Artificial Neural Network

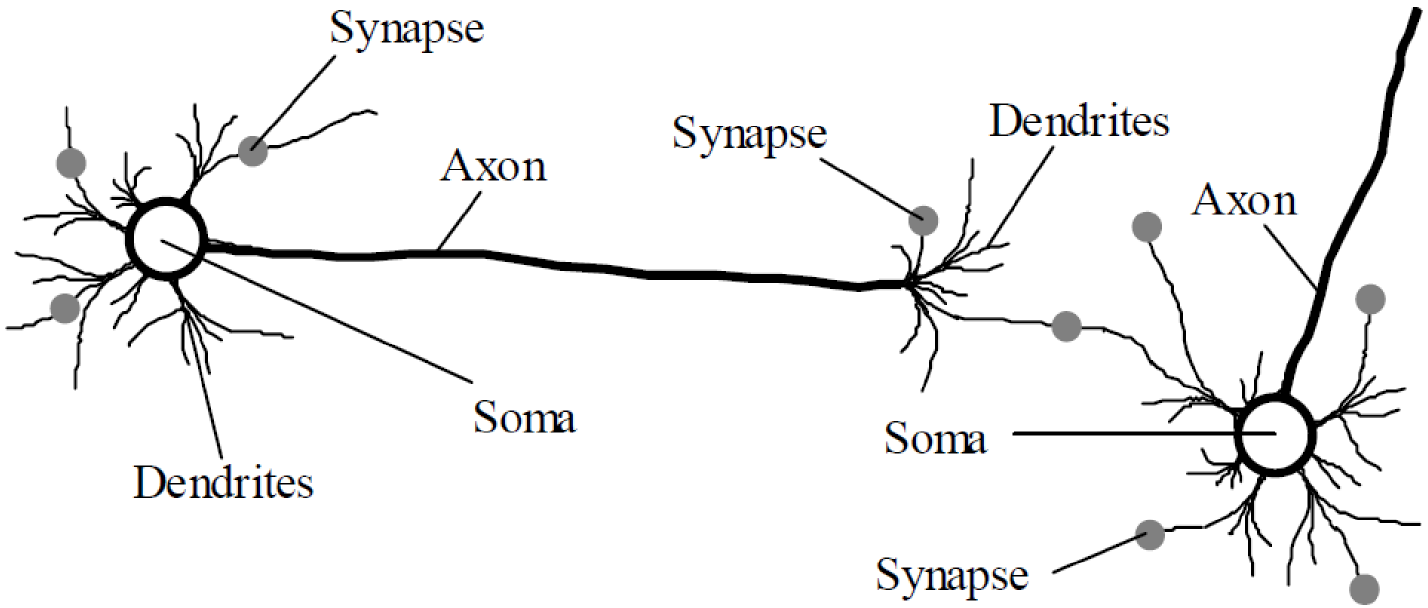
Dr. Avinash Kumar Singh
AI Consultant and Coach, Robaita

Discussion Points

- How brain works
- Similarity between brain and artificial neural network
- Perceptron learning
  - AND
  - OR
  - XOR
- Multilayer Perceptron
- Sigmoid, Tanh, ReLU: concepts and usage, Leaky ReLU and ELU, Effect on learning and gradient flow, Saturation and dead neurons

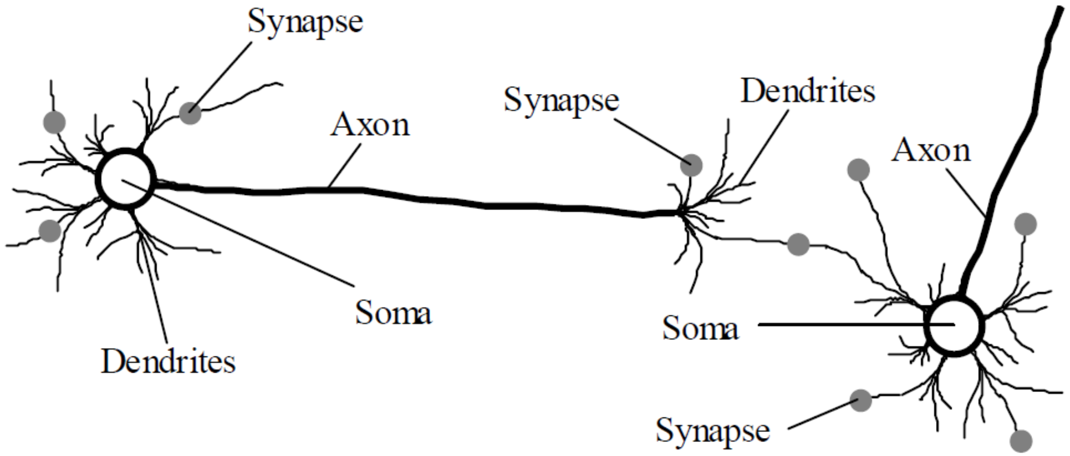
How Brain Works

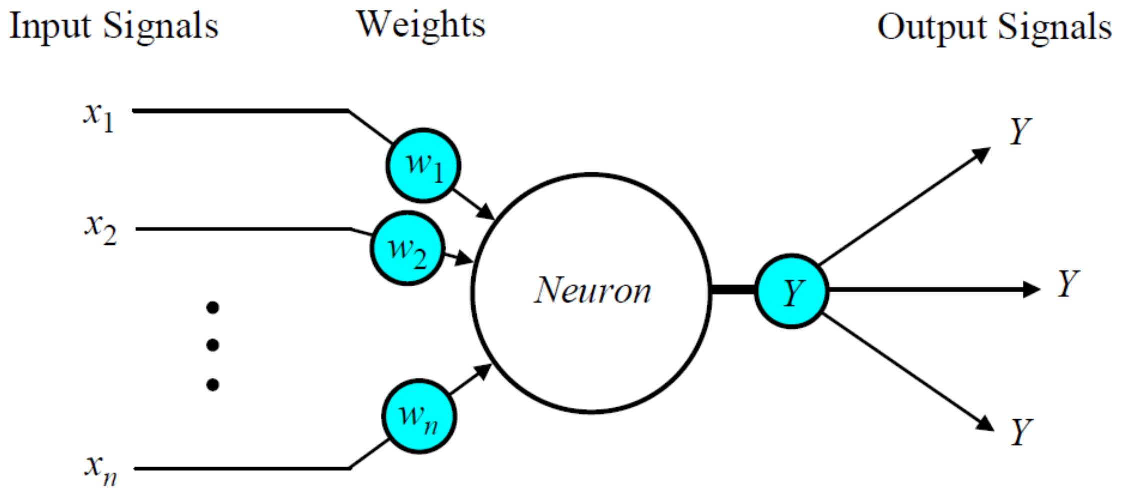
- The human brain incorporates nearly 10 billion neurons and 60 trillion connections, synapses, between them.
- A neuron consists of a cell body, soma, a number of fibers called dendrites, and a single long fiber called the axon.



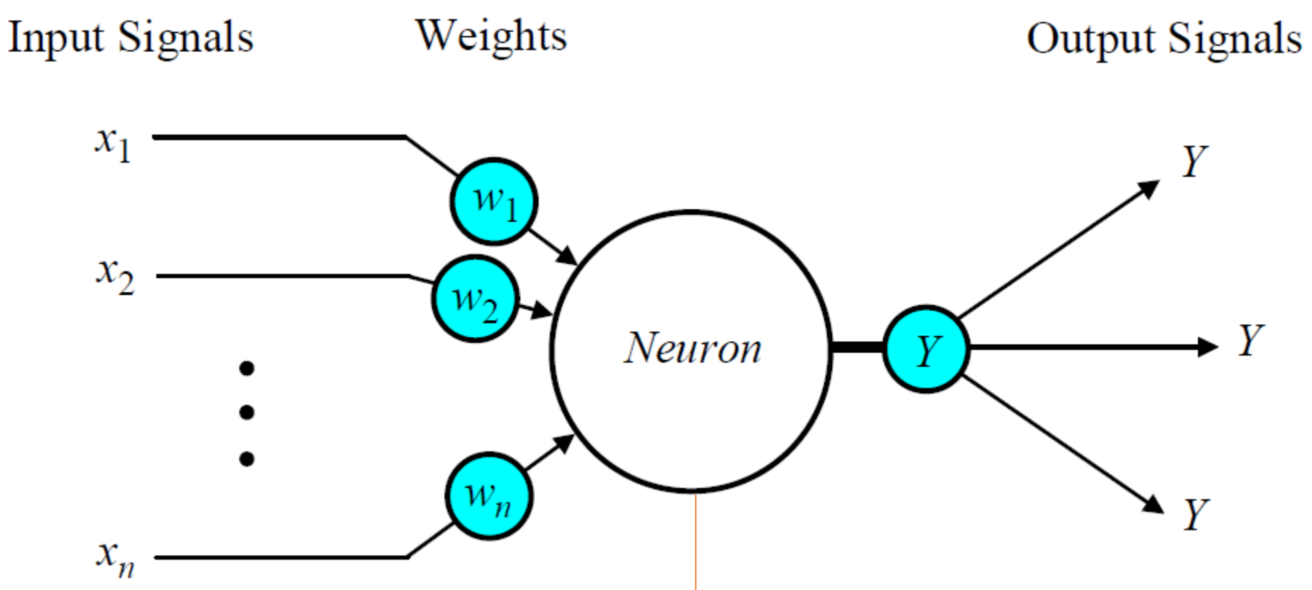
Similarity between brain and ANN

Biological Neural Network	*Artificial Neural Network*
Soma	Neuron
Dendrite	Input
Axon	Output
Synapse	Weight





Perceptron Learning



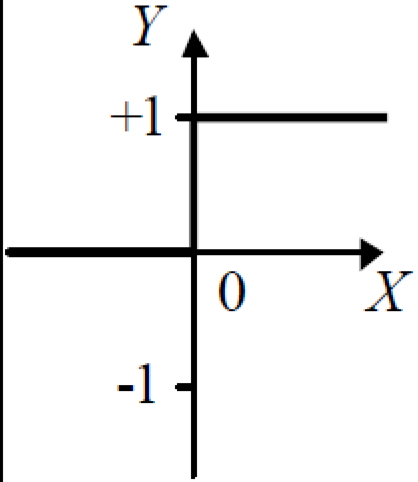
1 $$X = \sum_{i=1}^{n} x_i w_i$$

2 $$Y = \begin{cases} +1, & \text{if } X \geq \theta \\ -1, & \text{if } X < \theta \end{cases}$$

Sign Function

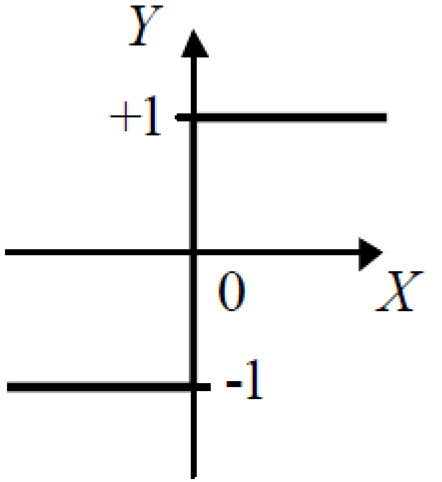
Activation Functions

Step function



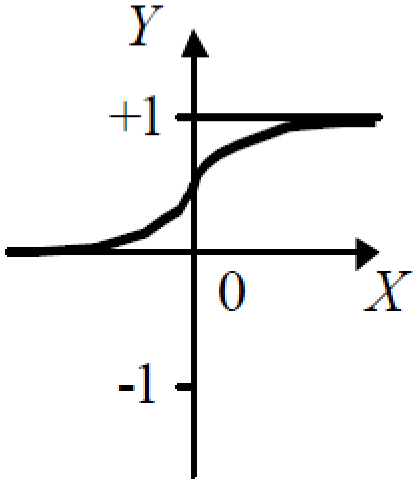
$$Y^{step} = \begin{cases} 1, \text{ if } X \geq 0 \\ 0, \text{ if } X < 0 \end{cases}$$

Sign function



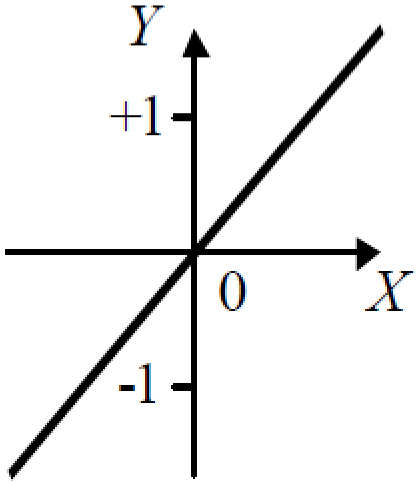
$$Y^{sign} = \begin{cases} +1, \text{ if } X \geq 0 \\ -1, \text{ if } X < 0 \end{cases}$$

Sigmoid function



$$Y^{sigmoid} = \frac{1}{1+e^{-X}}$$

Linear function

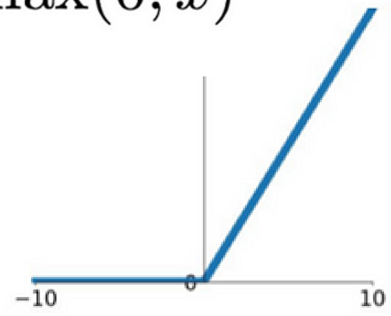


$$Y^{linear} = X$$

Rectified Linear Unit (RELU)

ReLU

$\max(0, x)$

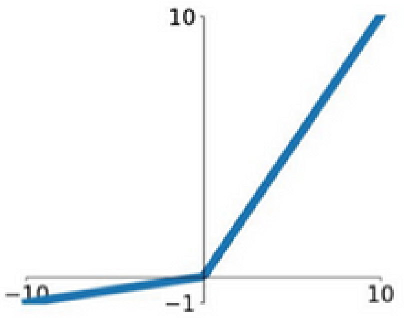


$$R(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Activation & Loss Functions

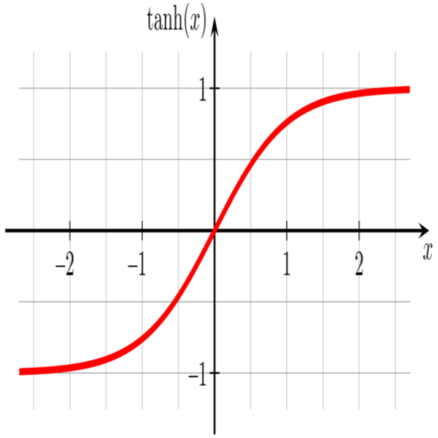
Leaky Rectified Linear Unit

Leaky ReLU
$\max(0.1x, x)$



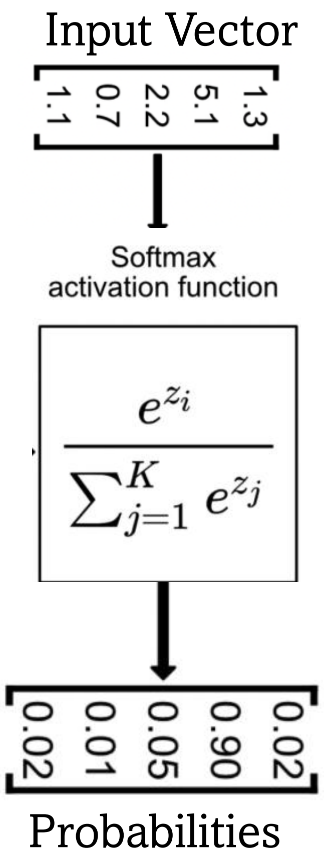
$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \le 0 \end{cases},$$

Hyperbolic tangent (Tan-h)



$$f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

SoftMax



Binary Cross Entropy

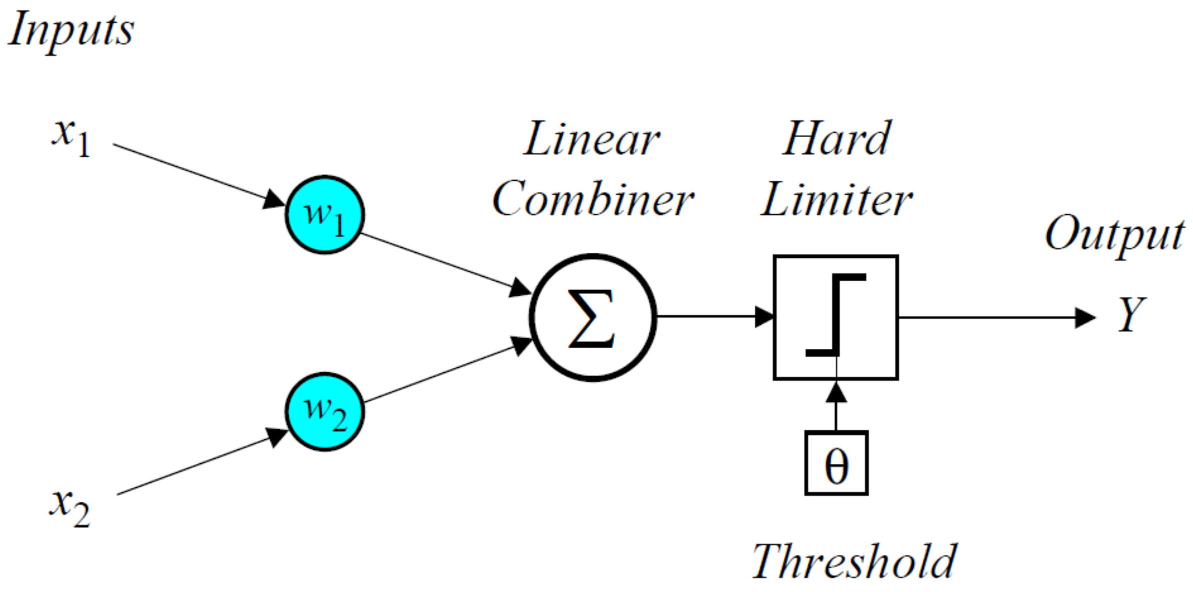
$$H_p(q) = -\frac{1}{N} \sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Categorical Cross Entropy

$$H(P^* | P) = -\sum_i \underbrace{P^*(i)}_{\text{TRUE CLASS DISTRIBUTION}} \log \underbrace{P(i)}_{\text{PREDICTED CLASS DISTRIBUTION}}$$

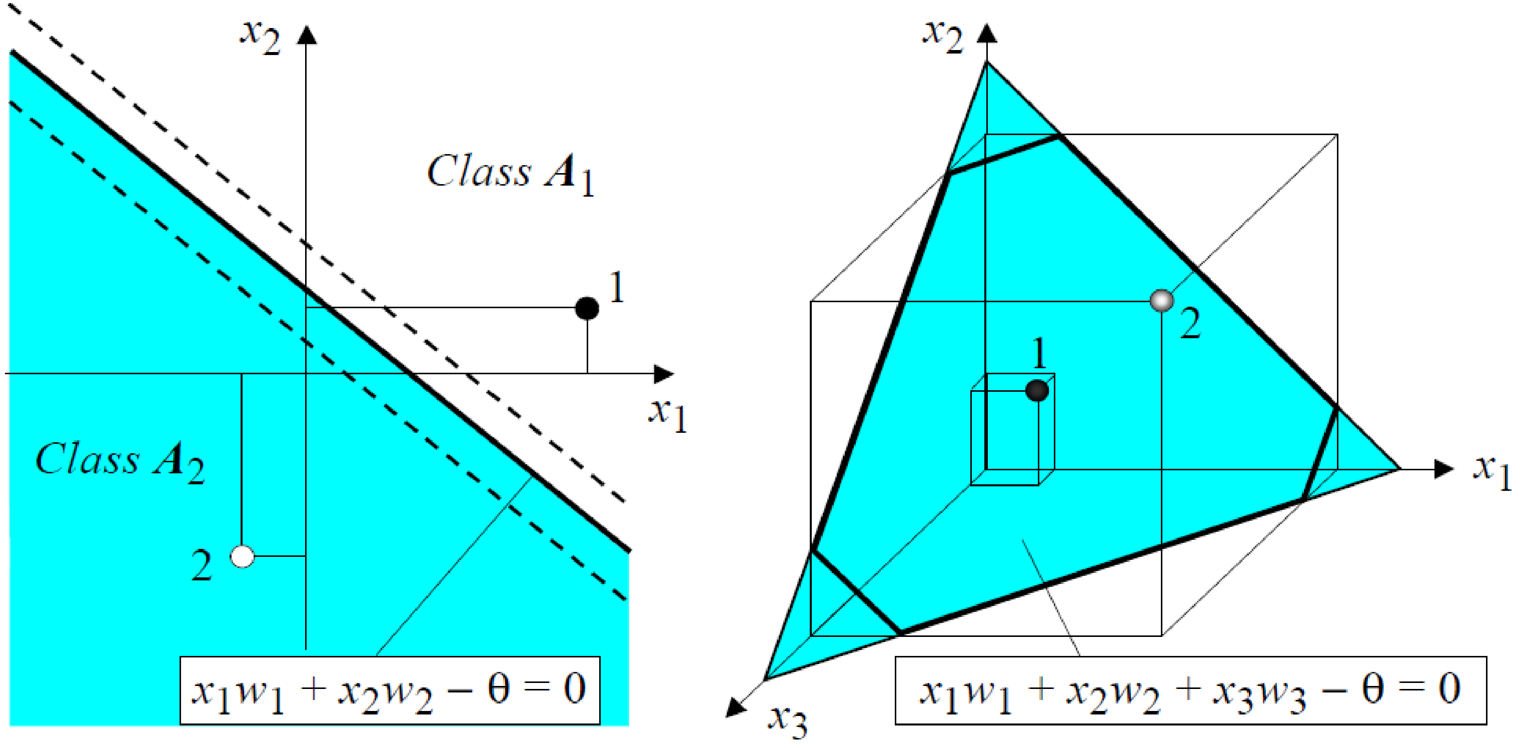
How a perceptron learns

- In 1958, Frank Rosenblatt introduced a training algorithm that provided the first procedure for training a simple ANN: a perceptron, inspired by **McCulloch and Pitts neuron model**



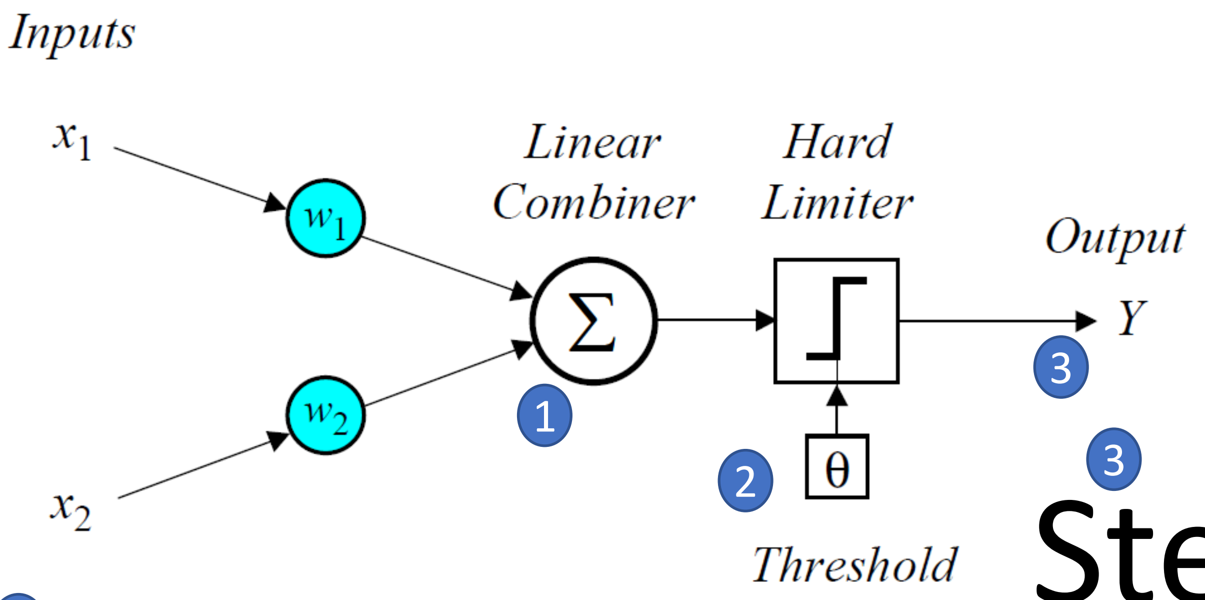
How a perceptron learns

- Decision boundary



(*a*) Two-input perceptron. (*b*) Three-input perceptron.

How a perceptron learns



AND Problem

x_1	x_2	y_d	w_1	w_2	y_a
0	0	0	0.3	-0.1	
0	1	0			
1	0	0			
1	1	1			

3

$$\text{Step}\left(\sum_{i=1}^{n} x_i w_i - \theta = 0\right)$$

1

$$X = \sum_{i=1}^{n} x_i w_i$$

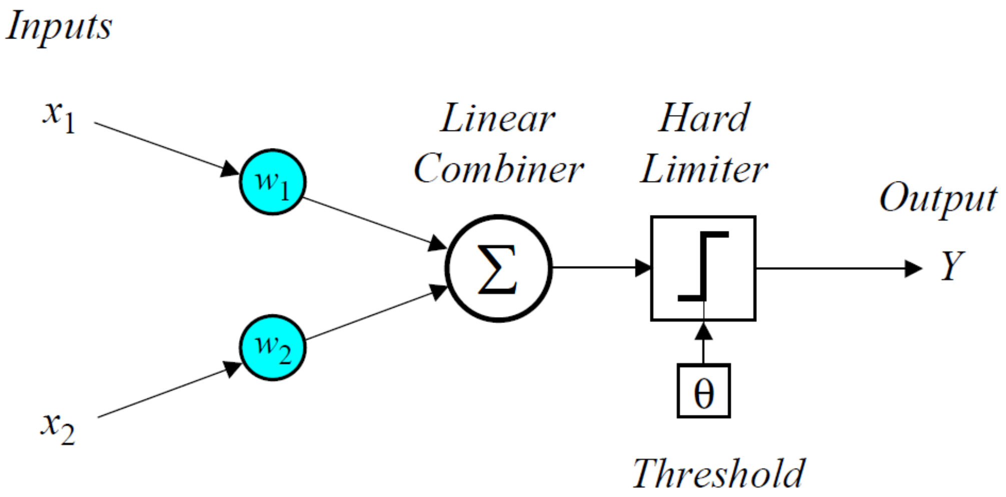
2

$$\sum_{i=1}^{n} x_i w_i - \theta = 0$$

Threshold: θ = 0.2; learning rate : α = 0.1; activation function : step

How a perceptron learns

x_1	x_2	y_d	w_1	w_2	y_a	$Error$	$w_1{}^{new}$	$w_2{}^{new}$
0	0	0	0.3	-0.1				
0	1	0						
1	0	0						
1	1	1						



Weight Updation

$$w_i(p+1) = w_i(p) + \Delta w_i(p)$$

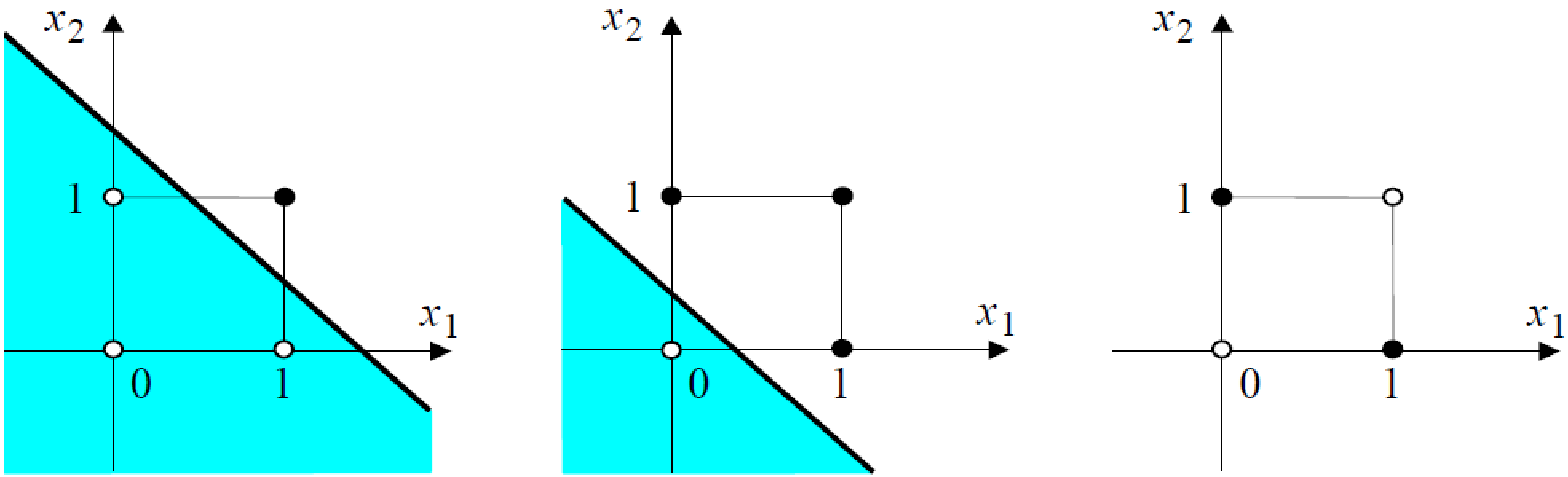
$$\Delta w_i(p) = \alpha \cdot x_i(p) \cdot e(p)$$

Threshold: $\theta = 0.2$; learning rate : $\alpha = 0.1$; activation function : step

How a perceptron learns

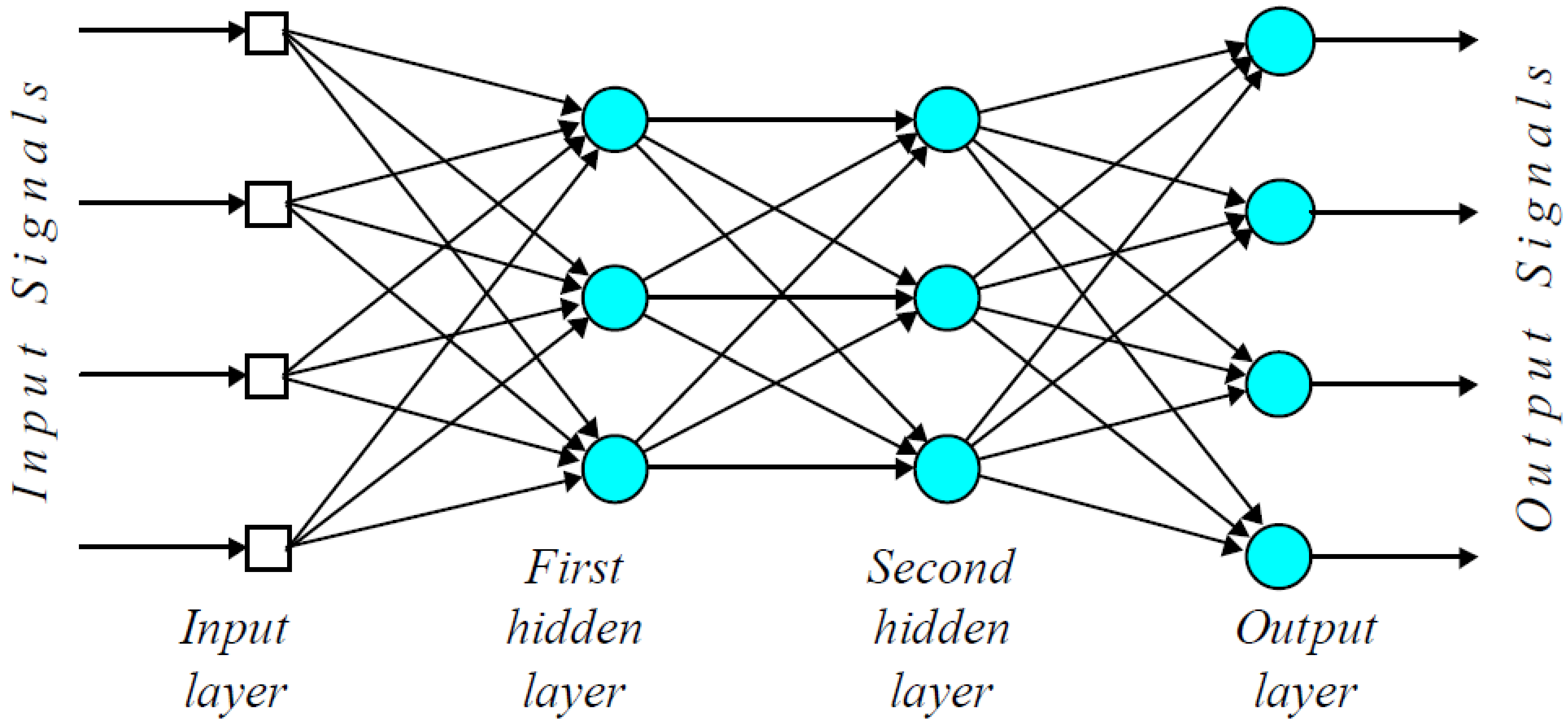
Epoch	x_1	x_2	y_d	w_1	w_2	y_a	$Error$	$w_1{}^{new}$	$w_2{}^{new}$	MSE
1	0	0	0	0.3	-0.1					
	0	1	0							
	1	0	0							
	1	1	1							
2	0	0	0							
	0	1	0							
	1	0	0							
	1	1	1							
3	0	0	0							
	0	1	0							
	1	0	0							
	1	1	1							

How a perceptron learns : Decision Boundary



(a) *AND* $(x_1 \cap x_2)$ (b) *OR* $(x_1 \cup x_2)$ (c) *Exclusive-OR* $(x_1 \oplus x_2)$

Multi Layer Perceptron



Multi Layer Perceptron: Weight Updation

Neuron output at layer “k”

$$y_k(p) = sigmoid\left[\sum_{j=1}^{m} x_{jk}(p) \cdot w_{jk}(p) - \theta_k\right]$$

Error at layer “k”

$$e_k(p) = y_{d,k}(p) - y_k(p)$$

Error gradient at layer “k”

$$\delta_k(p) = y_k(p) \cdot [1 - y_k(p)] \cdot e_k(p)$$

Change in weight

$$\Delta w_{jk}(p) = \alpha \cdot y_j(p) \cdot \delta_k(p)$$

Weight Updation

$$w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p)$$

Error gradient at layer “j”

$$\delta_j(p) = y_j(p) \cdot [1 - y_j(p)] \cdot \sum_{k=1}^{l} \delta_k(p)\, w_{jk}(p)$$

Change in weight

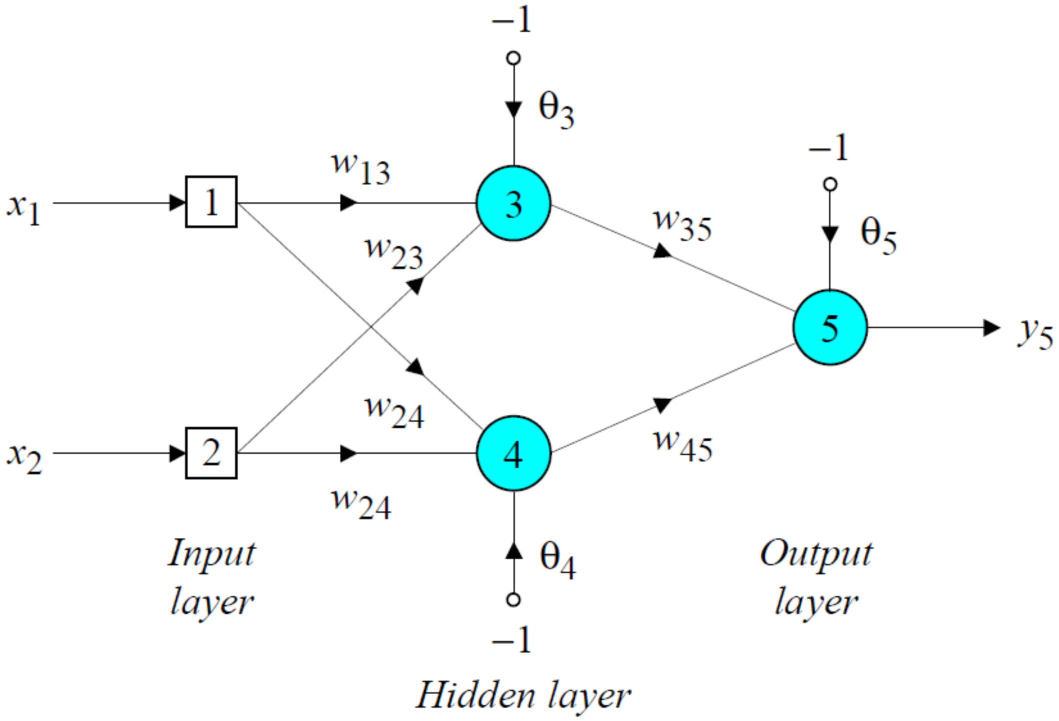
$$\Delta w_{ij}(p) = \alpha \cdot x_i(p) \cdot \delta_j(p)$$

Weight Updation

$$w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$$

XOR Problem

x_1	x_2	w_{13}	w_{23}	θ_3	y_3	w_{14}	w_{24}	θ_4	y_4	w_{35}	w_{45}	θ_5	y_d	y_5	E	w_{13}'	w_{23}'	θ_4'	w_{14}'	w_{24}'	θ_4'	w_{35}'	w_{45}'	θ_5'
1	1	0.5	0.4	0.8		0.9	1.0	-0.1		-1.2	1.1	0.3	0											
0	1												1											
1	0												1											
0	0												0											



$$y_3 = sigmoid\ (x_1 w_{13} + x_2 w_{23} - \theta_3) = 1/\left[1 + e^{-(1\cdot 0.5 + 1\cdot 0.4 - 1\cdot 0.8)}\right] = 0.5250$$

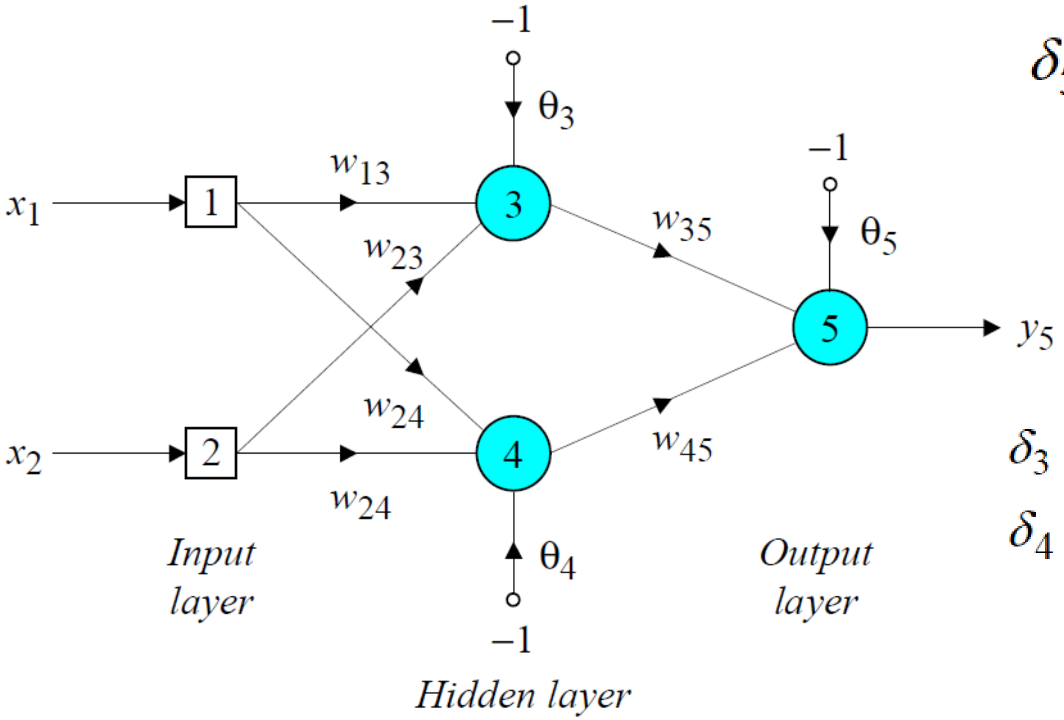
$$y_4 = sigmoid\ (x_1 w_{14} + x_2 w_{24} - \theta_4) = 1/\left[1 + e^{-(1\cdot 0.9 + 1\cdot 1.0 + 1\cdot 0.1)}\right] = 0.8808$$

$$y_5 = sigmoid(y_3 w_{35} + y_4 w_{45} - \theta_5) = 1/\left[1 + e^{-(-0.52501.2 + 0.88081.1 - 1\cdot 0.3)}\right] = 0.5097$$

$$e = y_{d,5} - y_5 = 0 - 0.5097 = -0.5097$$

XOR Problem

x_1	x_2	w_{13}	w_{23}	θ_3	y_3	w_{14}	w_{24}	θ_4	y_4	w_{35}	w_{45}	θ_5	y_d	y_5	E	w_{13}'	w_{23}'	θ_4'	w_{14}'	w_{24}'	θ_4'	w_{35}'	w_{45}'	θ_5'
1	1	0.5	0.4	0.8	0.52	0.9	1.0	-0.1	0.88	-1.2	1.1	0.3	0	0.50	-0.5									
0	1												1											
1	0												1											
0	0												0											



$$\delta_5 = y_5\,(1-y_5)\,e = 0.5097\cdot(1-0.5097)\cdot(-0.5097) = -0.1274$$

$$\Delta w_{35} = \alpha\cdot y_3\cdot\delta_5 = 0.1\cdot 0.5250\cdot(-0.1274) = -0.0067$$

$$\Delta w_{45} = \alpha\cdot y_4\cdot\delta_5 = 0.1\cdot 0.8808\cdot(-0.1274) = -0.0112$$

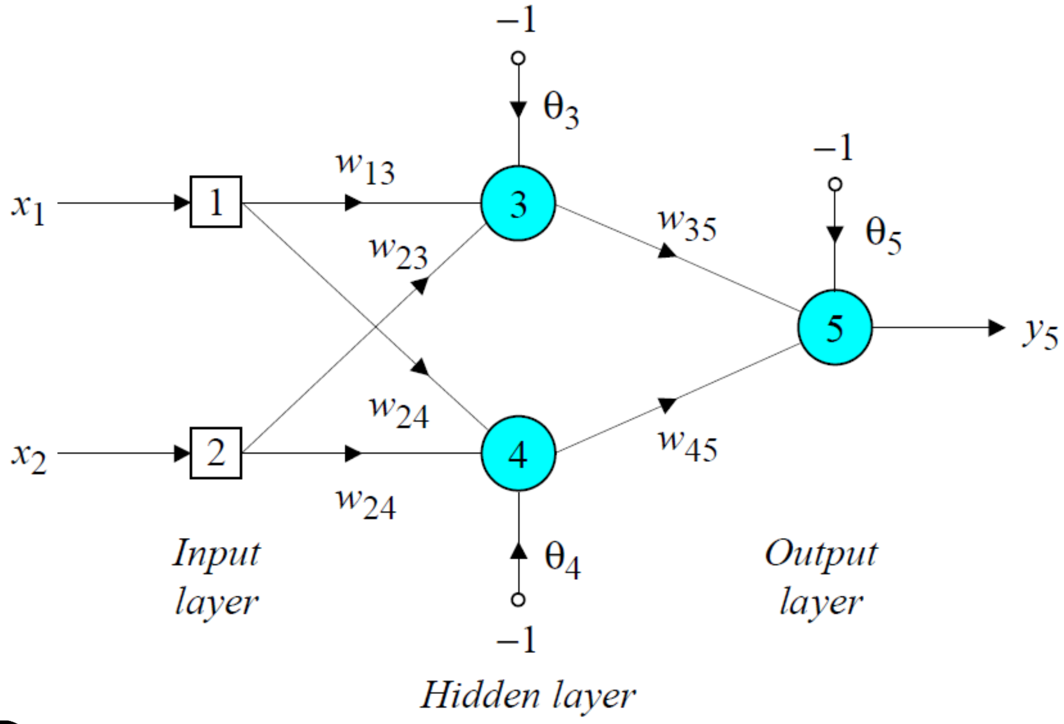
$$\Delta\theta_5 = \alpha\cdot(-1)\cdot\delta_5 = 0.1\cdot(-1)\cdot(-0.1274) = -0.0127$$

$$\delta_3 = y_3(1-y_3)\cdot\delta_5\cdot w_{35} = 0.5250\cdot(1-0.5250)\cdot(-0.1274)\cdot(-1.2) = 0.0381$$

$$\delta_4 = y_4(1-y_4)\cdot\delta_5\cdot w_{45} = 0.8808\cdot(1-0.8808)\cdot(-0.127\,4)\cdot 1.1 = -0.0147$$

XOR Problem

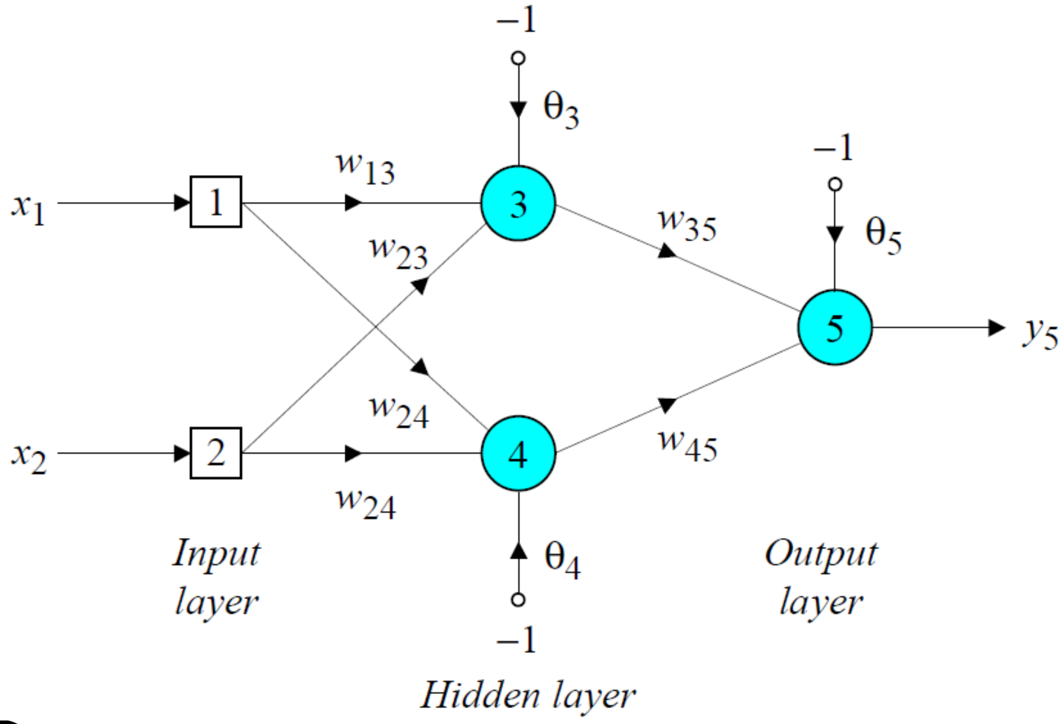
x_1	x_2	w_{13}	w_{23}	θ_3	y_3	w_{14}	w_{24}	θ_4	y_4	w_{35}	w_{45}	θ_5	y_d	y_5	E	w_{13}'	w_{23}'	θ_4'	w_{14}'	w_{24}'	θ_4'	w_{35}'	w_{45}'	θ_5'
1	1	0.5	0.4	0.8	0.52	0.9	1.0	-0.1	0.88	-1.2	1.1	0.3	0	0.50	-0.5									
0	1												1											
1	0												1											
0	0												0											



$$\Delta w_{13} = \alpha \cdot x_1 \cdot \delta_3 = 0.1 \cdot 1 \cdot 0.0381 = 0.0038$$
$$\Delta w_{23} = \alpha \cdot x_2 \cdot \delta_3 = 0.1 \cdot 1 \cdot 0.0381 = 0.0038$$
$$\Delta \theta_3 = \alpha \cdot (-1) \cdot \delta_3 = 0.1 \cdot (-1) \cdot 0.0381 = -0.0038$$
$$\Delta w_{14} = \alpha \cdot x_1 \cdot \delta_4 = 0.1 \cdot 1 \cdot (-0.0147) = -0.0015$$
$$\Delta w_{24} = \alpha \cdot x_2 \cdot \delta_4 = 0.1 \cdot 1 \cdot (-0.0147) = -0.0015$$
$$\Delta \theta_4 = \alpha \cdot (-1) \cdot \delta_4 = 0.1 \cdot (-1) \cdot (-0.0147) = 0.0015$$

XOR Problem

x_1	x_2	w_{13}	w_{23}	θ_3	y_3	w_{14}	w_{24}	θ_4	y_4	w_{35}	w_{45}	θ_5	y_d	y_5	E	w_{13}'	w_{23}'	θ_3'	w_{14}'	w_{24}'	θ_4'	w_{35}'	w_{45}'	θ_5'
1	1	0.5	0.4	0.8	0.52	0.9	1.0	-0.1	0.88	-1.2	1.1	0.3	0	0.50	-0.5	0.50	0.40	0.79	0.89	0.99	-.09	-1.2	1.08	0.31
0	1												1											
1	0												1											
0	0												0											



$$w_{13} = w_{13} + \Delta w_{13} = 0.5 + 0.0038 = 0.5038$$

$$w_{14} = w_{14} + \Delta w_{14} = 0.9 - 0.0015 = 0.8985$$

$$w_{23} = w_{23} + \Delta w_{23} = 0.4 + 0.0038 = 0.4038$$

$$w_{24} = w_{24} + \Delta w_{24} = 1.0 - 0.0015 = 0.9985$$

$$w_{35} = w_{35} + \Delta w_{35} = -1.2 - 0.0067 = -1.2067$$

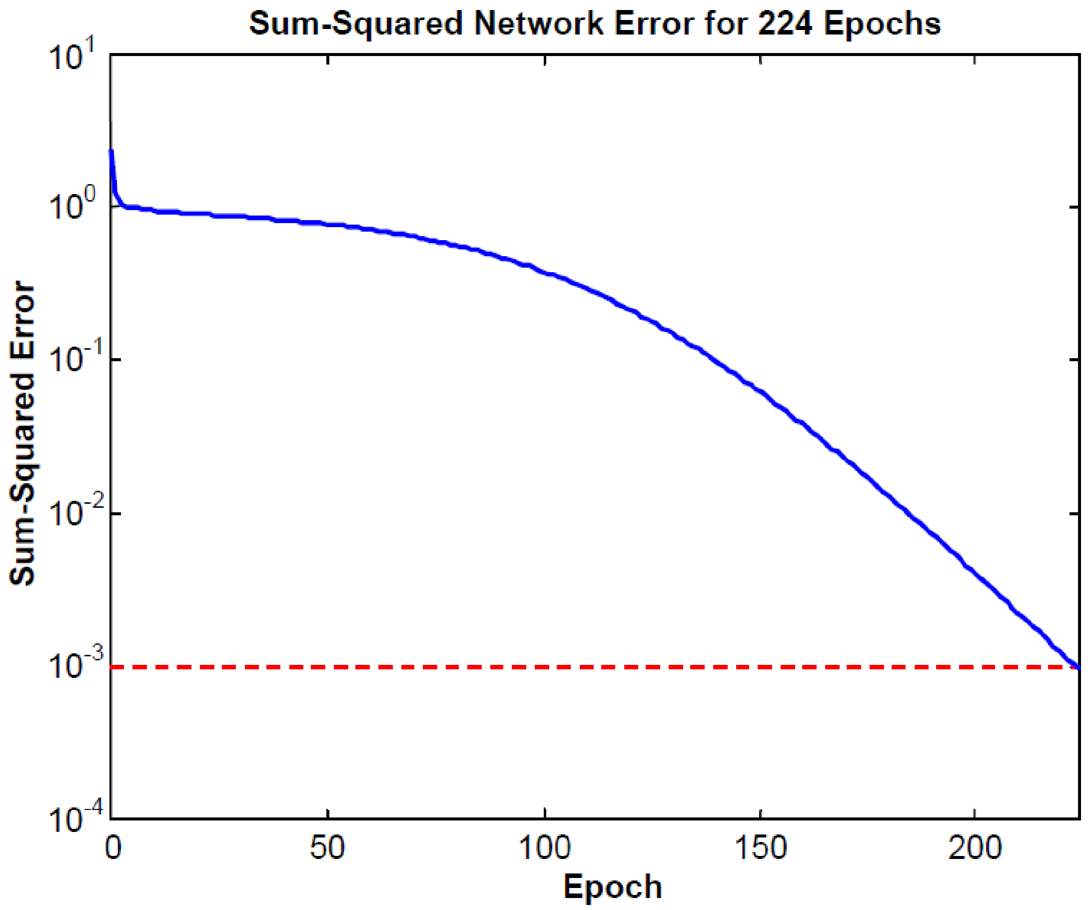
$$w_{45} = w_{45} + \Delta w_{45} = 1.1 - 0.0112 = 1.0888$$

$$\theta_3 = \theta_3 + \Delta\theta_3 = 0.8 - 0.0038 = 0.7962$$

$$\theta_4 = \theta_4 + \Delta\theta_4 = -0.1 + 0.0015 = -0.0985$$

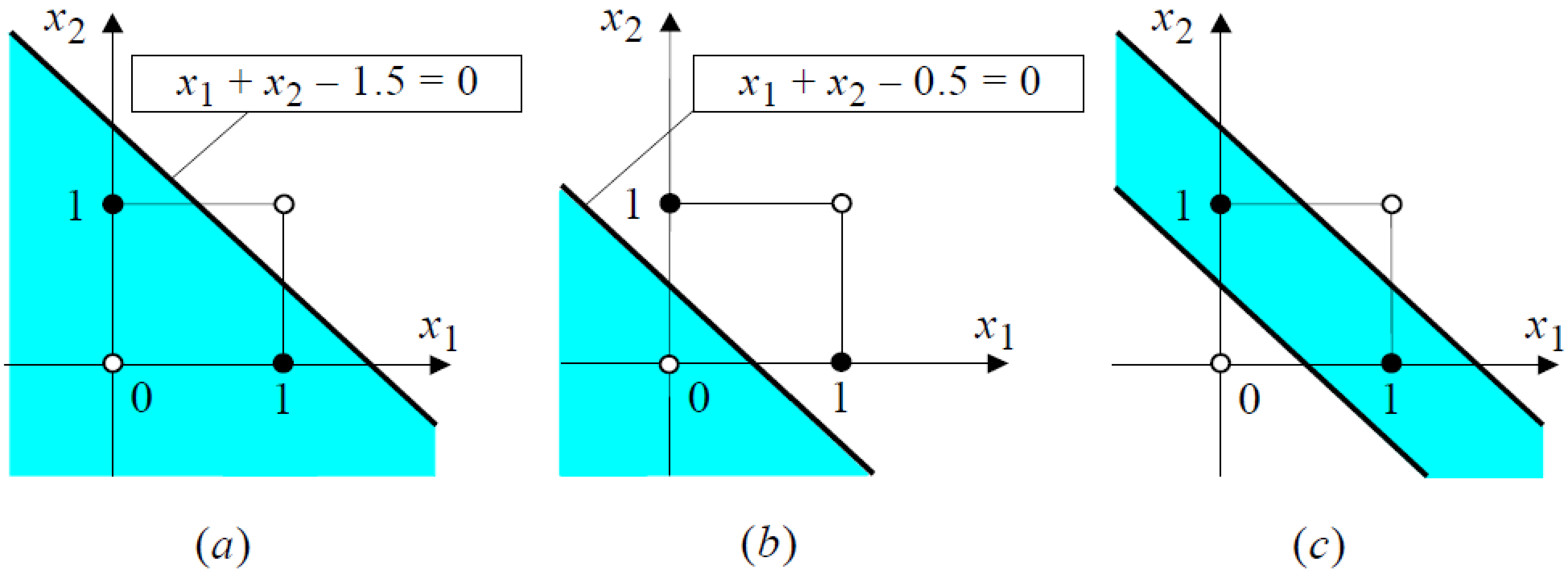
$$\theta_5 = \theta_5 + \Delta\theta_5 = 0.3 + 0.0127 = 0.3127$$

XOR Problem



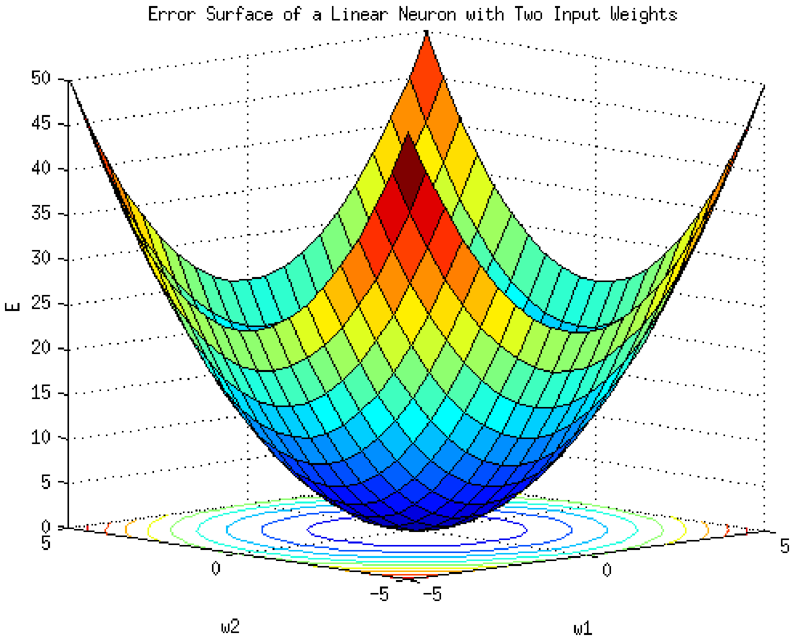
Inputs		Desired output	Actual output	Error	Sum of squared errors
x_1	x_2	y_d	y_5	e	
1	1	0	0.0155	–0.0155	0.0010
0	1	1	0.9849	0.0151	
1	0	1	0.9849	0.0151	
0	0	0	0.0175	–0.0175	

XOR Problem



- a) Decision boundary created by hidden neuron 3
- b) Decision boundary created by hidden neuron 4
- c) Decision boundary created by output neuron 5

Gradient Descent



🔁 Step 1: **Initialize Parameters**

Choose initial values for the model parameters (weights `w` and bias `b` or `θ`), usually random or zero.

> Example:

$$w = \text{random value}, \quad b = \text{random value}$$

🧮 Step 2: **Compute the Gradient**

Calculate the derivative of the **loss function** with respect to each parameter to understand the slope (direction and steepness) of the loss surface.

> For Mean Squared Error (MSE) loss:

$$L(w) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$\frac{\partial L}{\partial w} = -\frac{2}{n}\sum_{i=1}^{n}x_i(y_i - \hat{y}_i)$$

$$\frac{\partial L}{\partial b} = -\frac{2}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)$$

🏃 Step 3: **Update Parameters**

Move the parameters in the direction opposite to the gradient to minimize the loss.

> General update rule:

$$\theta := \theta - \eta \cdot \frac{\partial L}{\partial \theta}$$

Where:

- θ = parameter (like weight or bias),
- η = learning rate (small positive number),
- $\frac{\partial L}{\partial \theta}$ = gradient of loss w.r.t. parameter.

> Example for weight:

$$w := w - \eta \cdot \frac{\partial L}{\partial w}$$

$$b := b - \eta \cdot \frac{\partial L}{\partial b}$$