PERCEPTRON RULE

Step 1: Initialize all weights and bias are set to zero. Set learning rate =0to 1.
Step 2: while stopping condition is false do step 3-7
Step 3: For each training vector and target output pair (s, t) perform steps 4-6
Step 4: Set activations for input units with input vector
Step 5: Compute the output unit response

$$Y_{in} = b + \sum_i x_i * w_i$$

$$Y = f(Y_{in}) = \begin{cases} 1\_if\_Y_{in} > \theta \\ 0\_if\_-\theta < Y_{in} < \theta \\ -1\_if\_Y_{in} < -\theta \end{cases}$$

Step 6: The weights and bias are updated if the target is not equal to the output response. If t≠y and value of x_i is not zero

$$w_i(new) = w_i(old) + \alpha * t * x_i$$

Adjust the bias

$$b(new) = b(old) + \alpha * t$$

Step 7: Test for stopping condition.

AND MODEL

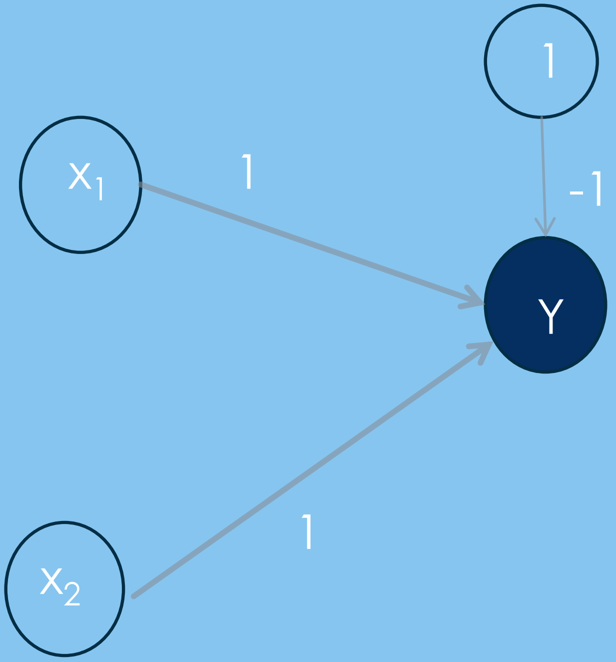
X_1	X_2	b	Y_{in}	y	t	ΔW_1	ΔW_2	Δb	W_1	W_2	B
									0	0	0
1	1	1	0	0	1	1	1	1	1	1	1
1	-1	1	1	1	-1	1	-1	-1	2	0	0
-1	1	1	2	1	-1	-1	1	-1	1	1	-1
-1	-1	1	-3	-1	-1	0	0	0	1	1	-1

$$\Delta w_1 = \alpha * x_1 * t \qquad \Delta b = \alpha * y$$

$$\Delta w_2 = \alpha * x_2 * t$$

$$w_1(new) = w_1(old) + \Delta w_1$$

$$w_2(new) = w_2(old) + \Delta w_2$$



AND MODEL

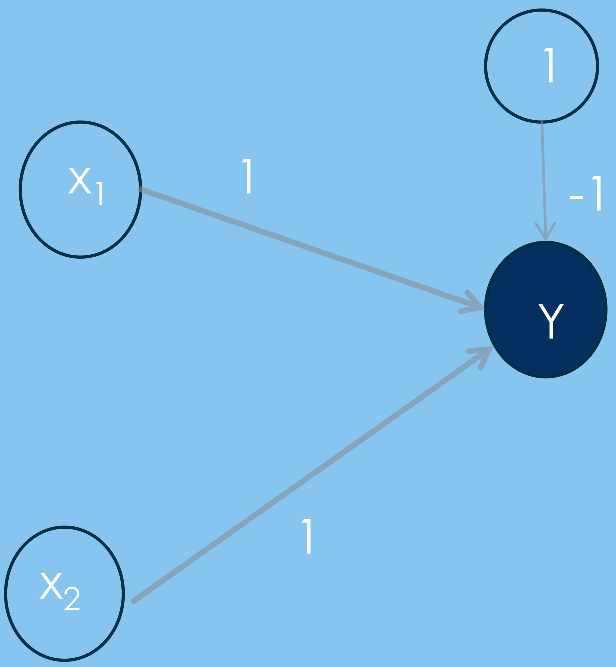
X_1	X_2	b	Y_{in}	y	t	ΔW_1	ΔW_2	Δb	W_1	W_2	B
									1	1	-1
1	1	1	0	0	1	1	1	1	1	1	-1
1	-1	1	1	-1	-1	1	-1	-1	1	1	-1
-1	1	1	2	-1	-1	-1	1	-1	1	1	-1
-1	-1	1	-3	-1	-1	0	0	0	1	1	-1

$$\Delta w_1 = \alpha * x_1 * t \qquad \Delta b = \alpha * y$$

$$\Delta w_2 = \alpha * x_2 * t$$

$$w_1(new) = w_1(old) + \Delta w_1$$

$$w_2(new) = w_2(old) + \Delta w_2$$



OR MODEL

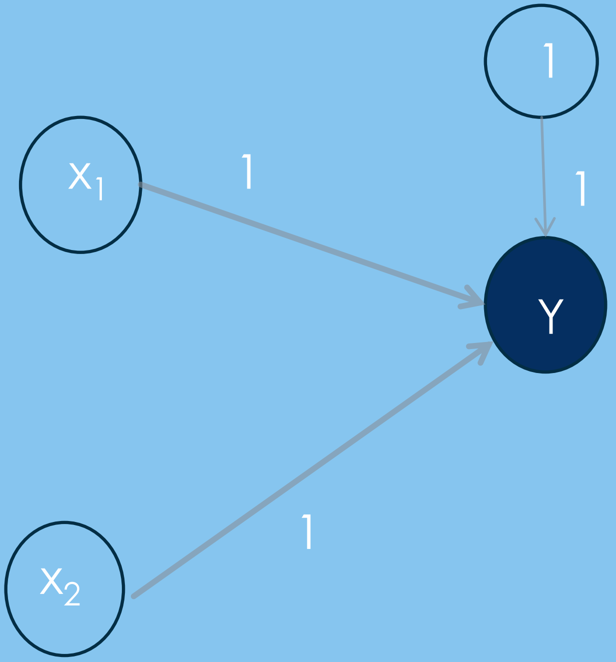
X_1	X_2	b	Y_{in}	y	t	ΔW_1	ΔW_2	Δb	W_1	W_2	B
									0	0	0
1	1	1	0	0	1	1	1	1	1	1	1
1	-1	1	1	1	1	0	0	0	1	1	1
-1	1	1	1	1	1	0	0	0	1	1	1
-1	-1	1	-1	-1	-1	0	0	0	1	1	1

$$\Delta w_1 = \alpha * x_1 * t \qquad \Delta b = \alpha * y$$

$$\Delta w_2 = \alpha * x_2 * t$$

$$w_1(new) = w_1(old) + \Delta w_1$$

$$w_2(new) = w_2(old) + \Delta w_2$$



AND MODEL

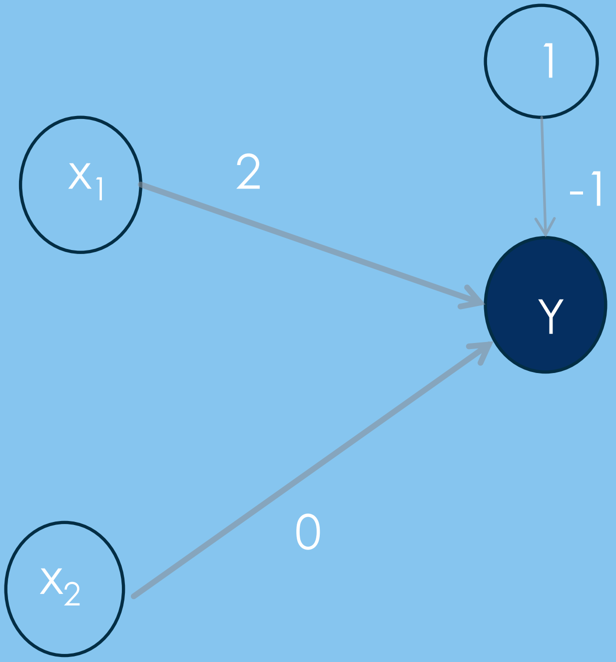
X_1	X_2	b	Y_{in}	y	t	ΔW_1	ΔW_2	Δb	W_1	W_2	B
									0	0	0
1	1	1	0	0	1	1	1	1	1	1	1
1	0	1	2	1	-1	1	0	-1	2	1	0
0	1	1	1	1	-1	0	-1	-1	2	0	-1
0	0	1	-1	-1	-1	0	0	0	2	0	-1

$$\Delta w_1 = \alpha * x_1 * t \qquad \Delta b = \alpha * y$$

$$\Delta w_2 = \alpha * x_2 * t$$

$$w_1(new) = w_1(old) + \Delta w_1$$

$$w_2(new) = w_2(old) + \Delta w_2$$



Example 3:

X_1	X_2	X_3	X_4	B	t
1	1	1	1	1	1
1	1	1	-1	1	-1
-1	1	-1	-1	1	1
1	-1	-1	1	1	-1

X_1	X_2	X_3	X_4	B	Y_{in}	y	t	ΔW_1	ΔW_2	ΔW_3	ΔW_4	Δb	W_1	W_2	W_3	W_4	b
													0	0	0	0	0
1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1
1	1	1	-1	1	3	1	-1	-1	-1	-1	1	-1	0	0	0	2	0
-1	1	-1	-1	1	-2	-1	1	-1	1	-1	-1	1	-1	1	-1	1	1
1	-1	-1	1	1	1	1	-1	-1	1	1	-1	-1	-2	2	0	0	0

Adaline

Step 1: Initialize all weights and bias any small value other than zero. Set learning rate =0to 1.
Step 2: while stopping condition is false do step 3-7
Step 3: For each bipolar training vector and target output pair (s, t) perform steps 4-6
Step 4: Set activations for input units with input vector
Step 5: Compute the output unit response

$$Y_{in} = b + \sum_i x_i * w_i$$

$$Y = f(Y_{in}) = \begin{cases} 1\_if\_Y_{in} >= 0 \\ -1\_if\_Y_{in} < -0 \end{cases}$$

Step 6: The weights and bias are updated

$$w_i(new) = w_i(old) + \alpha * (t - y_{in}) * x_i$$

Adjust the bias

$$b(new) = b(old) + \alpha * (t - y_{in})$$

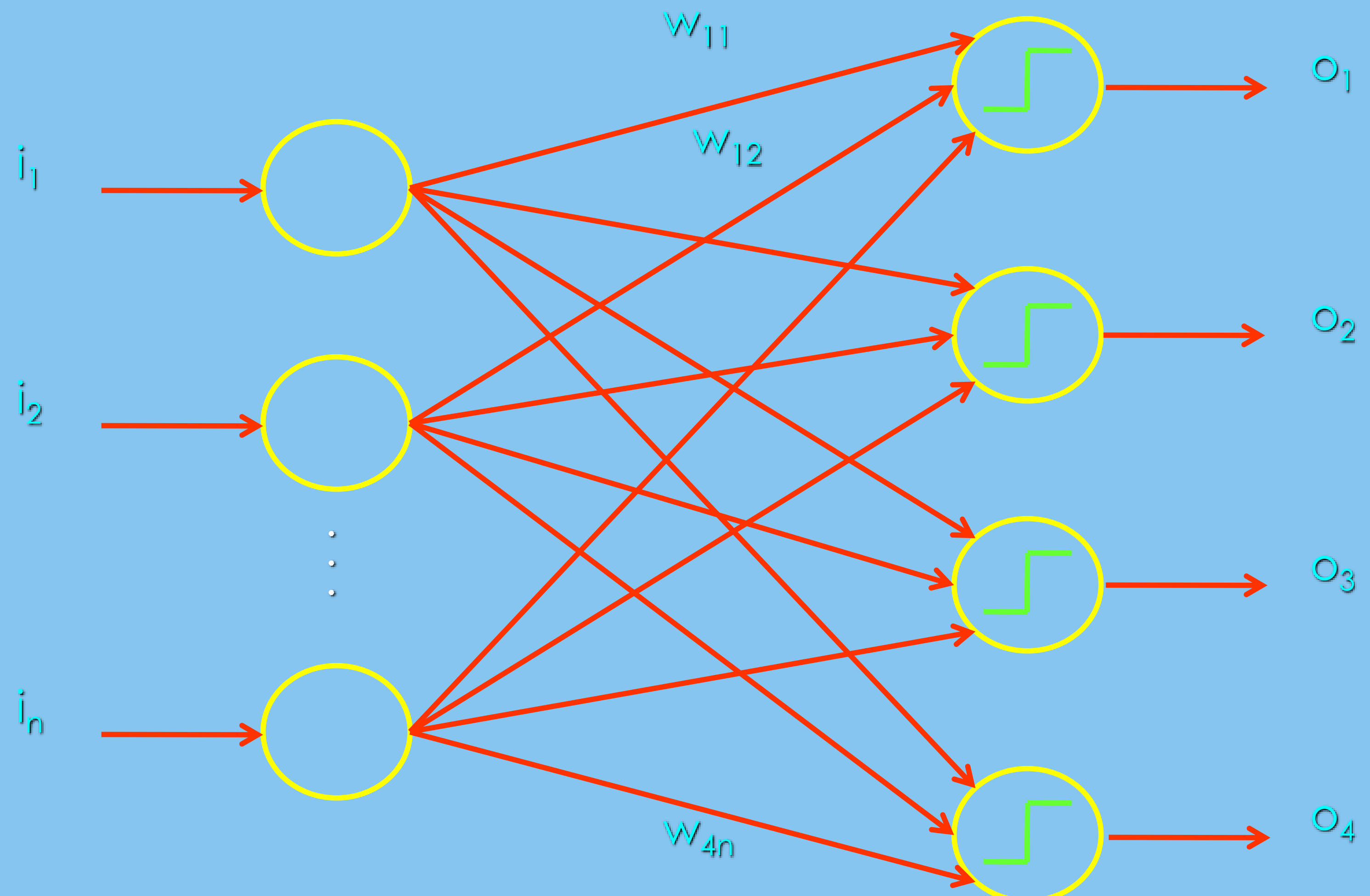
Step 7: Test for stopping condition.

X_1	X_2	b	Y_{in}	t	$t-Y_{in}$	ΔW_1	ΔW_2	Δb	W_1	W_2	B	$(t-Y_{in})^2$
									0.2	0.2	0.2	
1	1	1	0.6	1	0.4	0.08	0.08	0.08	0.28	0.28	0.28	0.16
1	-1	1	0.28	1	0.72	0.144	-0.144	0.144	0.424	0.136	0.424	0.51
-1	1	1	0.136	1	0.864	-0.173	0.173	0.173	0.251	0.309	0.597	0.74
-1	-1	1	0.037	1	0.963	-0.193	-0.193	0.963	0.0584	0.116	1.56	0.92

MULTICLASS DISCRIMINATION

- Often, our classification problems involve more than two classes.
- For example, character recognition requires at least 26 different classes.
- We can perform such tasks using layers of perceptrons or Adalines.

MULTICLASS DISCRIMINATION



- A four-node perceptron for a four-class problem in n-dimensional input space

MULTICLASS DISCRIMINATION

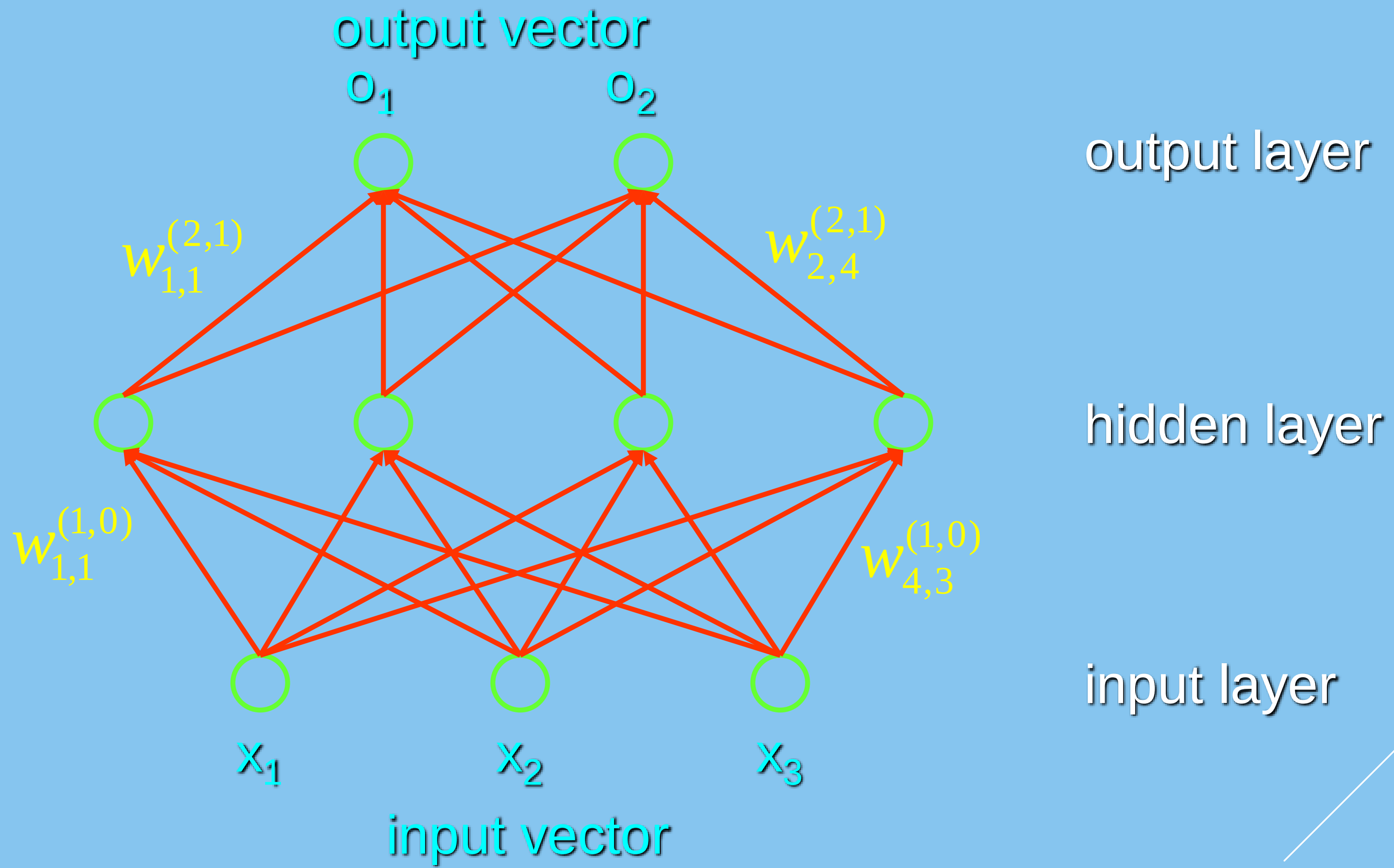
- Each perceptron learns to recognize one particular class, i.e., output 1 if the input is in that class, and 0 otherwise.
- The units can be trained separately and in parallel.
- In production mode, the network decides that its current input is in the k-th class if and only if $o_k = 1$, and for all $j \neq k$, $o_j = 0$, otherwise it is misclassified.
- For units with real-valued output, the neuron with maximal output can be picked to indicate the class of the input.
- This maximum should be significantly greater than all other outputs, otherwise the input is misclassified.

MULTILAYER NETWORKS

- Although single-layer perceptron networks can distinguish between any number of classes, they still require linear separability of inputs.
- To overcome this serious limitation, we can use multiple layers of neurons.
- Rosenblatt first suggested this idea in 1961, but he used perceptrons.
- However, their non-differentiable output function led to an inefficient and weak learning algorithm.
- The idea that eventually led to a breakthrough was the use of continuous output functions and gradient descent.

TERMINOLOGY

- **Example:** Network function f: $\mathbf{R}^3 \rightarrow \mathbf{R}^2$



MULTI LAYER ARTIFICIAL NEURAL NET

INPUT: records without class attribute with normalized attributes values.

INPUT VECTOR: $X = \{ x_1, x_2, \ldots, x_n \}$ where n is the number of (non-class) attributes.

INPUT LAYER: there are as many nodes as non-class attributes, i.e. as the length of the input vector.

HIDDEN LAYER: the number of nodes in the hidden layer and the number of hidden layers depends on implementation.

WEIGHT AND BIAS UPDATION

Per Sample Updating

- updating weights and biases after the presentation of each sample.

Per Training Set Updating (Epoch or Iteration)

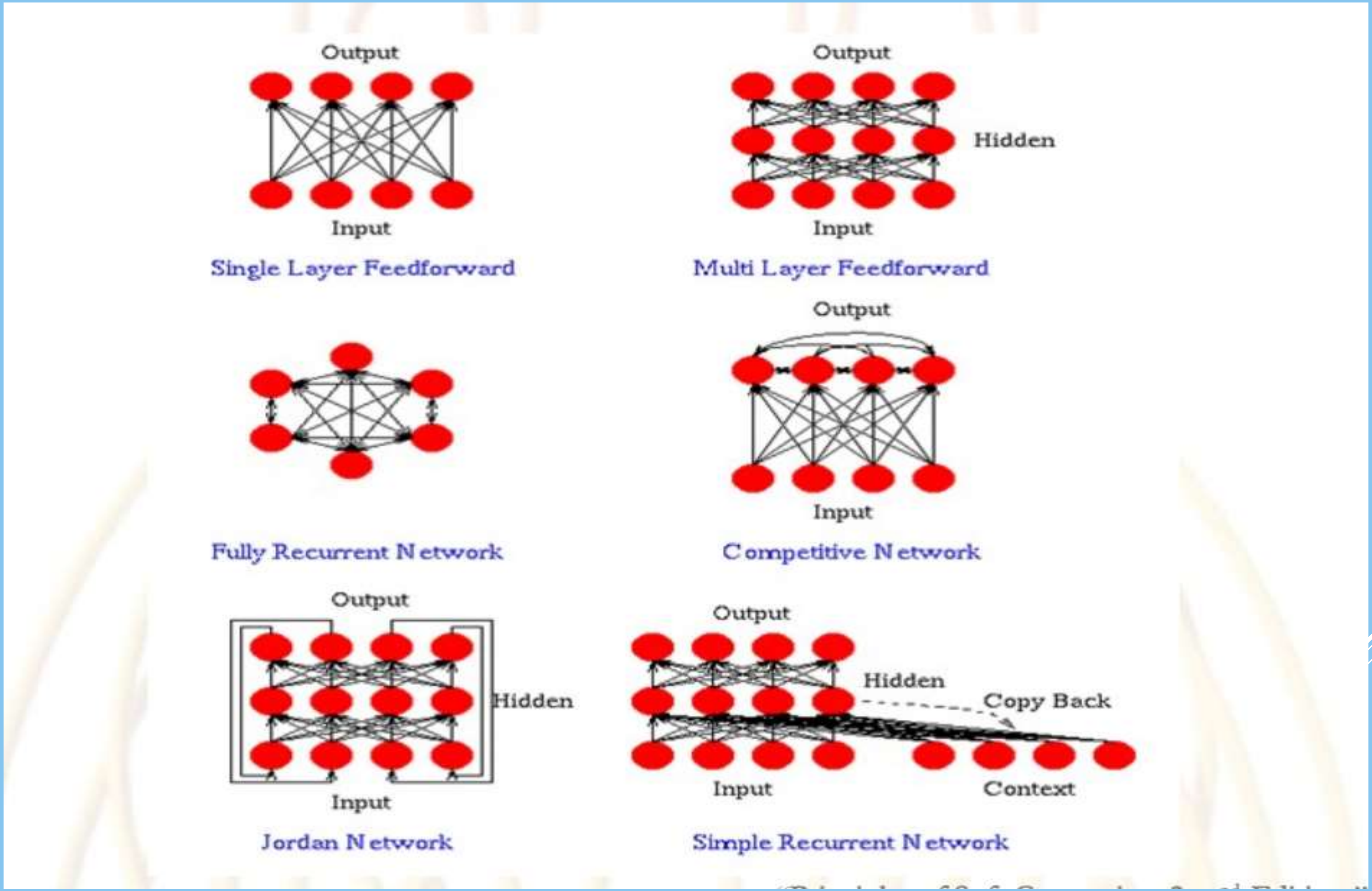
- weight and bias increments could be accumulated in variables and the weights and biases updated after all the samples of the training set have been presented.

STOPPING CONDITION

- All change in weights (Δwij) in the previous epoch are below some threshold, or
- The percentage of samples misclassified in the previous epoch is below some threshold, or
- A pre-specified number of epochs has expired.
- In practice, several hundreds of thousands of epochs may be required before the weights will converge.

BUILDING BLOCKS OF ARTIFICIAL NEURAL NET

- Network Architecture (Connection between Neurons)
- Setting the Weights (Training)
- Activation Function


Output
Input
Single Layer Feedforward
Output
Hidden
Input
Multi Layer Feedforward
Fully Recurrent Network
Output
Input
Competitive Network
Output
Hidden
Input
Jordan Network
Output
Hidden
Copy Back
Input
Context
Simple Recurrent Network

Madaline

Step 1: Initialize all weights and bias any small value other than zero. Set learning rate =0to 1.
Step 2: while stopping condition is false do step 3-7
Step 3: For each bipolar training vector and target output pair (s, t) perform steps 4-7
Step 4: Set activations for input units with input vector
Step 5: Compute the output unit response

$$Z_{inj} = b + \sum_i x_i * w_i$$

$$Z_j = f(Z_{inj}) = \begin{cases} 1\_if\_Z_{inj} >= 0 \\ -1\_if\_Z_{inj} < -0 \end{cases}$$

Step 6: Find the output of the net: $Y_{in} = b + \sum_i z_j * V_j$

$$Y = f(Y_{in}) = \begin{cases} 1\_if\_Y_{in} >= 0 \\ -1\_if\_Y_{in} < -0 \end{cases}$$

Step 7: Calculate the error and update the weights.

1. If t=y, no weight updation is required.

2. If t≠y and t=+1, update the weights on Z_j, where net input is closest to 0(zero)

$$w_i(new) = w_i(old) + \alpha * (1 - z_{inj}) * x_i$$

$$b(new) = b(old) + \alpha * (1 - z_{inj})$$

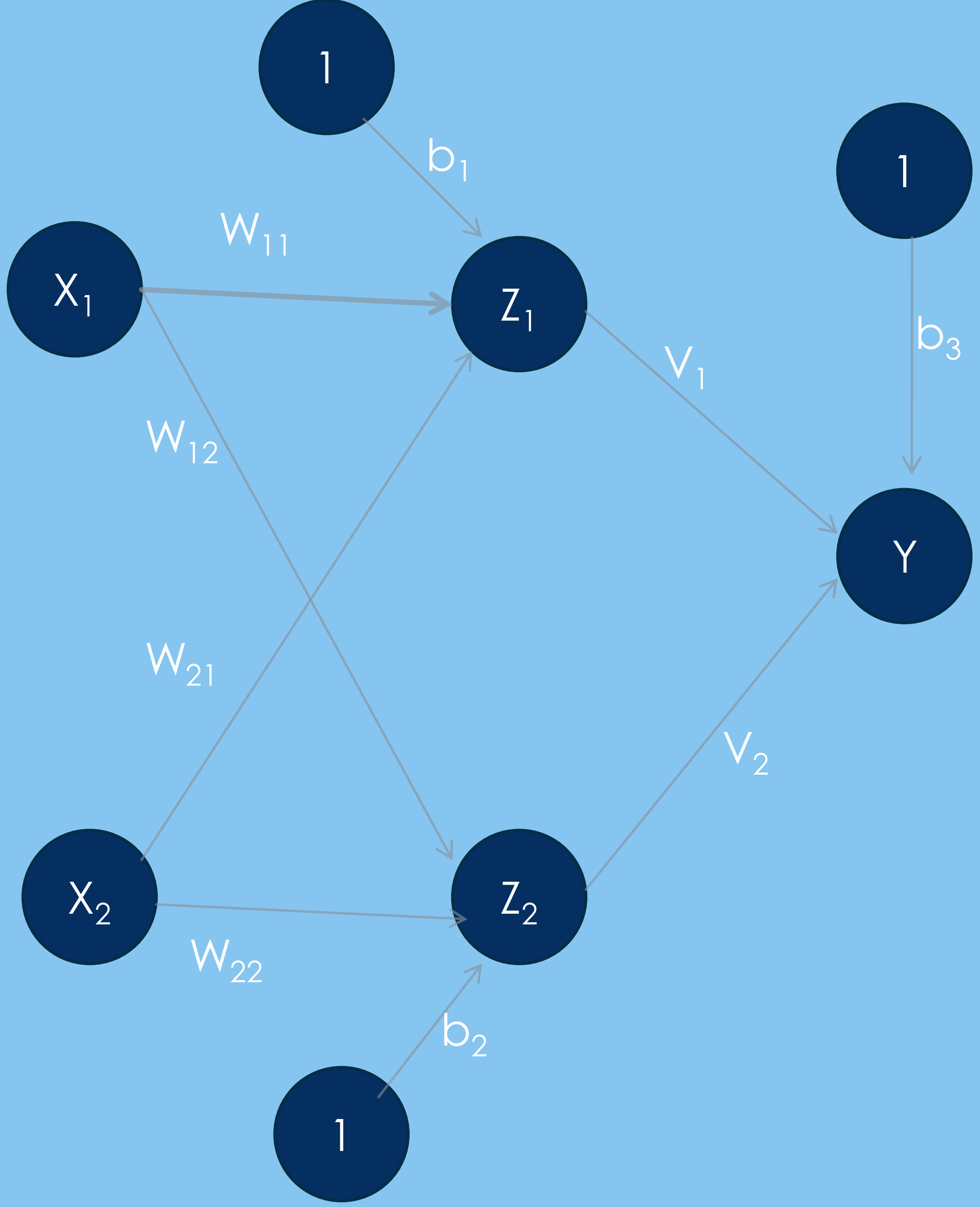
3. If t≠y and t=-1, update the weights on Z_j, where net input is closest to 0(zero)

$$w_i(new) = w_i(old) + \alpha * (-1 - z_{inj}) * x_i$$

$$b(new) = b(old) + \alpha * (-1 - z_{inj})$$

Step 8: Test for the stopping condition.

$W_{11} = 0.05$
$W_{12} = 0.1$
$W_{21} = 0.2$
$W_{22} = 0.2$
$V_1 = 0.5$
$V_2 = 0.5$
$b_1 = 0.3$
$b_2 = 0.15$
$b_3 = 0.5$



X_1	X_2	B	y
1	1	1	-1
1	-1	1	1
-1	1	1	1
-1	-1	1	-1

$$Z_{in1} = b_1 + x_1 w_{11} + x_1 w_{21}$$
$$Z_{in2} = b_2 + x_2 w_{12} + x_2 w_{22}$$

$$Z_1 = f(Z_{in1})$$
$$Z_2 = f(Z_{in2})$$
$$Y_{in} = b_2 + z_1 V_1 + z_2 V_2$$
$$Y = f(Y_{in})$$

$$w_i(new) = w_i(old) + \alpha * (1 - z_{inj}) * x_i$$

$$b(new) = b(old) + \alpha * (1 - z_{inj})$$

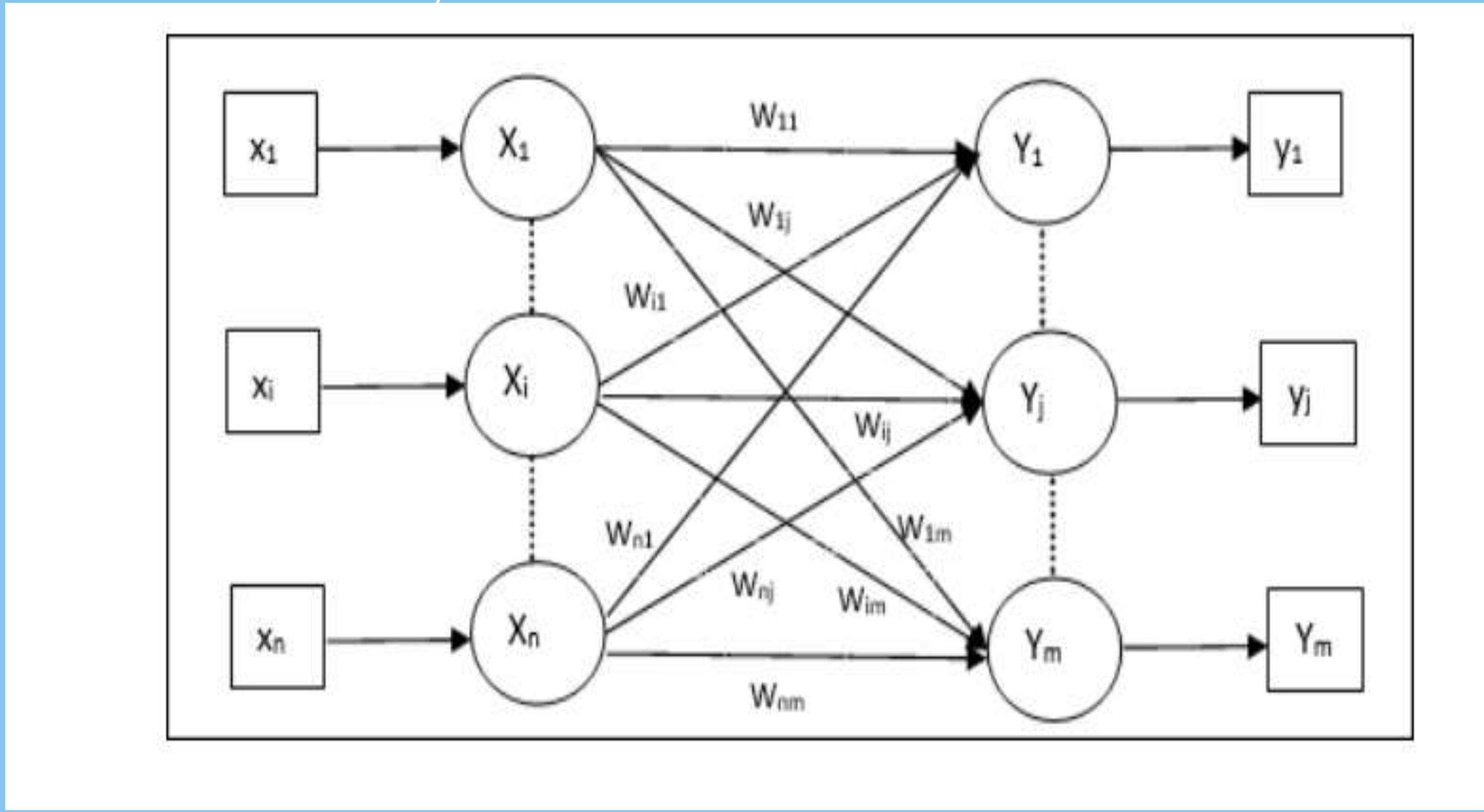
Hetero Associative Memory Neural Networks

Step 1: Initialize all weights using Hebb or Delta rule.
Step 2: whiFor each input vector do step 3-5
Step 3: For each bipolar training vector and target output pair (s, t) perform steps 4-7
Step 4: Set activations for input units with input vector
Step 5: Compute net input to the output units

$$Y_{inj} = \sum_i x_i * w_{ij}$$

$$Y_j = \begin{cases} 1\_if\_Y_{inj} > 0 \\ 0\_if\_Y_{inj} = 0 \\ -1\_if\_Z_{inj} < 0 \end{cases}$$

Hetero Associative Memory Neural Networks



Example 1: A hetero associative neural network is trained by Hebb outer product rule for input row vector S=(x1,x2,x3,x4) to the output row vectors t=(t1,t2). Find the weight matrix.

S1=(1 1 0 0) t1=(1 0)
S2=(1 1 1 0) t2=(0 1)
S3=(0 0 1 1) t3=(1 0)
S4=(0 1 0 0) t4=(1 0)

Step 1: Initialize all weights to zero
Step 2: Find the output for each input

S1=(1 1 0 0) t1=(1 0)

$$w_1 = S_1^t * t_1$$

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

Step 2: Find the output for each input

S2=(1 1 1 0)   t2=(0 1)

$$w_2 = S_2^t * t_2$$

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

Step 2: Find the output for each input

S3=(0 0 1 1)   t3=(1 0)

$$w_3 = S_3^t * t_3$$

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

Step 2: Find the output for each input

S4=(0 1 0 0) t4=(1 0)

$$w_4 = S_4^t * t_4$$

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

Step 3Weight matrix of all the four patterns is the sum of the weight matrix for each stored pattern

$$w = w_1 + w_2 + w_3 + w_4$$

$$= \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}$$

Example 2: A hetero associative neural network is trained by Hebb outer product rule for input row vector S=(x1,x2,x3,x4) to the output row vectors t=(t1,t2). Find the weight matrix.

S1=(1 1 0 0) t1=(1 0)
S2=(0 1 0 0) t2=(1 0)
S3=(0 0 1 1) t3=(0 1)
S4=(0 0 1 0) t4=(0 1)

Step 1: Initialize all weights to zero
Step 2: Find the output for each input
S1=(1 1 0 0) t1=(1 0)

$$w_1 = S_1^t * t_1$$

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

Step 2: Find the output for each input

S2=(0 1 0 0)   t2=(1 0)

$$w_2 = S_2^t * t_2$$

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

Step 2: Find the output for each input

S3=(0 0 1 1)   t3=(0 1)

$$w_3 = S_3^t * t_3$$

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

Step 2: Find the output for each input

S4=(0 0 1 0)   t4=(0 1)

$$w_4 = S_4^t * t_4$$

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

Step 3Weight matrix of all the four patterns is the sum of the weight matrix for each stored pattern

$$w = w_1 + w_2 + w_3 + w_4$$

$$= \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 2 & 0 \\ 0 & 2 \\ 0 & 1 \end{bmatrix}$$

Test the weight using different input set:

$$y_{inj} = \sum_{i=1}^{4} x_i * w_{ij}$$

$$y_{in1} = x_1 w_{11} + x_2 w_{21} + x_3 w_{31} + x_4 w_{41}$$

$$y_{in1} = x_1 w_{12} + x_2 w_{22} + x_3 w_{32} + x_4 w_{42}$$

S1=(1 1 0 0) t1=(1 0)

$$= \begin{bmatrix} 1 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 2 & 0 \\ 0 & 2 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

S2=(0 1 0 0) t2=(1 0)

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

S3=(0 0 1 1)   t3=(0 1)

$$=\begin{bmatrix}0 & 0 & 1 & 1\end{bmatrix}\begin{bmatrix}1 & 0\\ 2 & 0\\ 0 & 2\\ 0 & 1\end{bmatrix}=\begin{bmatrix}0 & 3\end{bmatrix}=\begin{bmatrix}0 & 1\end{bmatrix}$$

S4=(0 0 1 0)   t4=(0 1)

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

Test the weight using different input set:

S3=(1 1 1 0)   t3=?

$$= \begin{bmatrix} 1 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 2 & 0 \\ 0 & 2 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \end{bmatrix}$$

Example 3: A hetero associative neural network is trained by Hebb outer product rule for input row vector S=(x1,x2,x3,x4) to the output row vectors t=(t1,t2). Find the weight matrix.

S1=(1 1 -1 -1) t1=(1 -1)
S2=(-1 1 -1 -1) t2=(1 -1)
S3=(-1 -1 1 1) t3=(-1 1)
S4=(-1 -1 1 -1) t4=(-1 1)

Step 1: Initialize all weights to zero
Step 2: Find the output for each input

S1=(1 1 -1 -1) t1=(1 -1)

$$w_1 = S_1^t * t_1$$

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

Step 2: Find the output for each input

S2=(-1 1 -1 -1)   t2=(1 -1)

$$w_2 = S_2^t * t_2$$

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

Step 2: Find the output for each input

S3=(-1 -1 1 1)   t3=(-1 1)

$$w_3 = S_3^t * t_3$$

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

Step 2: Find the output for each input

S4=(-1 -1 1 -1)   t4=(-1 1)

$$w_4 = S_4^t * t_4$$

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

Step 3Weight matrix of all the four patterns is the sum of the weight matrix for each stored pattern

$$w = w_1 + w_2 + w_3 + w_4$$

$$= \begin{bmatrix} 1 & -1 \\ 1 & -1 \\ -1 & 1 \\ -1 & 1 \end{bmatrix} + \begin{bmatrix} -1 & 1 \\ 1 & -1 \\ -1 & 1 \\ -1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & -1 \\ 1 & -1 \\ -1 & 1 \\ -1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & -1 \\ 1 & -1 \\ -1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 2 & -2 \\ 4 & -4 \\ -4 & 4 \\ -2 & 2 \end{bmatrix}$$

Test the weight using different input set:

$$y_{inj} = \sum_{i=1}^{4} x_i * w_{ij}$$

$$y_{in1} = x_1 w_{11} + x_2 w_{21} + x_3 w_{31} + x_4 w_{41}$$

$$y_{in1} = x_1 w_{12} + x_2 w_{22} + x_3 w_{32} + x_4 w_{42}$$

S1=(1 1 -1 -1) t1=(1 -1)

$$= \begin{bmatrix} 1 & 1 & -1 & -1 \end{bmatrix} \begin{bmatrix} 2 & -2 \\ 4 & -4 \\ -4 & 4 \\ -2 & 2 \end{bmatrix} = \begin{bmatrix} 12 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

S2=(-1 1 -1 -1) t2=(1 -1)

$$= \begin{bmatrix} -1 & 1 & -1 & -1 \end{bmatrix} \begin{bmatrix} 2 & -2 \\ 4 & -4 \\ -4 & 4 \\ -2 & 2 \end{bmatrix} = \begin{bmatrix} 8 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

S3=(-1 -1 1 1) t3=(-1 1)

$$=\begin{bmatrix}-1 & -1 & 1 & 1\end{bmatrix}\begin{bmatrix}2 & -2\\ 4 & -4\\ -4 & 4\\ -2 & 2\end{bmatrix}=\begin{bmatrix}-12 & 12\end{bmatrix}=\begin{bmatrix}0 & 1\end{bmatrix}$$

S4=(-1 -1 1 -1) t4=(-1 1)

$$=\begin{bmatrix}-1 & -1 & 1 & -1\end{bmatrix}\begin{bmatrix}2 & -2\\ 4 & -4\\ -4 & 4\\ -2 & 2\end{bmatrix}=\begin{bmatrix}-8 & 8\end{bmatrix}=\begin{bmatrix}0 & 1\end{bmatrix}$$

Test the weight using different input set:

$$y_{inj} = \sum_{i=1}^{4} x_i * w_{ij}$$

$$y_{in1} = x_1 w_{11} + x_2 w_{21} + x_3 w_{31} + x_4 w_{41}$$

$$y_{in1} = x_1 w_{12} + x_2 w_{22} + x_3 w_{32} + x_4 w_{42}$$

S1=(1 1 0 0) t1=(1 0)

$$= \begin{bmatrix} 1 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 2 & -2 \\ 4 & -4 \\ -4 & 4 \\ -2 & 2 \end{bmatrix} = \begin{bmatrix} 6 & -6 \end{bmatrix} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

S2=(0 1 0 0) t2=(1 0)

$$= \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 2 & -2 \\ 4 & -4 \\ -4 & 4 \\ -2 & 2 \end{bmatrix} = \begin{bmatrix} 4 & -4 \end{bmatrix} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

S3=(0 0 1 1)   t3=(0 1)

$$=\begin{bmatrix}0 & 0 & 1 & 1\end{bmatrix}\begin{bmatrix}2 & -2\\ 4 & -4\\ -4 & 4\\ -2 & 2\end{bmatrix}=\begin{bmatrix}-6 & 6\end{bmatrix}=\begin{bmatrix}0 & 1\end{bmatrix}$$

S4=(0 0 1 0)   t4=(0 1)

$$=\begin{bmatrix}0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}2 & -2\\ 4 & -4\\ -4 & 4\\ -2 & 2\end{bmatrix}=\begin{bmatrix}-4 & 4\end{bmatrix}=\begin{bmatrix}0 & 1\end{bmatrix}$$

S3=(1 0 0 0)   t3=?

$$=\begin{bmatrix}1 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}2 & -2\\ 4 & -4\\ -4 & 4\\ -2 & 2\end{bmatrix}=\begin{bmatrix}2 & -2\end{bmatrix}=\begin{bmatrix}1 & 0\end{bmatrix}$$

S4=(1 1 1 0)   t4=?

$$=\begin{bmatrix}1 & 1 & 1 & 0\end{bmatrix}\begin{bmatrix}2 & -2\\ 4 & -4\\ -4 & 4\\ -2 & 2\end{bmatrix}=\begin{bmatrix}2 & -2\end{bmatrix}=\begin{bmatrix}1 & 0\end{bmatrix}$$

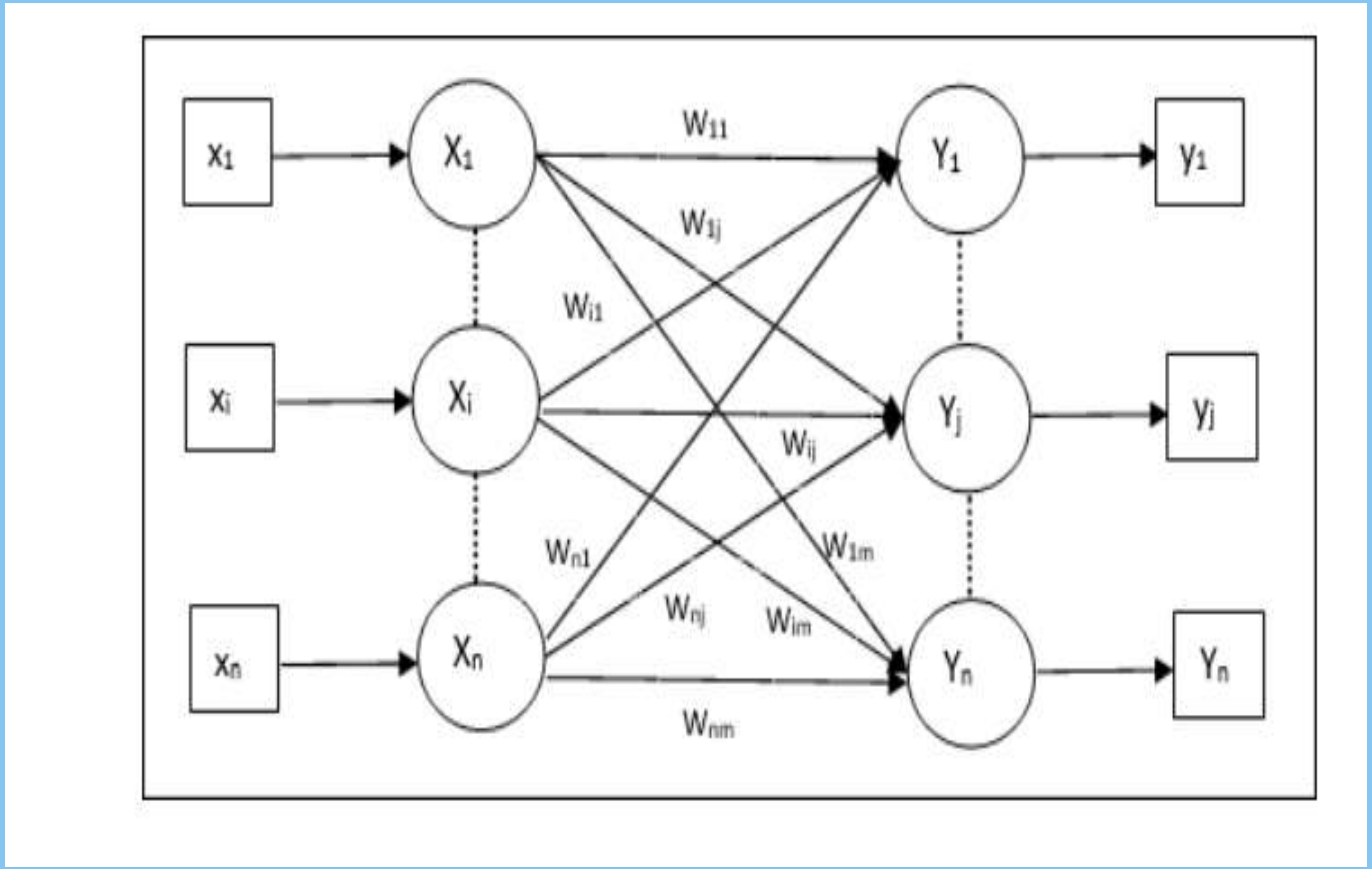
S5=(1 1 1 1) t5=?

$$=\begin{bmatrix}1 & 1 & 1 & 1\end{bmatrix}\begin{bmatrix}2 & -2\\ 4 & -4\\ -4 & 4\\ -2 & 2\end{bmatrix}=[0 \quad 0]=[0 \quad 0]$$

S6=(0 1 -1 0) t6=?

$$=\begin{bmatrix}0 & 1 & -1 & 0\end{bmatrix}\begin{bmatrix}2 & -2\\ 4 & -4\\ -4 & 4\\ -2 & 2\end{bmatrix}=[8 \quad -8]=[1 \quad 0]$$

Auto Associative Memory Neural Networks



Mutually Orthogonal Pairs

Two vectors x and y are orthogonal if

$$\sum_i x_i * y_i = 0$$

$$= \begin{bmatrix} 1 & 1 & -1 & -1 \end{bmatrix} * \begin{bmatrix} -1 \\ 1 \\ 1 \\ -1 \end{bmatrix} = [0]$$

Example 3: A auto associative neural network is trained by Hebb outer product rule for input row vector S=(1 1 -1 -1). Find the weight matrix.
S1=(1 1 -1 -1) t1=(1 1 -1 -1)

Step 1: Initialize all weights to zero
Step 2: Find the output for each input
S1=(1 1 -1 -1) t1=(1 1 -1 -1)

$$w_1 = S_1^t * t_1$$

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

Test the weight using different input set:

$$y_{inj} = \sum_{i=1}^{4} x_i * w_{ij}$$

$$y_{in1} = x_1 w_{11} + x_2 w_{21} + x_3 w_{31} + x_4 w_{41}$$

$$y_{in2} = x_1 w_{12} + x_2 w_{22} + x_3 w_{32} + x_4 w_{42}$$

$$y_{in3} = x_1 w_{13} + x_2 w_{23} + x_3 w_{33} + x_4 w_{43}$$

$$y_{in4} = x_1 w_{14} + x_2 w_{24} + x_3 w_{34} + x_4 w_{44}$$

S1=(1 1 0 0) t1=(1 0)

$$= [1 \quad 1 \quad -1 \quad -1] \begin{bmatrix} 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 \end{bmatrix} = [1 \quad 1 \quad -1 \quad -1]$$

S2=(0 1 0 0) t2=(1 0)

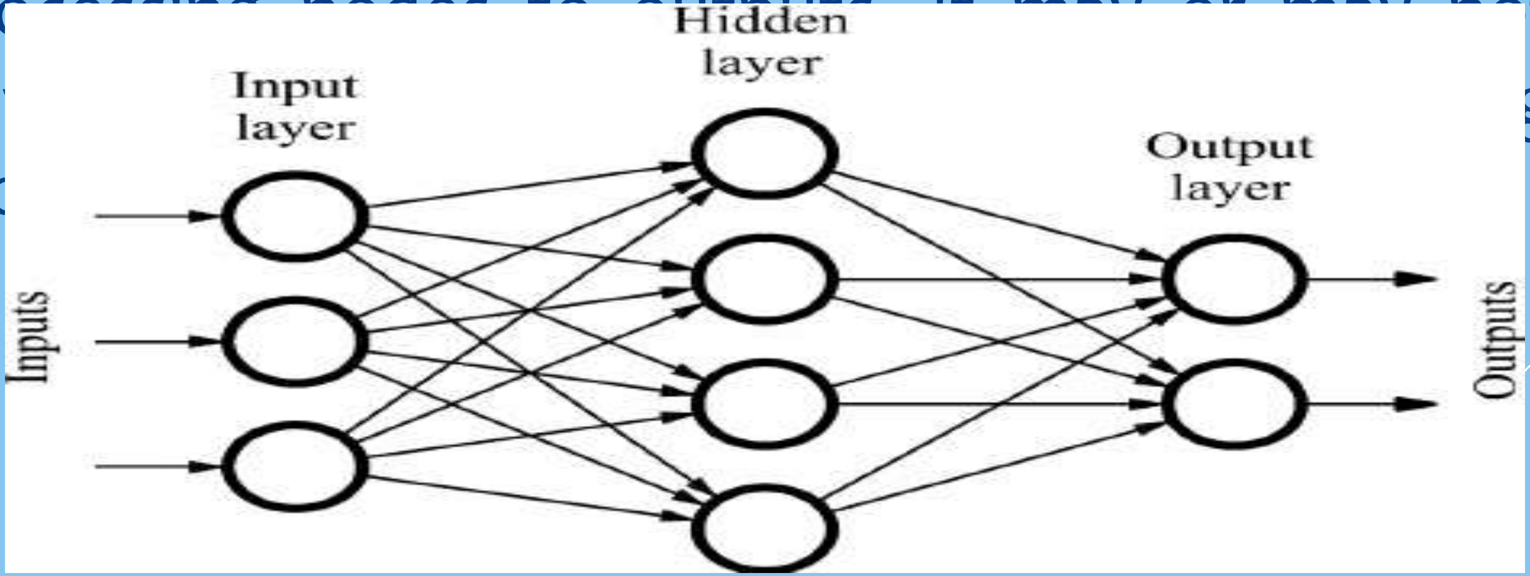
$$= [0 \quad 1 \quad 0 \quad 0] \begin{bmatrix} 2 & -2 \\ 4 & -4 \\ -4 & 4 \\ -2 & 2 \end{bmatrix} = [4 \quad -4] = [1 \quad 0]$$

DIFFERENT TYPES OF NEURAL NETWORKS

- Variations on the classic neural network design allow various forms of forward and backward propagation of information among tiers.
- 1. feed-forward
- 2.backpropagation

FEED-FORWARD NEURAL NETWORK

This type of artificial neural network algorithm passes information straight through from input to pr ot ha g mc

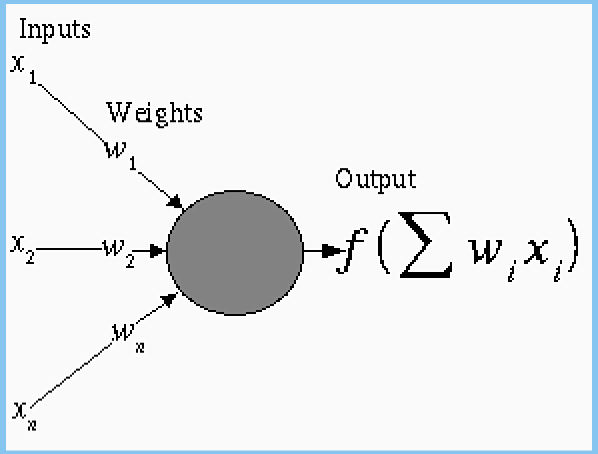


BACK PROPAGATION NEURAL NETWORK

- Recurrent Neural Networks (Back-Propagating)
- Information passes from input layer to output layer to produce result. Error in result is then communicated back to previous layers now. Nodes get to know how much they contributed in the answer being wrong. Weights are re-adjusted. Neural network is improved. It learns. There is bi-directional flow of information.

BASIC NEURON MODEL IN A FEEDFORWARD NETWORK

- **Inputs x_i** arrive through pre-synaptic connections
- Synaptic efficacy is modeled using real **weights w_i**
- The response of the neuron is a **nonlinear function *f*** of its weighted inputs



- Arise from other neurons or from outside the network
- Nodes whose inputs arise outside the network are called *input nodes* and simply copy values
- An input may *excite* or *inhibit* the response of the neuron to which it is applied, depending upon the weight of the connection

INPUTS TO NEURONS

WEIGHTS

- Represent synaptic efficacy and may be *excitatory* or *inhibitory*
- Normally, positive weights are considered as excitatory while negative weights are thought of as inhibitory
- ***Learning*** is the process of modifying the weights in order to produce a network that performs some function

OUTPUT

- The response function is normally nonlinear
- Samples include
  - Sigmoid

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

  - Piecewise linear

$$f(x) = \begin{cases} x, if \ \ x \ge \theta \\ 0, if \ \ x < \theta \end{cases}$$

- **Training Set**
  A collection of input-output patterns that are used to train the network
- **Testing Set**
  A collection of input-output patterns that are used to assess network performance
- **Learning Rate-*a***
  A scalar parameter, analogous to step size in numerical integration, used to set the rate of adjustments

BACKPROPAGATION PREPARATION

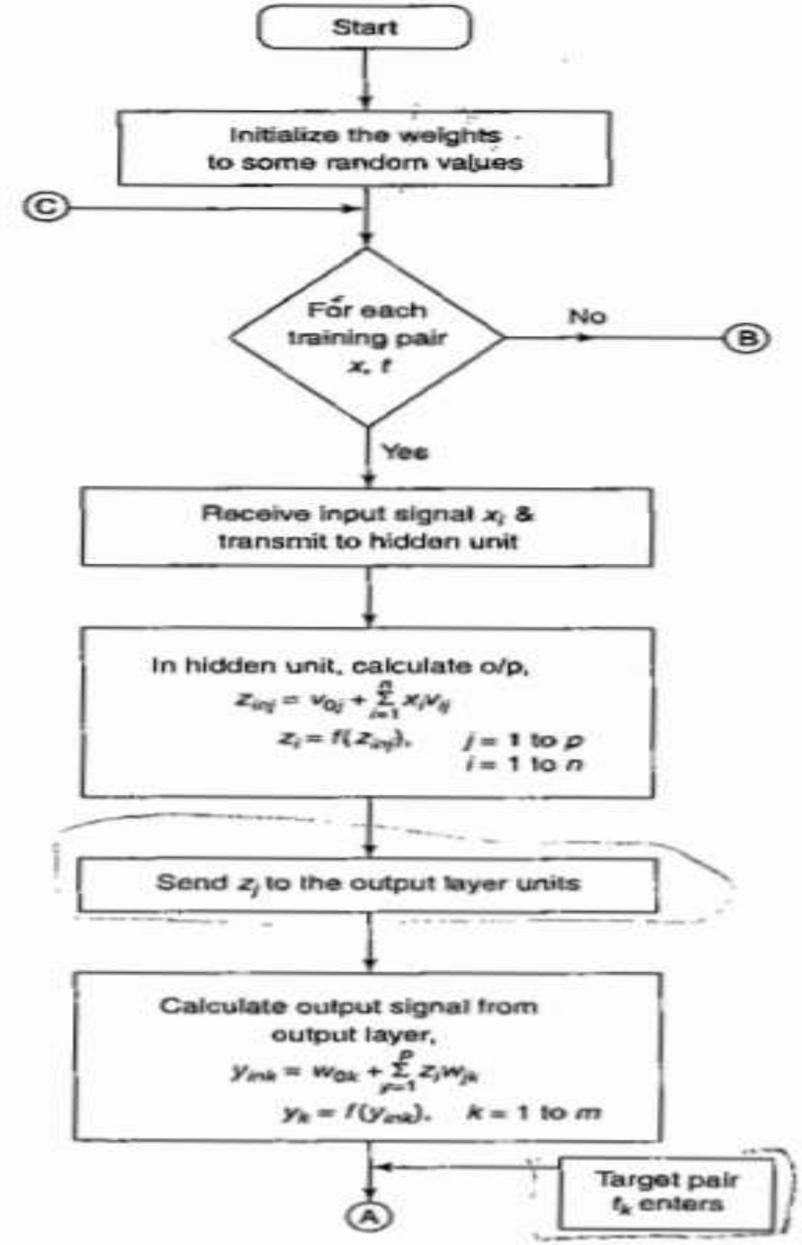
NETWORK ERROR

- Total-Sum-Squared-Error (TSSE)

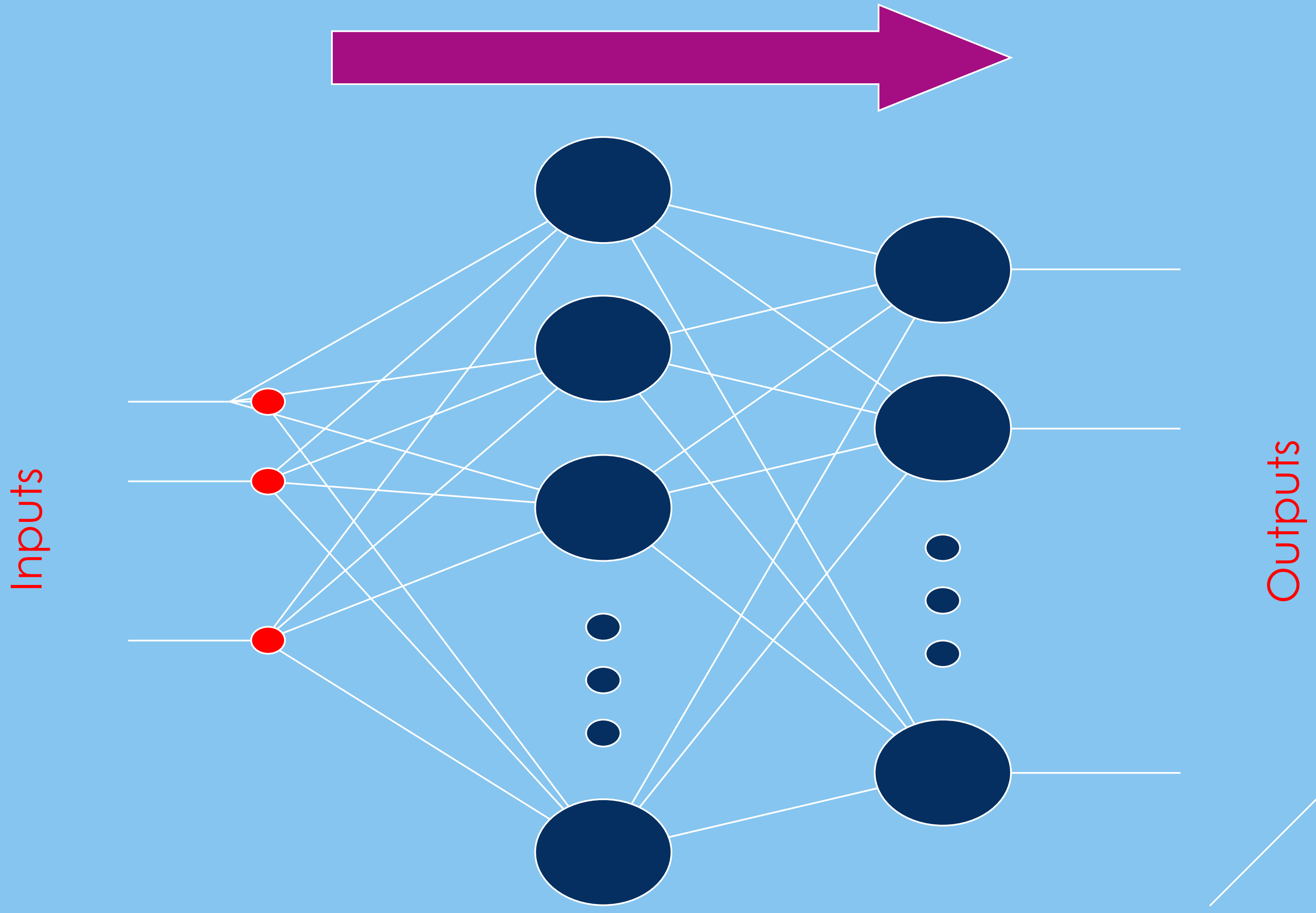
$$TSSE = \frac{1}{2}\sum_{patterns}\sum_{outputs}(desired - actual)^2$$

- Root-Mean-Squared-Error (RMSE)

$$RMSE = \sqrt{\frac{2 * TSSE}{\# patterns * \# outputs}}$$


Start
Initialize the weights to some random values
C
For each training pair x, t
No
B
Yes
Receive input signal x_i & transmit to hidden unit
In hidden unit, calculate o/p,
$z_{inj} = v_{0j} + \sum_{i=1}^{n} x_i v_{ij}$
$z_j = f(z_{inj})$, $j = 1$ to p
$i = 1$ to n
Send z_j to the output layer units
Calculate output signal from output layer,
$y_{ink} = w_{0k} + \sum_{j=1}^{p} z_j w_{jk}$
$y_k = f(y_{ink})$, $k = 1$ to m
Target pair t_k enters
A

Feedforward



$$Y_{ink} = \sum_{i} z_j V_{jk}$$

$$Z_{-inj} = \sum_{i} x_i w_{ij}$$

$$Y_k = f(Y_{ink})$$

$$E = 0.5\sum_{k}[t_k - y_k]^2$$

By use of chain rule we have

$$Y_{ink} = \sum_{i} z_j V_{jk}$$

$$Z_{-inj} = \sum_{i} x_i w_{ij}$$

$$Y_k = f(Y_{ink})$$

$$\frac{\partial E}{\partial V_{jk}} = \frac{\partial}{\partial V_{jk}}(0.5\sum_{k}[t_k - y_k]^2)$$

$$= \frac{\partial}{\partial V_{jk}}(0.5\sum_{k}[t_k - f(Y_{ink})]^2)$$

$$= \frac{\partial}{\partial V_{jk}}(0.5\sum_{k}[t_k - f(Y_{ink})]^2)$$

$$= -[t_k - y_k]\frac{\partial}{\partial V_{jk}} f(Y_{ink})$$

$$= -[t_k - y_k]\frac{\partial}{\partial V_{jk}} f^1(Y_{ink})z_j$$

$$\delta_k = -[t_k - y_k] f^1(Y_{ink})$$

$$\frac{\partial E}{\partial W_{ij}} = -[t_k - y_k]\frac{\partial}{\partial W_{ij}} f(Y_{ink})$$

$$= -[t_k - y_k]\frac{\partial}{\partial W_{ij}} f^1(Y_{ink})z_j$$

$$= -\delta_k \frac{\partial}{\partial W_{ij}} \sum_k z_j$$

$$= -\delta_k w_{ij} f^1(Z_{inj})x_i$$

$$\delta_j = -\sum_k \delta_k w_{ij} f^1(Z_{inj})$$

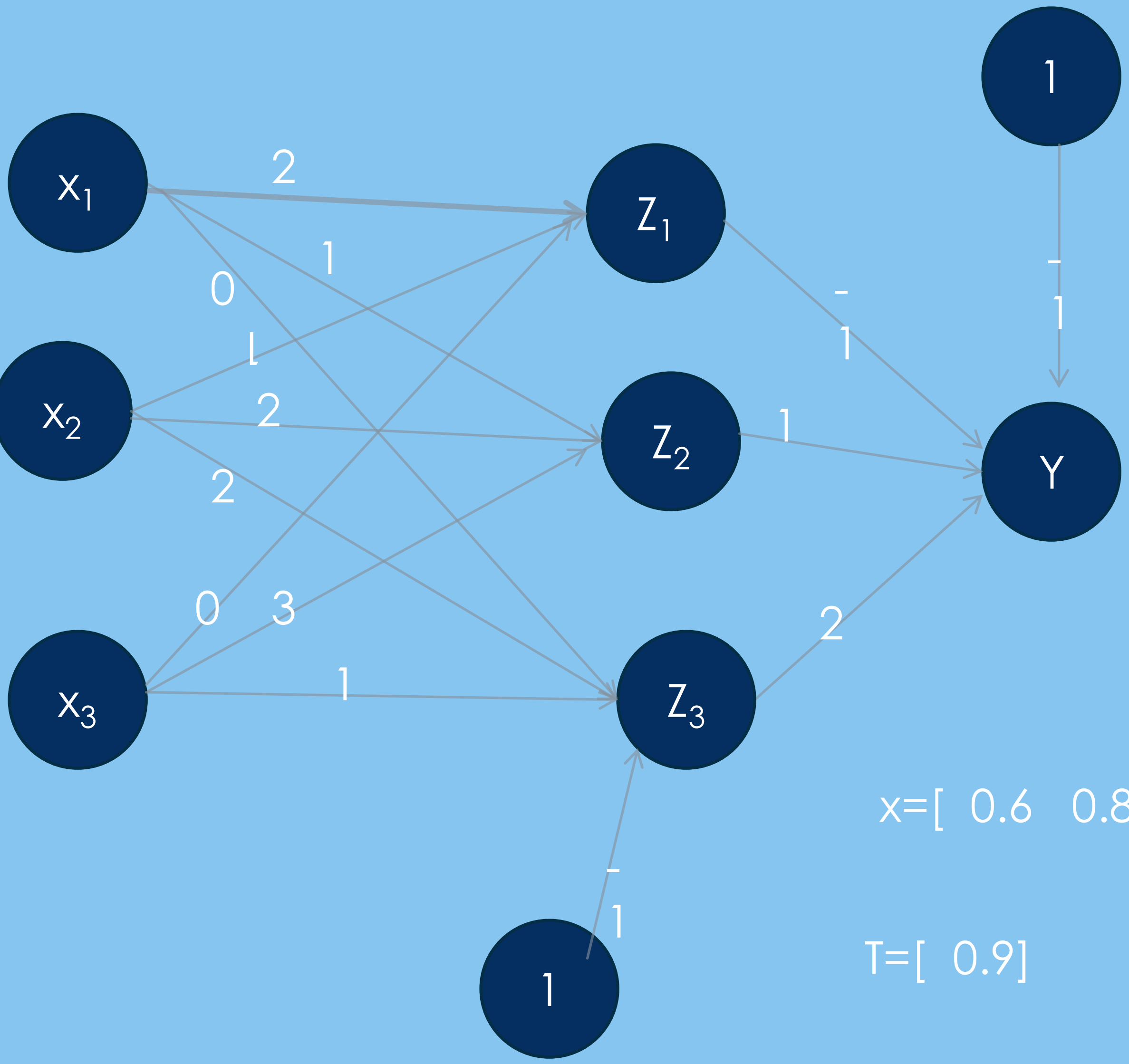
$$\Delta V_{jk} = -\alpha \frac{\partial E}{\partial V_{jk}}$$

$$= \alpha[t_k - y_k] f^1(Y_{ink})z_j$$

$$= \alpha * \delta_k * z_j$$

$$\Delta W_{ij} = -\alpha * \frac{\partial E}{\partial W_{ij}}$$

$$= \alpha * \delta_j * x_i$$



$V = [\,-1 \quad 1 \quad 2\,]$

$W = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 2 \\ 0 & 3 & 1 \end{bmatrix}$

$Vo = [-1]$

$Wo = [\,0 \quad 0 \quad -1\,]$

$x = [\,0.6 \quad 0.8 \quad 0\,]$

$T = [\,0.9\,]$

$$Z_{inj} = w_{oj} + \sum_i x_i w_{ij}$$

$$Z_{in1} = w_{o1} + \sum_{i=1}^{3} x_i w_{i1}$$

$$= w_{o1} + x_1 w_{11} + x_2 w_{21} + x_3 w_{31}$$

$$= 0 + 0.6 * 2 + 0.8 * 1 + 0 * 0$$

$$= 1.2 + 0.8 = 2$$

$$Z_{inj} = w_{oj} + \sum_i x_i w_{ij}$$

$$Z_{in2} = w_{o2} + \sum_{i=1}^{3} x_i w_{i2}$$

$$= w_{o2} + x_1 w_{12} + x_2 w_{22} + x_3 w_{32}$$

$$= 0 + 0.6 + 0.8 * 2 + 0 * 3$$

$$= 2.2$$

$$Z_{inj} = w_{oj} + \sum_i x_i w_{ij}$$

$$Z_{in3} = w_{o3} + \sum_{i=1}^{3} x_i w_{i3}$$

$$= w_{o3} + x_1 w_{13} + x_2 w_{23} + x_3 w_{33}$$

$$z_1 = f(z_{in1}) = \frac{1}{1+e^{-2}}$$

$$z_2 = f(z_{in2}) = \frac{1}{1+e^{-2.2}}$$

$$z_3 = f(z_{in3}) = \frac{1}{1+e^{-0.6}}$$

$$Y_{ink} = V_{ok} + \sum_{j} Z_j V_{jk}$$

$$Y_{in1} = V_{o1} + \sum_{j=1}^{3} Z_j V_{j1}$$

$$= V_{o1} + z_1 V_{11} + z_2 V_{21} + z_3 V_{31}$$

$$Y_1 = f(Y_{in1}) = \frac{1}{1 + e^{-Y_{in1}}}$$

$$z_1 = f(z_{in1}) = \frac{1}{1 + e^{-2}}$$

$$z_2 = f(z_{in2}) = \frac{1}{1 + e^{-2.2}}$$

$$z_3 = f(z_{in3}) = \frac{1}{1 + e^{-0.6}}$$

$$\delta_k = (t_k - Y_k)\, f'(Y_{ink})$$
$$\delta_1 = (t_1 - Y_1)\, f'(Y_{in1})$$
$$f'(Y_{in1}) = f(Y_{in1})(1 - f(Y_{in1}))$$
$$= Y_1 * (1 - Y_1)$$

$$\delta_{inj} = \sum_{k=1}^{m} \delta_k V_{jk}$$

$$\Delta_j = \delta_{inj} * f'(z_{inj})$$

$$\Delta_1 = \delta_{in1} * f(z_{in1})(1 - f(z_{in1}))$$

$$\Delta_2 = \delta_{in2} * f(z_{in2})(1 - f(z_{in2}))$$

$$\Delta_3 = \delta_{in3} * f(z_{in3})(1 - f(z_{in3}))$$

$$\Delta w_{ij} = \alpha * \Delta_j * x_i$$
$$\Delta w_{11} = \alpha * \Delta_1 * x_1$$
$$\Delta w_{12} = \alpha * \Delta_2 * x_1$$
$$\Delta w_{13} = \alpha * \Delta_3 * x_1$$
$$\Delta w_{21} = \alpha * \Delta_1 * x_2$$
$$\Delta w_{22} = \alpha * \Delta_2 * x_2$$
$$\Delta w_{23} = \alpha * \Delta_3 * x_2$$
$$\Delta w_{31} = \alpha * \Delta_1 * x_3$$
$$\Delta w_{32} = \alpha * \Delta_2 * x_3$$
$$\Delta w_{33} = \alpha * \Delta_3 * x_3$$
$$\Delta w_{01} = \alpha * \Delta_1$$
$$\Delta w_{02} = \alpha * \Delta_2$$
$$\Delta w_{03} = \alpha * \Delta_3$$

$$\Delta w_{ij} = \alpha * \Delta_j * x_i$$

$$w_{11}(new) = w_{11}(old) + \Delta w_{11}$$

$$w_{12}(new) = w_{12}(old) + \Delta w_{12}$$

$$w_{13}(new) = w_{13}(old) + \Delta w_{13}$$

$$w_{21}(new) = w_{21}(old) + \Delta w_{21}$$

$$w_{22}(new) = w_{22}(old) + \Delta w_{22}$$

$$w_{23}(new) = w_{23}(old) + \Delta w_{23}$$

$$w_{31}(new) = w_{31}(old) + \Delta w_{31}$$

$$w_{32}(new) = w_{32}(old) + \Delta w_{32}$$

$$w_{33}(new) = w_{33}(old) + \Delta w_{33}$$

$$w_{01}(new) = w_{01}(old) + \Delta w_{01}$$

$$w_{02}(new) = w_{02}(old) + \Delta w_{02}$$

$$w_{03}(new) = w_{03}(old) + \Delta w_{03}$$

$$\Delta V_{jk} = \alpha * \delta_k * z_j$$

$$\Delta V_{11} = \alpha * \delta_1 * z_1$$

$$\Delta V_{12} = \alpha * \delta_1 * z_2$$

$$\Delta V_{13} = \alpha * \delta_1 * z_2$$

$$\Delta V_{01} = \alpha * \delta_1$$

$$V_{11}(new) = V_{11}(old) + \Delta V_{11}$$

$$V_{12}(new) = V_{12}(old) + \Delta V_{12}$$

$$V_{13}(new) = V_{13}(old) + \Delta V_{13}$$

$$V_{01}(new) = V_{01}(old) + \Delta V_{01}$$

- We can include additional structure in the network so that the net is forced to make a decision as to which one unit will respond.
- The mechanism by which it is achieved is called *competition.*
- It can be used in unsupervised learning.
- A common use for unsupervised learning is clustering based neural networks.

UNSUPERVISED LEARNING

- In a clustering net, there are as many units as the input vector has components.
- Every output unit represents a cluster and the number of output units limit the number of clusters.
- During the training, the network finds the best matching output unit to the input vector.
- The weight vector of the winner is then updated according to learning algorithm.

UNSUPERVISED LEARNING

- A variety of nets use Kohonen Learning
  - New weight vector is the linear combination of old weight vector and the current input vector.
  - The weight update for cluster unit (output unit) *j* can be calculated as:
  - the learning rate alpha decreases as the learning process proceeds.

$$\mathbf{w}_{\cdot j}(\text{new}) = \mathbf{w}_{\cdot j}(\text{old}) + \alpha[\mathbf{x} - \mathbf{w}_{\cdot j}(\text{old})]$$
$$= \alpha\mathbf{x} + (1 - \alpha)\mathbf{w}_{\cdot j}(\text{old}),$$

KOHONEN LEARNING

- Since it is unsupervised environment, so the name is Self Organizing Maps.

- Self Organizing NNs are also called Topology Preserving Maps which leads to the idea of neighborhood of the clustering unit.

- During the self-organizing process, the weight vectors of winning unit and its neighbors are updated.

KOHONEN SOM (SELF ORGANIZING MAPS)

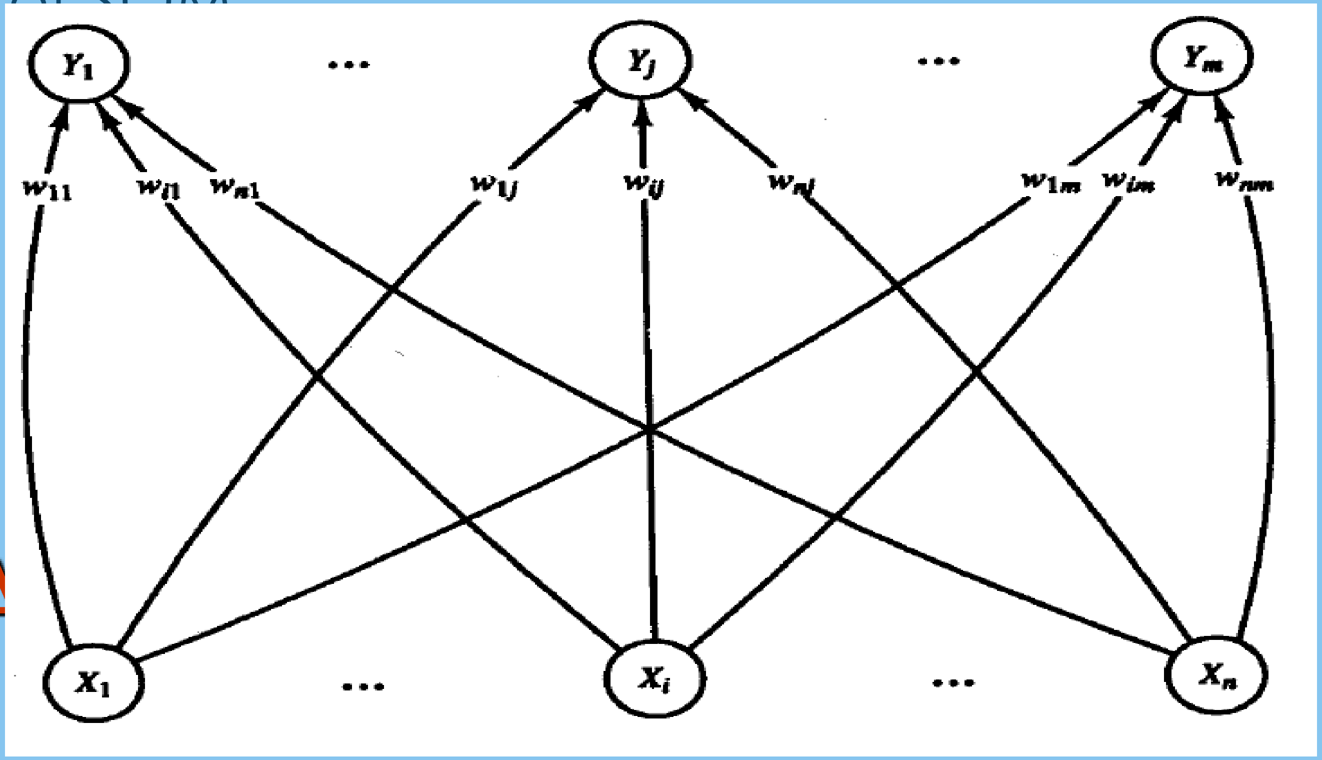
- Normally, Euclidean distance measure is used to find the cluster unit whose weight vector matches most closely to the input vector.

- For a linear array of cluster units, the neighborhood of radius *R* around cluster unit *J* consists of all units *j* such that:

KOHONEN SOM (SEI

$$\max(1, J - R) \le j \le \min(J + R, m)).$$

- Architecture of SOM



KOHONEN SOM

- Structure of Neighborhoods

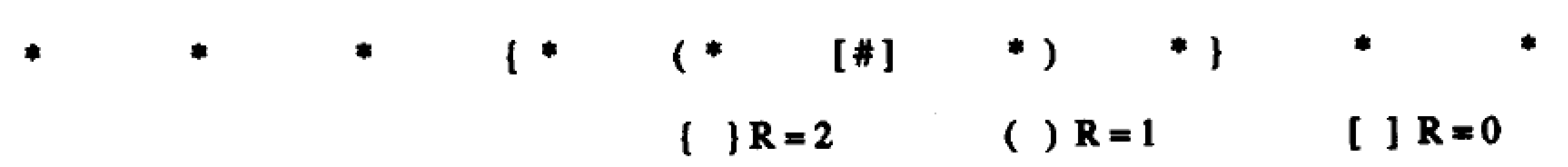


Figure 4.6 Linear array of cluster units.

KOHONEN SOM (SELF ORGANIZING MAPS)

- Structure of Neighborhoods

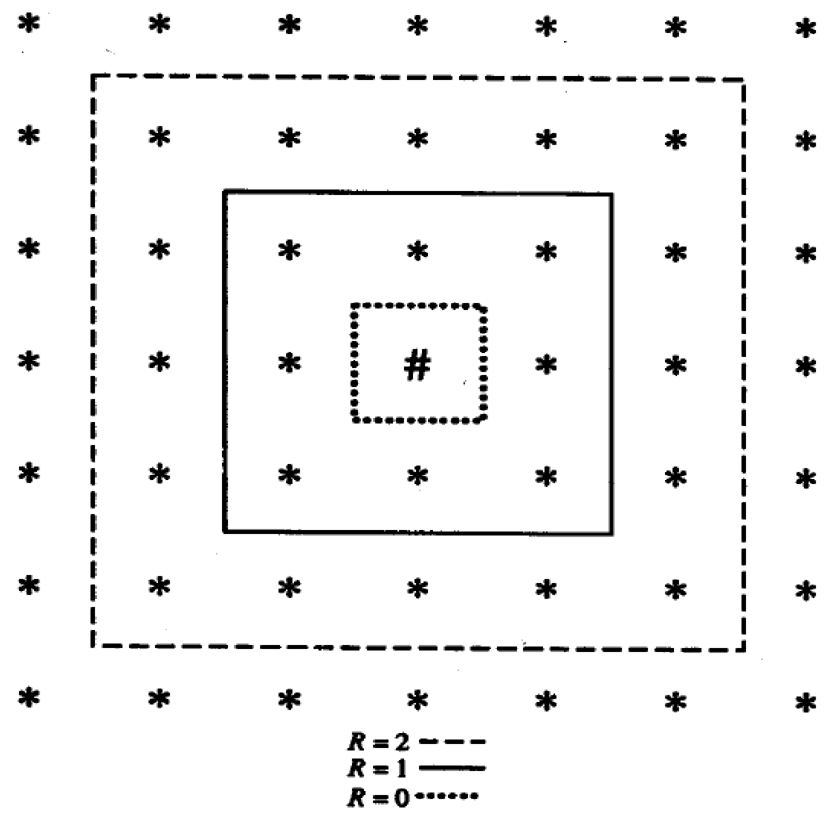


Figure 4.7 Neighborhoods for rectangular grid.

KOHONEN SOM (S

- Structure of Neighborhoods

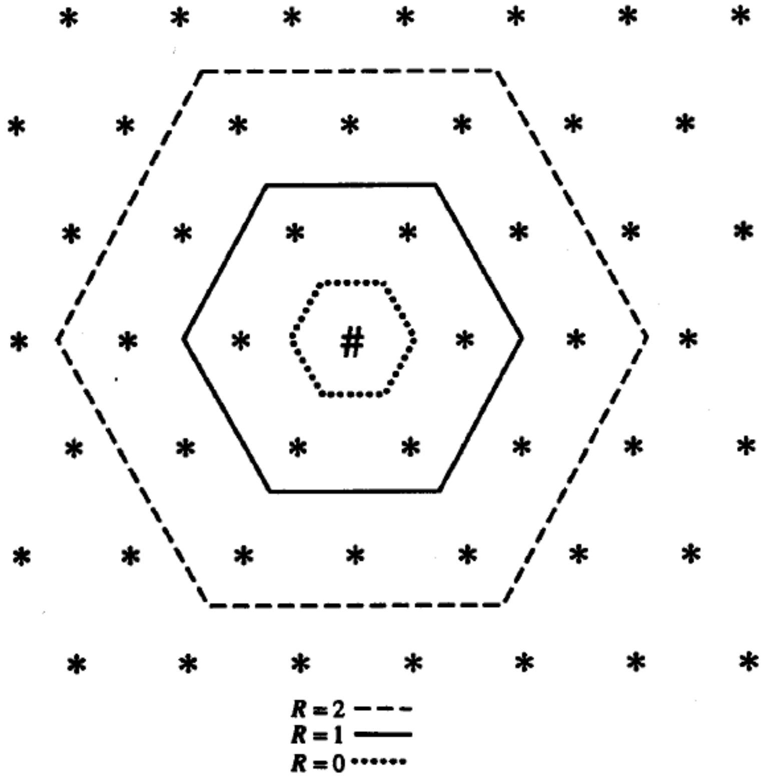


Figure 4.8 Neighborhoods for hexagonal grid.

KOHONEN SOM (S

KOHONEN SOM (SELF ORGANIZING MAPS)

- Neighborhoods do not wrap around from one

Step 0. Initialize weights w_{ij}. (Possible choices are discussed below.)
Set topological neighborhood parameters.
Set learning rate parameters.

Step 1. While stopping condition is false, do Steps 2–8.

Step 2. For each input vector **x**, do Steps 3–5.

Step 3. For each j, compute:

$$D(j) = \sum_i (w_{ij} - x_i)^2.$$

Step 4. Find index J such that $D(J)$ is a minimum.

Step 5. For all units j within a specified neighborhood of J, and for all i:

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

Step 6. Update learning rate.

- Algorithm:

Step 7. Reduce radius of topological neighborhood at specified times.

Step 8. Test stopping condition.

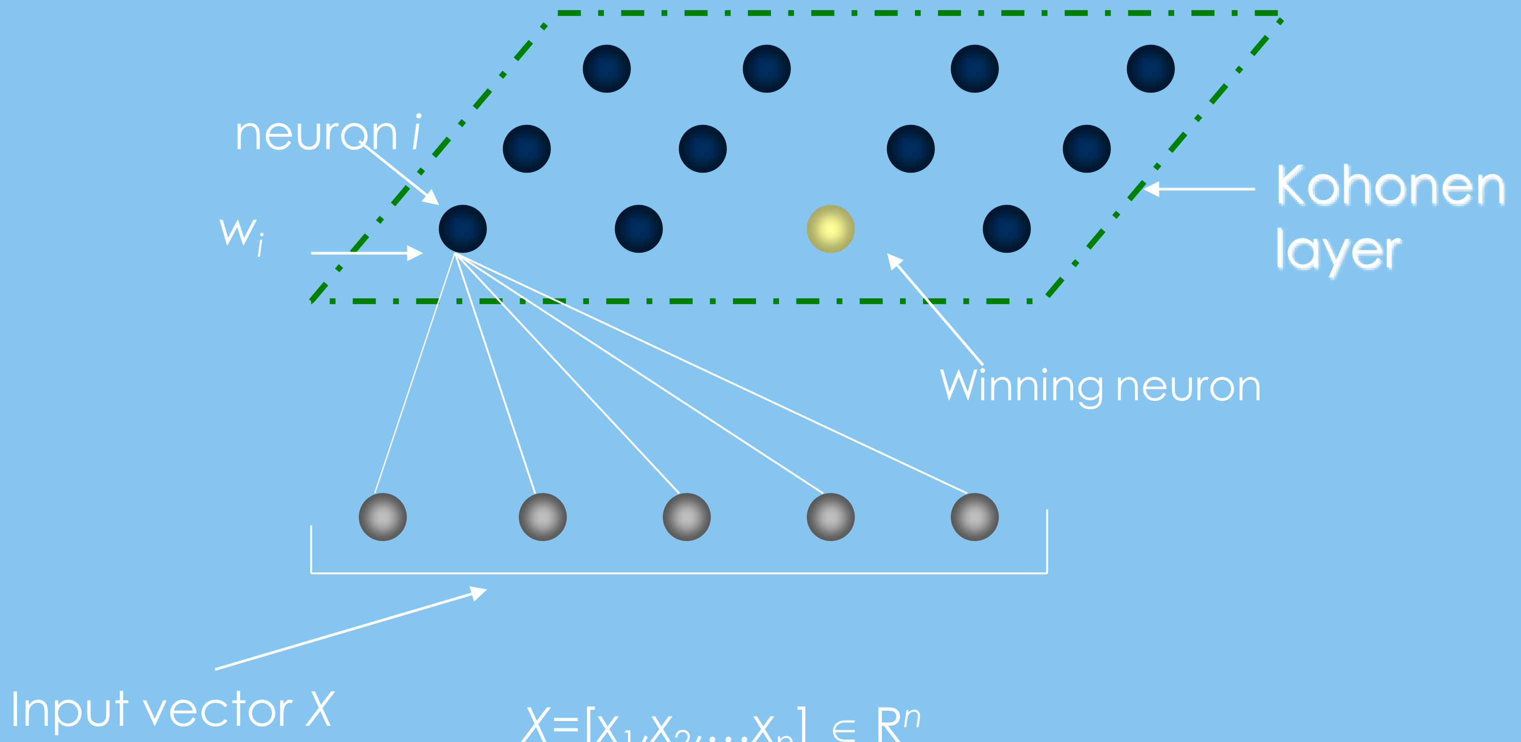
- Radius o
each ep

- Learning rate decrease may be either linear or geometric.

KOHONEN SOM (SELF ORGANIZING MAPS)

KOHONEN SELF ORGANIZING MAPS

Architecture



$X=[x_1,x_2,\ldots x_n] \in R^n$

$w_i=[w_{i1},w_{i2},\ldots,w_{in}] \in R^n$

- Example

Let the vectors to be clustered be

$$(1, 1, 0, 0); (0, 0, 0, 1); (1, 0, 0, 0); (0, 0, 1, 1).$$

The maximum number of clusters to be formed is

$$m = 2.$$

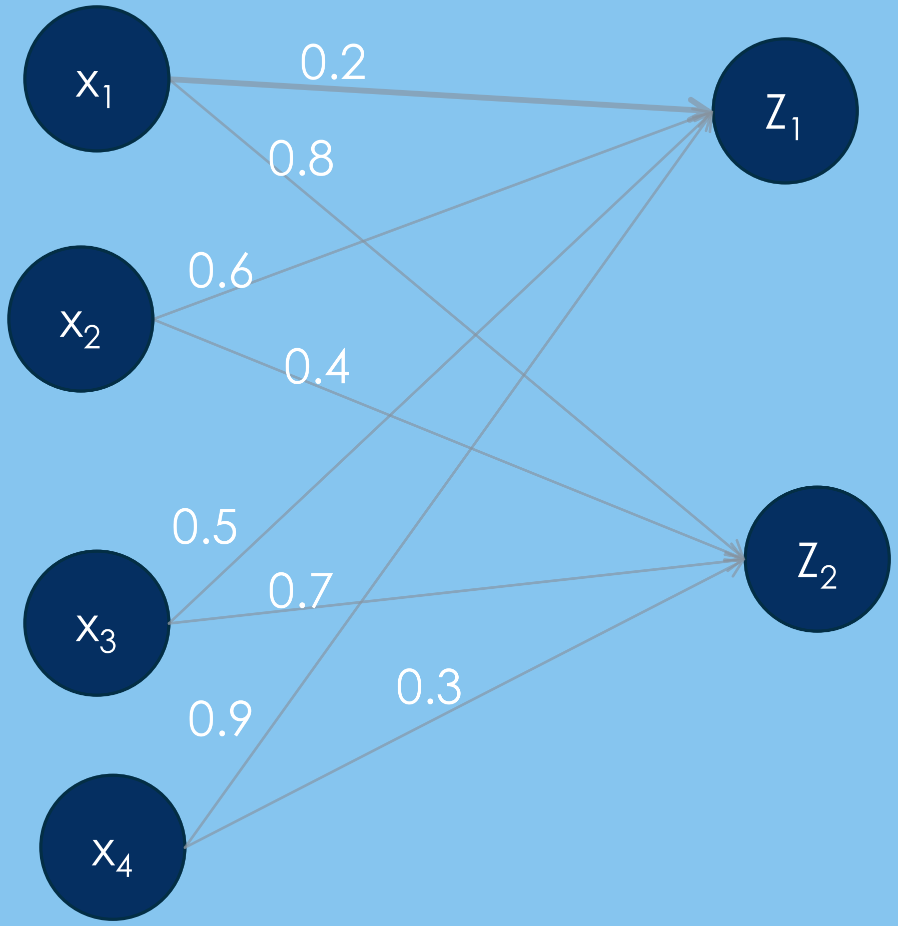
Suppose the learning rate (geometric decrease) is

$$\alpha(0) = .6,$$

$$\alpha(t + 1) = .5\, \alpha(t).$$

KOHO

KOHONEN SOM (SELF ORGANIZING MAPS)



KOHONEN SOM (SELF ORGANIZING MAPS)

With only two clusters available, the neighborhood of node J (Step 4) is set so that only one cluster updates its weights at each step (i.e., $R = 0$).

Step 0. Initial weight matrix:

$$\begin{bmatrix} .2 & .8 \\ .6 & .4 \\ .5 & .7 \\ .9 & .3 \end{bmatrix}.$$

Initial radius:

$$R = 0.$$

Initial learning rate:

$$\alpha(0) = 0.6.$$

Step 1. Begin training.

Step 2. For the first vector, (1, 1, 0, 0), do Steps 3–5.

Step 3.
$$D(1) = (.2 - 1)^2 + (.6 - 1)^2 + (.5 - 0)^2 + (.9 - 0)^2 = 1.86;$$
$$D(2) = (.8 - 1)^2 + (.4 - 1)^2 + (.7 - 0)^2 + (.3 - 0)^2 = 0.98.$$

Step 4. The input vector is closest to output node 2, so

$$J = 2.$$

Step 5. The weights on the winning unit are updated:

$$w_{i2}(\text{new}) = w_{i2}(\text{old}) + .6\,[x_i - w_{i2}(\text{old})]$$
$$= .4\,w_{i2}(\text{old}) + .6\,x_i.$$

This gives the weight matrix

$$\begin{bmatrix} .2 & .92 \\ .6 & .76 \\ .5 & .28 \\ .9 & .12 \end{bmatrix}.$$

Step 2. For the second vector, (0, 0, 0, 1), do Steps 3–5.

Step 3.

$$D(1) = (.2 - 0)^2 + (.6 - 0)^2 + (.5 - 0)^2 + (.9 - 1)^2 = 0.66;$$

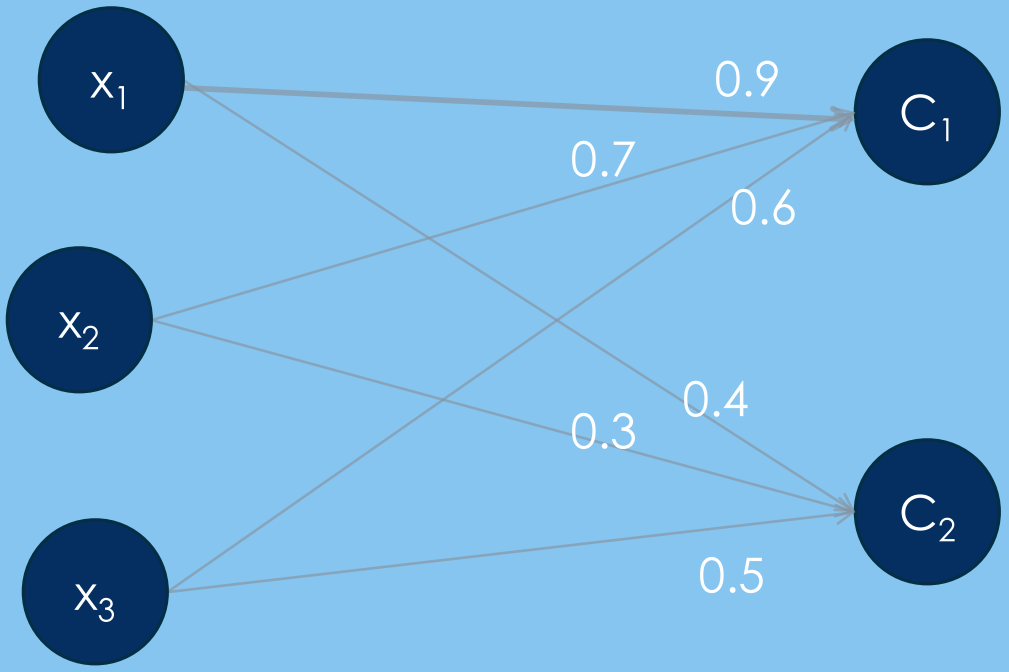
$$D(2) = (.92 - 0)^2 + (.76 - 0)^2 + (.28 - 0)^2 + (.12 - 1)^2 = 2.2768.$$

Step 4. The input vector is closest to output node 1, so $J = 1$.

KOHONE

KOHONEN SOM (SELF ORGANIZING MAPS)

The weight vector for the cluster unit are (0.9,0.7,0.6) and (0.4,0.2,0.1). Find the winning cluster for the input vector (0.4,0.2,0.1). Use learning rate of 0.2. Find the new weight for the winning unit.



$$D(j) = \sum (w_{ij} - x_i)^2$$

$$D(1) = (0.9 - 0.4)^2 + (0.7 - 0.2)^2 + (0.6 - 0.1)^2$$

$$D(2) = (0.4 - 0.4)^2 + (0.3 - 0.2)^2 + (0.5 - 0.1)^2$$

$$w_{ij}(new) = w_{ij}(old) + \alpha * (x_i - w_{ij}(old))$$
$$w_{12}(new) = 0.4 + 0.2 * (0.4 - 0.4)$$
$$w_{22}(new) = 0.3 + 0.2 * (0.2 - 0.3)$$
$$w_{32}(new) = 0.5 + 0.2 * (0.1 - 0.5)$$

$$w = \begin{bmatrix} 0.9 & 0.4 \\ 0.7 & 0.28 \\ 0.6 & 0.42 \end{bmatrix}$$

KOHONEN SOM (SELF ORGANIZING MAPS)

$X=[0.3,0.4]$
$a=0.3$