REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Khider-BISKRA.

Faculté des sciences exactes et des sciences de la nature et de la vie

Département d'informatique

Machine Learning (ML)

Dr. TIBERMACINE Ahmed

Content Summary

Chapter 1: Artificial Neural network

Introduction

- In this chapter :
- we will explore the foundational concepts of neural networks, starting with the perceptron and the multi-layer perceptron (MLP).

- We will then delve into deep learning, focusing on Convolutional Neural Networks (CNNs) – a critical advancement in the field of artificial intelligence.

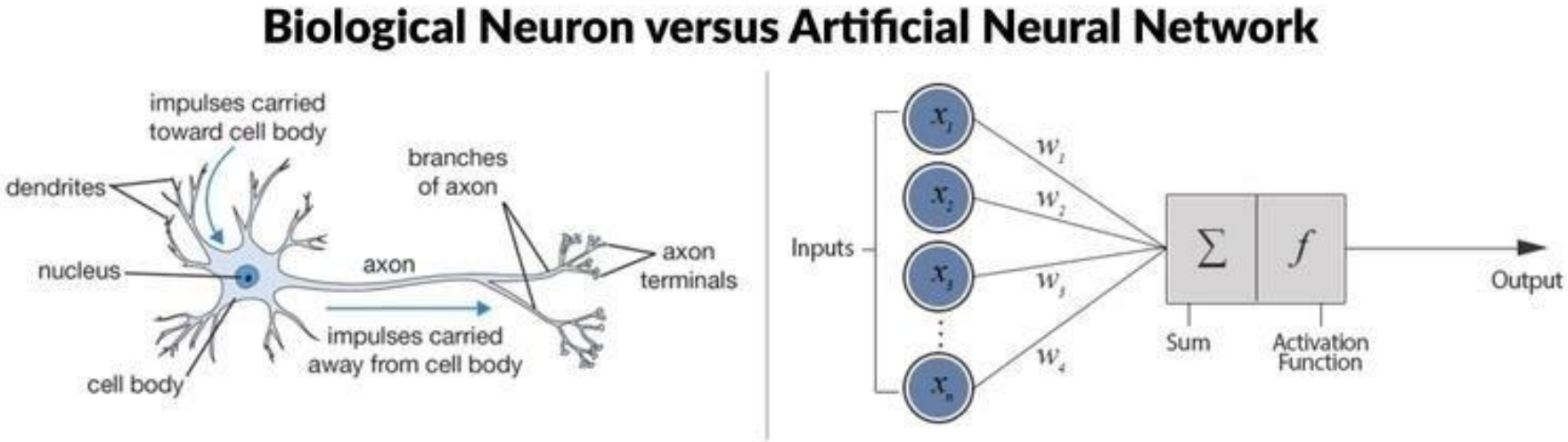
Learning Objectives

- Understand the basic structure and functioning of a perceptron.
- Learn the limitations of a single-layer perceptron.
- Explore the need for multi-layer perceptron in solving complex tasks.
- Gain insights into deep learning, its applications, and the emergence of CNNs.
- Comprehend the architecture and training of CNNs for image-related tasks.

Perceptron

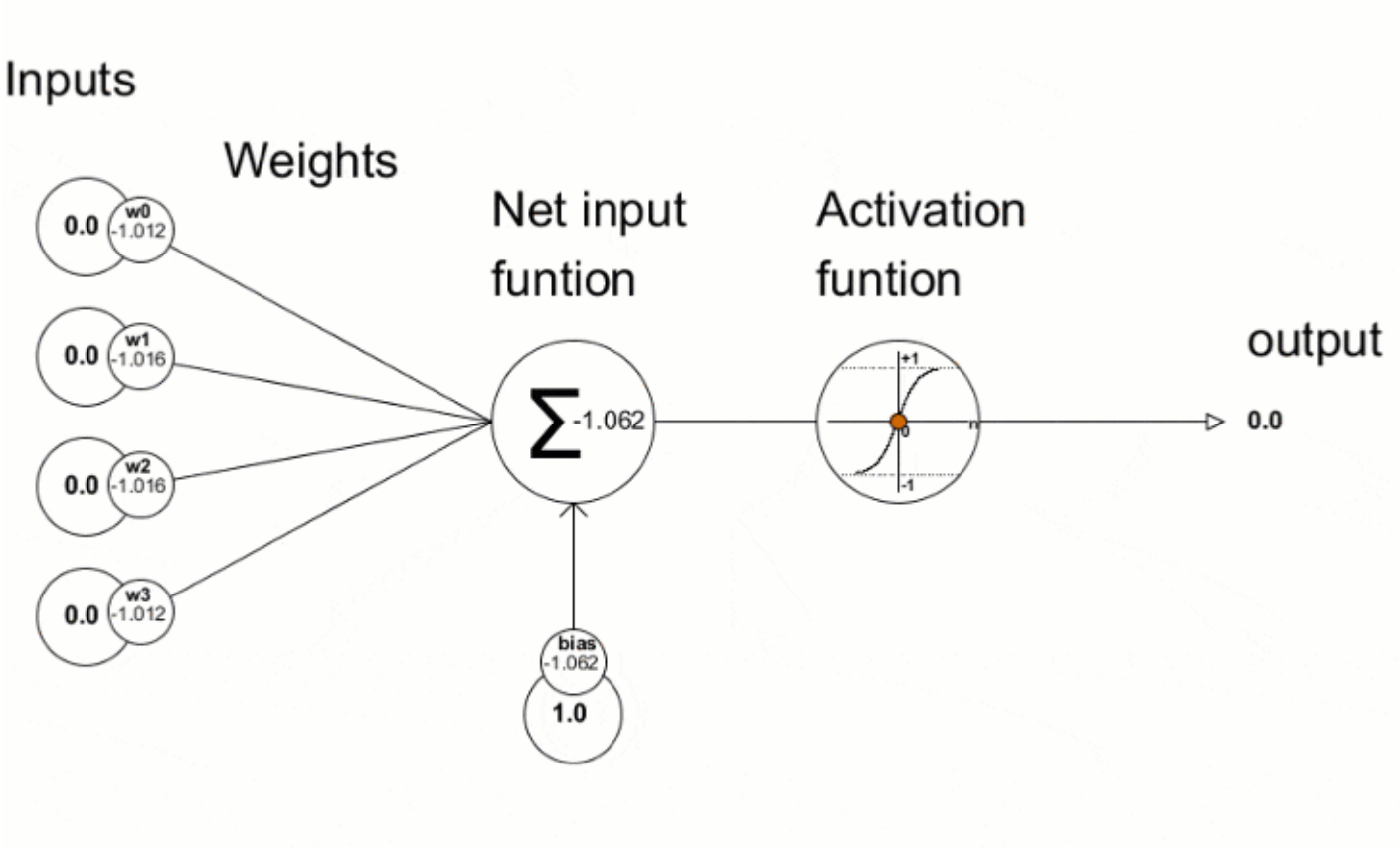
Perceptron: The Building Blocks

- **Biological Inspiration**



Perceptron: The Building Blocks

- **Structure of a Perceptron**



PERCEPTRON

Perceptron Training Algorithm - Single Output Class

- Initialize the weights and the bias. Also initialize the learning rate α ($0 < \alpha \leq 1$).
- Until the final stopping condition is false.
  - for each training pair indicated by s:t.
    - Set each input unit i = 1 to n: $x_i = s_i$
    - Calculate the output of the network.

$$y_{in} = b + \sum_{i=1}^{n} x_i w_i$$

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > \theta \\ 0 & \text{if } -\theta \leq y_{in} \leq \theta \\ -1 & \text{if } y_{in} < -\theta \end{cases}$$

    - Weight and bias adjustment:

If $y \neq t$, then

$$w_i(\text{new}) = w_i(\text{old}) + \alpha t x_i$$

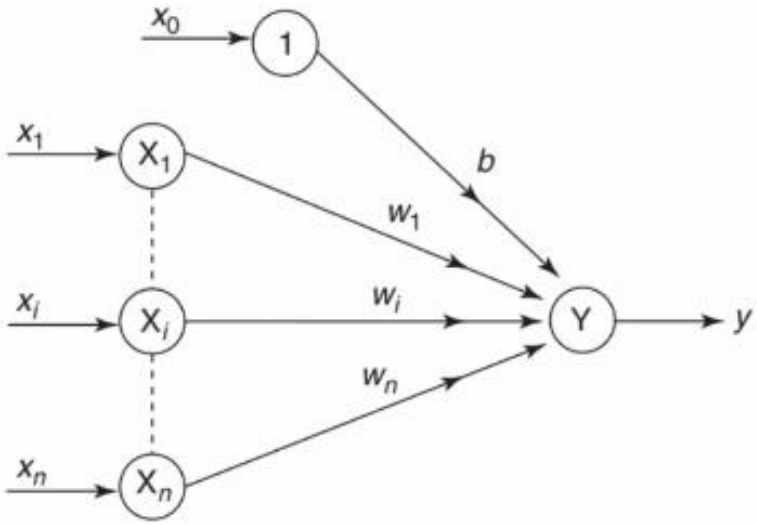
$$b(\text{new}) = b(\text{old}) + \alpha t$$

else we have

$$w_i(\text{new}) = w_i(\text{old})$$

$$b(\text{new}) = b(\text{old})$$

  - Train the network until there is no weight change.



PERCEPTRON

Perceptron Network Testing Algorithm

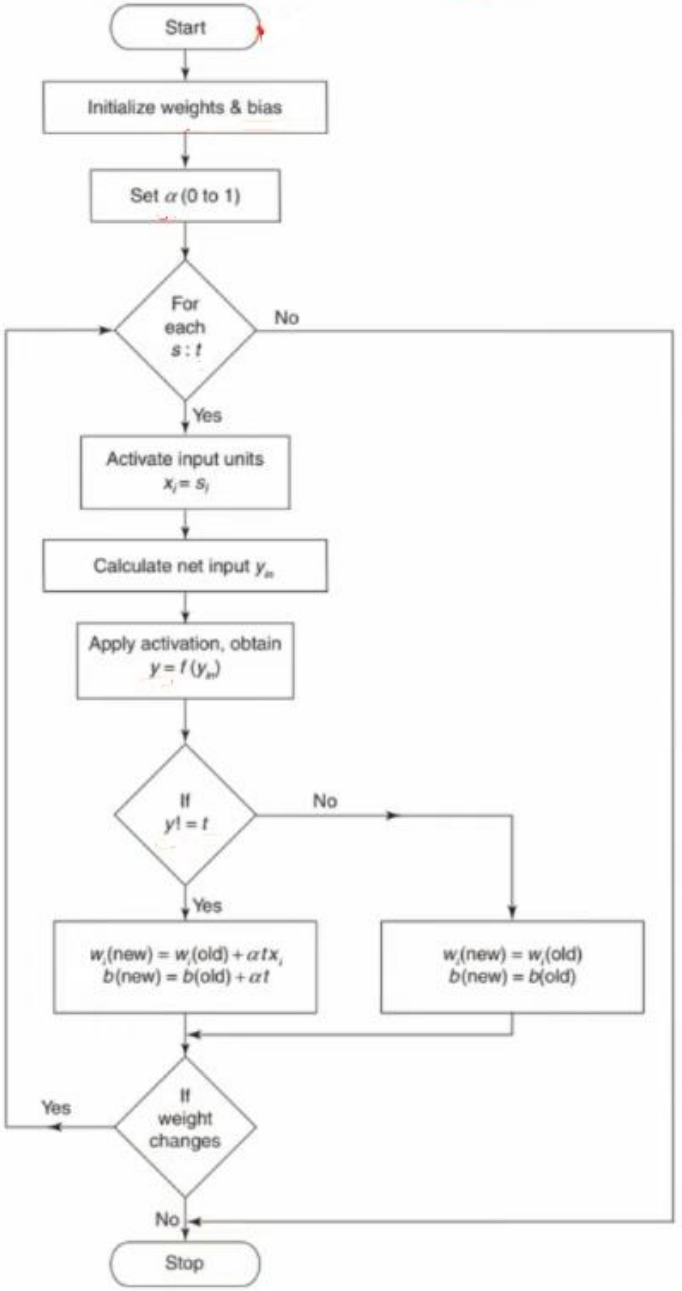
- The initial weights to be used here are taken from the training algorithms (the final weights obtained during training).
- For each input vector X to be classified, perform the following
  - Calculate the net input of the unit.
  - Obtain the response of output unit.

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

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > \theta \\ 0 & \text{if } -\theta \le y_{in} \le \theta \\ -1 & \text{if } y_{in} < -\theta \end{cases}$$

Flowchart of Perceptron Learning Rule

Single Output Class

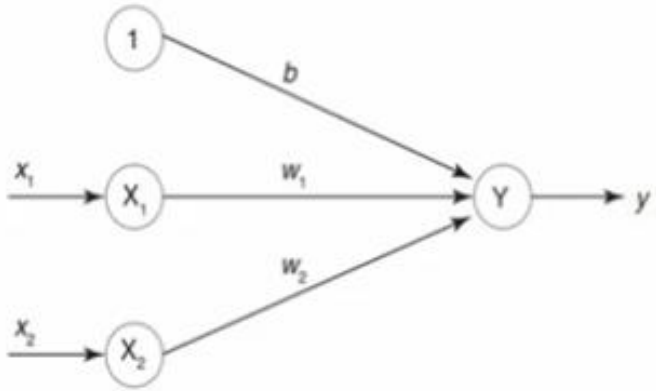


PERCEPTRON

AND function using Perceptron Rule Solved Example

- The perceptron network, which uses perceptron learning rule, is used to train the AND function.
- The input patterns are presented to the network one by one.
- When all the four input patterns are presented, then one epoch is said to be completed.
- The initial weights and threshold are set to zero.
- The learning rate a is set equal to 1.

x_1	x_2	t
1	1	1
1	−1	−1
−1	1	−1
−1	−1	−1



PERCEPTRON

AND function using Perceptron Rule Solved Example

$$y_{in} = b + x_1 w + x_2 w$$

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > 0 \\ 0 & \text{if } y_{in} = 0 \\ -1 & \text{if } y_{in} < 0 \end{cases}$$

$$\Delta w_1 = \alpha t x_1; \quad \Delta w_2 = \alpha t x_2; \quad \Delta b = \alpha t$$

If $y \neq t$, then

$$w_i(\text{new}) = w_i(\text{old}) + \alpha t x_i$$

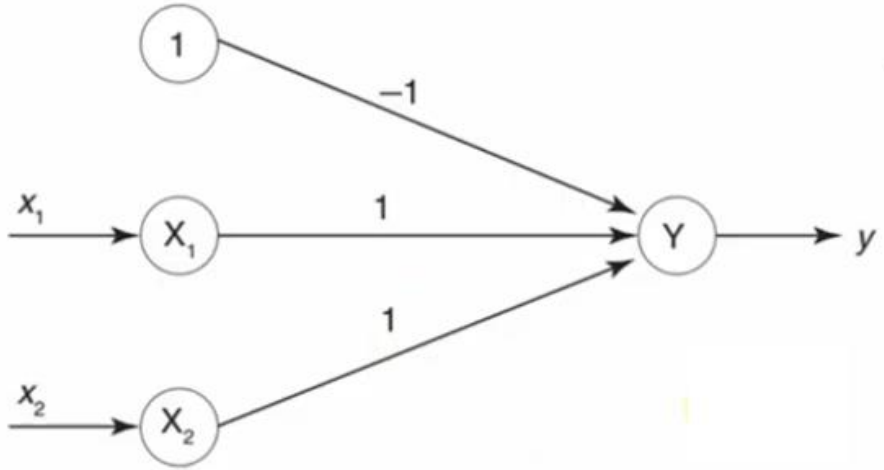
$$b(\text{new}) = b(\text{old}) + \alpha t$$

else we have

$$w_i(\text{new}) = w_i(\text{old})$$

$$b(\text{new}) = b(\text{old})$$

Input		Target	Net input	Calculated output	Weight changes			Weights		
x_1	x_2	(t)	(y_{in})	(y)	Δw_1	Δw_2	Δb	w_1 (0	w_2 0	b 0)
EPOCH-1										
1	1	1	0	0	1	1	1	1	1	1
1	−1	−1	1	1	−1	1	−1	0	2	0
−1	1	−1	2	1	+1	−1	−1	1	1	−1
−1	−1	−1	−3	−1	0	0	0	1	1	−1
					0	0	0	1	1	−1
					0	0	0	1	1	−1
					0	0	0	1	1	−1
					0	0	0	1	1	−1



PERCEPTRON

Perceptron Network (Rule) Solved Example

- Find the weights required to perform the following classification using perceptron network.
- The vectors (1, 1, 1, 1) and (- 1, 1 - 1, - 1) are belonging to the class 1, vectors (1, 1, 1, - 1) and (1, - 1, - 1, 1) are belonging to the class -1.
- Assume learning rate as 1
- and Initial weights as 0.

Input					Target
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

PERCEPTRON

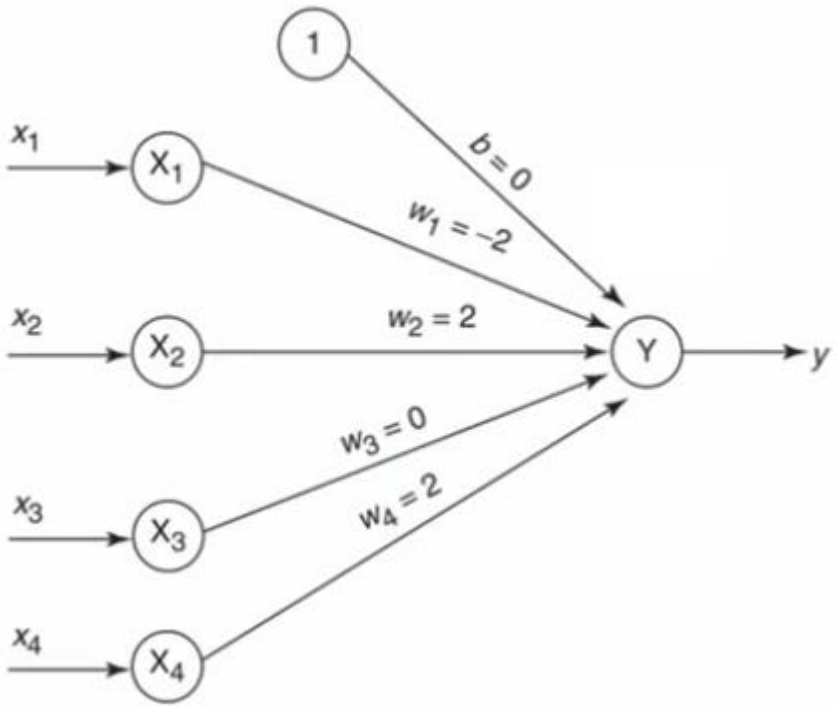
Perceptron Network (Rule) Solved Example

$$y_{in} = b + x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4$$

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > 0 \\ 0 & \text{if } y_{in} = 0 \\ -1 & \text{if } y_{in} < 0 \end{cases}$$

$$\Delta w_1 = \alpha t x_1;\quad \Delta w_2 = \alpha t x_2;\quad \Delta w_3 = \alpha t x_3;\quad \Delta w_4 = \alpha t x_4;\quad \Delta b = \alpha t$$

Inputs				Target	Net input	output	Weight changes					Weights				
$(x_1$	x_2	x_3	x_4	(t)	(y_{in})	(y)	$(\Delta w_1$	Δw_2	Δw_3	Δw_4	$\Delta b)$	w_1 (0	w_2 0	w_3 0	w_4 0	b 0)
EPOCH-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	0	2	0	0	2
(1	1	1	-1	-1	4	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
EPOCH-2																
(1	1	1	1	1	0	0	1	1	1	1	1	-1	3	1	1	1
												0	0	-1	3	1	1	1
												1	-1	-2	2	0	2	0
												0	0	-2	2	0	2	0
												0	0	-2	2	0	2	0
												0	0	-2	2	0	2	0
												0	0	-2	2	0	2	0
												0	0	-2	2	0	2	0

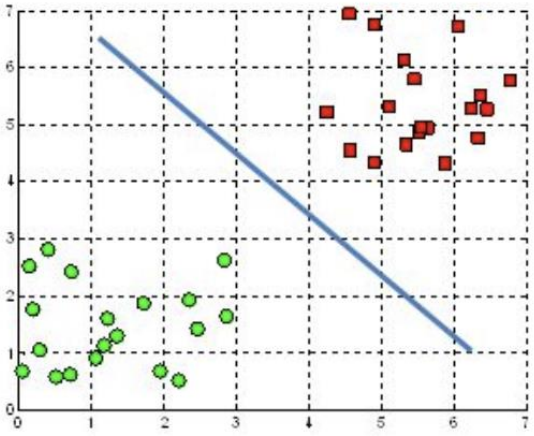


Perceptron

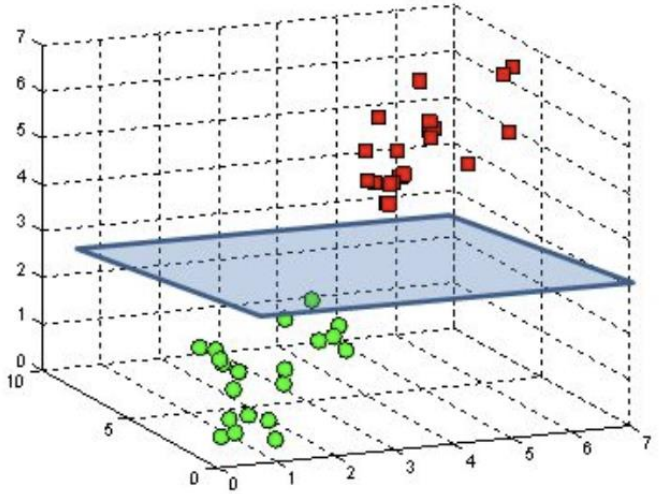
Perceptron for Binary Classification

- perceptron can be used as a **binary classification model**, defining a **linear decision boundary**.
- It finds the separating hyperplane that minimizes the distance between misclassified points and the decision boundary.

A hyperplane in $\mathbb{R}^2$ is a line



A hyperplane in $\mathbb{R}^3$ is a plane

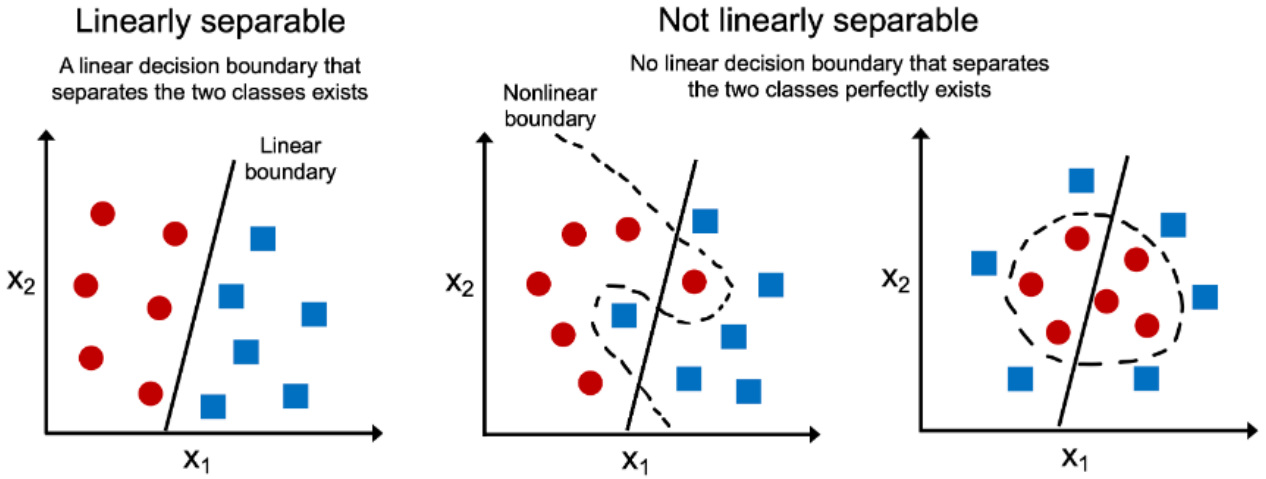


PERCEPTRON

Biggest criticism:

- it couldn't represent the **XOR gate** **(exclusive OR).**
- Perceptron can't be applied to non-linear data (Minsky and Papert, in 1969[5]).

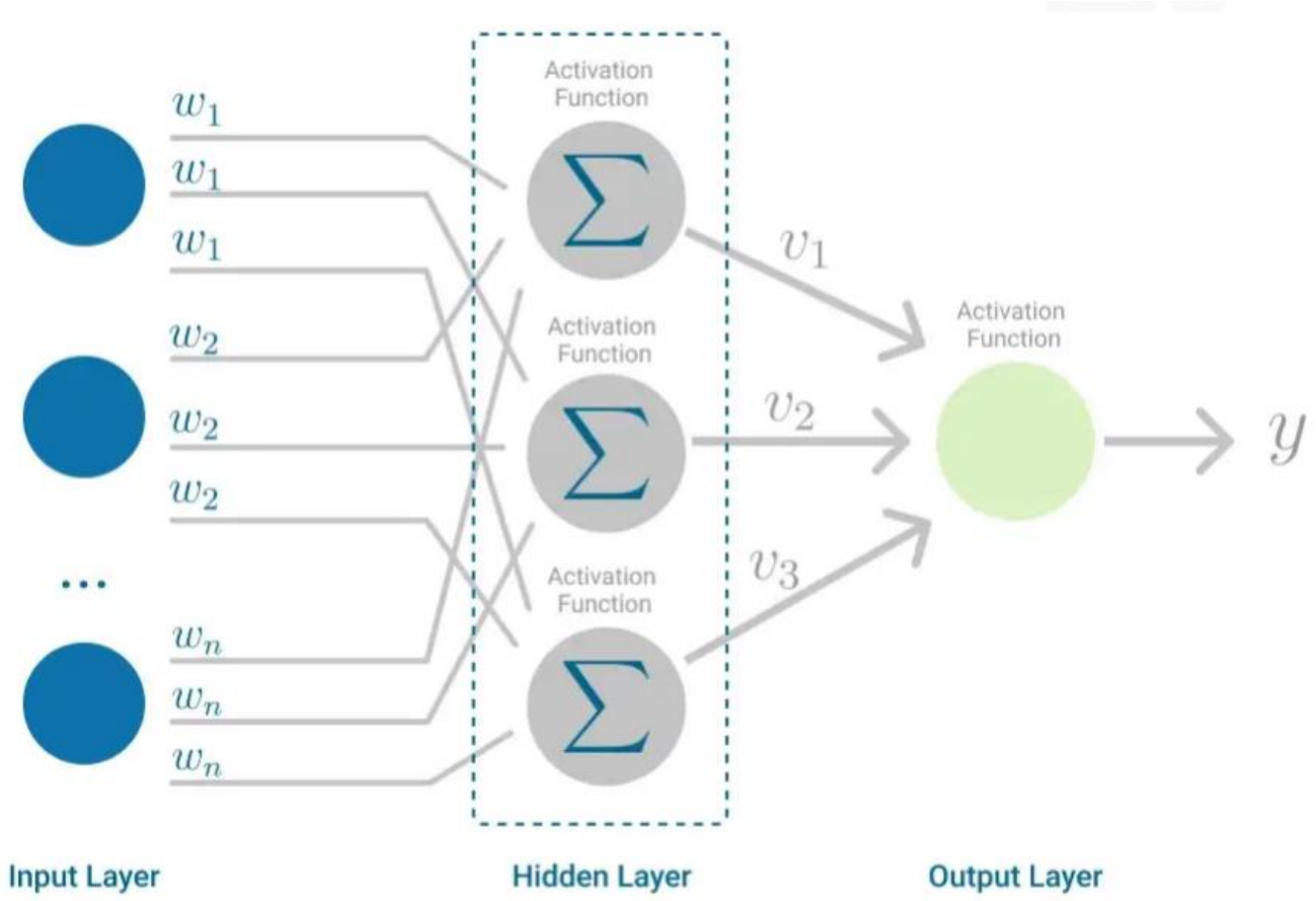
linearly separable and non linearly separable data:



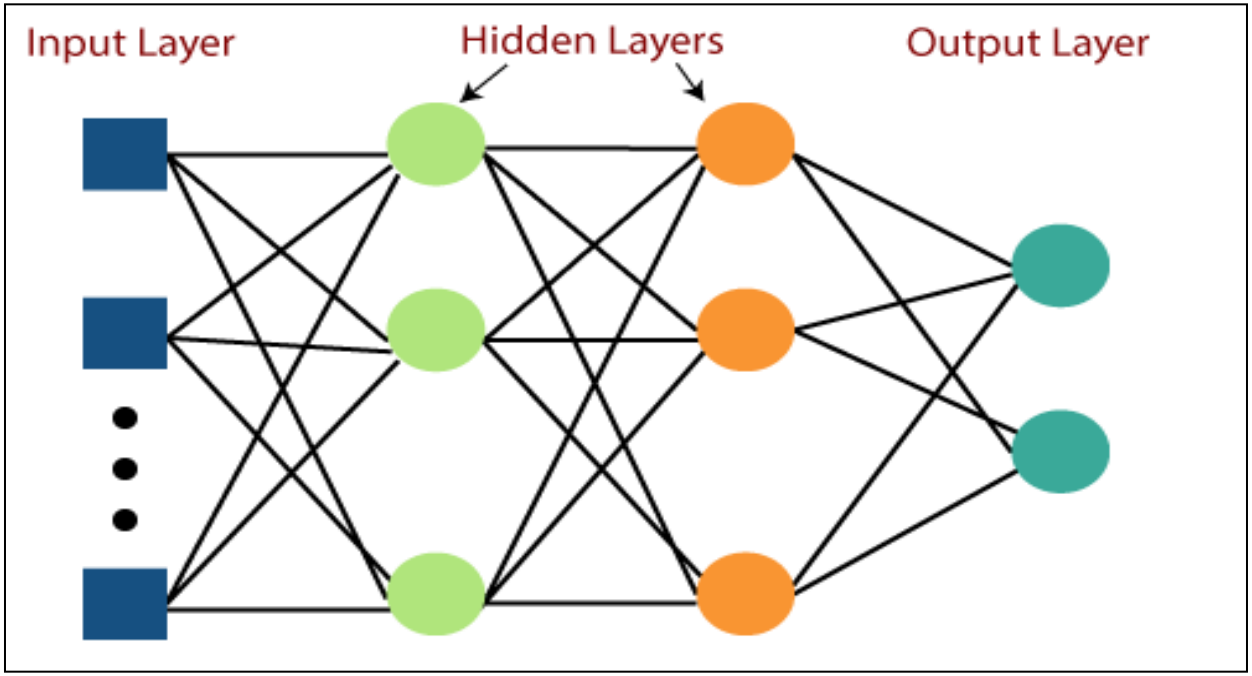
Multilayer Perceptron

"vanilla" neural networks

Multilayer Perceptron



Multilayer Perceptron



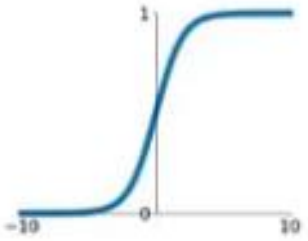
Activation Function

❖Types of Activation Functions

1. *Linear Activation Functions* (f(x)=x)
2. Non-Linear Activation Functions

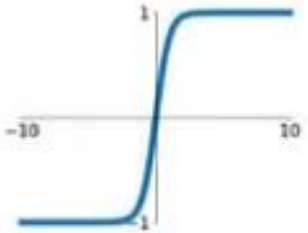
Sigmoid

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



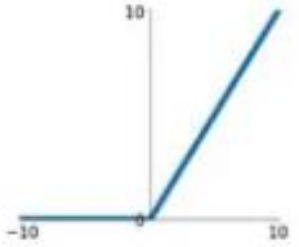
tanh

$$\tanh(x)$$



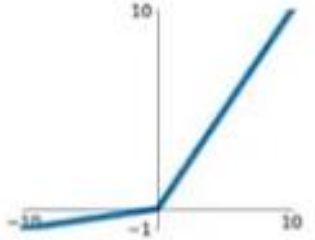
ReLU

$$\max(0, x)$$



Leaky ReLU

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

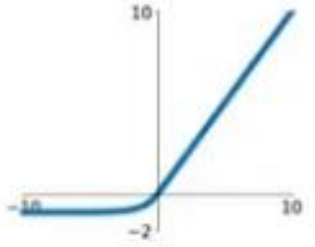


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Cheatsheet of most frequently used Activation Functions

Multilayer Perceptron

Training a Multi-Layer Perceptron (MLP)

- The main algorithm used to train MLPs is a combination of two key techniques:
  - Backpropagation
  - Gradient Descent.
- This combination is often referred to as:

  Backpropagation with Gradient Descent

- The primary goal is to adjust the weights and biases of the network in such a way that it can make accurate predictions

Multilayer Perceptron

- **Training a Multi-Layer Perceptron (MLP)**
- Here's an overview of the main steps in training an MLP using this algorithm:

1. Initialization
2. Forward Propagation
3. Loss Computation
4. Back-propagation (gradient calculation)
5. Gradient Descent
6. Iteration
7. Evaluation

Multilayer Perceptron

- **Training a Multi-Layer Perceptron (MLP)**
  1. Initialization
  2. Forward Propagation
  3. **Loss Computation**

- In this step, we measure the error between the predicted outputs and the ground truth (target) values.

- The choice of loss function depends on the task, but common choices include mean squared error (MSE) for regression and cross-entropy for classification.

Multilayer Perceptron

- **Training a Multi-Layer Perceptron (MLP)**
  - **Loss Computation**
- **Mean Squared Error (MSE) Loss (Regression):**

$$\text{MSE Loss} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

- Where N is the number of data points, y_i is the true target value for data point i, and $\hat{y}_i$ is the predicted value for data point i.

- **Cross-Entropy Loss (Classification):**

$$Cross - Entropy\ Loss = \frac{1}{N}\sum_{i=1}^{N}(y_i log(\hat{y}_i) + (1 + y_i)\log(1 - \hat{y}_i))$$

- Here, N is the number of data points, y_i is the true class label (0 or 1), and $\hat{y}_i$ is the predicted probability of belonging to class 1 for data point i.

Multilayer Perceptron

Gradient Descent Algorithm

- **Learning rate (step size) :** is the size of the steps that are taken to reach the minimum.
- It is a small value, evaluated and updated based on the behavior of the cost function.
- Two types of learning rates:
  - **High learning rates**
  - **Low learning rate**

Multilayer Perceptron

- **Gradient Descent Algorithm**
- **The cost (or loss) function**
- measures the difference (error) between target and predicted output.
- This improves the machine learning model's efficacy by providing feedback to the model.
- It continuously iterates, moving along the direction of steepest descent until the cost function is close to or at zero.

Multilayer Perceptron

- **How does gradient descent work?**
- **Back-propagation (gradient calculation)**
- Calculate the gradient of the loss with respect to the network's weights and biases by propagating errors backward through the network.

- This involves:
  - **Error Computation**
  - **Weight and Bias Update**

Multilayer Perceptron

- **How does gradient descent work?**
- **Gradient Descent:**

  Use the computed gradients to update the weights and biases of the network.
- **Iteration**:

  Repeat steps 2 to 5 for multiple iterations (epochs).
- **Evaluation**:

  After training, evaluate the trained MLP on a separate test dataset to assess its generalization performance.

MULTILAYER PERCEPTRON

Back Propagation Algorithm

BACKPROPAGATION (training_example, η, n_{in}, n_{out}, n_{hidden})

- Each training example is a pair of the form (x, t), where (x) is the vector of network input values, and (t) is the vector of target network output values.
- η is the learning rate (e.g., 0.05).
- n_i, is the number of network inputs,
- n_{hidden} the number of units in the hidden layer, and
- n_{out} the number of output units.
- The input from unit i into unit j is denoted x_{ji}, and the weight from unit i to unit j is denoted w_{ji}

Multilayer Perceptron

Back Propagation Algorithm

- Create a feed-forward network with n_i inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers
- Until the termination condition is met, Do
  - For each (x, t), in training examples, Do
    - Propagate the input forward through the network:

      1. Input the instance x, to the network and compute the output o_u of every unit u in the network.

    - Propagate the errors backward through the network

      2. For each network unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

      3. For each network unit h, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k}\delta_k$$

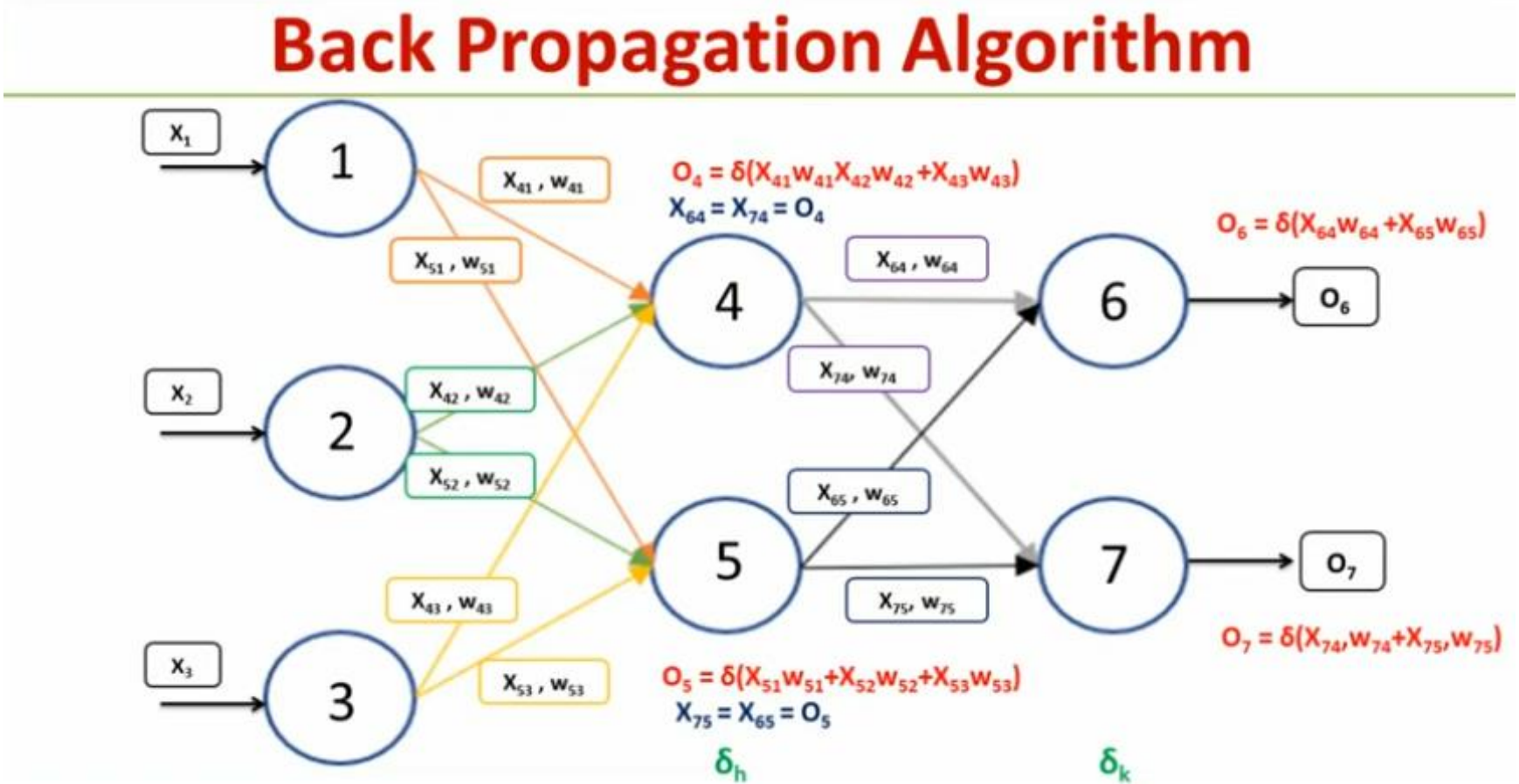
4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

Where

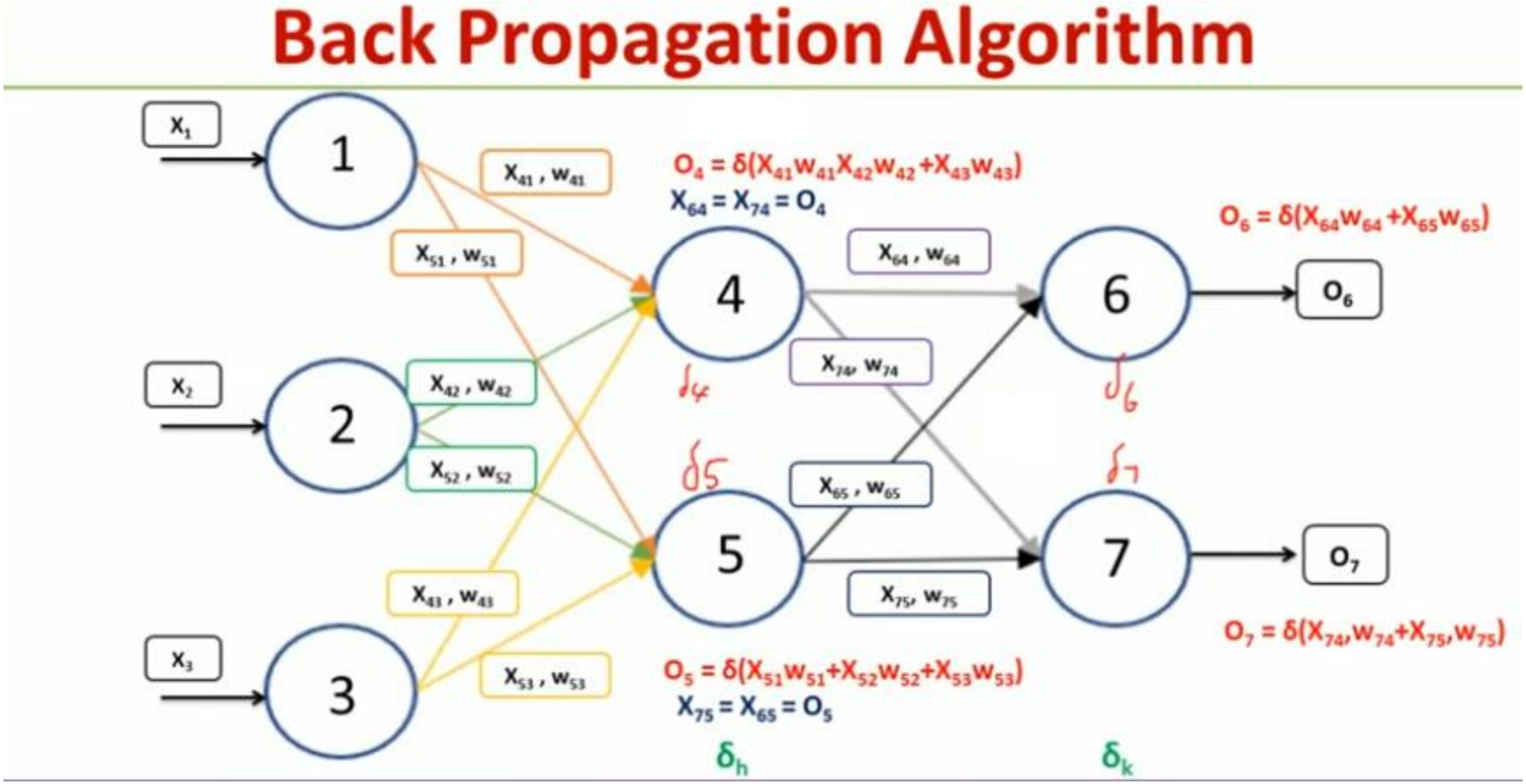
$$\Delta w_{ji} = \eta \delta_j x_{ji}$$

Multilayer Perceptron



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

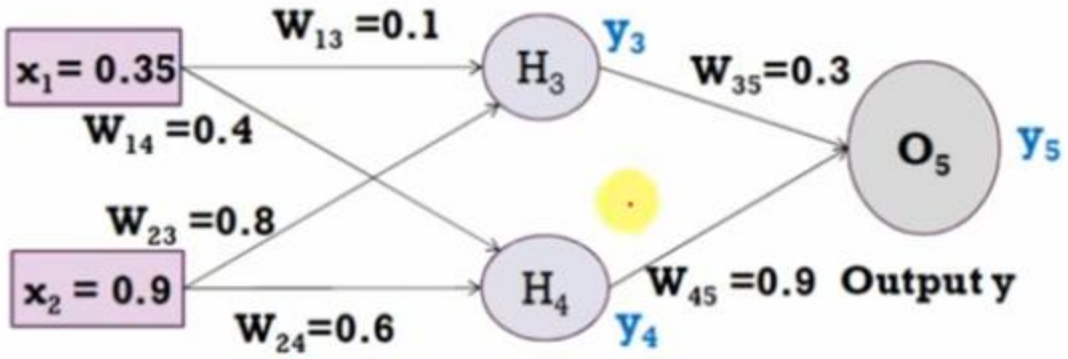
Multilayer Perceptron



Multilayer Perceptron

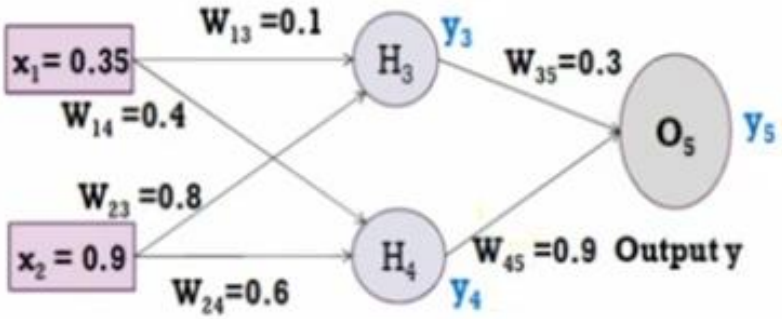
Back Propagation Solved Example - 1

- Assume that the neurons have a sigmoid activation function, perform a forward pass and a backward pass on the network. Assume that the actual output of y is 0.5 and learning rate is 1. Perform another forward pass.



Multilayer Perceptron

Back Propagation Solved Example - 1



- Forward Pass: Compute output for y3, y4 and y5.

$$a_j = \sum_j (w_{i,j} * x_i) \qquad yj = F(aj) = \frac{1}{1 + e^{-a_j}}$$

?

?

Back Propagation Solved Example - 1

- Each weight changed by:

$$\Delta w_{ji} = \eta \delta_j o_i$$

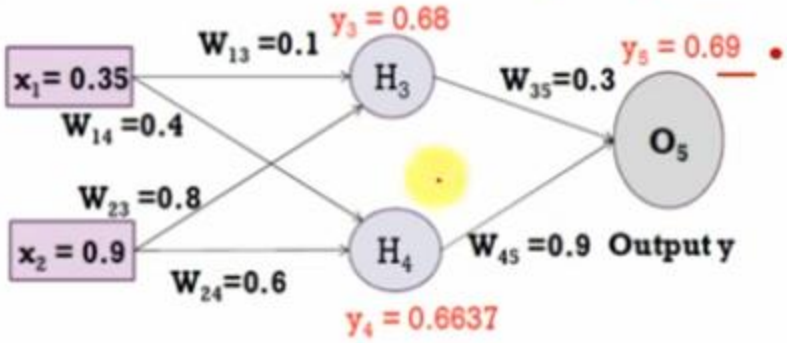
$$\delta_j = o_j(1-o_j)(t_j - o_j) \qquad \text{if } j \text{ is an output unit}$$

$$\delta_j = o_j(1-o_j)\sum_k \delta_k w_{kj} \qquad \text{if } j \text{ is a hidden unit}$$

- where η is a constant called the learning rate
- tj is the correct teacher output for unit j
- δj is the error measure for unit j

Multilayer Perceptron

Back Propagation Solved Example - 1



Backward Pass: Compute δ3, δ4 and δ5.

For output unit:

?

For hidden unit:

$$\Delta w_{ji} = \eta \delta_j o_i$$

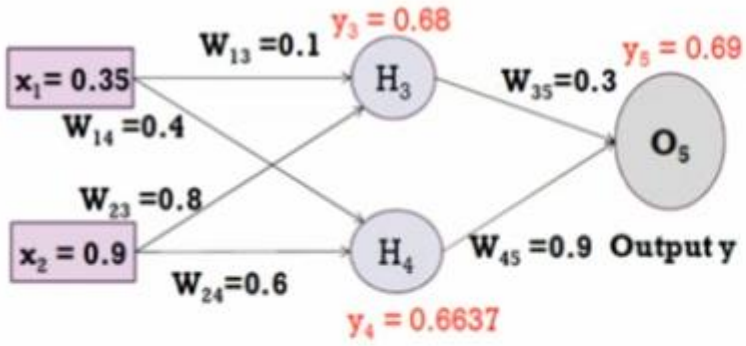
?

$$\delta_j = o_j(1-o_j)(t_j - o_j) \quad \text{if } j \text{ is an output unit}$$

$$\delta_j = o_j(1-o_j)\sum_k \delta_k w_{kj} \quad \text{if } j \text{ is a hidden unit}$$

Multilayer Perceptron

Back Propagation Solved Example - 1



Compute new weights

$$\Delta w_{ji} = \eta \delta_j o_i$$

?

- Backward Pass: Compute δ3, δ4 and δ5.

For output unit:

$\delta_5 = y(1-y)\,(y_{target} - y)$

$= 0.69*(1-0.69)*(0.5-0.69) = -0.0406$

For hidden unit:

$\delta_3 = y_3(1-y_3)\, w_{35} * \delta_5$

$= 0.68*(1 - 0.68)*(0.3 * -0.0406) = -0.00265$

$\delta_4 = y_4(1-y_4) w_{45} * \delta_5$

$= 0.6637*(1 - 0.6637)* (0.9 * -0.0406) = -0.0082$

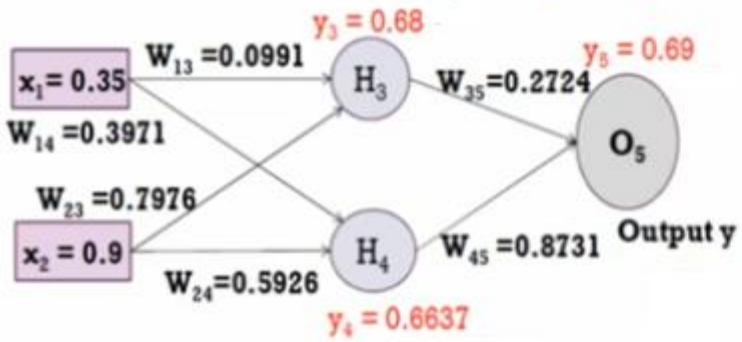
Back Propagation Solved Example - 1

- Similarly, update all other weights

i	j	w_{ij}	δ_j	x_i	η	Updated w_{ij}
1	3	0.1	-0.00265	0.35	1	0.0991
2	3	0.8	-0.00265	0.9	1	0.7976
1	4	0.4	-0.0082	0.35	1	0.3971
2	4	0.6	-0.0082	0.9	1	0.5926
3	5	0.3	-0.0406	0.68	1	0.2724
4	5	0.9	-0.0406	0.6637	1	0.8731

Multilayer Perceptron

Back Propagation Solved Example - 1



- Forward Pass: Compute output for y3, y4 and y5.

$$a_j = \sum_i (w_{i,j} * x_i) \qquad yj = F(aj) = \frac{1}{1 + e^{-a_j}}$$

$a_1 = (w_{13} * x_1) + (w_{23} * x_2)$
$= (0.0991 * 0.35) + (0.7976 * 0.9) = 0.7525$
$y_3 = f(a_1) = 1/ (1 + e^{-0.7525}) = 0.6797$

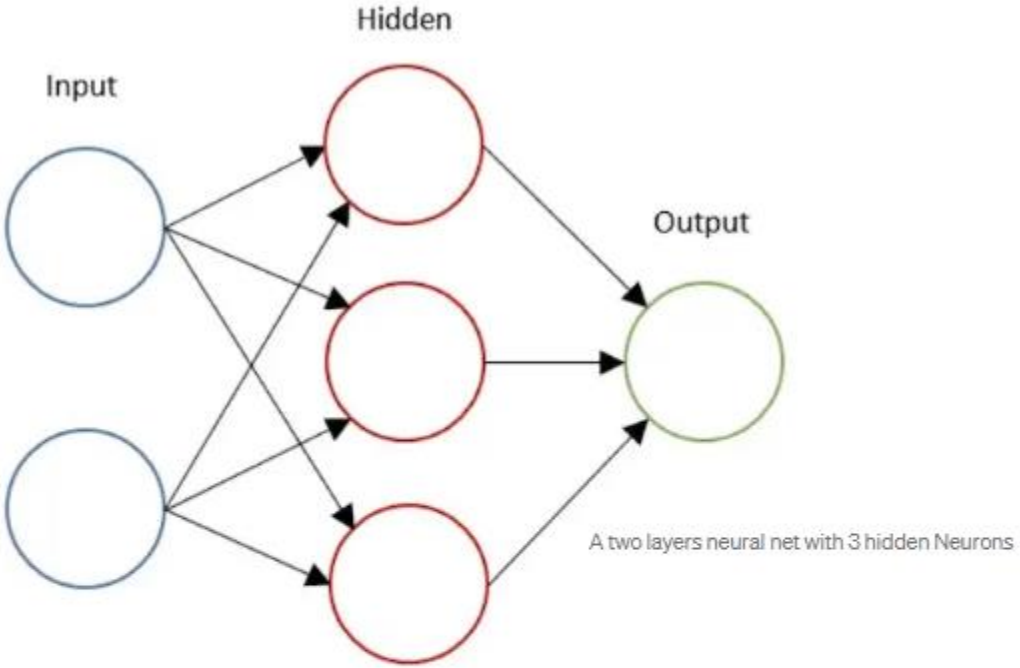
$a_2 = (w_{14} * x_1) + (w_{24} * x_2)$
$= (0.3971 * 0.35) + (0.5926 * 0.9) = 0.6723$
$y_4 = f(a_2) = 1/ (1 + e^{-0.6723}) = 0.6620$

$a_3 = (w_{35} * y_3) + (w_{45} * y_4)$
$= (0.2724 * 0.6797) + (0.8731 * 0.6620) = 0.7631$
$y_5 = f(a_3) = 1/ (1 + e^{-0.7631}) =$ **0.6820** (Network Output)

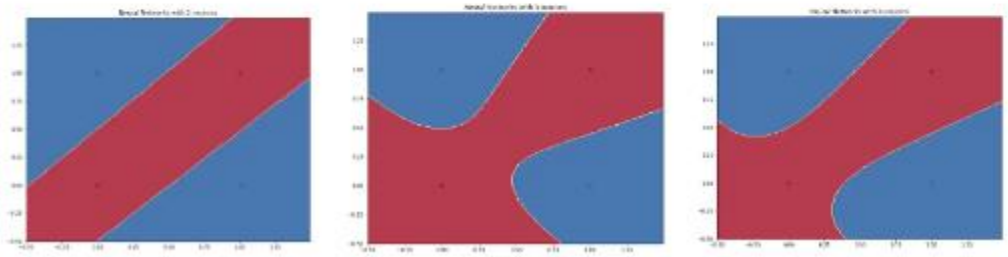
Error $= y_{target} - y_5 = -0.182$

Multilayer Perceptron

- **Neural Nets for Solving the XOR Problem**



A two layers neural net with 3 hidden Neurons



Decision boundaries generated by one hidden layer with several hidden neurons

Multilayer Perceptron

- **Applications of MLPs**
  - Image Classification
  - Natural Language Processing (NLP)
  - Financial Forecasting
  - Healthcare
  - Recommendation Systems

Multilayer Perceptron

- **Problems**
  1. **Amount of data**: quantity of data increased → the parameters of ANN also increased.
  2. **Computationally expensive**: new type of ANN is needed
- **Let's see the problems ourselves !**
- Suppose we want to apply classification problems to a 16-bit image.
- 2D-grayscale image would 16 * 16 = 256 pixels,
  - → neural network with a input layer of 256 Neurons.
- color image → 16 * 16 * 16 = 4096 Neurons in the input layer.
- **Need for an another type of NN**

  → **Convolution neural network (CNN)**