

MACHINE LEARNING

CHAP 10

Artificial Neural Network (ANN)

Jean Tabet

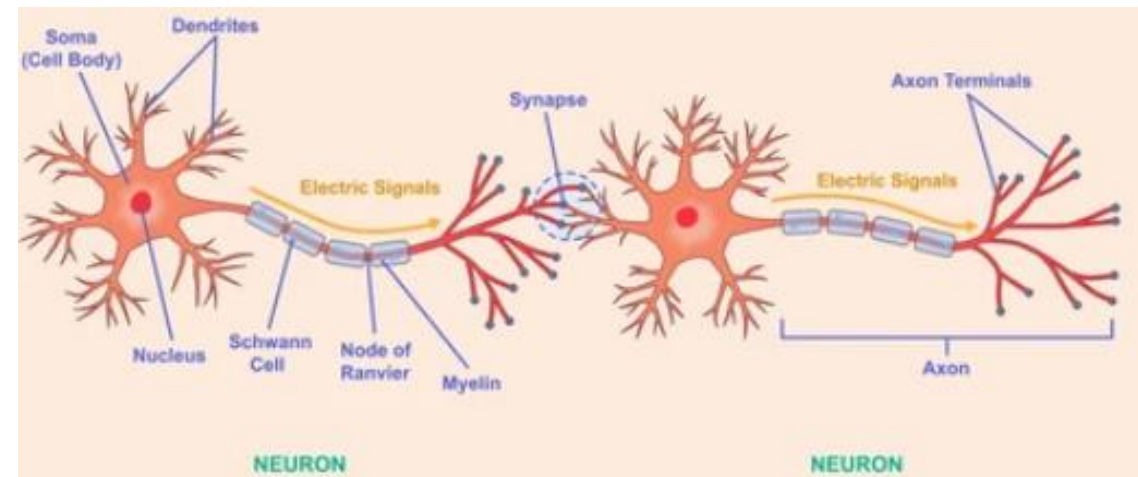
j.tabet@ul.edu.lb

➔ OUTLINE

- Introduction to ANN
- Artificial neuron and perceptron
- Activation functions
- Forward propagation
- Loss evaluation
- Backpropagation and gradient descent
- MLP architectures

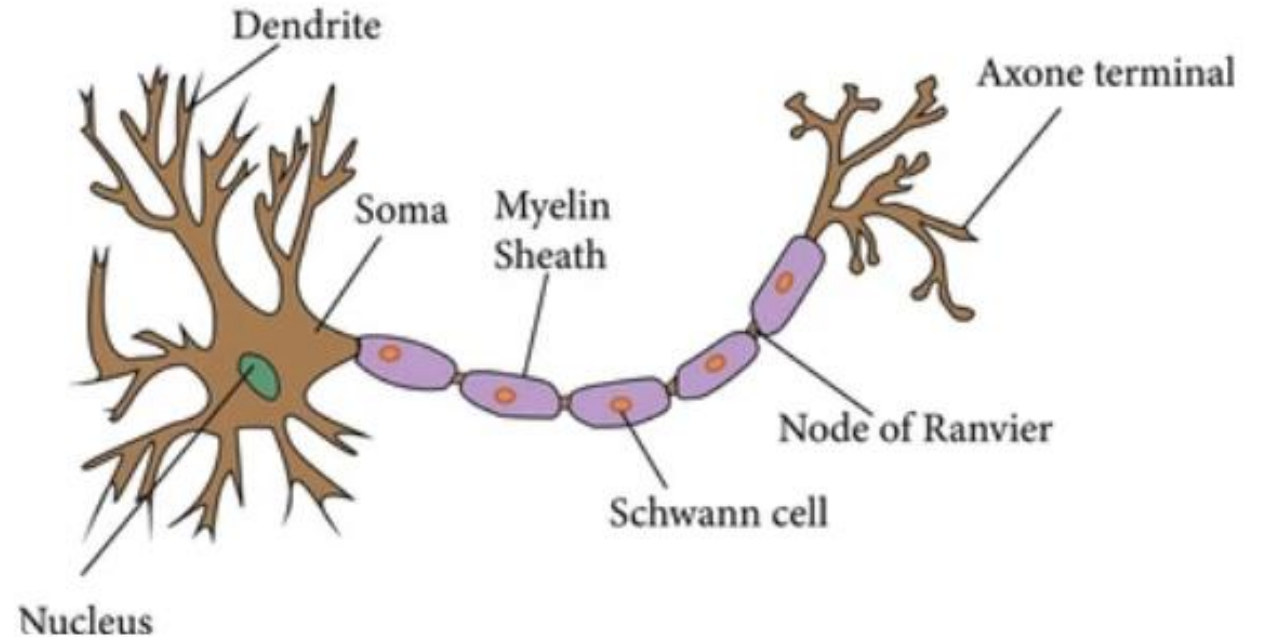
➔ INTRODUCTION TO ANN

- ANN are the most used machine learning algorithms today.
- The basic structure of a neural network, both artificial and natural ones is the neurone.
- A neuron in biology consists of three major parts: the soma (cell body), the dendrites (inputs) and the axon (output).



➔ INTRODUCTION TO ANN

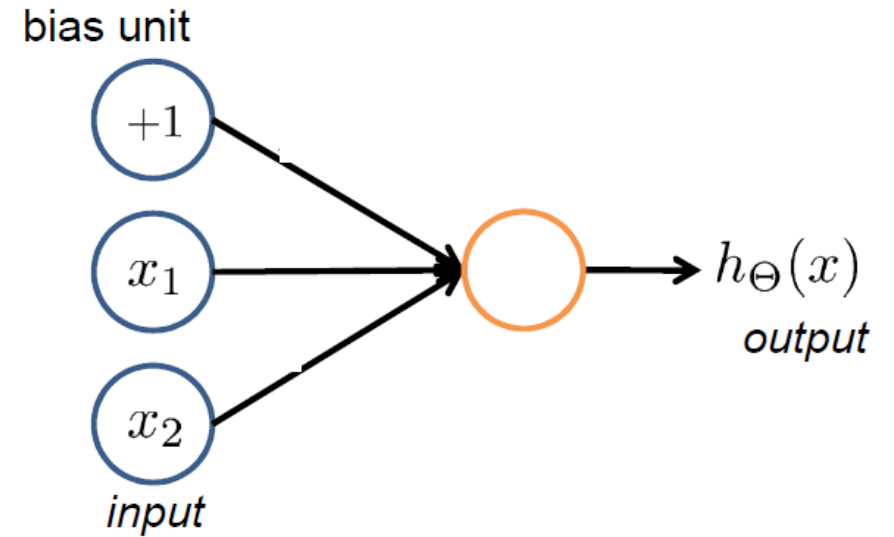
- Dendrites receive signals (nerve impulses) from other neurons through synapses.
- The axon transmits the output signal of the neuron to other neurons through synaptic connections.



- Dendrites: Input
- Synapses: Weights
- Nucleus: Transfer function
- Axon: Output

→ SINGLE NEURONE PERCEPTRON

- The perceptron is a single layer neural network
- The perceptron is formed of a first layer of units (or neurons) which allow to read the data. Each unit corresponds to one of the input variables.



- A bias unit is added. Once activated it transmits 1 whatever the data.
- These units are connected to a single output unit, which receives the sum of the units connected to it, weighed by connection weights.

→ SINGLE NEURONE PERCEPTRON

- For a neural network with p input variables ($x_1, x_2, \dots, x_p$), each input is multiplied by a corresponding weight ($w_1, w_2, \dots, w_p$). A bias term w_0 is also added to the weighted sum. The neuron first computes the weighted sum entering the neuron: z

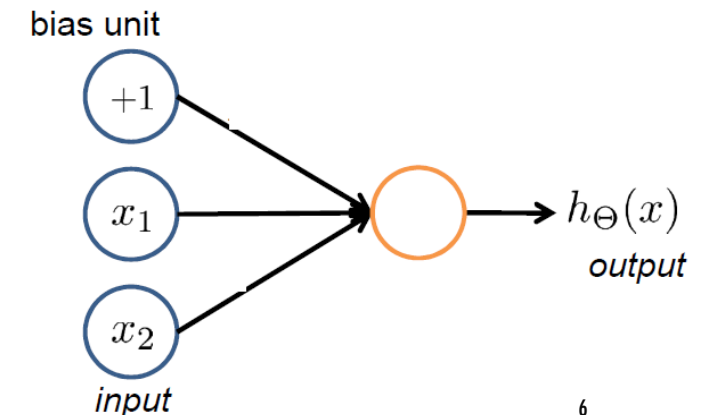
$$z = w_0 + \sum_{j=1}^p w_j x_j$$

- The neuron then applies an activation function to this weighted sum:

$$y = h_{\theta} \left(w_0 + \sum_{j=1}^p w_j x_j \right)$$

Main idea

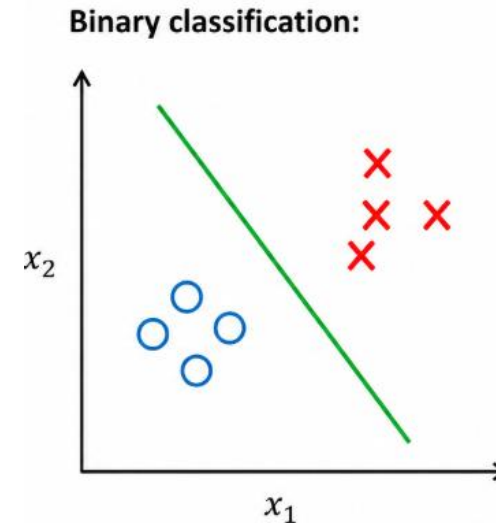
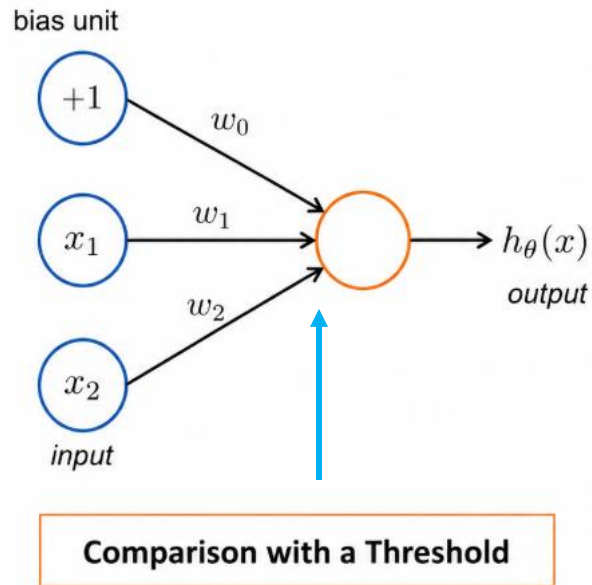
- Inputs are combined using weights
- The bias shifts the result
- The activation function determines the final neuron output



→ SINGLE NEURONE PERCEPTRON

- The threshold function converts this value into a binary output:

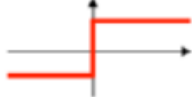
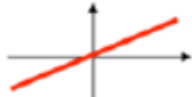
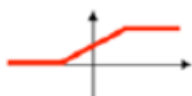
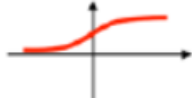
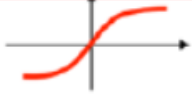
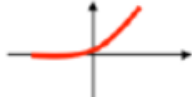
$$\text{Threshold}(z) = \begin{cases} 0 & \text{if } z \leq 0 \\ 1 & \text{otherwise} \end{cases}$$



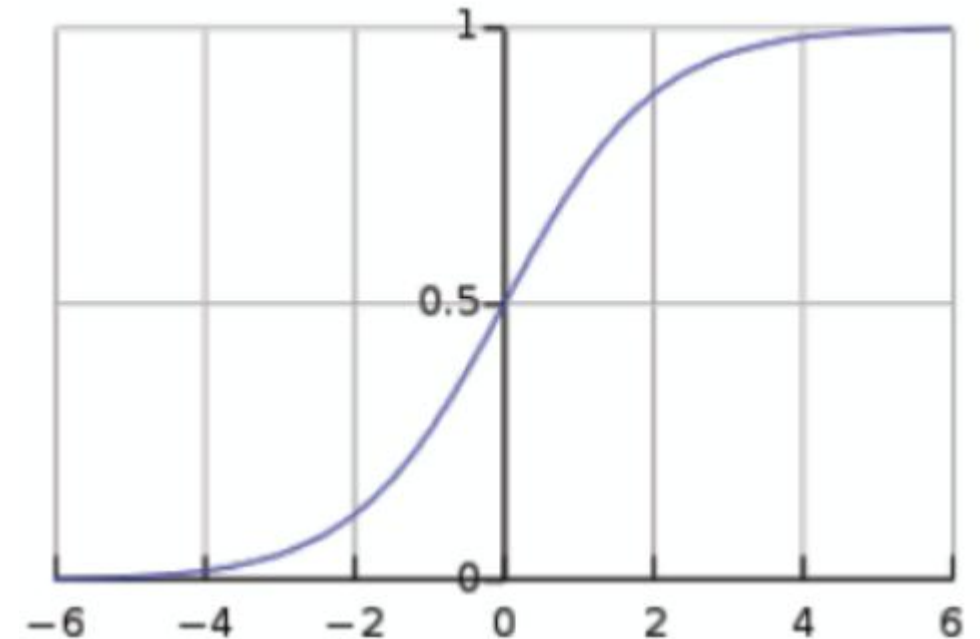
Main idea

- If the weighted sum is below or equal to the threshold $\rightarrow$ output = 0
- If the weighted sum is above the threshold $\rightarrow$ output = 1

→ SINGLE NEURONE PERCEPTRON

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	

Copyright © Sebastian Raschka 2016
(<http://sebastianraschka.com>)



Logistic activation function

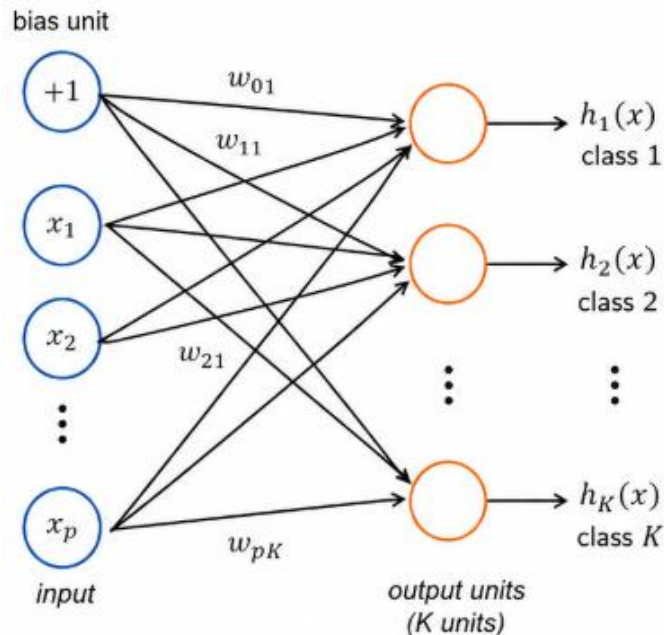
$$h_{\theta}(X) = \frac{1}{1 + e^{-\theta X}}$$

→ SINGLE NEURONE PERCEPTRON

- Instead of using a single output unit, it uses as many as classes. Each of these units is connected to all input units.

→ **K output units e.g.,**

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$



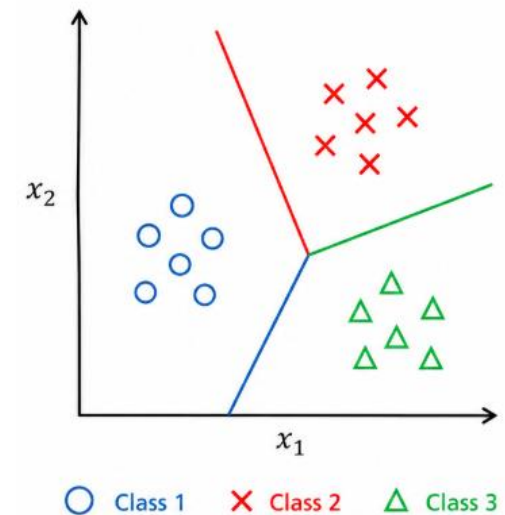
Example ($K = 4$ classes):

Output (after SoftMax):

$\begin{bmatrix} 0.70 \\ 0.10 \\ 0.05 \\ 0.15 \end{bmatrix}$	→ class 1
	→ class 2
	→ class 3
	→ class 4

The class with the highest probability is selected.

Multi-class classification:



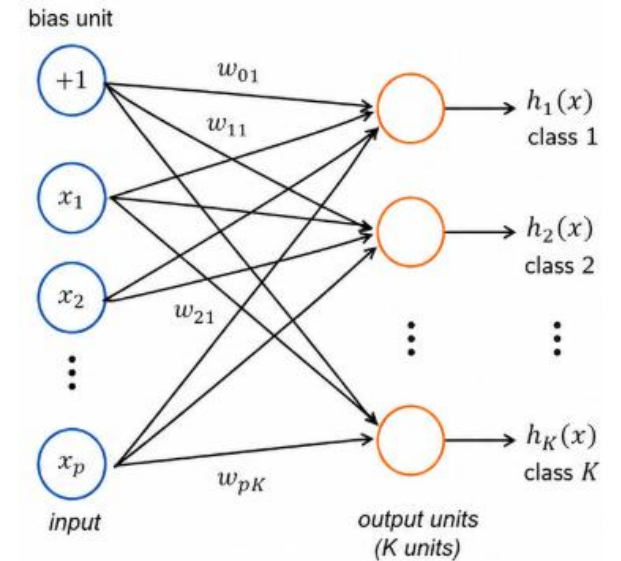
→ SINGLE NEURONE PERCEPTRON (SOFTMAX)

- The activation function used in this case is the SoftMax function.
- It is a general form of the sigmoid:

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

- For K classes, we use this activation function:

$$\sigma_k(u_k) = \frac{e^{u_k}}{\sum_{j=1}^K e^{u_j}}$$



Example ($K = 4$ classes):

Output (after SoftMax):

0.70	→ class 1
0.10	→ class 2
0.05	→ class 3
0.15	→ class 4

The class with the highest probability is selected.

→ SINGLE NEURONE PERCEPTRON (SOFTMAX)

- SoftMax converts a vector of real-valued scores (one per class) into a probability distribution over K classes.
- All values are positive and sum to 1.
- Exponentiation makes higher scores much larger.
- Normalization by the sum turns them into probabilities.

1 Scores (before activation)

For class $k \in \{1, 2, \dots, K\}$, the score (net input) is:

$$u_k = w_{0k} + \sum_{j=1}^p w_{jk} x_j$$

x_j : input j ($j = 1, \dots, p$)

w_{jk} : weight from input j to class k

w_{0k} : bias for class k

2 Softmax activation

The softmax activation for class k is:

$$\sigma_k(u_k) = \frac{e^{u_k}}{\sum_{j=1}^K e^{u_j}}$$

- $\sigma_k(u_k)$ is the output (probability) for class k

- $0 < \sigma_k(u_k) < 1$

- $\sum_{k=1}^K \sigma_k(u_k) = 1$

3 Prediction

The predicted class is the one with the highest probability:

$$\hat{y} = \arg \max_k \sigma_k(u_k)$$

→ SINGLE NEURONE PERCEPTRON (ADVANTAGES)

- Perceptron allows to learn parametric models based on a linear combination of variables.
- Perceptron makes it possible to learn regression models (identity activation), binary (logistic activation) or multi-class classification (SoftMax function).
- The perceptron is trained by iterative updates of its weights based on gradient algorithms. The same rule for updating weights applies in the case of regression, binary classification or multi-class classification.

➔ MULTI-LAYER PERCEPTRON (MLP)

NETWORK ARCHITECTURE

Input layer

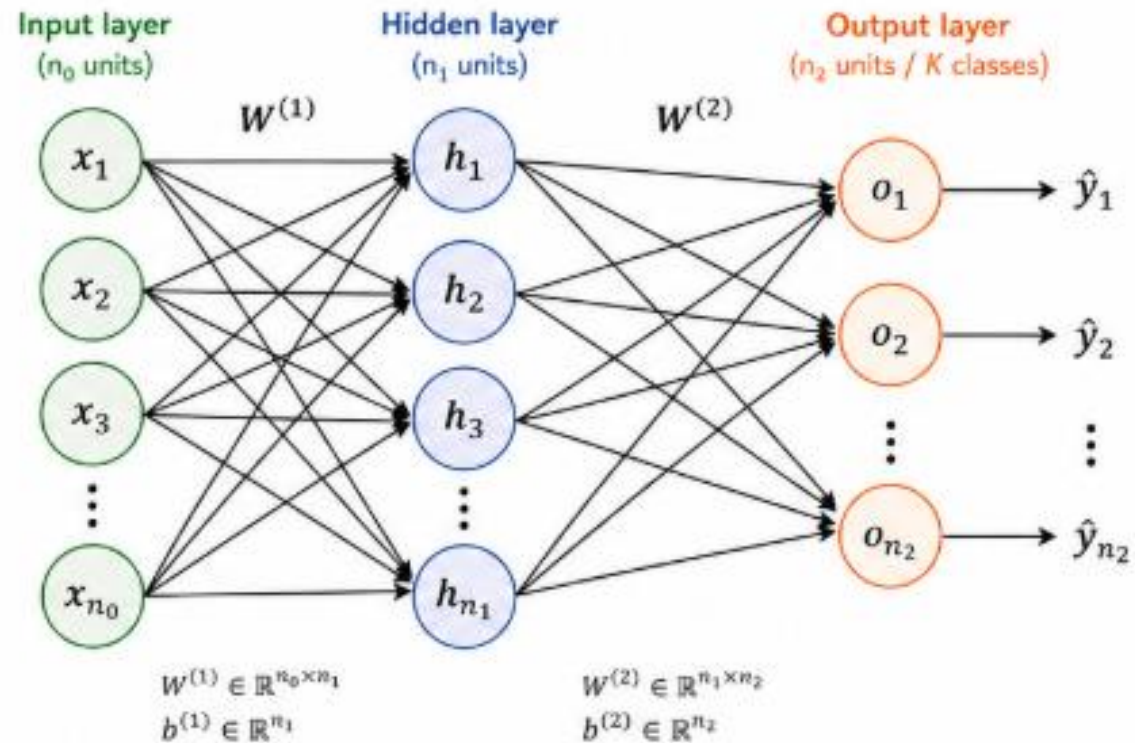
- Receives the input features
- No computation (just passes the values)

Hidden layer (s)

- Learns complex patterns through weights
- Each neuron applies an activation function

Output layer

- Produces the final output
- For classification: usually uses Softmax activation



ROLE OF THE ACTIVATION FUNCTION

Each neuron computes a weighted sum of its inputs and then applies a non-linear activation function $\varphi(\cdot)$.

$$z = Wx + b \quad (\text{linear combination})$$

$$a = \varphi(z) \quad (\text{activation})$$

Common choices:

- Hidden layers: ReLU, tanh, sigmoid
→ introduce non-linearity
- Output layer (classification):
Softmax → converts scores to probabilities (sum = 1)

➔ MULTI-LAYER VS SINGLE PERCEPTRON

- Can solve nonlinear problems
- Uses one or more hidden layers
- Learns complex patterns and relationships
- Higher prediction accuracy
- Supports multi-class classification
- Better feature extraction capability
- More powerful and flexible than a single perceptron

➔ LEARNING STEPS OF ANN

- Step 1: Input data enters the network
- Step 2: Forward propagation computes outputs
- Step 3: Loss function measures prediction error
- Step 4: Backpropagation computes gradients
- Step 5: Optimizer updates weights
- The cycle repeats during training

→ LEARNING STEPS OF ANN: BACKPROPAGATION

- An iterative gradient algorithm is used, and an error backpropagation is applied.
- Example: Binary Classification with Logistic Loss + L2 Regularization

$$J(W) = -\frac{1}{m} \sum_{i=1}^m \underbrace{\left[y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right]}_{\text{First Term} \rightarrow \text{Prediction Loss}} + \underbrace{\frac{\lambda}{2m} \sum_l \sum_i \sum_j \left(W_{ij}^{(l)} \right)^2}_{\text{Second Term} \rightarrow \text{Regularization}}$$

First Term → Prediction Loss

Measures the prediction error between:

- true values y
- predicted outputs $\hat{y}$

Second Term → Regularization

Penalizes large weights to:

- reduce overfitting
- improve generalization
- simplify the model

→ ANN OPTIMIZATION VIA GRADIENT DESCENT

- Gradient Descent updates the weights in the direction that minimizes the loss function. The update rule is:

$$W^{new} = W^{old} - \eta \frac{\partial J}{\partial W}$$

where:

- η = learning rate
- $\frac{\partial J}{\partial W}$ = gradient of the loss
- Similarly, biases are updated as:

$$b^{new} = b^{old} - \eta \frac{\partial J}{\partial b}$$

- Alternative optimizers: SGD, Adam, RMSProp

➔ MULTI-LAYER PERCEPTRON (MLP) CLASSIFIER

```
mlp_gs = MLPClassifier(max_iter=1)

parameter_space = {
    'hidden_layer_sizes': [(10,), (20,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant', 'adaptive'],
}

from sklearn.model_selection import GridSearchCV

clf = GridSearchCV(mlp_gs, parameter_space, n_jobs=-1, cv=2)
clf.fit(X, y)
```

➔ NEURAL NETWORKS: RNN

There are several types of Artificial Neural Networks. The most widely used architectures are Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNN).

1. Recurrent Neural Network (RNN)

Recurrent Neural Networks are specialized neural networks designed to process sequential data. Unlike traditional feedforward networks, RNNs use information from previous steps when computing the current output. Thus, the output depends not only on the current input, but also on previously calculated outputs and hidden states.

RNN applications: Time-series prediction, Speech recognition, Natural language processing Chatbots and text generation language translation.

➔ NEURAL NETWORKS: CNN

There are several types of Artificial Neural Networks. The most widely used architectures are Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNN).

2. Convolutional Neural Network (CNN)

Convolutional Neural Networks are neural networks specialized for processing image-like data. CNNs use convolution filters (kernels) that automatically extract important features from the input before passing the information to deeper layers of the network.

RNN applications: Image classification, Object detection, Face recognition, Medical image analysis, Computer vision applications

➔ **TRANSITION FROM ANN TO DEEP LEARNING**

- Traditional ANN: Input Layer → Hidden Layer → Output Layer
- Deep Learning Network: Input Layer → Multiple Hidden Layers → Output Layer
- More hidden layers allow learning of complex patterns
- Deep Learning applications include image recognition, NLP, speech recognition, and medical diagnosis