



ADALINE (Adaptive Linear Neural)

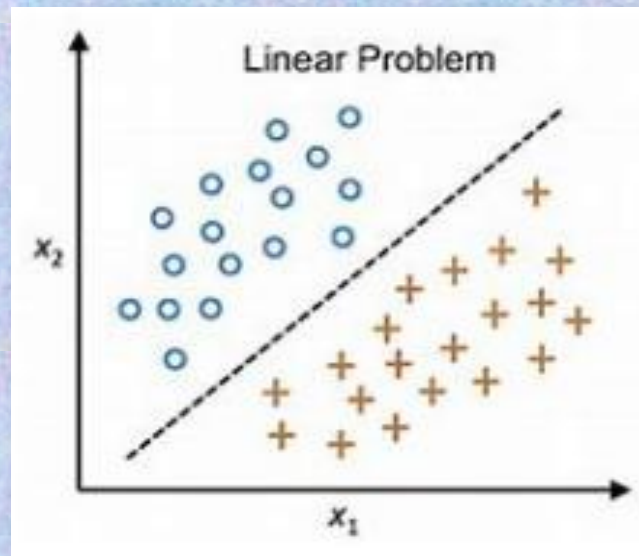
Widrow-Hoff Learning Rule

ADALINE (Adaptive Linear Neural)

- ❖ **A network with a single linear unit is called Adaline (Adaptive Linear Neural).**
- ❖ **A unit with a linear activation function is called a linear unit.**
- ❖ **In Adaline, there is only one output unit .**
- ❖ **Like a Perceptron, ADALINE can solve linearly separable problems.**

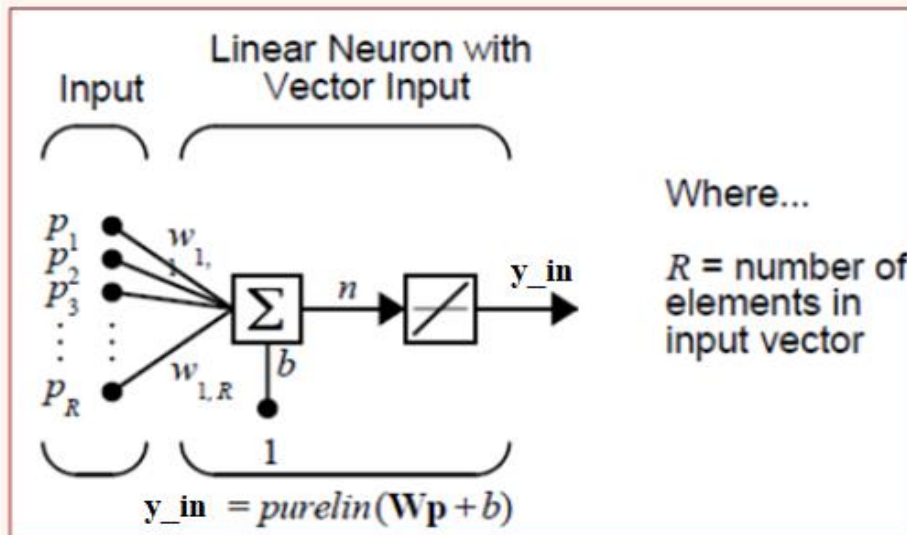
ADALINE (Adaptive Linear Neural)

A linearly separable problem



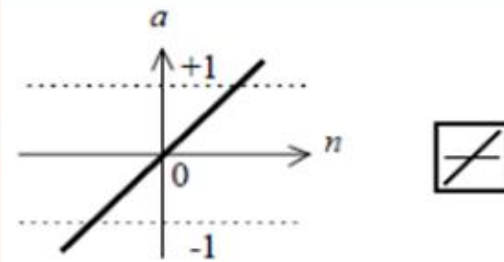
ADALINE (Adaptive Linear Neural)

The structure of ADALINE



Where...

R = number of elements in input vector



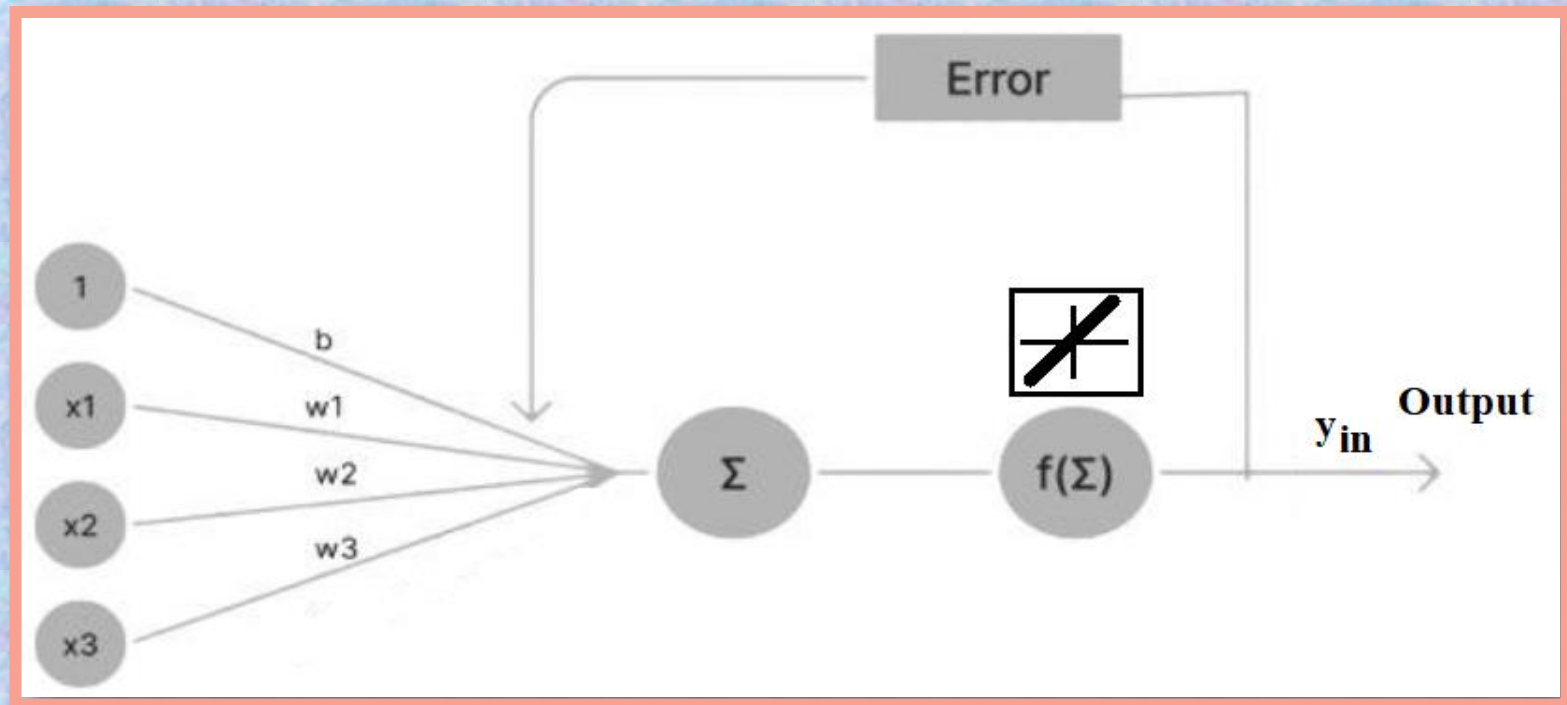
$$y_{in} = \text{purelin}(n)$$

Linear Transfer Function

ADALINE (Adaptive Linear Neural)

- **The learning rule is found to minimize the mean square error between neural network outputs and target values.**
- **ADALINE consists of trainable weights, it compares actual output with calculated target, and based on error training algorithm is applied.**

ADALINE (Adaptive Linear Neural)



$$error : (t - y_{in})^2$$

ADALINE : Training Algorithm

Step 1: Initialize weight not zero but small random values are used.
Set learning rate α .

Step 2: While the stopping condition is False do steps 3 to 7.

Step 3: For each training set perform steps 4 to 6.

Step 4: Set x_i for ($i=1$ to n).

Step 5: Compute net input to output unit $y_{in} = \sum w_i x_i + b$

Here, b is the bias and n is the total number of neurons.

Step 6: Update the weights and bias for $i=1$ to n

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

and calculate

$$\text{error} : (t - y_{in})^2$$

Step 7: Test the stopping condition.

Design OR gate using Adaline Network

Solution :

- Initially, all weights are assumed to be small random values, say 0.1, and set learning rule to 0.1.
- The weights will be updated until the total error is greater than the least squared error.

x_1	x_2	t
1	1	1
1	-1	1
-1	1	1
-1	-1	-1

Design OR gate using Adaline Network

- Calculate

$$y_{in} = \sum w_i x_i + b$$

$$y_{in} = 0.1 \times 1 + 0.1 \times 1 + 0.1 = 0.3 \quad (\text{when } x_1=x_2=1)$$

- Now compute, $(t - y_{in}) = (1 - 0.3) = 0.7$
- Now, update the weights and bias

$$w_i(\text{new}) = w_i(\text{old}) + \alpha(t - y_{in})x_i$$
$$w_1(\text{new}) = 0.1 + 0.1(1 - 0.3)1 = 0.17$$
$$w_2(\text{new}) = 0.1 + 0.1(1 - 0.3)1 = 0.17$$

$$b(\text{new}) = b(\text{old}) + \alpha(t - y_{in})$$
$$b(\text{new}) = 0.1 + 0.1(1 - 0.3) = 0.17$$

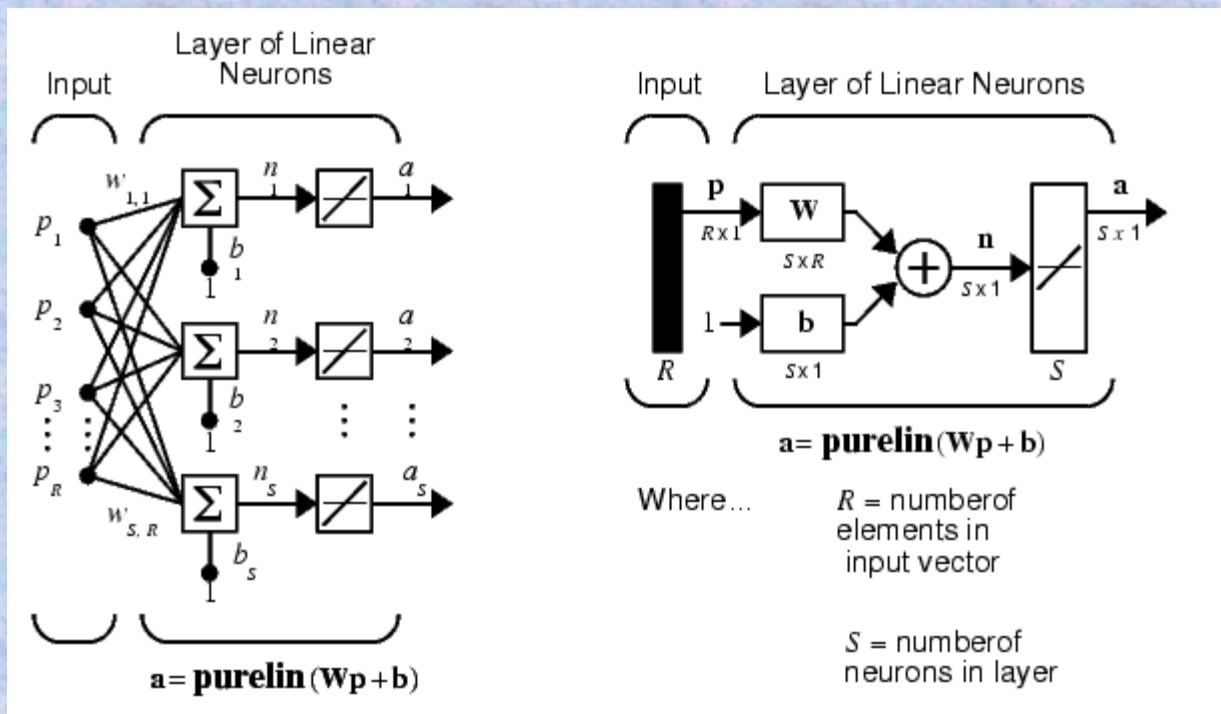
Design OR gate using Adaline Network

x_1	x_2	t	y_{in}	$(t-y_{in})$	Δw_1	Δw_2	Δb	w_1 (0.1)	w_2 (0.1)	b (0.1)	error ²
1	1	1	0.3	0.7	0.07	0.07	0.07	0.17	0.17	0.17	0.49
1	-1	1	0.17	0.83	0.083	-0.083	0.083	0.253	0.087	0.253	0.69
-1	1	1	0.087	0.913	-0.0913	0.0913	0.0913	0.1617	0.1783	0.3443	0.83
-1	-1	-1	0.0043	-1.0043	0.1004	0.1004	-0.1004	0.2621	0.2787	0.2439	1.01

$$e = E \{(y-t)^2\} = \sum_{p=1}^4 [\{x_1(p)w_1 + x_2(p)w_2 + w_0\} - t(p)]^2$$

$$e = 0.49 + 0.69 + 0.83 + 1.01 = 3.02$$

MADALINE: Multiple Adaptive Linear Neurons



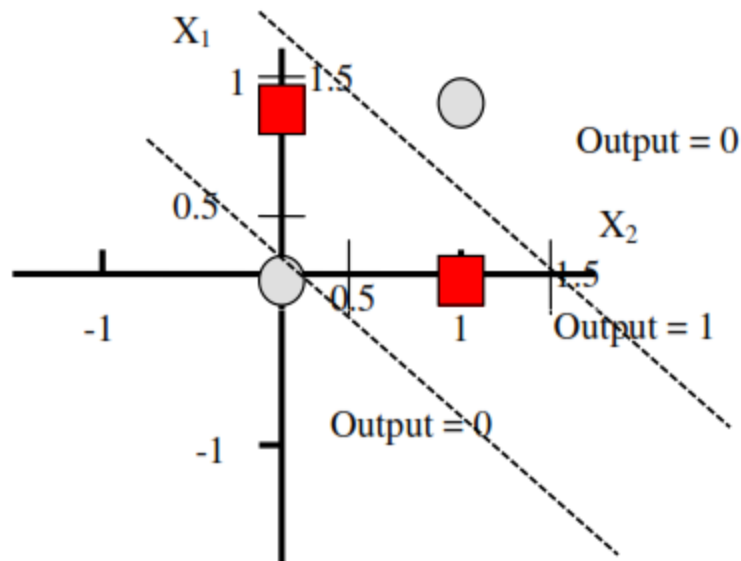
MADALINE (Many ADALINE) is a three-layer (input, hidden, output), fully connected, feed-forward artificial neural network architecture for classification that uses ADALINE units in its hidden and output layers.

Problem:

A Perceptron and ADALINE can solve linearly separable problems.

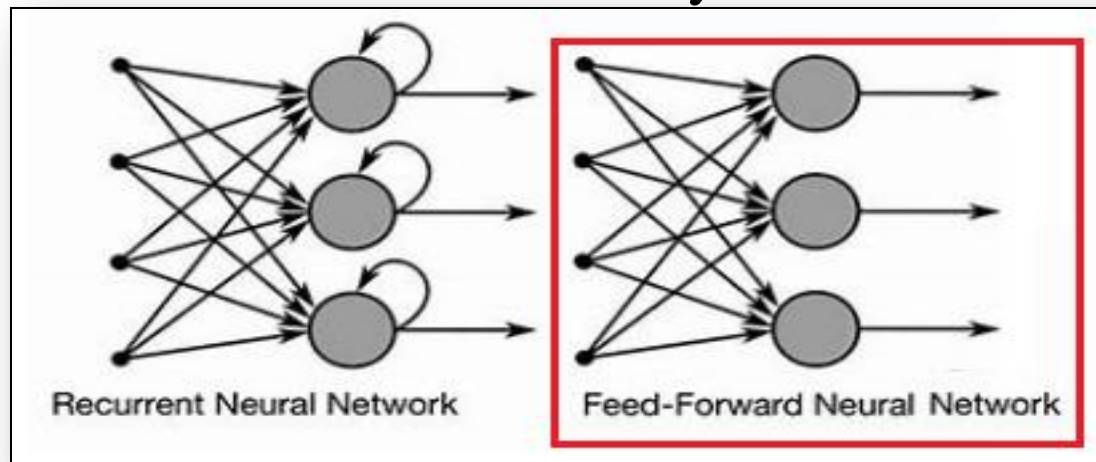
How to solve XOR?

A Possible Solution to the XOR Problem By Using Two Lines to Separate the Plane into Three Regions



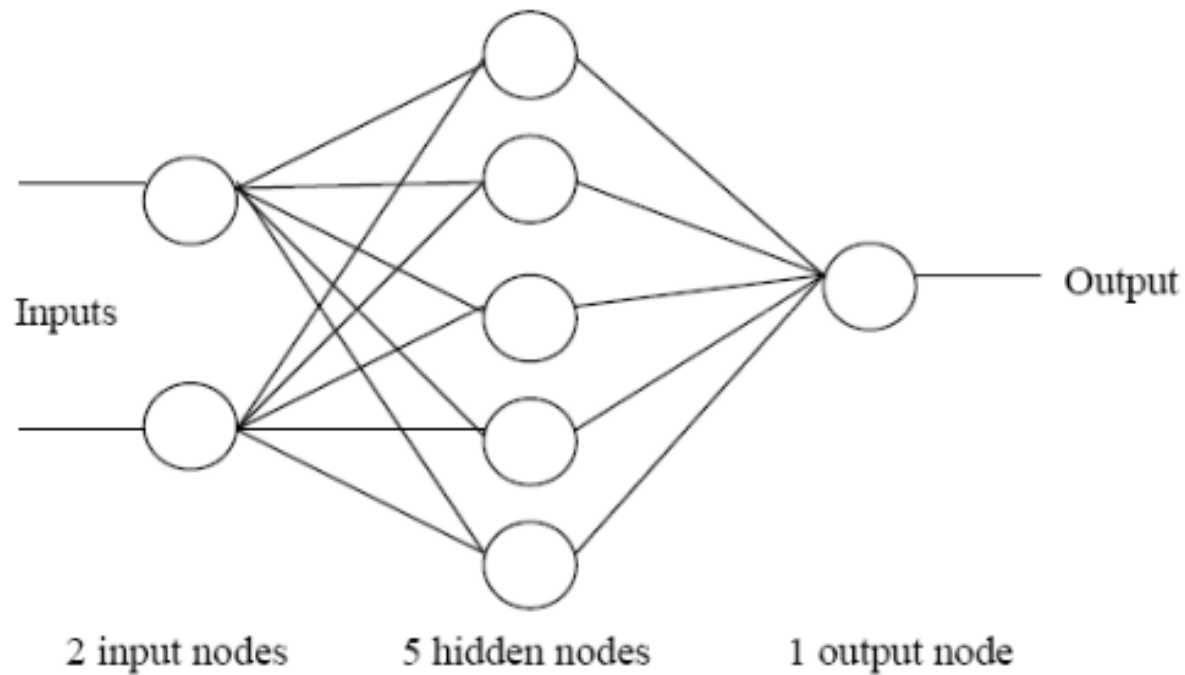
Multilayer Perceptron (MLP)

- ❖ A multilayer perceptron (MLP) is a **feedforward** neural network with one or more hidden layers.

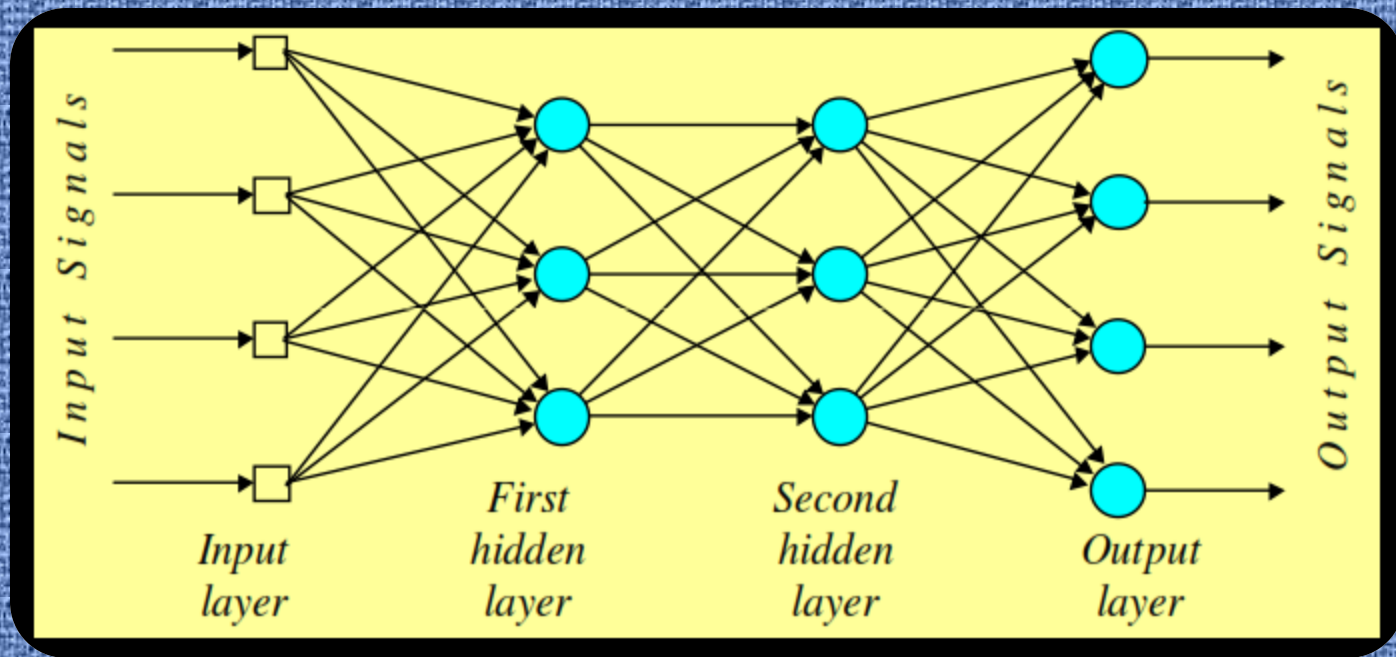


- ❖ The network consists of an input layer of source neurons, at least one middle or hidden layer of computational neurons, and an layer of computational neurons, and an output layer of computational neurons.
- ❖ The input signals are propagated in a forward direction on a layer-by-layer basis.

Multilayer Perceptron with ONE Hidden Layer



Multilayer Perceptron with Two Hidden Layers



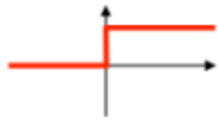
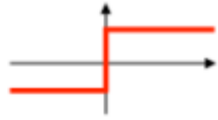
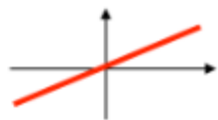

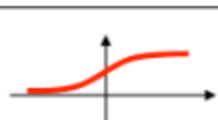
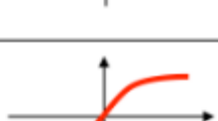


Multilayer Perceptron (MLP)

Properties of MLP Architecture

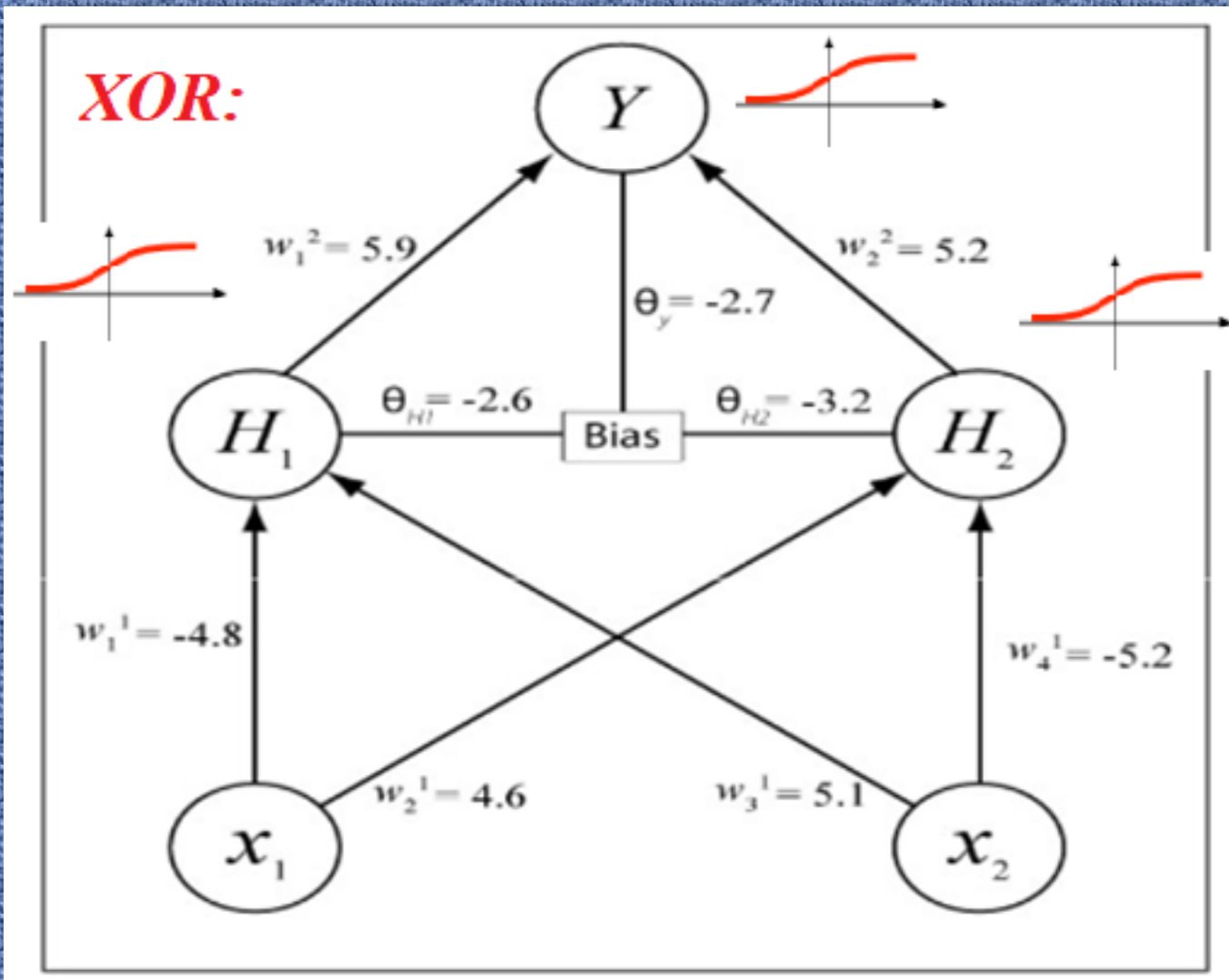
1. No connections within a layer
2. No direct connections between input and output layers
3. Fully connected between layers
4. Number of output units need not equal number of input units
5. Number of hidden units per layer can be more or less than input or output units

MULTILAYER PERCEPTRON MECHANISM (Supervised Learning)

- ✓ Learning in a multilayer network proceeds the same way as for a perceptron.
- ✓ A training set of input patterns is presented to the network.
- ✓ The network computes its output pattern, and if there is an error – or in other words, there is a difference between the output obtained by MLP and target – the weights are adjusted to reduce this error.

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = \max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^z)$	Multi-layer Neural Networks	

MULTILAYER PERCEPTRON MECHANISM (Supervised Learning)



XOR

Input		Target Output	Network Output (Y)
0	1	1	0.9401
1	0	1	
0	0	0	
1	1	0	

$$Y = f(\text{net}) = 1 / (1 + e^{-\text{net}})$$

$$\text{net} = \sum w_i^j x_i + \theta_j$$

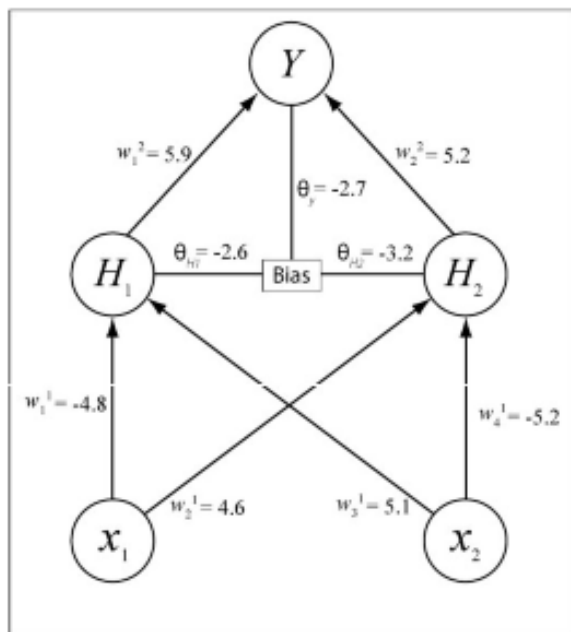
Generally,

$$\text{net}_{H_1} = \sum_{i=1}^2 x_i w_i + \theta_{H_1}$$

$$\text{net}_{H_1} = x_1 w_1^1 + x_2 w_2^1 + \theta_{H_1}$$

$$\text{net}_{H_1} = 0 * (-4.8) + 1 * (5.1) + (-2.6) = 2.5$$

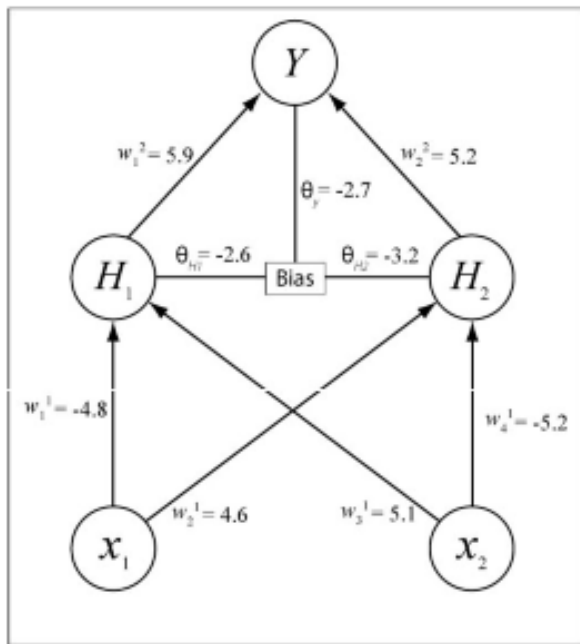
$$H_1 = f(\text{net}_{H_1}) = \frac{1}{1 + e^{-2.5}} = 0.9241$$



Thus, output for hidden nodes $H_1 = 0.9241$

XOR

Output for hidden node H_2 ,



$$net_{H_2} = \sum_{i=1}^2 x_i w_i + \theta_{H_2},$$

$$net_{H_2} = x_1 w_2^1 + x_2 w_4^1 + \theta_{H_2},$$

$$net_{H_2} = 0 * (4.6) + 1 * (-5.2) + (-3.2) = -8.4$$

$$H_2 = f(net_{H_2}) = \frac{1}{1 + e^{-(-8.4)}} \approx 0.00022$$

Thus, output for hidden nodes $H_2 = 0.00022$

XOR

Calculate the weighted sum going into the output neuron:

$$net_Y = \sum_{i=1}^2 H_i w_i + \theta_Y,$$

$$net_Y = H_1 w_1^2 + H_2 w_2^2 + \theta_Y,$$

$$net_Y = 0.9241 * (5.9) + 0.00022 * (5.2) + (-2.7) = 2.7533$$

$$Y = f(net_Y) = \frac{1}{1 + e^{-(2.7533)}} \approx 0.9401$$

Mean-squared error is used to quantify the error of the network:

$$err = 0.5 * (\text{Target Output} - \text{Actual Output})^2$$

$$err = 0.5 * (1 - 0.9401)^2 \approx 0.0018$$

XOR

Input		Target Output	Network Output (Y)
0	1	1	0.9401
1	0	1	0.8138
0	0	0	0.1102
1	1	0	0.1142