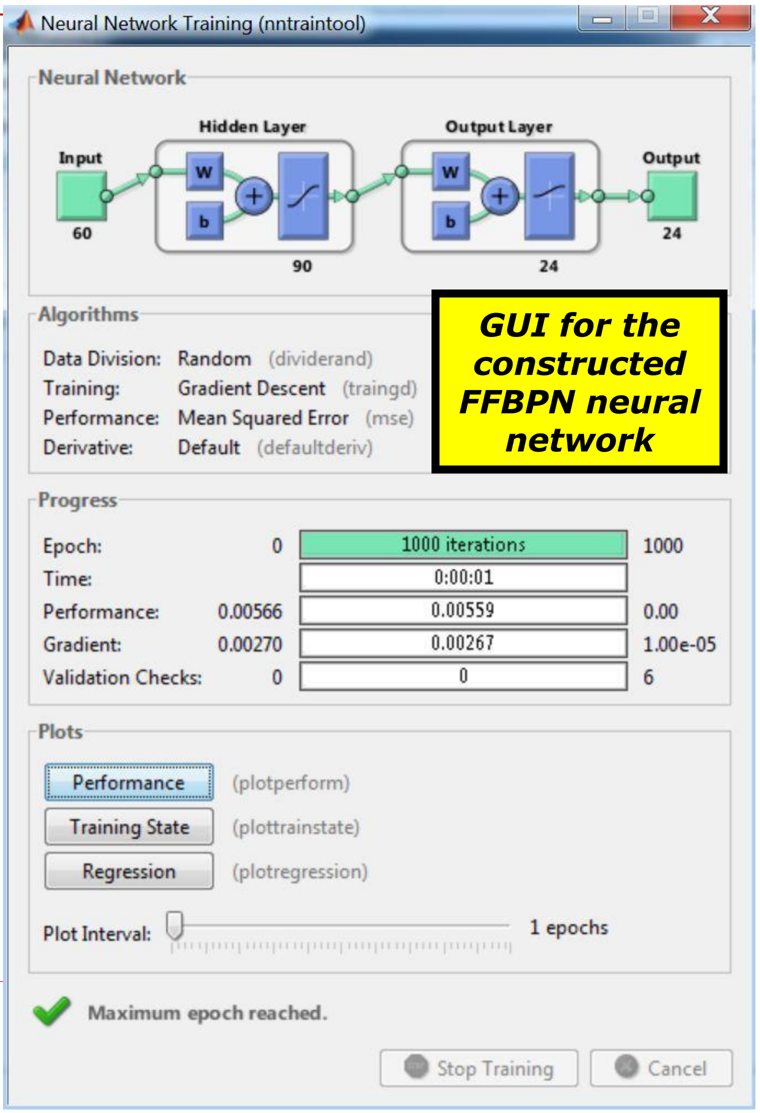
*Click the 'Train Network' button to begin training of the neural network*

# ELECTRICAL LOAD FORECASTING USING MATLAB NN TOOLBOX

## *MATLAB NN toolbox*

***Current status of the training is shown in the NN Training GUI***

***The neural network is said to be trained when the weight values are optimized such that the sum squared error of the training data is below a certain threshold or the number of validation checks have exceeded a set point.***



Algorithms

| | | |
|---|---|---|
| Data Division: | Random | (dividerand) |
| Training: | Gradient Descent | (traingd) |
| Performance: | Mean Squared Error | (mse) |
| Derivative: | Default | (defaultderiv) |

***GUI for the constructed FFBPN neural network***

Progress

| | | | |
|---|---|---|---|
| Epoch: | 0 | 1000 iterations | 1000 |
| Time: | | 0:00:01 | |
| Performance: | 0.00566 | 0.00559 | 0.00 |
| Gradient: | 0.00270 | 0.00267 | 1.00e-05 |
| Validation Checks: | 0 | 0 | 6 |

Plots

| | |
|---|---|
| Performance | (plotperform) |
| Training State | (plottrainstate) |
| Regression | (plotregression) |

Plot Interval: 1 epochs

Maximum epoch reached.

# ELECTRICAL LOAD FORECASTING USING MATLAB NN TOOLBOX

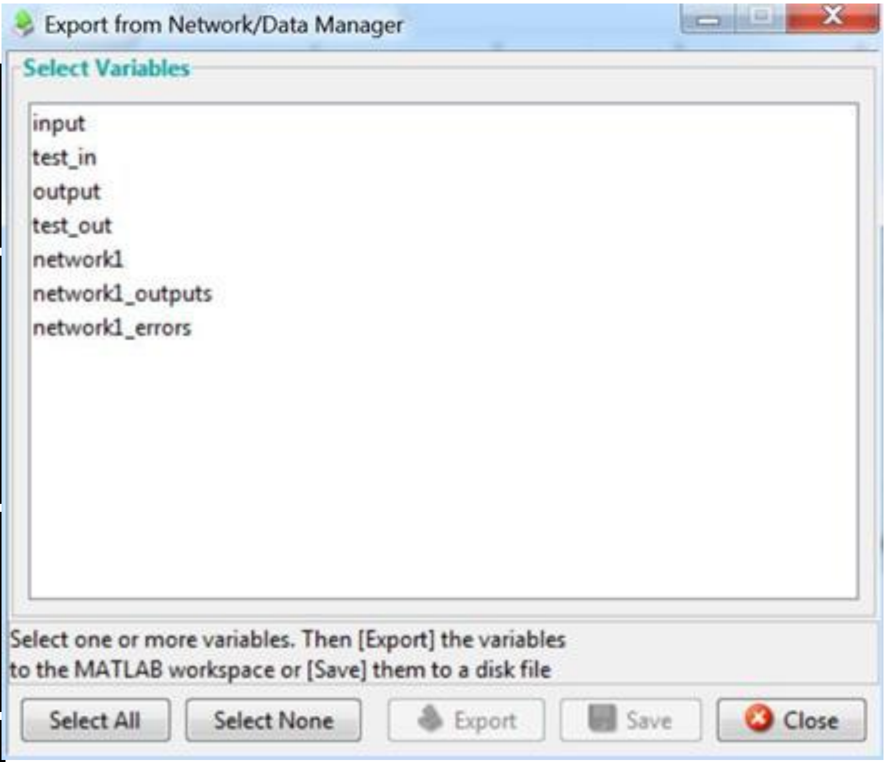
## *MATLAB NN toolbox*

***The suitability of the neural network for load forecasting can be known by testing it against data not in the training set***

***To test the neural network, go to the network properties in GUI for Network1 and simulate after selecting 'test_in_sun' as Inputs and 'test_out_sun' as Targets.***

***The output of the simulation will be stored in the nntool GUI under the Output Data section.***

***Click Export button in nntool GUI to display the Export from Network/Data Manager window. Select the simulated output data variable and press Export. The data will now be saved to the workspace.***



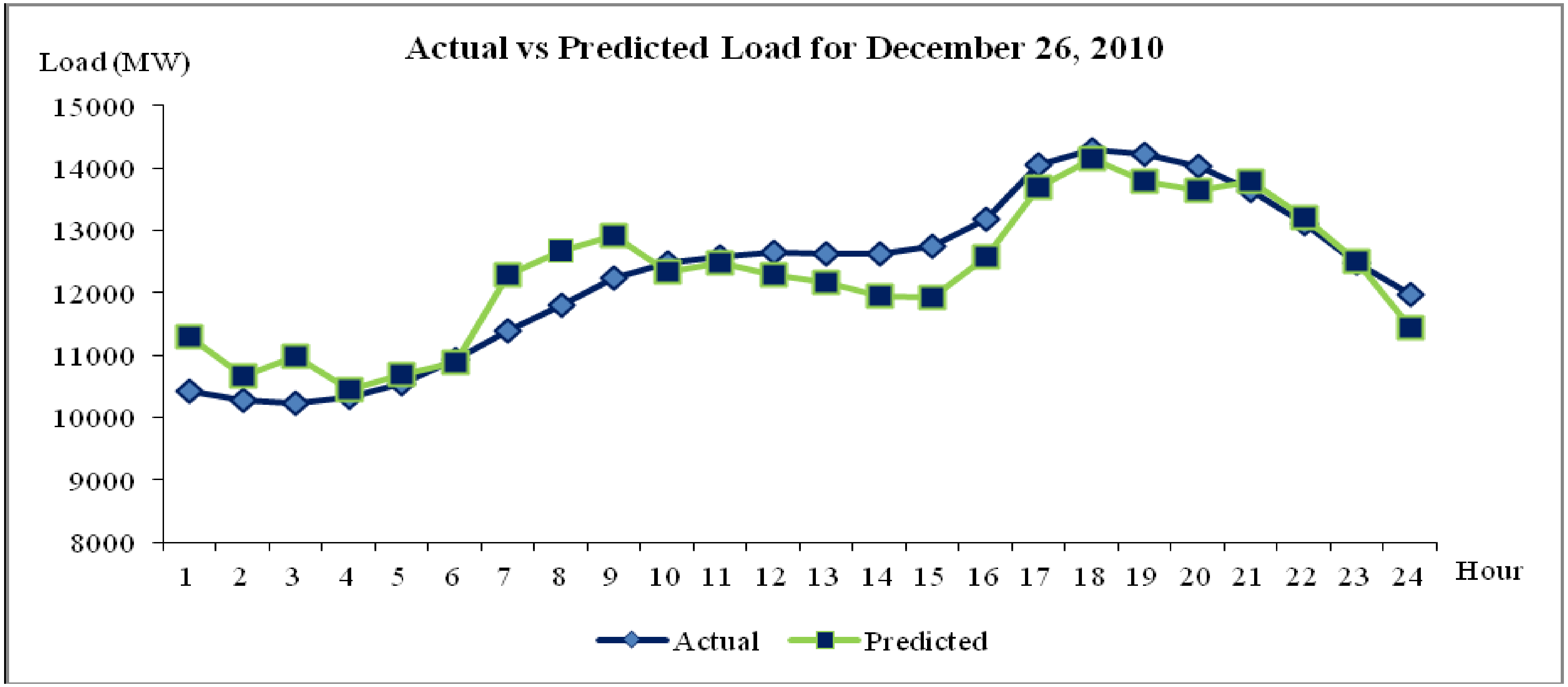
input
test_in
output
test_out
network1
network1_outputs
network1_errors

Select one or more variables. Then [Export] the variables to the MATLAB workspace or [Save] them to a disk file

# ELECTRICAL LOAD FORECASTING USING MATLAB NN TOOLBOX

***MATLAB NN toolbox***

***The graph in shows the variation actual Vs expected load for 24 hours on 26 December 2010, which is a Sunday.***



## ARTIFICIAL NEURAL NETWORKS II