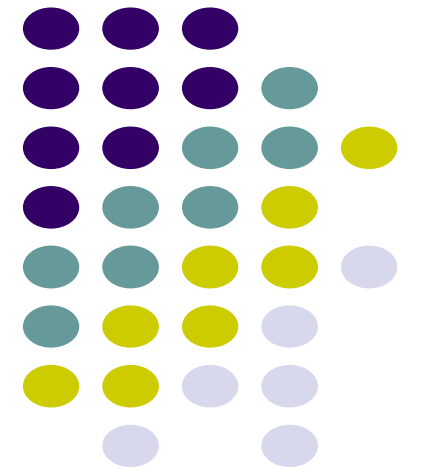
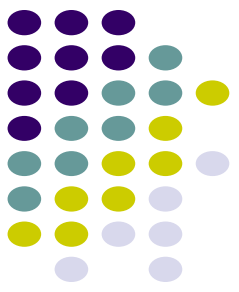


# Neural Networks

---



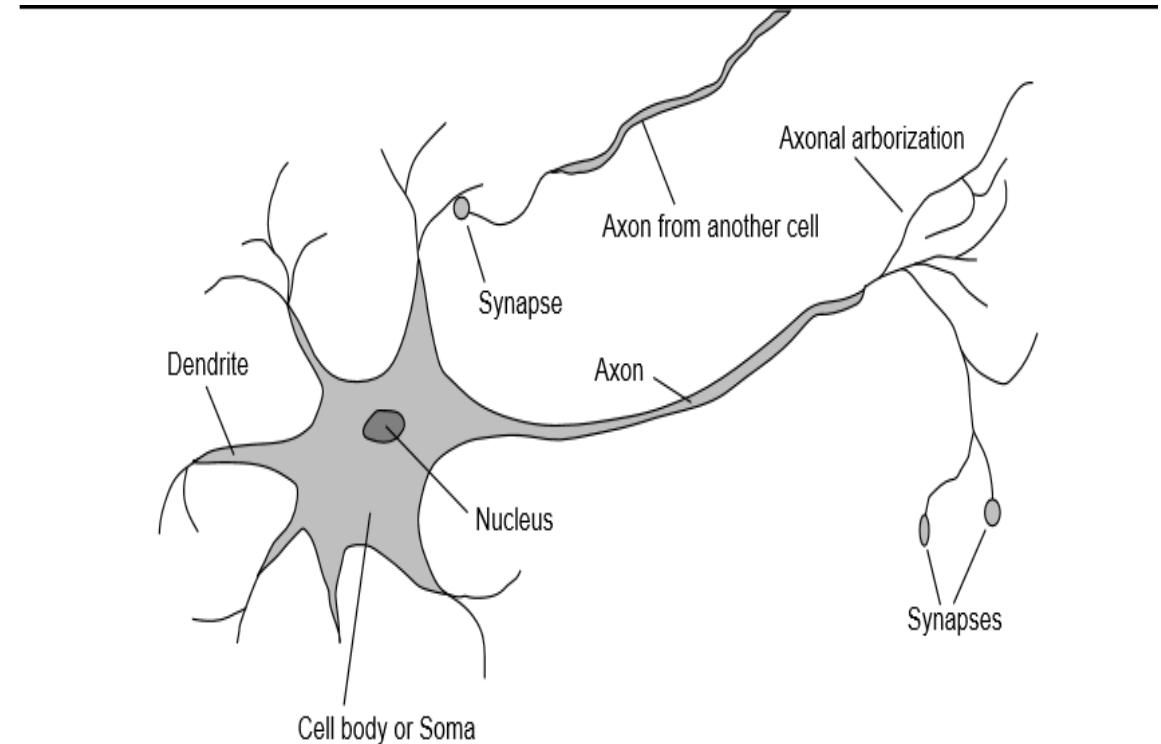


# Biological neuron model

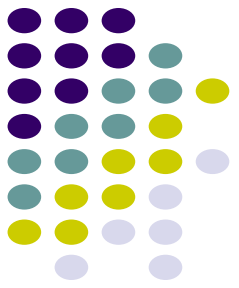
The brain consists of a densely interconnected set of nerve cells, or basic information-processing units, called neurons.

A neuron consists of:

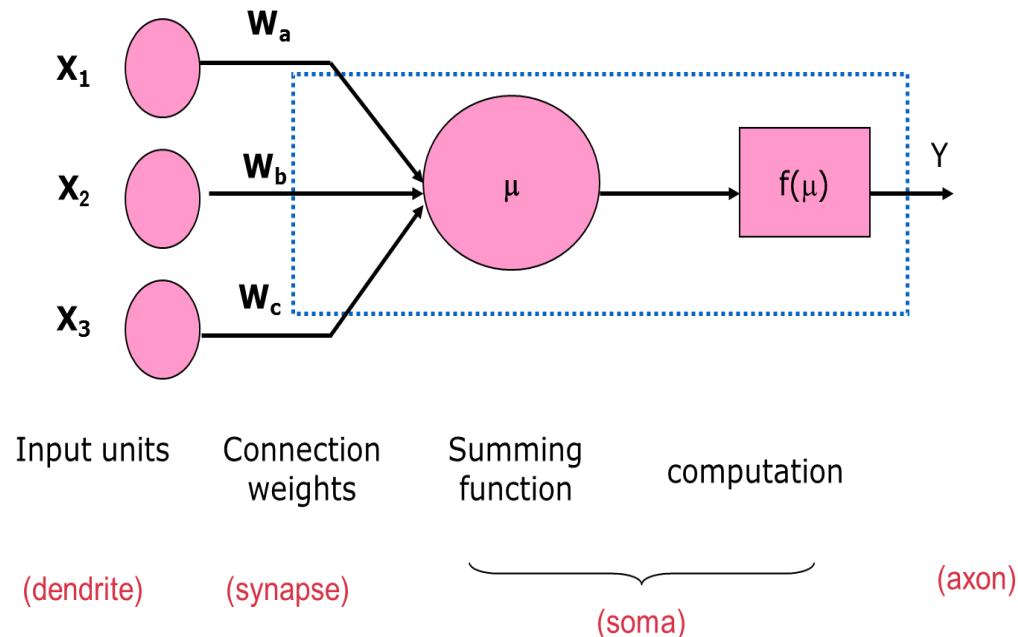
- **Dendrites:** They are tree-like branches, responsible for receiving the information from other neurons it is connected to. In other sense, we can say that they are like the ears of neuron.
- **Soma:** It is the cell body of the neuron and is responsible for processing of information, they have received from dendrites.
- **Axon:** It is just like a cable through which neurons send the information
- **Synapses:** It is the connection between the axon and other neuron dendrites.



# How working of the brain inspires?



- Inputs are received on dendrites, and if the input levels are over a threshold, the neuron fires, passing a signal through the axon to the synapse which then connects to another neuron.

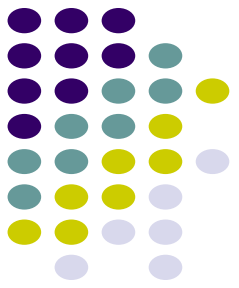


- Human brain contains approx  $10^{11}$  neurons and approx.  $10^{14}$  connections

Brain	Neural Network
Neuron	Node
Connection of neurons	Connection weight

**Information processing model that is inspired by the way biological nervous system (i.e) the brain, process information.**

# Artificial Neural Networks



*Nodes* – interconnected processing elements (units or neurons)

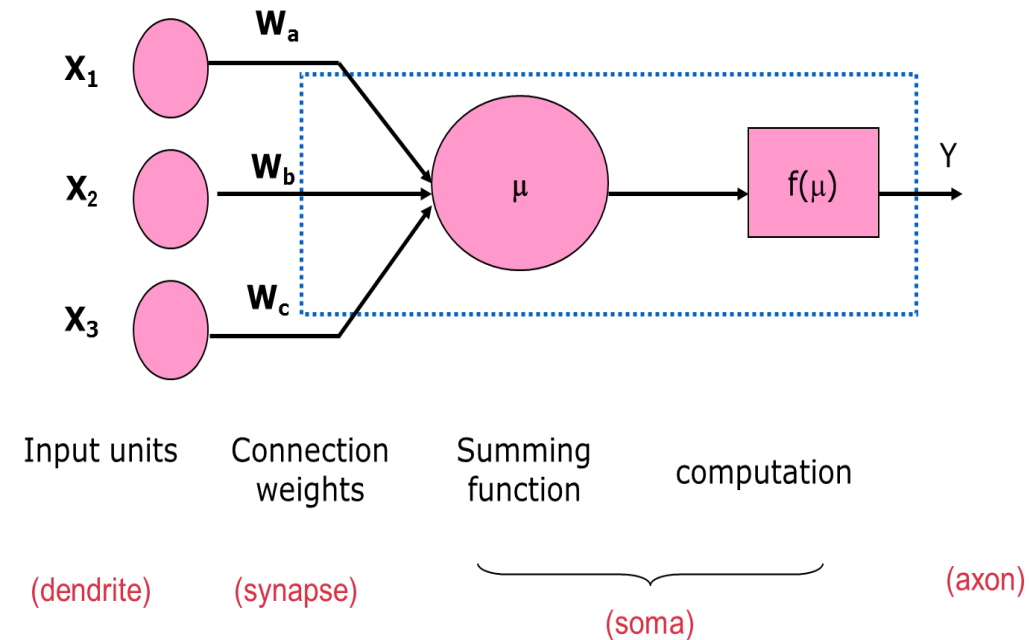
Neuron is connected to other by a *connection link*.

Each connection link is associated with *weight* which has information about the input signal.

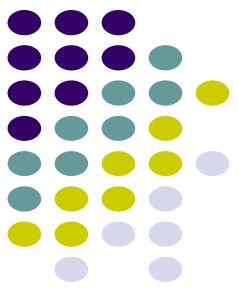
ANN processing elements are called as *neurons* or *artificial neurons*, since they have the capability to model networks of original neurons as found in brain.

Internal state of neuron is called *activation* or *activity level* of neuron, which is the function of the inputs the neurons receives.

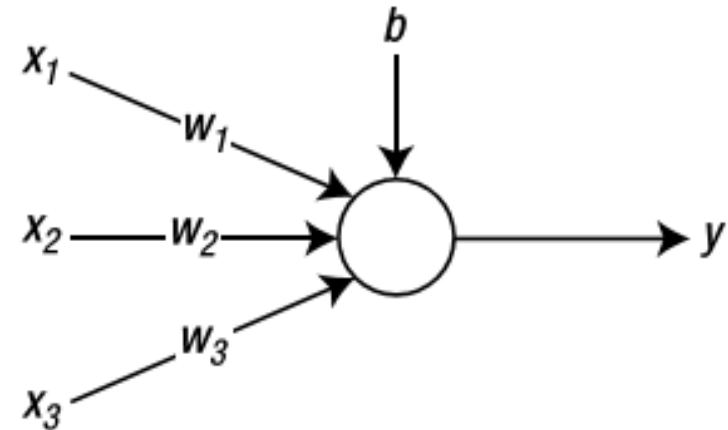
Neuron can send only one signal at a time.

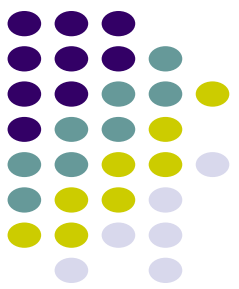


# Artificial Neuron



- The circle and arrow of the figure denote the node and signal flow, respectively.
- $x_1$ ,  $x_2$ , and  $x_3$  are the input signals.
- $w_1$ ,  $w_2$ , and  $w_3$  are the weights for the corresponding signals.
- Lastly,  $b$  is the bias, which is another factor associated with the storage of information.
- In other words, the information of the neural net is stored in the form of weights and bias.



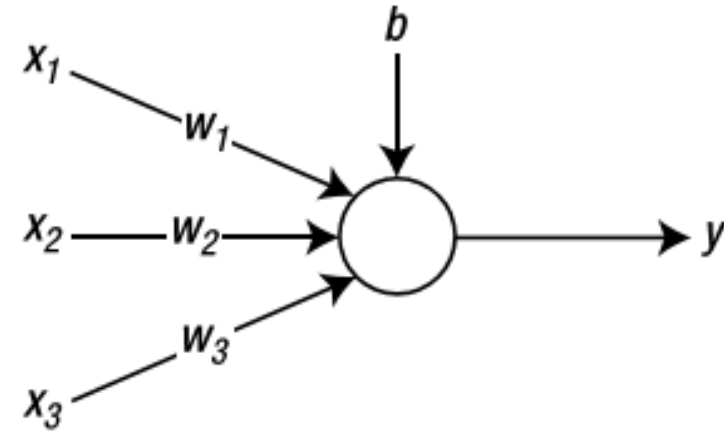


The weighted sum of the input signals is calculated.

$$\begin{aligned}v &= w_1x_1 + w_2x_2 + w_3x_3 + b \\ &= wx + b\end{aligned}$$

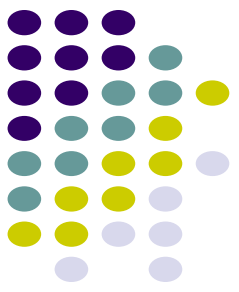
The output from the activation function to the weighted sum is passed outside.

$$y = \varphi(v) = \varphi(wx + b)$$

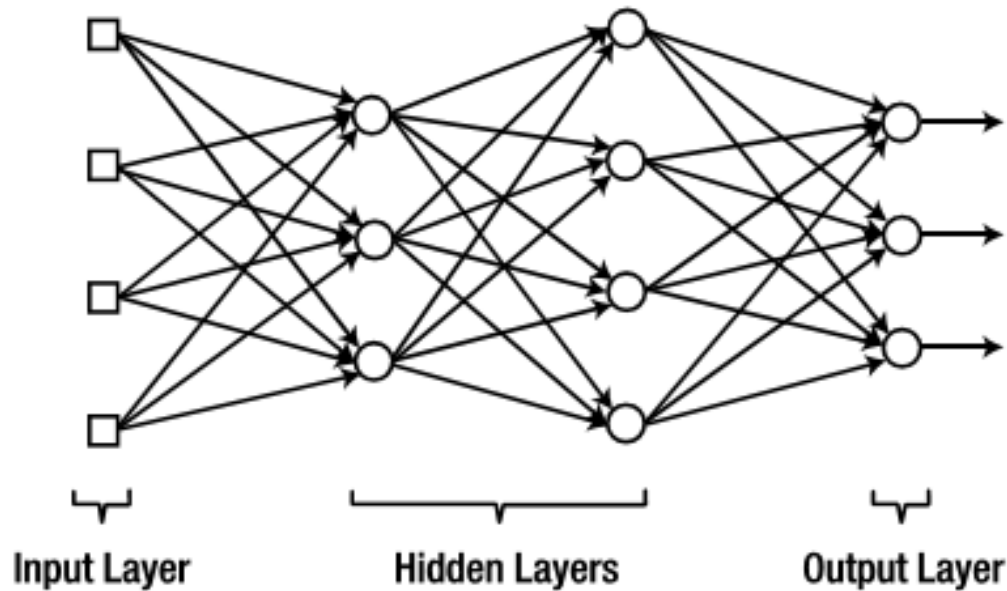


$$w = [w_1 \quad w_2 \quad w_3] \quad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

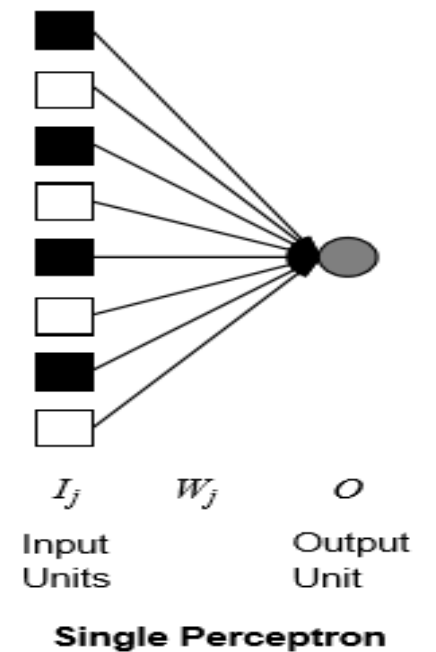
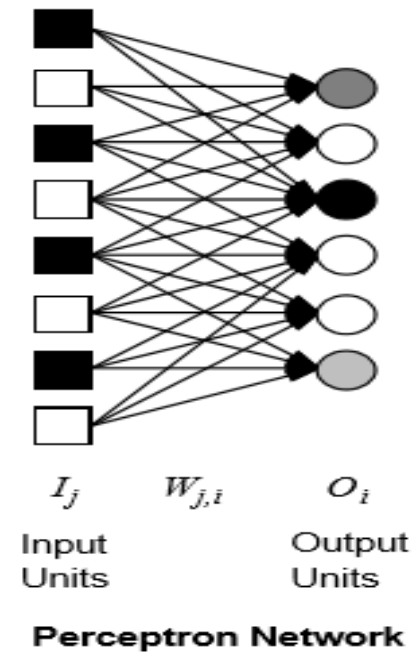
# Layers of Neural Network

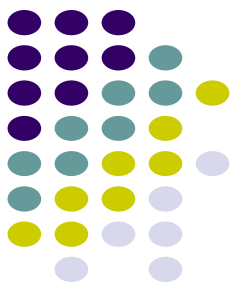


- As the brain is a gigantic network of the neurons, the neural network is a network of nodes



- Introduced in the late 50s – Minsky and Papert.
- A single layer perceptron (SLP) is the simplest feed-forward artificial neural network that can learn to classify any linearly separable set of inputs

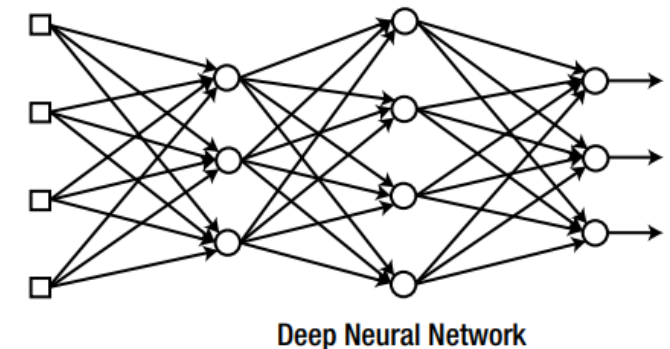
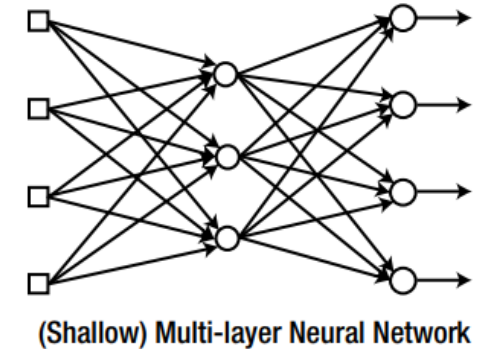
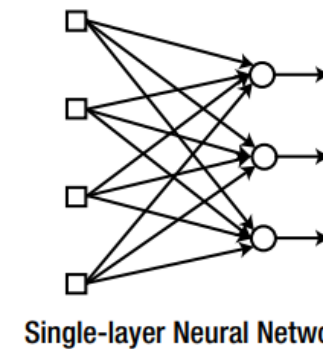




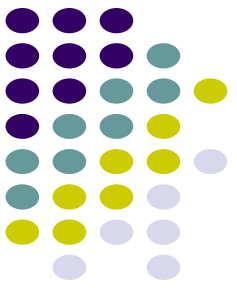
# Network Architectures

- This describes the set of connections between the neurons, the number of layers, and the number of neurons in each layer.

Single-Layer Neural Network		Input Layer - Output Layer
Multi-Layer Neural Network	Shallow Neural Network	Input Layer - Hidden Layer - Output Layer
	Deep Neural Network	Input Layer - Hidden Layers - Output Layers

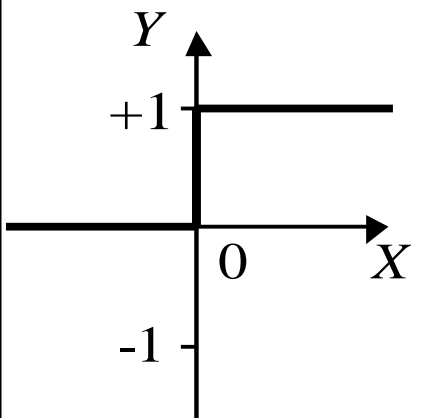
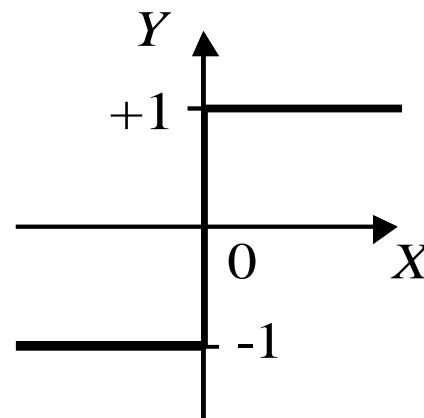
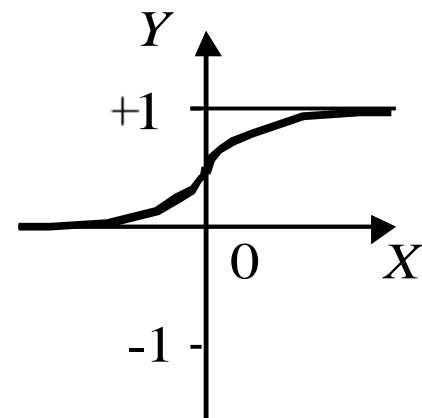
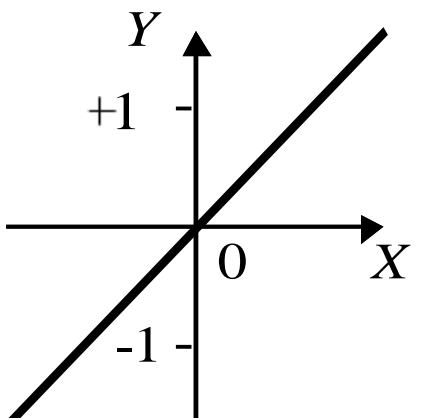




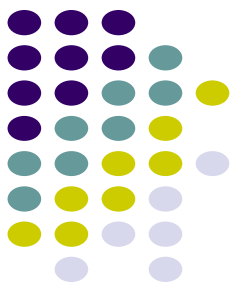


# Activation Functions

- Activation function is applied over the net input to calculate the output of an ANN.
- Information processing of processing element has two major parts: input and output.

<i>Step function</i>	<i>Sign function</i>	<i>Sigmoid function</i>	<i>Linear function</i>
			
$Y^{step} = \begin{cases} 1, & \text{if } X \geq 0 \\ 0, & \text{if } X < 0 \end{cases}$	$Y^{sign} = \begin{cases} +1, & \text{if } X \geq 0 \\ -1, & \text{if } X < 0 \end{cases}$	$Y^{sigmoid} = \frac{1}{1 + e^{-X}}$	$Y^{linear} = X$

# Activation Functions



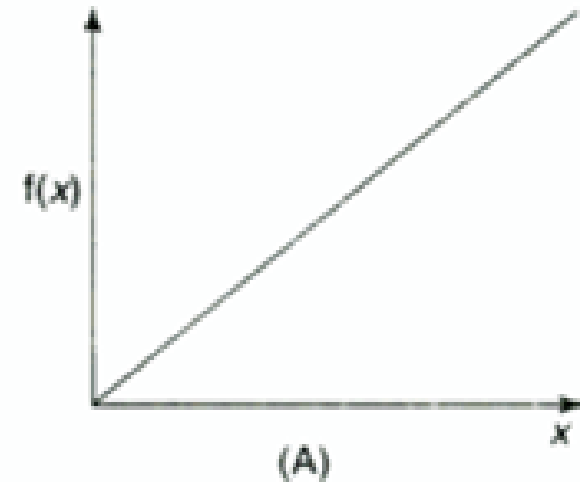
- Activation function is applied over the net input to calculate the output of an ANN.
- Information processing of processing element has two major parts: input and output.

## 1. Identity function:

- It is a linear function which is defined as

$$f(x) = x \text{ for all } x$$

- The output is same as the input.





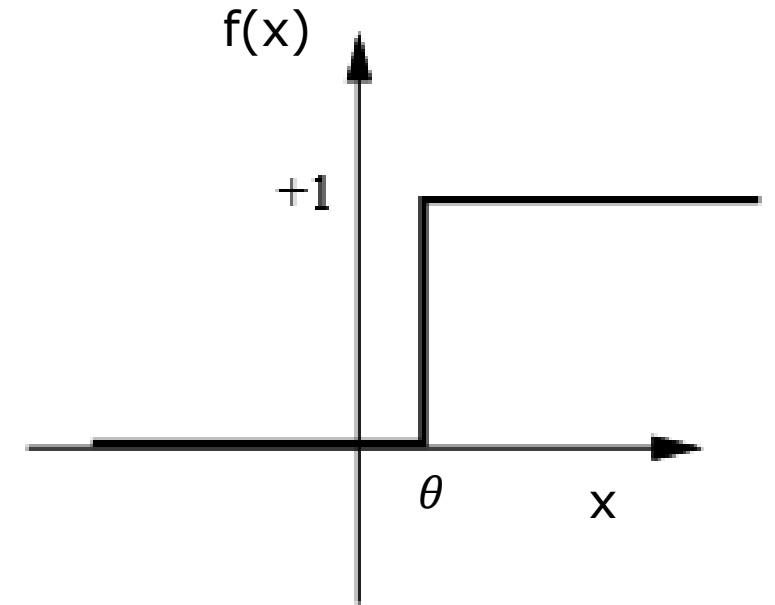
## Binary step function

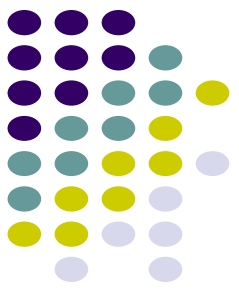
It is defined as

$$\begin{aligned} f(x) &= 1 \text{ if } x \geq \theta \\ &= 0 \text{ if } x < \theta \end{aligned}$$

where  $\theta$  represents thresh hold value.

It is used in single layer nets to convert the net input to an output that is binary.





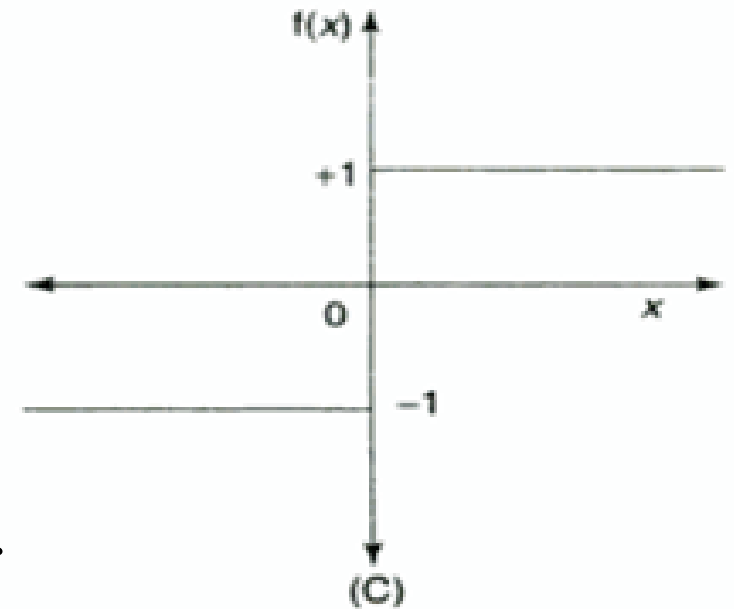
### 3. Bipolar step function:

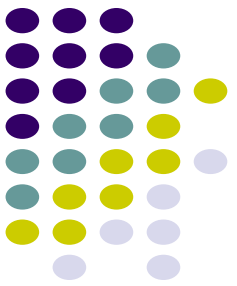
It is defined as

$$f(x) = 1 \text{ if } x \geq \theta$$
$$= -1 \text{ if } x < \theta$$

where  $\theta$  represents threshold value.

- It is used in single layer nets to convert the net input to an output that is bipolar (+1 or -1).





#### 4. Sigmoid function:

It is defined as

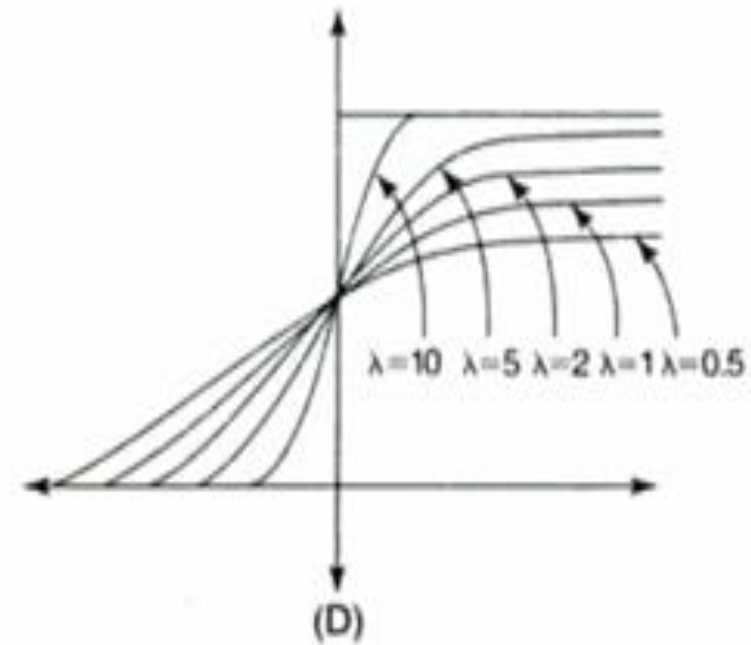
$$f(x) = \frac{1}{1 + e^{-\lambda x}}$$

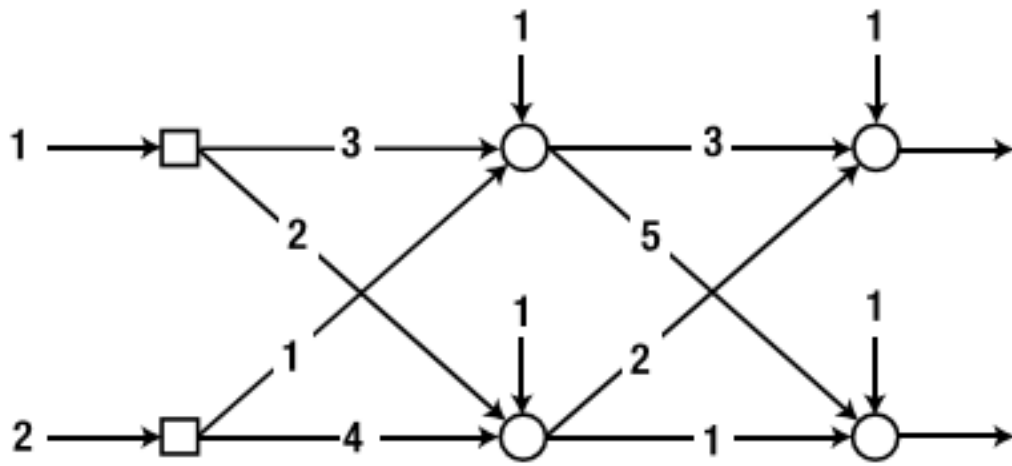
where  $\lambda$  – steepness parameter.

The derivative of this function is

$$f'(x) = \lambda f(x)[1-f(x)].$$

The range of sigmoid function is 0 to 1.

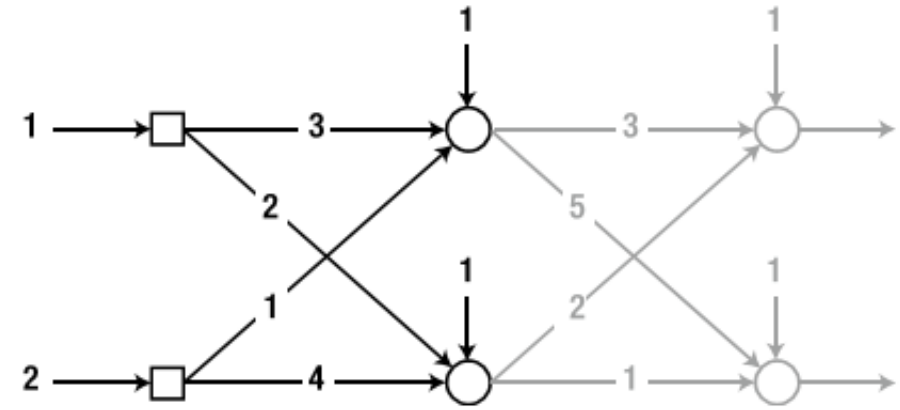




The activation function of each node is assumed to be a linear function

$$\varphi(x) = x$$

$$v = \begin{bmatrix} 3 \times 1 + 1 \times 2 + 1 \\ 2 \times 1 + 4 \times 2 + 1 \end{bmatrix} = \begin{bmatrix} 3 & 1 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 6 \\ 11 \end{bmatrix}$$



The first node of the hidden layer calculates the output as:

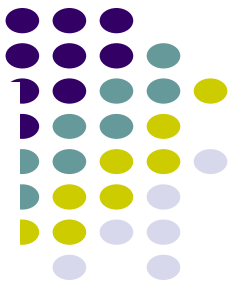
$$\text{Weighted sum: } v = (3 \times 1) + (1 \times 2) + 1 = 6$$

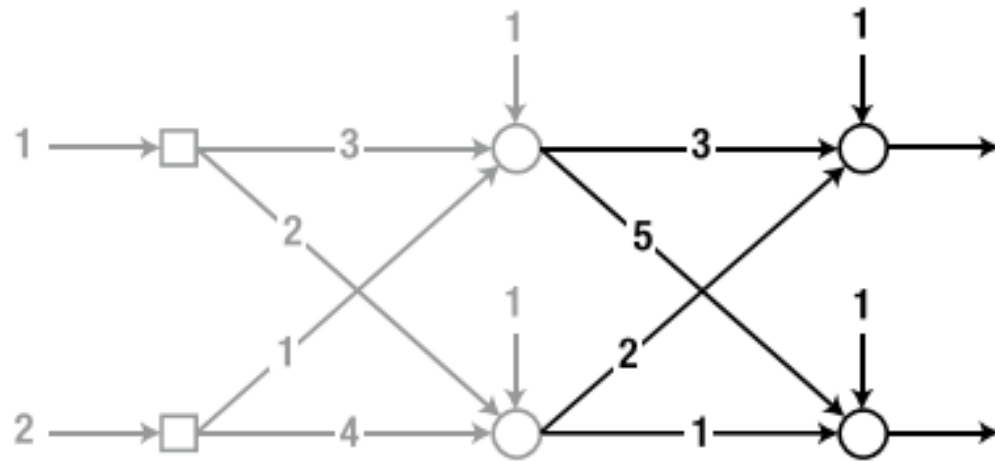
$$\text{Output: } y = \varphi(v) = v = 6$$

In a similar manner, the second node of the hidden layer calculates the output as:

$$\text{Weighted sum: } v = (2 \times 1) + (4 \times 2) + 1 = 11$$

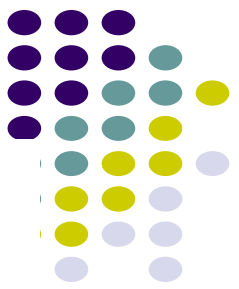
$$\text{Output: } y = \varphi(v) = v = 11$$



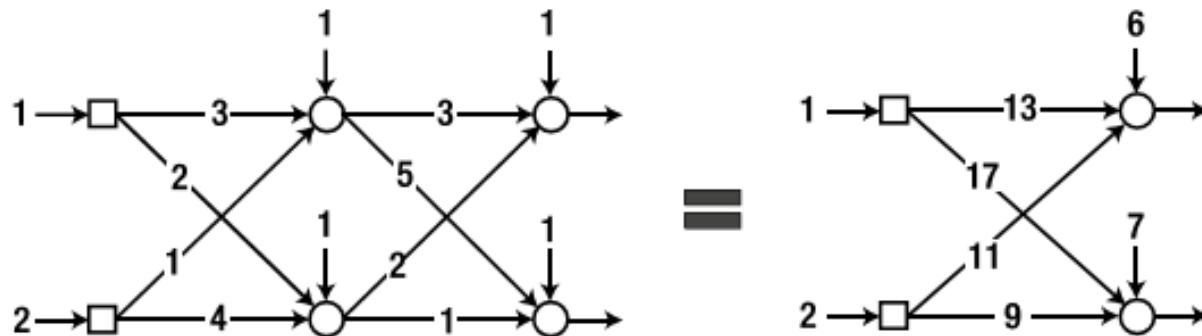


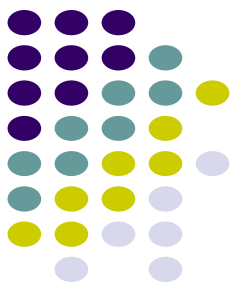
$$\text{Weighted sum: } v = \begin{bmatrix} 3 & 2 \\ 5 & 1 \end{bmatrix} \begin{bmatrix} 6 \\ 11 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 41 \\ 42 \end{bmatrix}$$

$$\text{Output: } y = \varphi(v) = v = \begin{bmatrix} 41 \\ 42 \end{bmatrix}$$



- Effect of linear activation function





# Training Method or Learning :

Training Method	Training Data
Supervised Learning	{ input, correct output }
Unsupervised Learning	{ input }
Reinforced Learning	{ input, some output, grade for this output }

Supervised Learning:

**Application:** Recognizing hand-written digits, pattern recognition and etc.

**Examples:** Perceptron, feed-forward neural network, radial basis function

Reinforcement Learning:

**Application:** Text Prediction

**Examples:** Recurrent neural networks

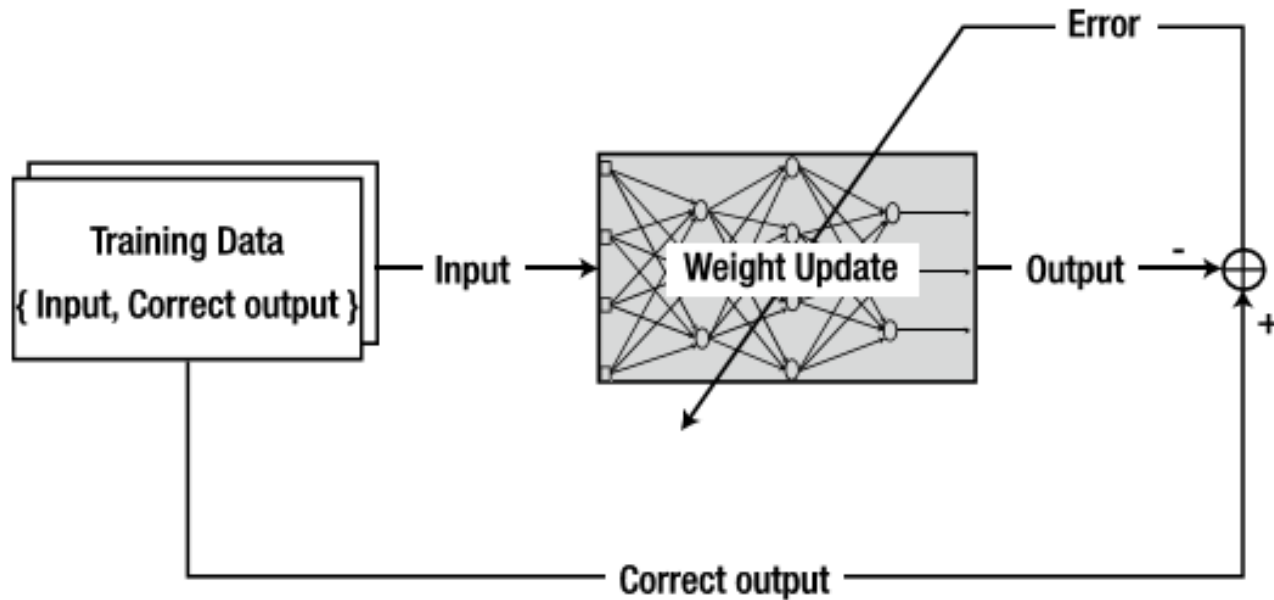
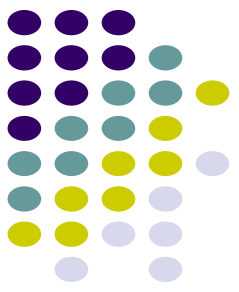
Unsupervised Learning:

**Application:** Clustering

**Examples:** Kohonen, SOM , Hopfield networks.

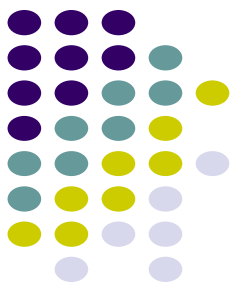


# Supervised Learning

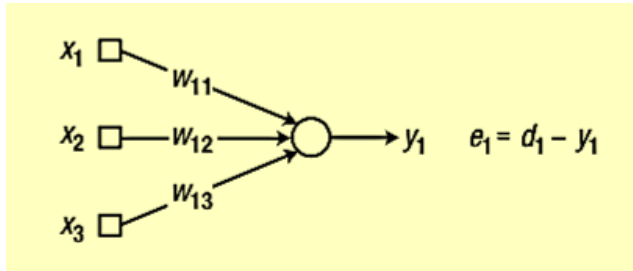


1. Initialize the weights with adequate values.
2. Take the "input" from the training data, which is formatted as { input, correct output }, and enter it into the neural network. Obtain the output from the neural network and calculate the error from the correct output.
3. Adjust the weights to reduce the error.
4. Repeat Steps 2-3 for all training data

# Training of a Single-Layer Neural Network: Delta Rule



- It's a single-unit network.
- Let  $d_i$  is the correct output of the output node  $i$
- Change the weight by an amount proportional to the difference between the desired output and the actual output



where

$x_j$  = The output from the input node  $j$ , ( $j=1, 2, 3$ )

$e_i$  = The error of the output node  $i$

$w_{ij}$  = The weight between the output node  $i$  and input node  $j$

$\alpha$  = Learning rate ( $0 < \alpha \leq 1$ )

## Delta Rule

1. Initialize the weights at adequate values.
2. Take the “input” from the training data of { input, correct output } and enter it to the neural network. Calculate the error of the output,  $y_i$ , from the correct output,  $d_i$ , to the input.

$$e_i = d_i - y_i$$

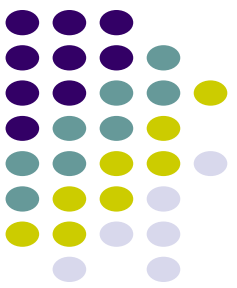
3. Calculate the weight updates according to the following delta rule:

$$\Delta w_{ij} = \alpha e_i x_j$$

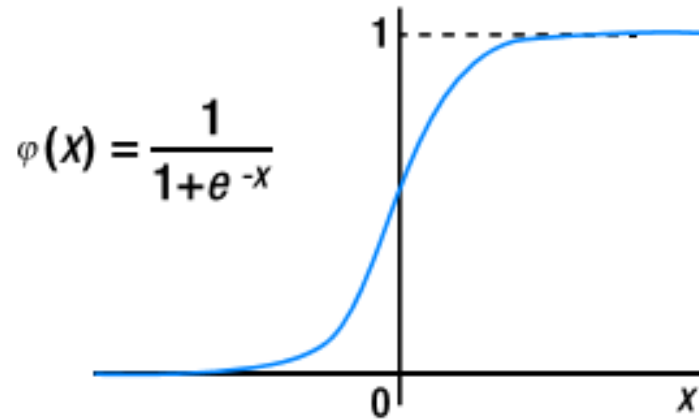
4. Adjust the weights as:

$$w_{ij} \leftarrow w_{ij} + \alpha e_i x_j$$

5. Perform Steps 2-4 for all training data.
6. Repeat Steps 2-5 until the error reaches an acceptable tolerance level.



We need the derivative of this function, which is given as:



$$\varphi'(x) = \varphi(x)(1 - \varphi(x))$$



$$\delta_i = \varphi'(v_i)e_i = \varphi(v_i)(1 - \varphi(v_i))e_i$$



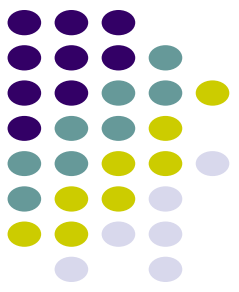
$$w_{ij} \leftarrow w_{ij} + \alpha \varphi(v_i)(1 - \varphi(v_i))e_i x_j$$

$$\delta_i = \varphi'(v_i)e_i$$

$$w_{ij} \leftarrow w_{ij} + \alpha \delta_i x_j$$

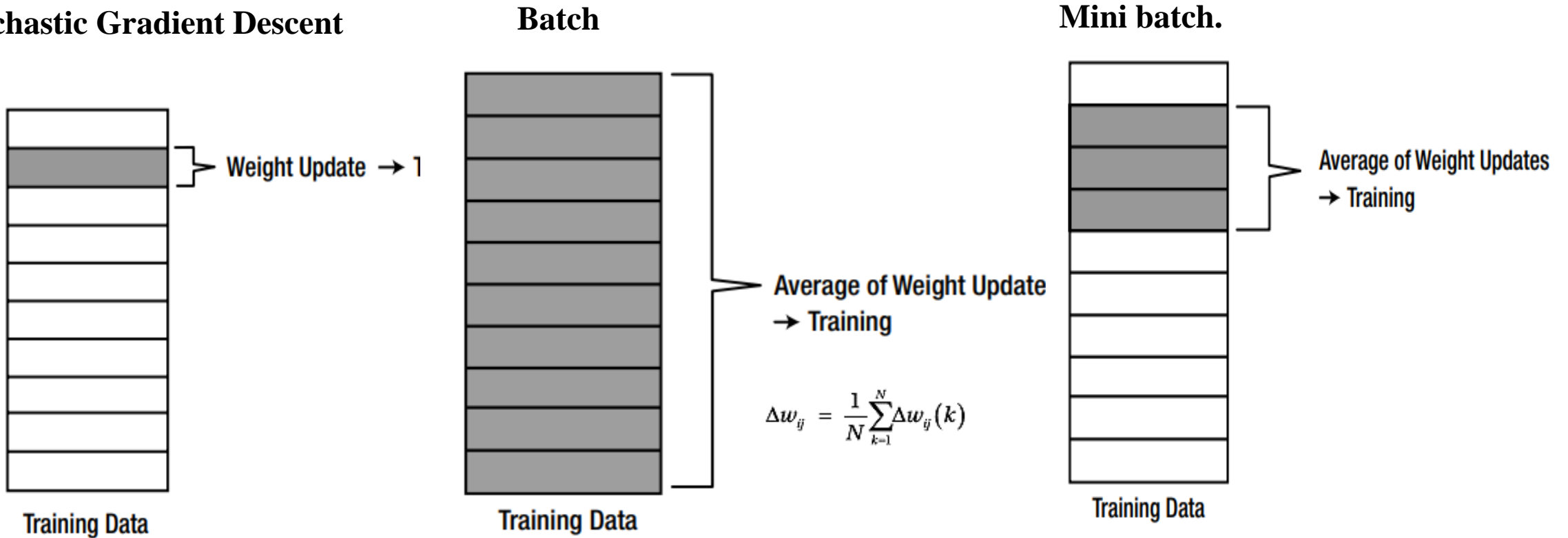
$v_i$  = The weighted sum of the output node  $i$

$\varphi'$  = The derivative of the activation function  $\varphi$  of the output node  $i$

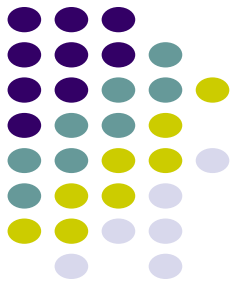


# Error Calculations in ANN

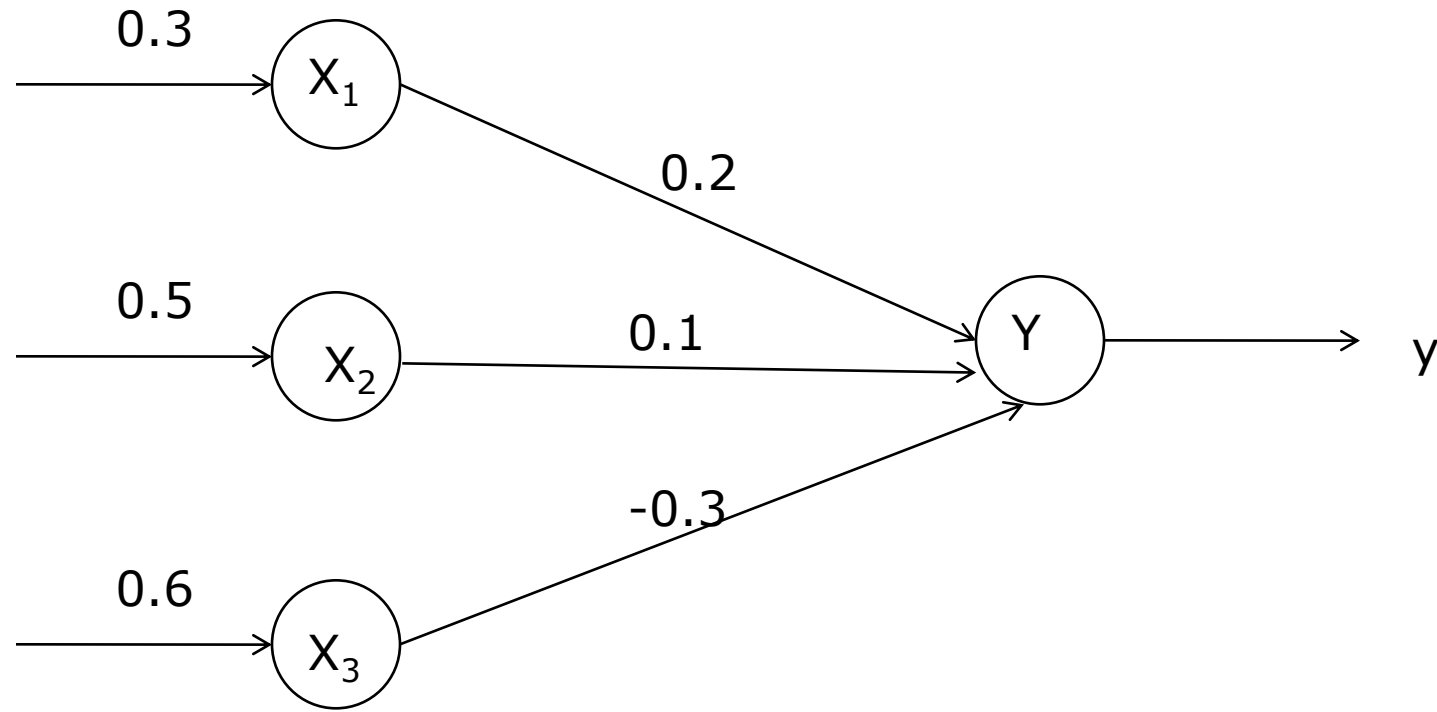
- Stochastic Gradient Descent

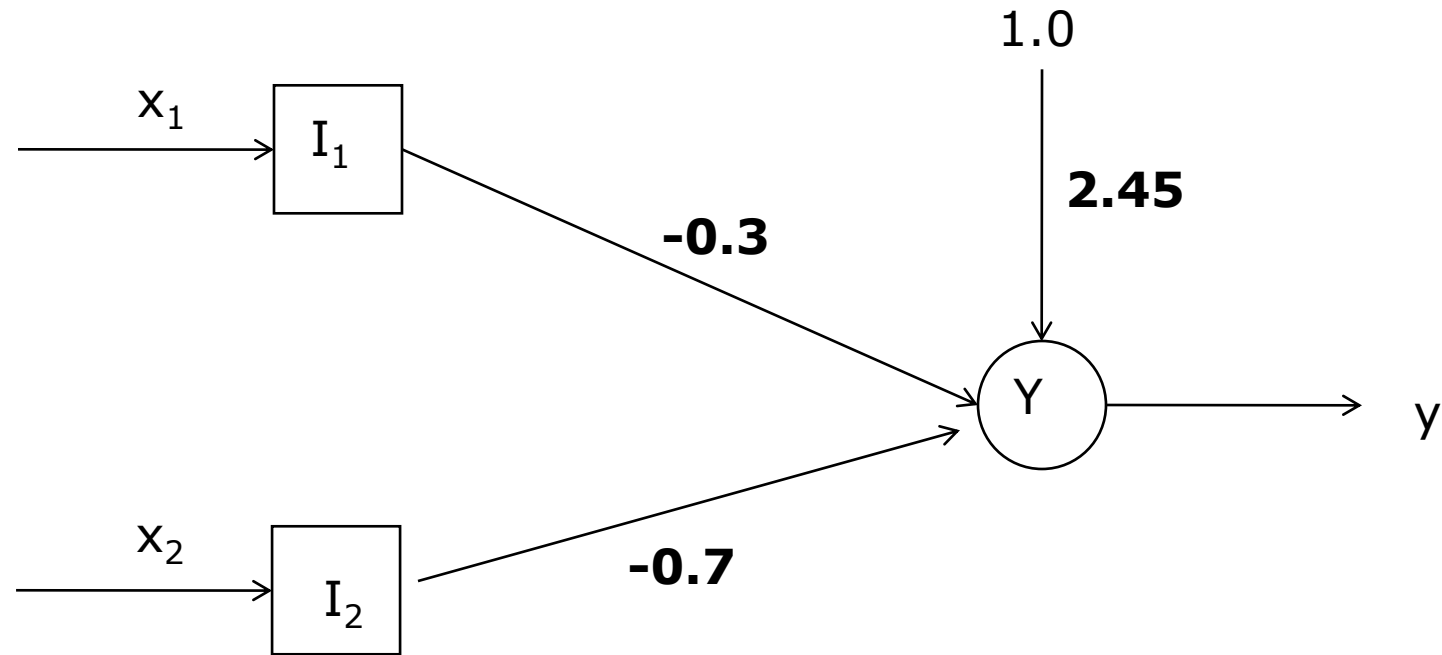
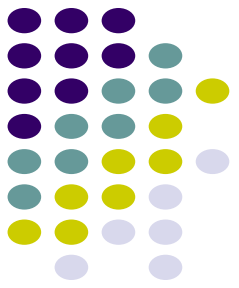


# Examples

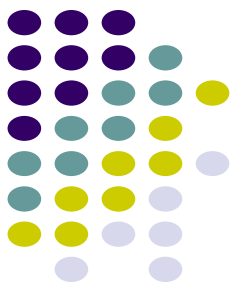


- Calculate the net input to the output neuron





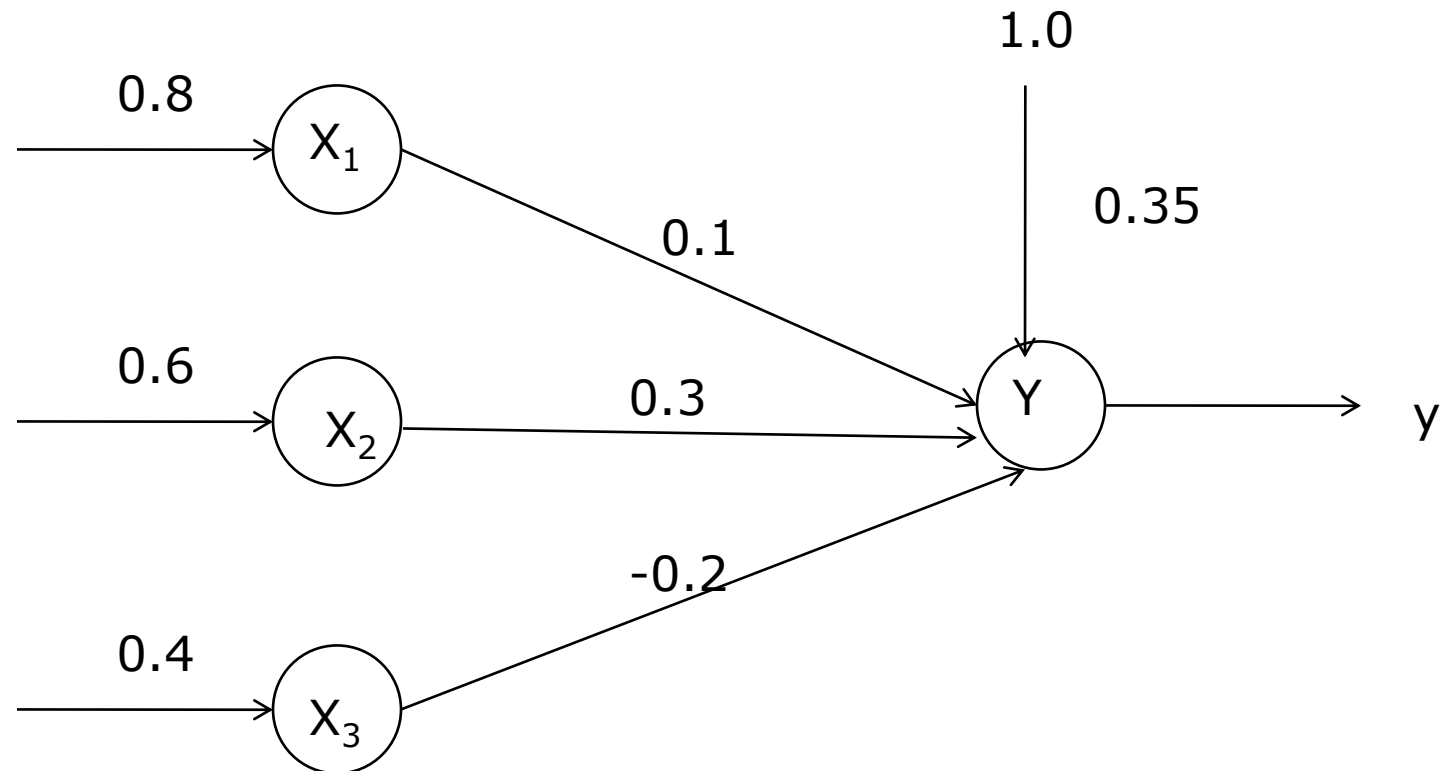
**Calculate the output of the neuron Y using activation function**



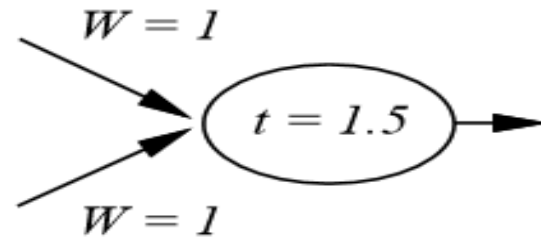
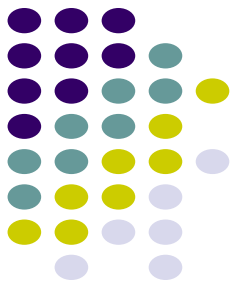
**1) Step function with threshold 0.5**

**2) Sign function with threshold 0.5**

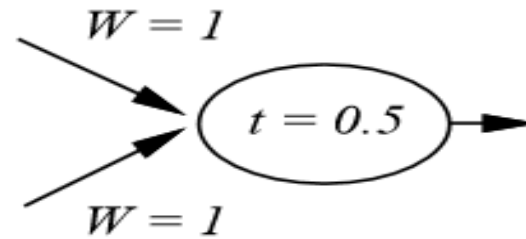
**3) Linear function**



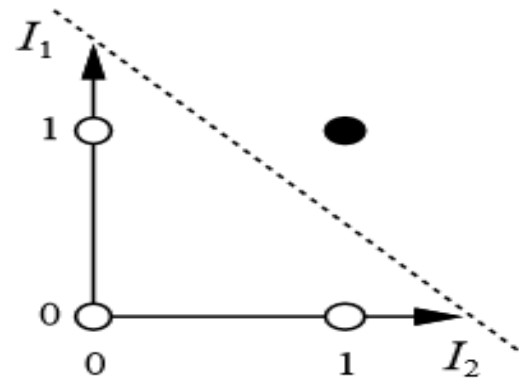
# Implementing AND and OR Logic function



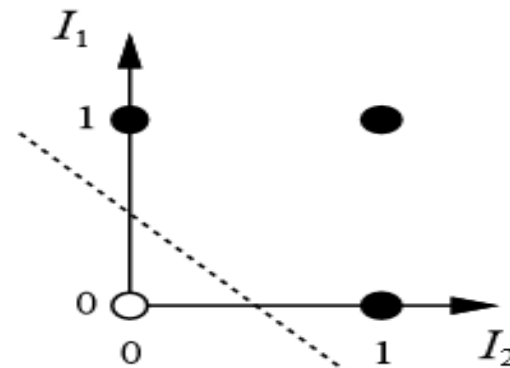
**AND**



**OR**

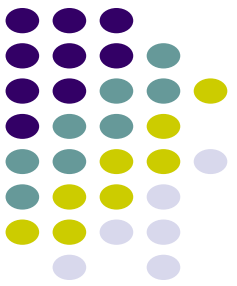


**(a)**  $I_1$  and  $I_2$



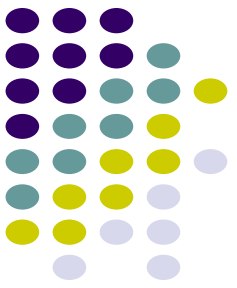
**(b)**  $I_1$  or  $I_2$





# Learning of Perceptron

- Training set  $S$  of examples  $\{\mathbf{x}, \mathbf{t}\}$ 
  - $\mathbf{x}$  is an input vector and
  - $\mathbf{t}$  the desired target vector
  - Example: Logical OR  
 $S = \{(0,0),0\}, \{(0,1),1\}, \{(1,0),1\}, \{(1,1),1\}$
- Iterative process
  - Present a training example  $x$  , compute network output  $y$  , compare output  $y$  with target  $t$ , adjust weights and thresholds



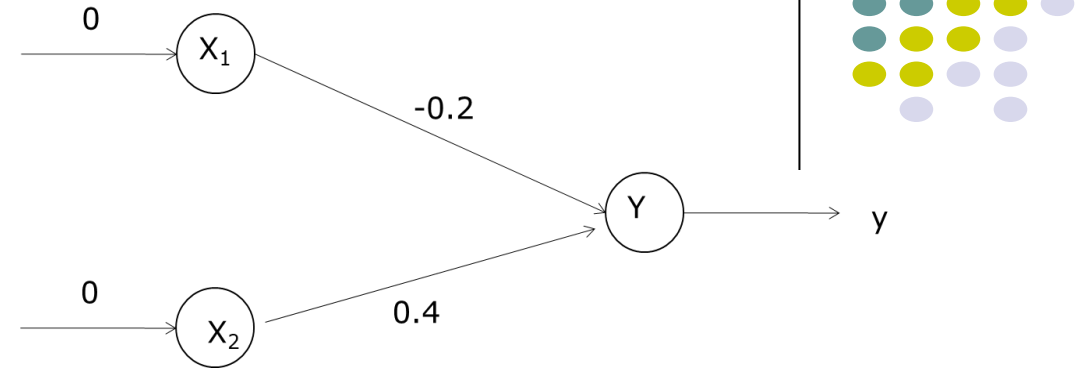
# Perceptron Learning Rule

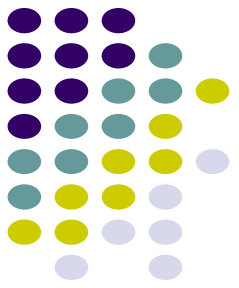
$$\mathbf{w}' = \mathbf{w} + \alpha (t - y) \mathbf{x}$$

- The parameter  $\alpha$  is called the *learning rate*.
- If the output is correct ( $t=y$ ) the weights are not changed ( $\Delta w_i = 0$ ).
- If the output is incorrect ( $t \neq y$ ) the weights  $w_i$  are changed.

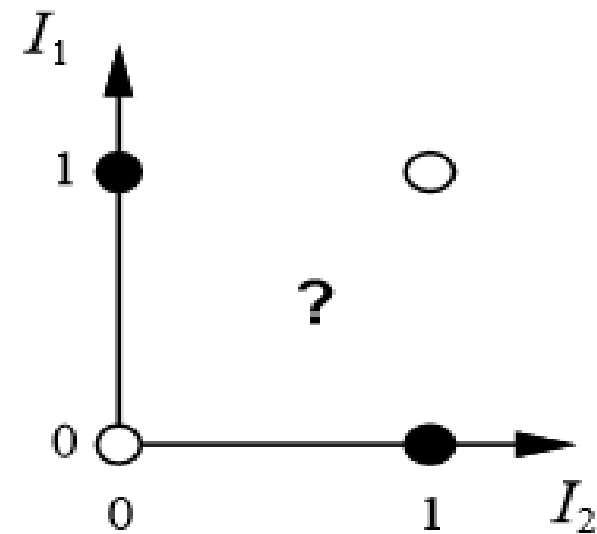
# Implementing OR

- Initialize  $W$ :
- $Y = \text{Step}(W_1 * X_1 + W_2 * X_2) = 0$ ; OK
- $X_1 = 0$  and  $X_2 = 1$  and actual output = 1
- $Y = \text{Step}(W_1 * X_1 + W_2 * X_2) = \text{Step}(0.4) = 1$ ; OK
- $X_1 = 1$  and  $X_2 = 0$  and actual output = 1
- $Y = \text{Step}(W_1 * X_1 + W_2 * X_2) = \text{Step}(-0.2) = 0$ ; change  $e = 1 - 0 = 1$
- $\eta = 0.2$ ,  $\Delta W_i = \eta * X_i * e$ ,  $W_1 = (-0.2 + (0.2 * 1 * 1)) = 0$ , ,
- $W_2 = (0.4 + (0.2 * 0 * 1)) = 0.4$ ;
- $X_1 = 1$  and  $X_2 = 1$  and actual output = 1
- $Y = \text{Step}(W_1 * X_1 + W_2 * X_2) = \text{Step}(0.4) = 1$ ; OK



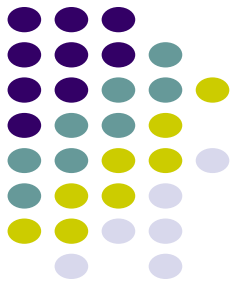


**Is this problem  
linearly separable?**

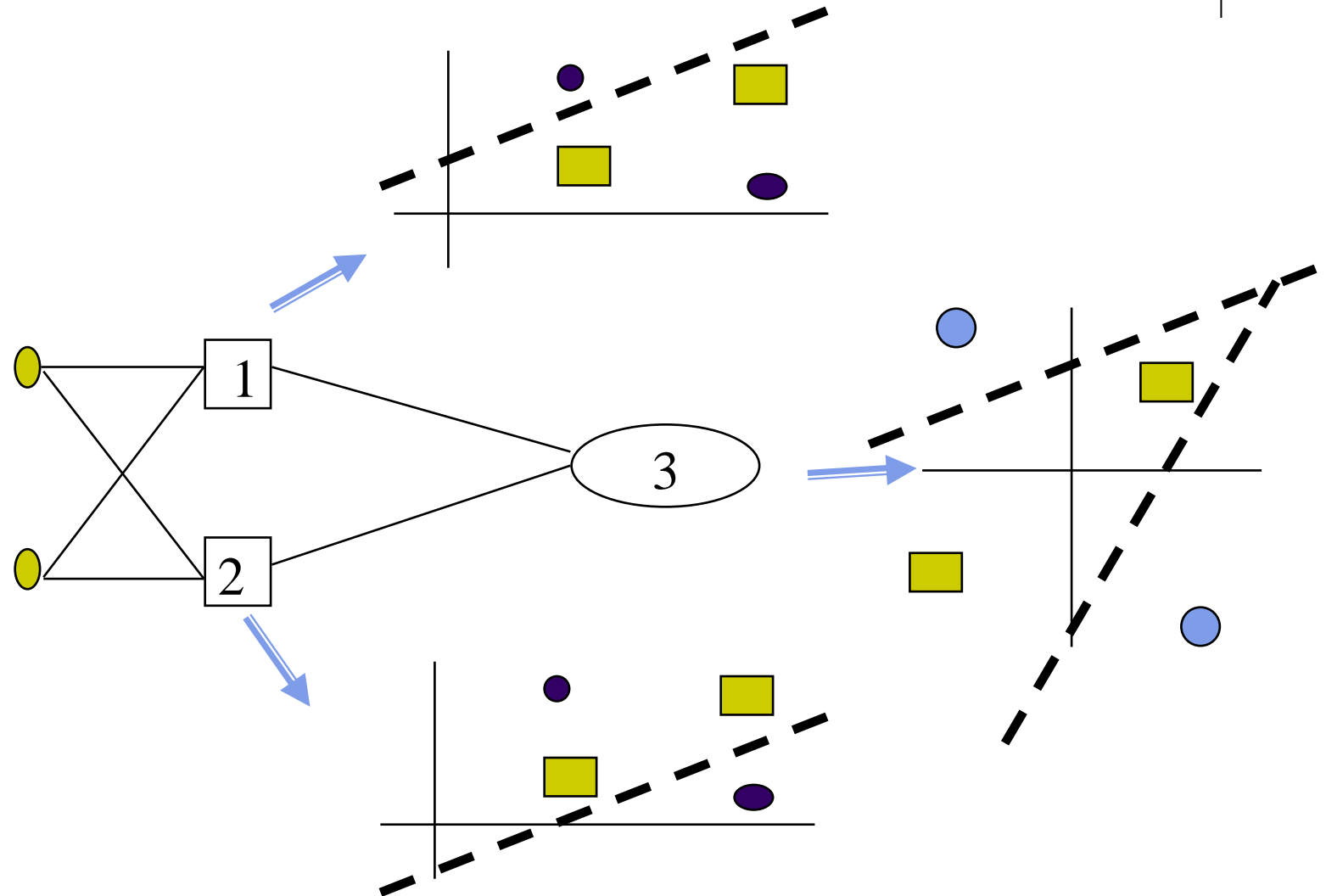


(c)  $I_1 \text{ xor } I_2$

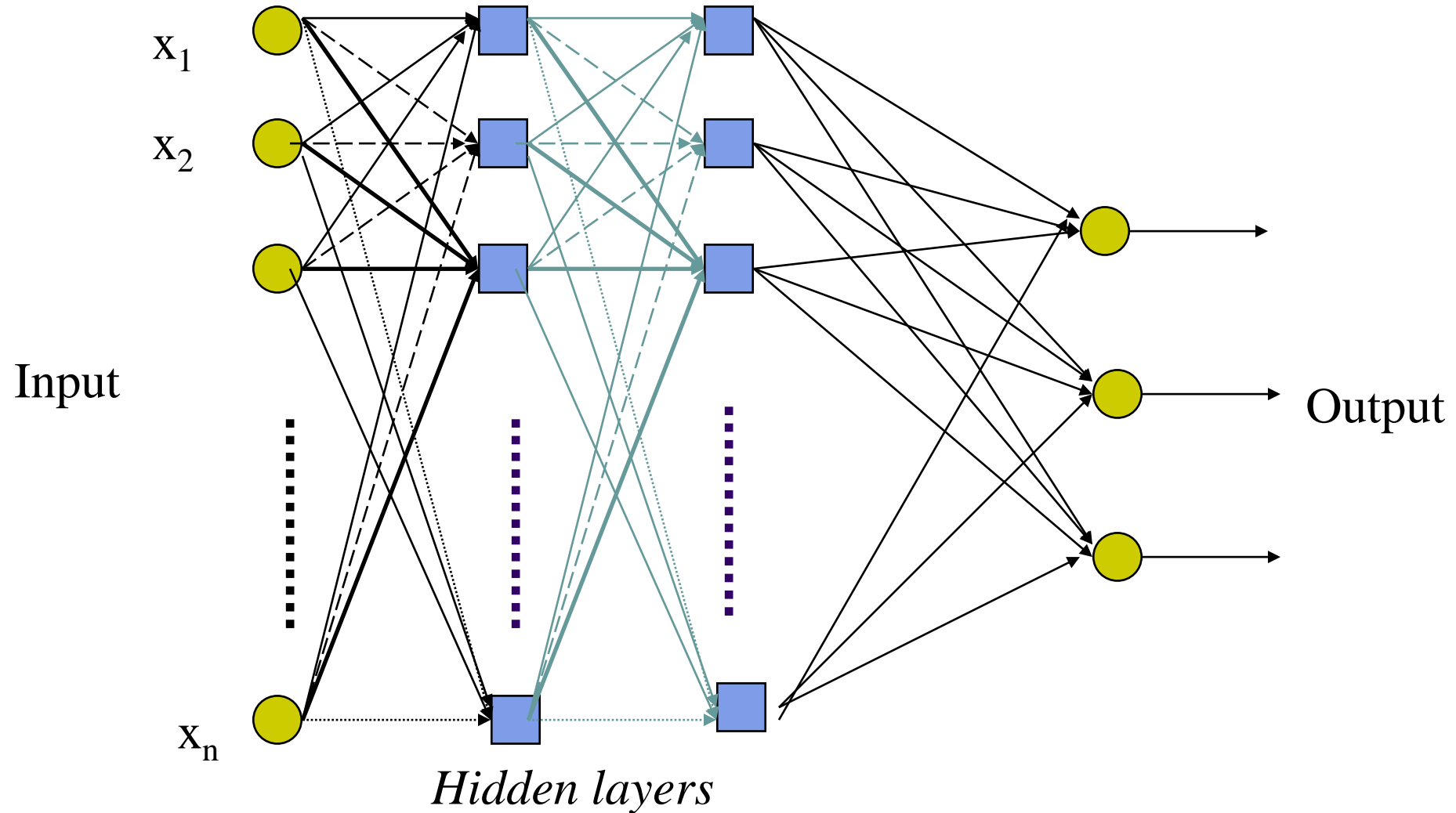
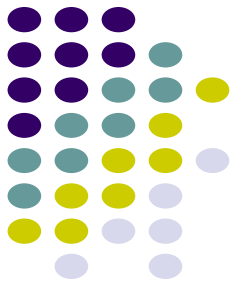
# Solution of XOR problem

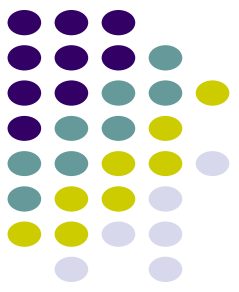


Minsky & Papert (1969) offered solution to XOR problem by combining perceptron unit responses using a second layer of perceptron. Piecewise linear classification using an Multi-layer perceptron.



# Multi-layer Perceptron



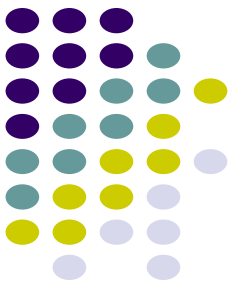


- No connections within a layer
- No direct connections between input and output layers
- Fully connected between layers
- Can have more than 3 layers
- Number of output units need not equal number of input units
- Number of hidden units per layer can be more or less than input or output units

Each unit is a perceptron

$$y_i = f \left( \sum_{j=1}^m w_{ij} x_j + b_i \right)$$

Often include bias as an extra weight



# Back-propagation Algorithm

- *Back-propagation* algorithm has two phases:

Forward pass phase: computes ‘functional signal’, feed forward propagation of input pattern signals through network

Backward pass phase: computes ‘error signal’, *propagates* the error *backwards* through network starting at output units (where the error is the difference between actual and desired output values)



# Forward Pass

First, the weighted sum of the hidden node is calculated as:

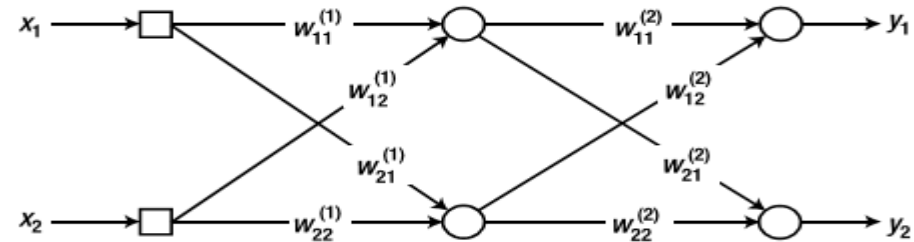
$$\begin{bmatrix} v_1^{(1)} \\ v_2^{(1)} \end{bmatrix} = \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$\triangleq W_1 x$$

The weighted sum of the output nodes is calculated as:

$$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} w_{11}^{(2)} & w_{12}^{(2)} \\ w_{21}^{(2)} & w_{22}^{(2)} \end{bmatrix} \begin{bmatrix} y_1^{(1)} \\ y_2^{(1)} \end{bmatrix}$$

$$\triangleq W_2 y^{(1)}$$

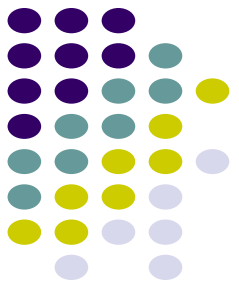


We obtain the output from the hidden nodes:

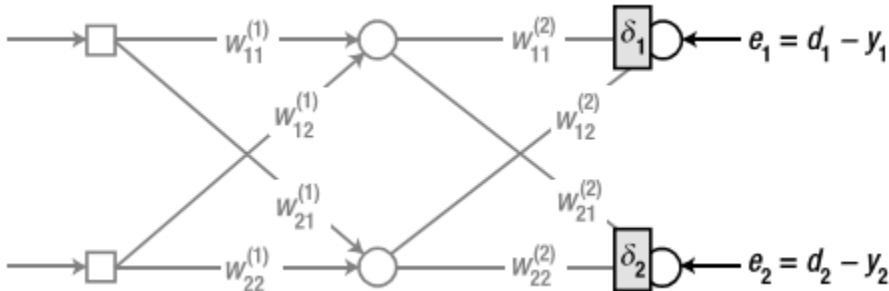
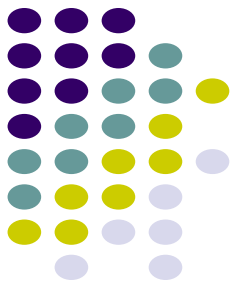
$$\begin{bmatrix} y_1^{(1)} \\ y_2^{(1)} \end{bmatrix} = \begin{bmatrix} \varphi(v_1^{(1)}) \\ \varphi(v_2^{(1)}) \end{bmatrix}$$

We obtain the output from the output nodes:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \varphi(v_1) \\ \varphi(v_2) \end{bmatrix}$$



Now, we will train the neural network using the back-propagation algorithm.



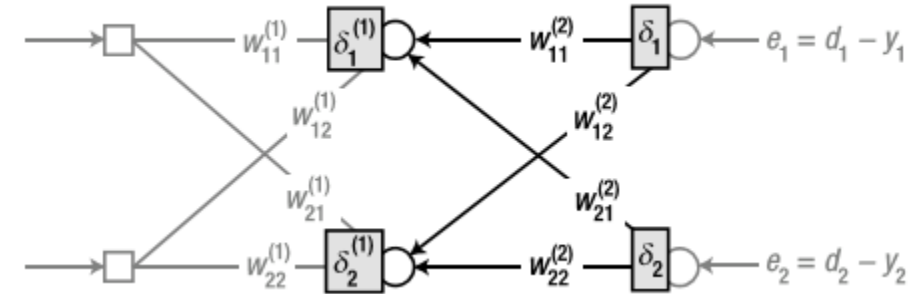
$$e_1 = d_1 - y_1$$

$$\delta_1 = \varphi'(v_1)e_1$$

$$e_2 = d_2 - y_2$$

$$\delta_2 = \varphi'(v_2)e_2$$

Proceed leftward to the hidden nodes and calculate the delta



$$e_1^{(1)} = w_{11}^{(2)}\delta_1 + w_{21}^{(2)}\delta_2$$

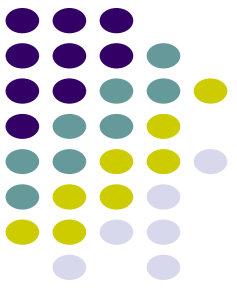
$$\delta_1^{(1)} = \varphi'(v_1^{(1)})e_1^{(1)}$$

$$e_2^{(1)} = w_{12}^{(2)}\delta_1 + w_{22}^{(2)}\delta_2$$

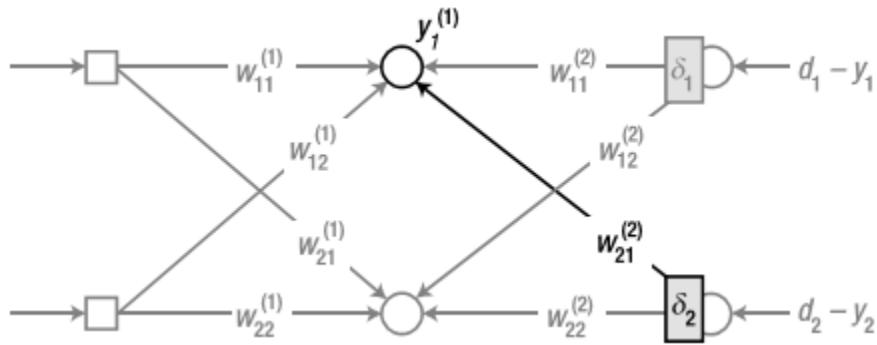
$$\delta_2^{(1)} = \varphi'(v_2^{(1)})e_2^{(1)}$$

$$\begin{bmatrix} e_1^{(1)} \\ e_2^{(1)} \end{bmatrix} = \begin{bmatrix} w_{11}^{(2)} & w_{21}^{(2)} \\ w_{12}^{(2)} & w_{22}^{(2)} \end{bmatrix} \begin{bmatrix} \delta_1 \\ \delta_2 \end{bmatrix} \Rightarrow \begin{bmatrix} e_1^{(1)} \\ e_2^{(1)} \end{bmatrix} = W_2^T \begin{bmatrix} \delta_1 \\ \delta_2 \end{bmatrix}$$

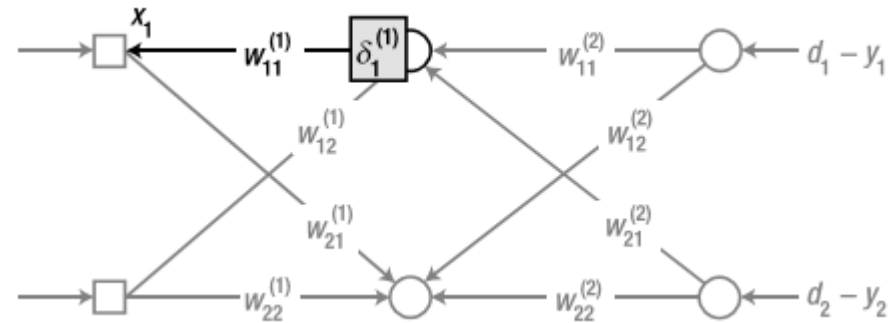
# Update weights



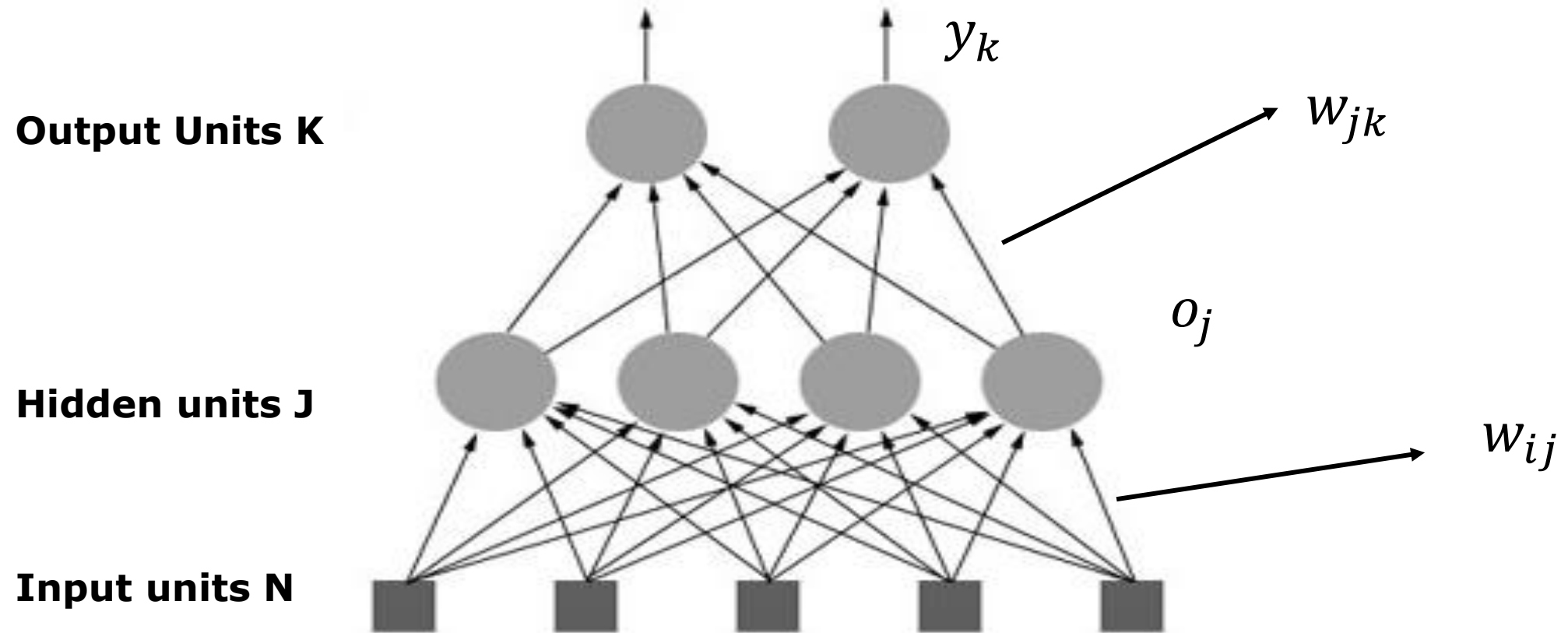
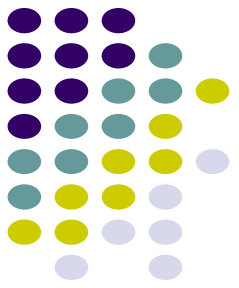
$$\Delta w_{ij} = \alpha \delta_i x_j$$
$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$$



$$w_{21}^{(2)} \leftarrow w_{21}^{(2)} + \alpha \delta_2 y_1^{(1)}$$



$$w_{11}^{(1)} \leftarrow w_{11}^{(1)} + \alpha \delta_1^{(1)} x_1$$



- **Step 1:** Initialize weights at random, choose a learning rate  $\eta$
- **Step 2:** For all labeled examples train the network.

For J hidden nodes, N input nodes and K output nodes

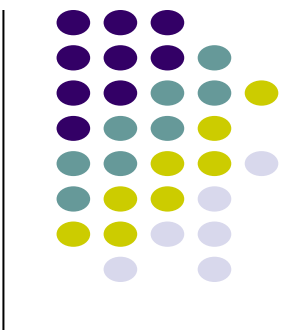
Total weights :  $(N*J)+(J*K)$

Output of k-th output node:

$$y_k = f\left(\sum_{j=1}^J w_{jk} o_j\right)$$

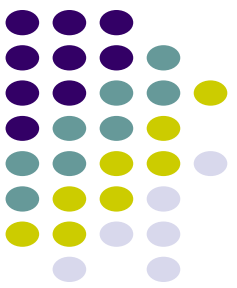
where  $o_j$  is the output of j-th hidden node

$$o_j = f\left(\sum_{i=1}^N w_{ij} x_i\right)$$



Activation function

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



For each output unit compute delta

$$\delta_k = (y_{target} - y_k)y_k(1 - y_k)$$

For each hidden unit compute delta

$$\delta_j = o_j(1 - o_j) \sum_{k=1}^K w_{jk} \delta_k$$

Change in weight:

For weight from input layer i to hidden layer j

$$\Delta w_{ij} = \eta \delta_j x_i$$

For weight from hidden layer j to output layer k

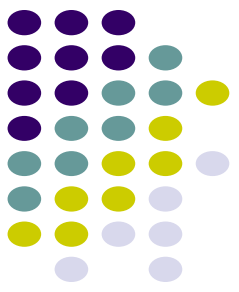
$$\Delta w_{jk} = \eta \delta_k o_j$$

# Training MLP

- The updation rules for weights are computed using Gradient Descent Approach.
- We try to minimize the squared error in classification

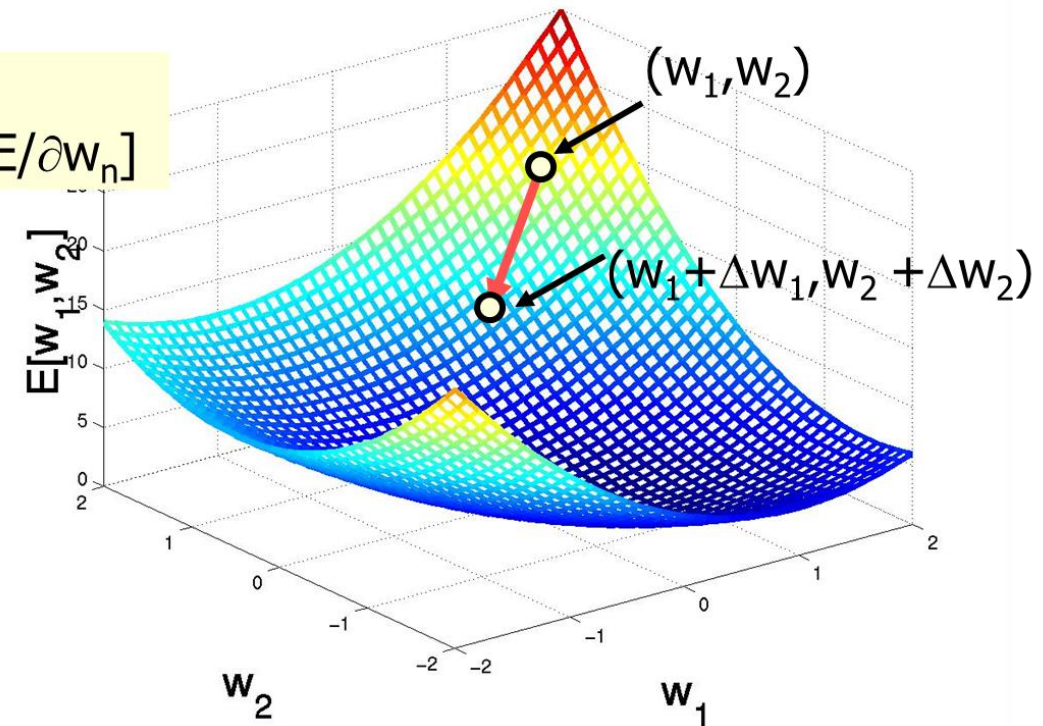
$$E_d[w] = \frac{1}{2} (\text{targetOutput} - \text{recievedOutput})^2$$

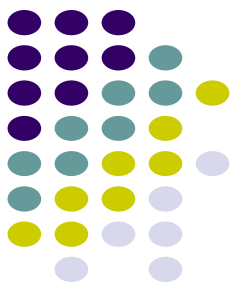
where d is the labeled sample in the dataset D



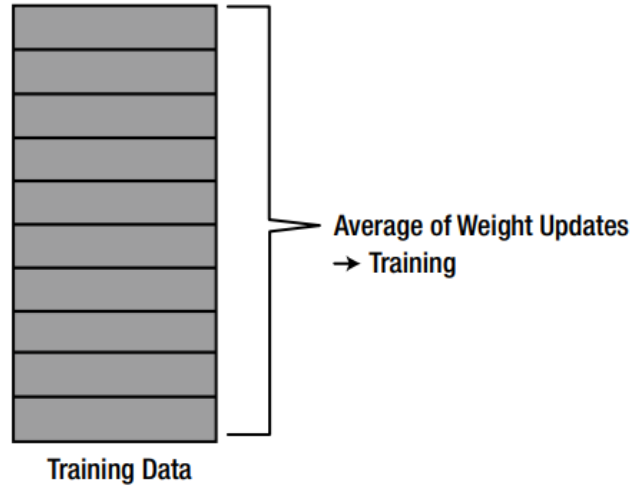
Gradient:

$$\nabla E[w] = [\partial E / \partial w_0, \dots, \partial E / \partial w_n]$$

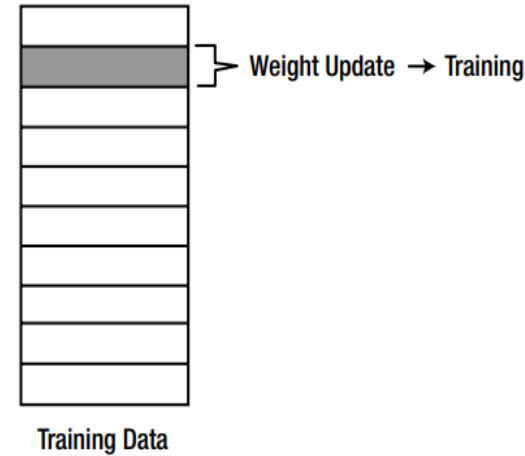




## Batch



## Incremental



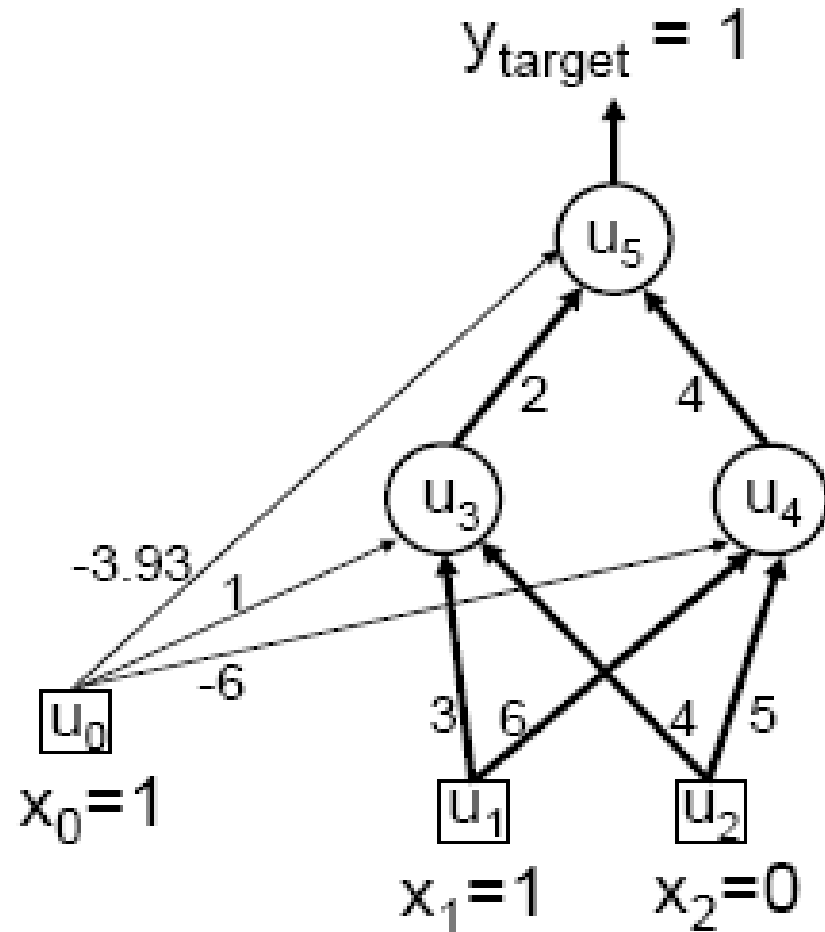
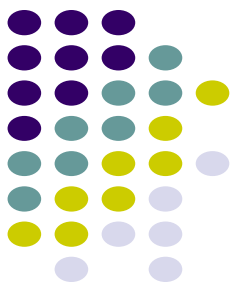
- Learning rate  $\eta$

If  $\eta$  small  $\Rightarrow$  Slow rate of learning

If  $\eta$  large  $\Rightarrow$  Large changes of weights  
 $\Rightarrow$  learning can become unstable



# Worked Example

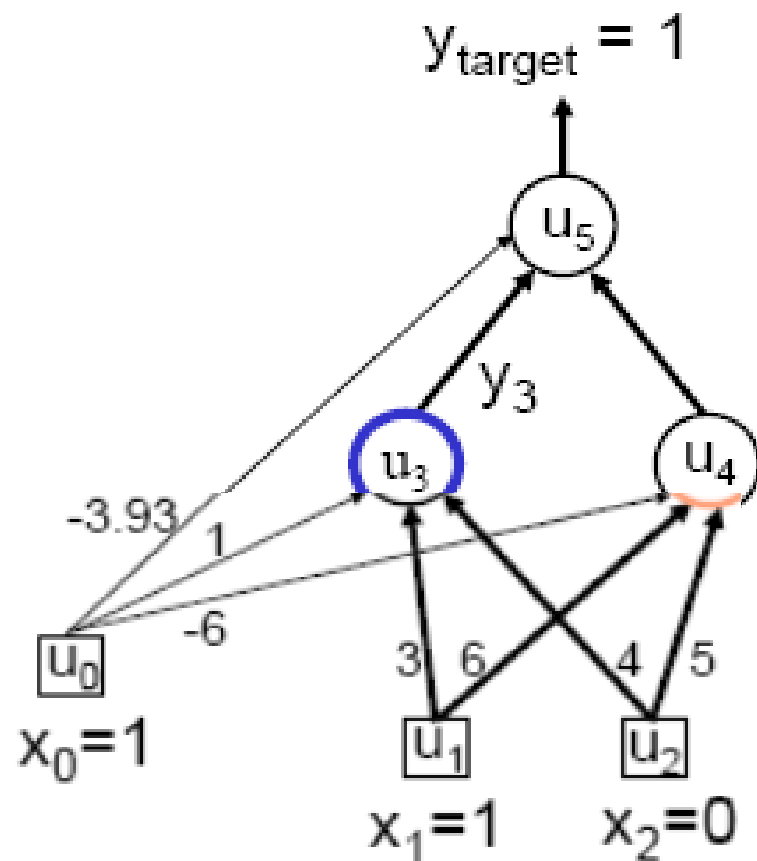
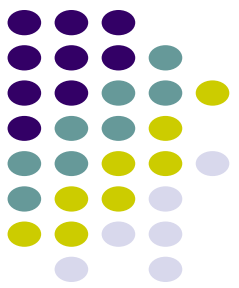


Current state:

- Weights on arrows e.g.  $w_{13} = 3$ ,  $w_{35} = 2$ ,  $w_{24} = 5$
- Bias weights, e.g. bias for unit 4 ( $u_4$ ) is  $w_{04} = -6$

Training example (e.g. for logical OR problem):

- Input pattern is  $x_1=1$ ,  $x_2=0$
- Target output is  $y_{\text{target}}=1$



Output for any neuron/unit  $j$   
can be calculated from:

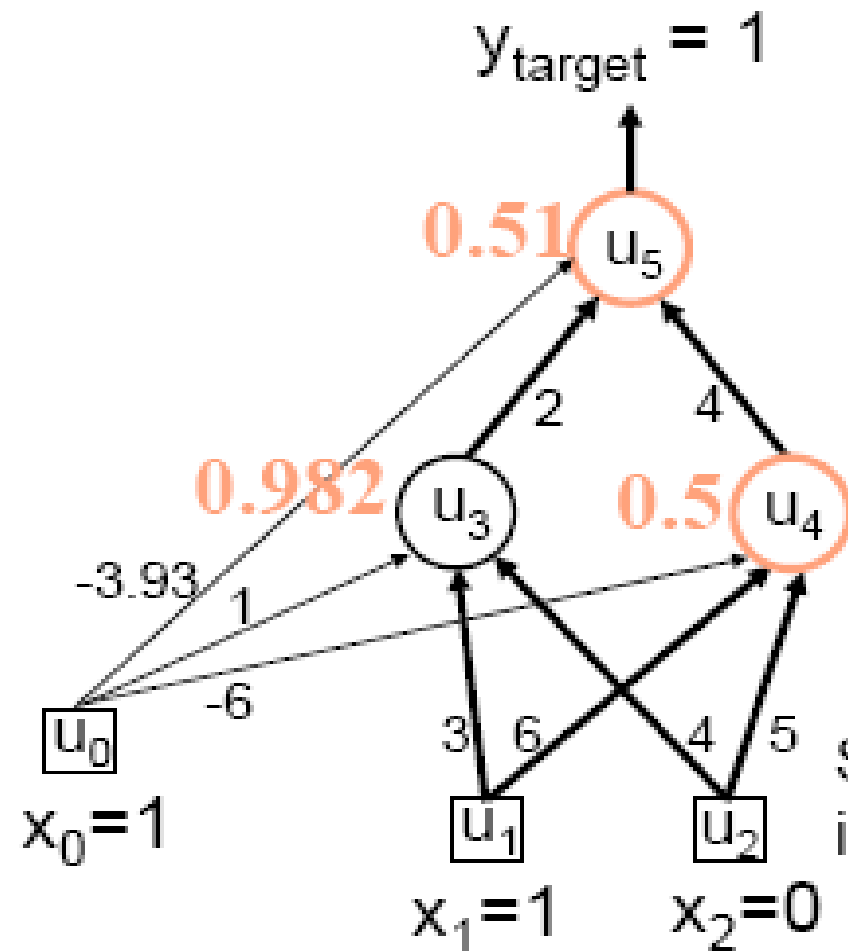
$$a_j = \sum_i w_{ij} x_i$$

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

e.g Calculating output for  
Neuron/unit 3 in hidden layer:

$$a_3 = 1 * 1 + 3 * 1 + 4 * 0 = 4$$

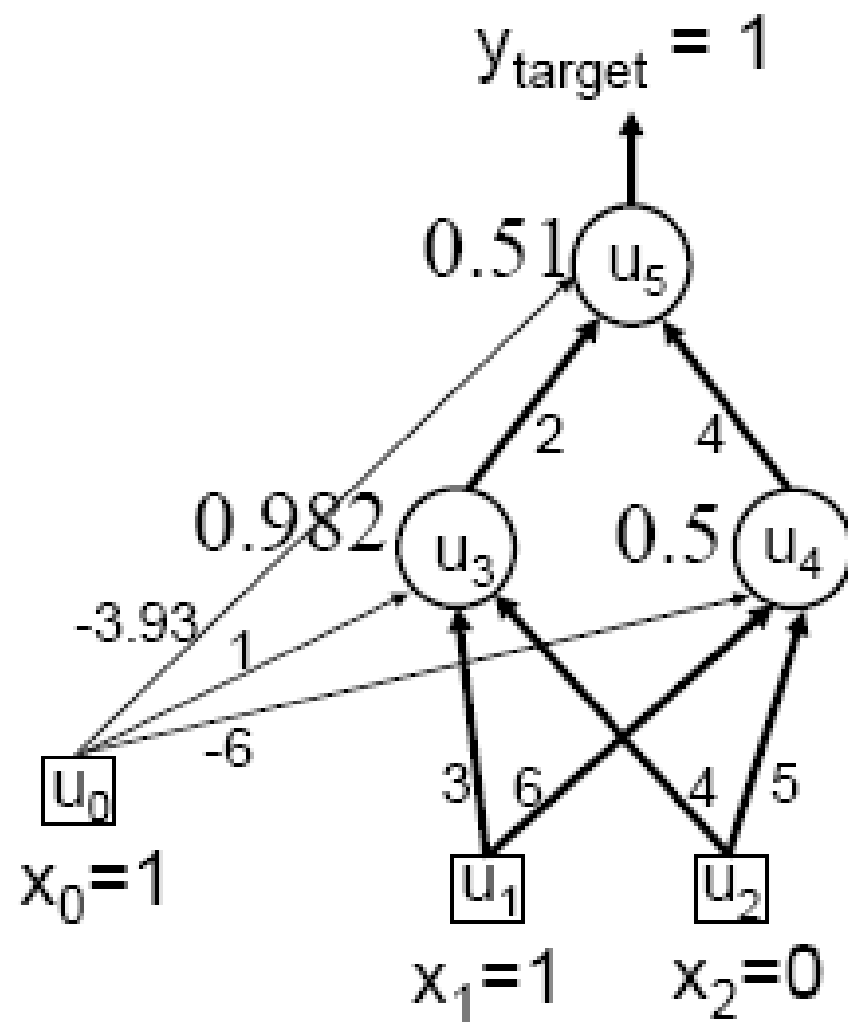
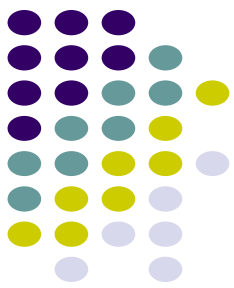
$$y_3 = f(4) = \frac{1}{1 + e^{-4}} = 0.982$$



Unit	activation $a_j$	output $y_j$
$u_3$	4.00	0.982
$u_4$	0.00	0.500
$u_5$	0.04	<b>0.510</b>

So the error for this training example is:  $(y_{\text{target}} - y_5) = (1 - 0.510) = 0.490$

(network output)



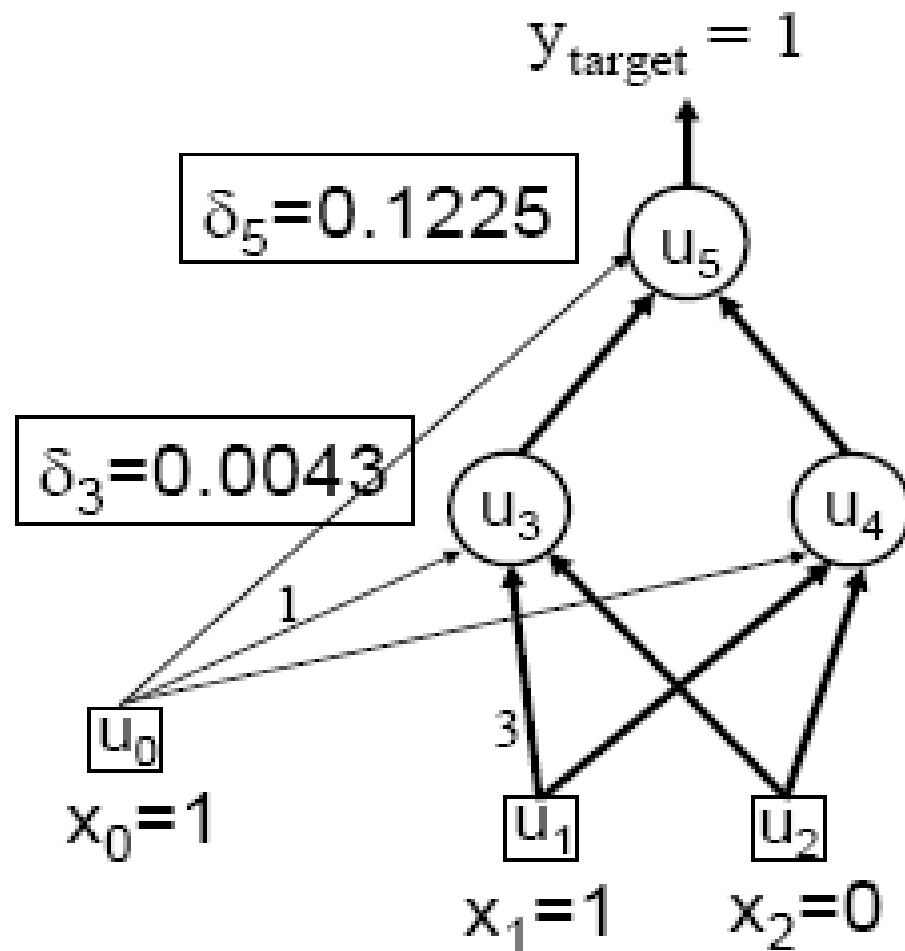
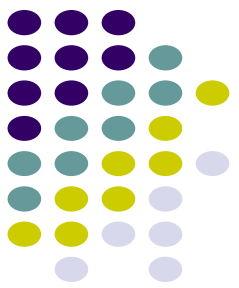
Now compute delta values starting at the output:

$$\begin{aligned}\delta_5 &= y_5(1 - y_5)(y_{\text{target}} - y_5) \\ &= 0.51(1 - 0.51) \times 0.49 \\ &= \mathbf{0.1225}\end{aligned}$$

Then for hidden units:

$$\begin{aligned}\delta_4 &= y_4(1 - y_4) w_{45} \delta_5 \\ &= 0.5(1 - 0.5) \times 4 \times 0.1225 \\ &= \mathbf{0.1225}\end{aligned}$$

$$\begin{aligned}\delta_3 &= y_3(1 - y_3) w_{35} \delta_5 \\ &= 0.982(1 - 0.982) \times 2 \times 0.1225 \\ &= \mathbf{0.0043}\end{aligned}$$



- ◆ Set learning rate  $\eta = 0.1$   
Change weights by:  
$$\Delta w_{ij} = \eta \delta_j y_i$$

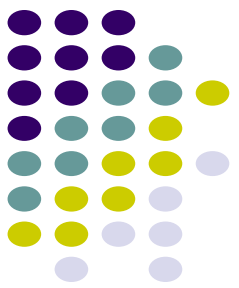
- ◆ e.g. bias weight on  $u_3$ :  
$$\begin{aligned} \Delta w_{03} &= \eta \delta_3 x_0 \\ &= 0.1 * 0.0043 * 1 \\ &= 0.0004 \end{aligned}$$

So, new  $w_{03} \leftarrow$

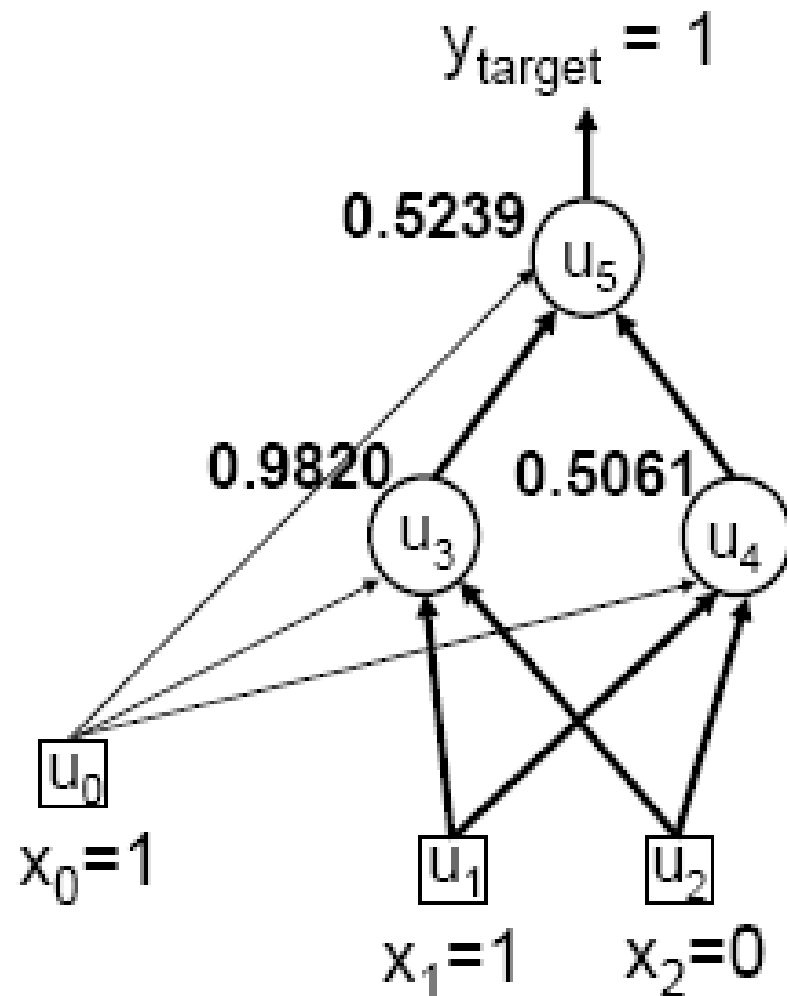
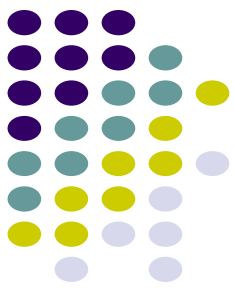
$$\begin{aligned} w_{03}(\text{old}) + \Delta w_{03} \\ = 1 + 0.0004 = 1.0004 \end{aligned}$$

- ◆ and likewise:

$$\begin{aligned} w_{13} &\leftarrow 3 + 0.0004 \\ &= 3.0004 \end{aligned}$$



i	j	$w_{ij}$	$\delta_j$	$y_i$	Updated $w_{ij}$
0	3	<b>1</b>	0.0043	1.0	<b>1.0004</b>
1	3	<b>3</b>	0.0043	1.0	<b>3.0004</b>
2	3	<b>4</b>	0.0043	0.0	<b>4.0000</b>
0	4	<b>-6</b>	0.1225	1.0	<b>-5.9878</b>
1	4	<b>6</b>	0.1225	1.0	<b>6.0123</b>
2	4	<b>5</b>	0.1225	0.0	<b>5.0000</b>
0	5	<b>-3.92</b>	0.1225	1.0	<b>-3.9078</b>
3	5	<b>2</b>	0.1225	0.9820	<b>2.0120</b>
4	5	<b>4</b>	0.1225	0.5	<b>4.0061</b>



On next forward pass:

The new activations are:

$$y_3 = f(4.0008) = 0.9820$$

$$y_4 = f(0.0245) = 0.5061$$

$$y_5 = f(0.0955) = \mathbf{0.5239}$$

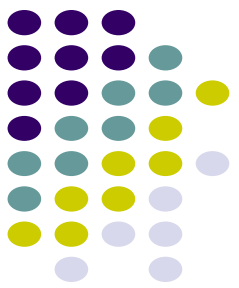
Thus the new error

$$(y_{\text{target}} - y_5) = (1 - 0.5239) = 0.476$$

has been reduced by 0.014

(from **0.490** to **0.476**)

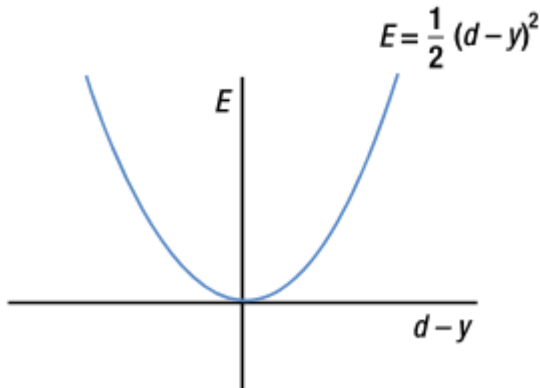
*Ref: "Neural Network Learning & Expert Systems" by Stephen Gallant*



# Cost Function

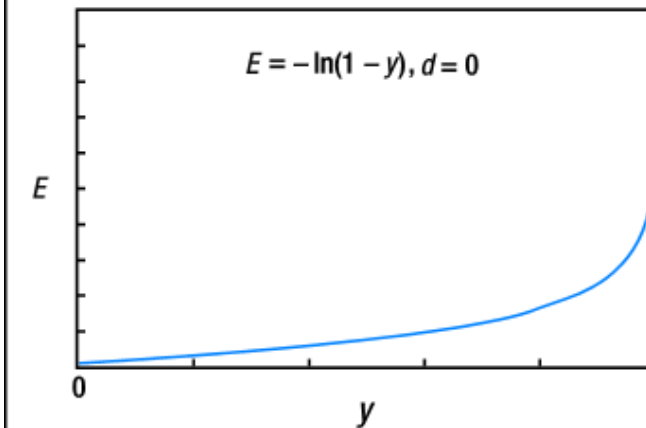
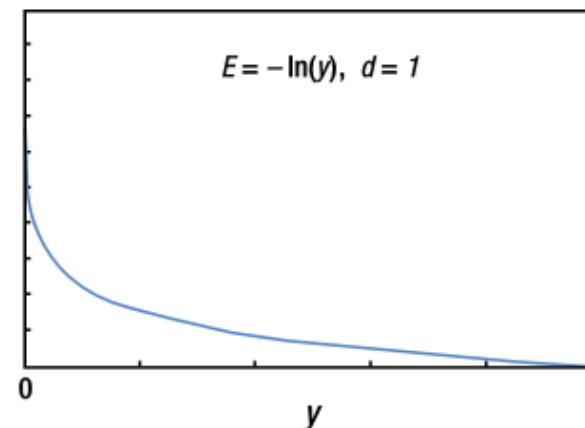
- The measure of the neural network's error is the cost function.
- The greater the error of the neural network, the higher the value of the cost function is.
- There are two primary types of cost functions for the neural network's supervised learning: Sum of squared error and Cross entropy

$$J = \sum_{i=1}^M \frac{1}{2} (d_i - y_i)^2$$



The cross entropy function is much more sensitive to the error.

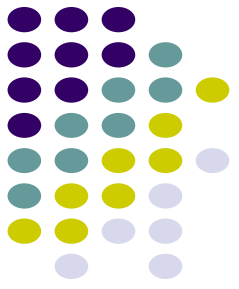
$$J = \sum_{i=1}^M \{-d_i \ln(y_i) - (1 - d_i) \ln(1 - y_i)\}$$



where  $y_i$  is the output from the output node,  $d_i$  is the correct output from the training data, and  $M$  is the number of output nodes.



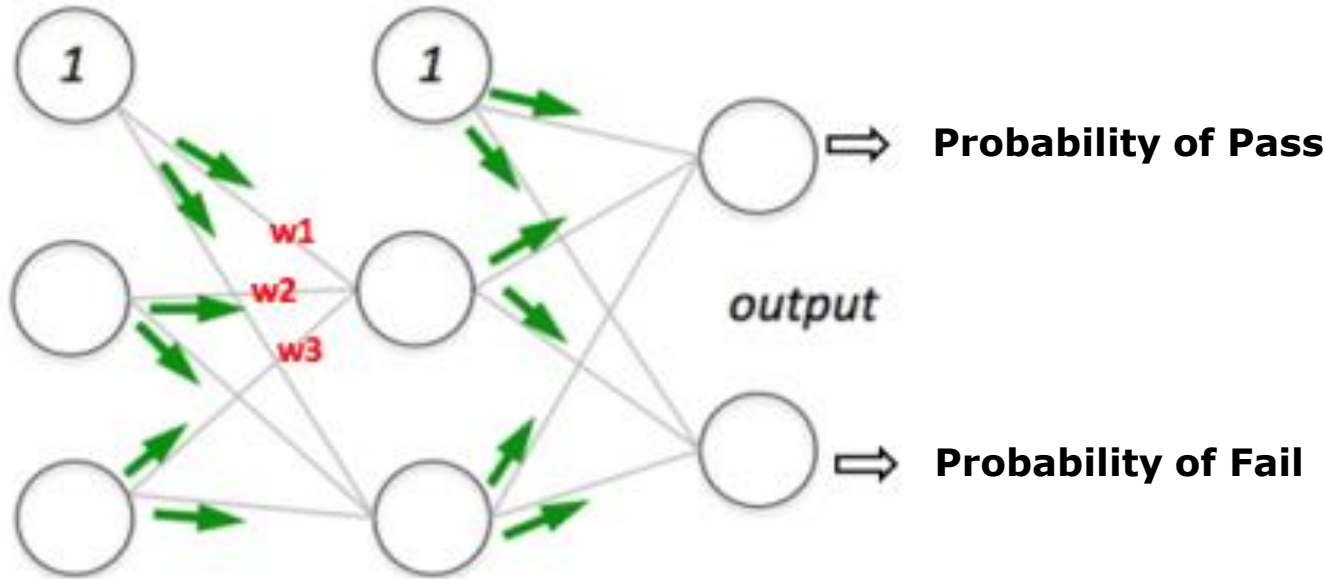
# How to apply MLP in real world problems



bias

Hours  
Studied

Marks  
Obtained



Labeled Data

Hours Studied	Marks Obtained	Result
35	67	1(Pass)
12	75	0(Fail)
16	89	1(Pass)
45	56	1(Pass)
10	50	0(Fail)

Predict

Hours Studied	Marks Obtained	Result
25	70	??

