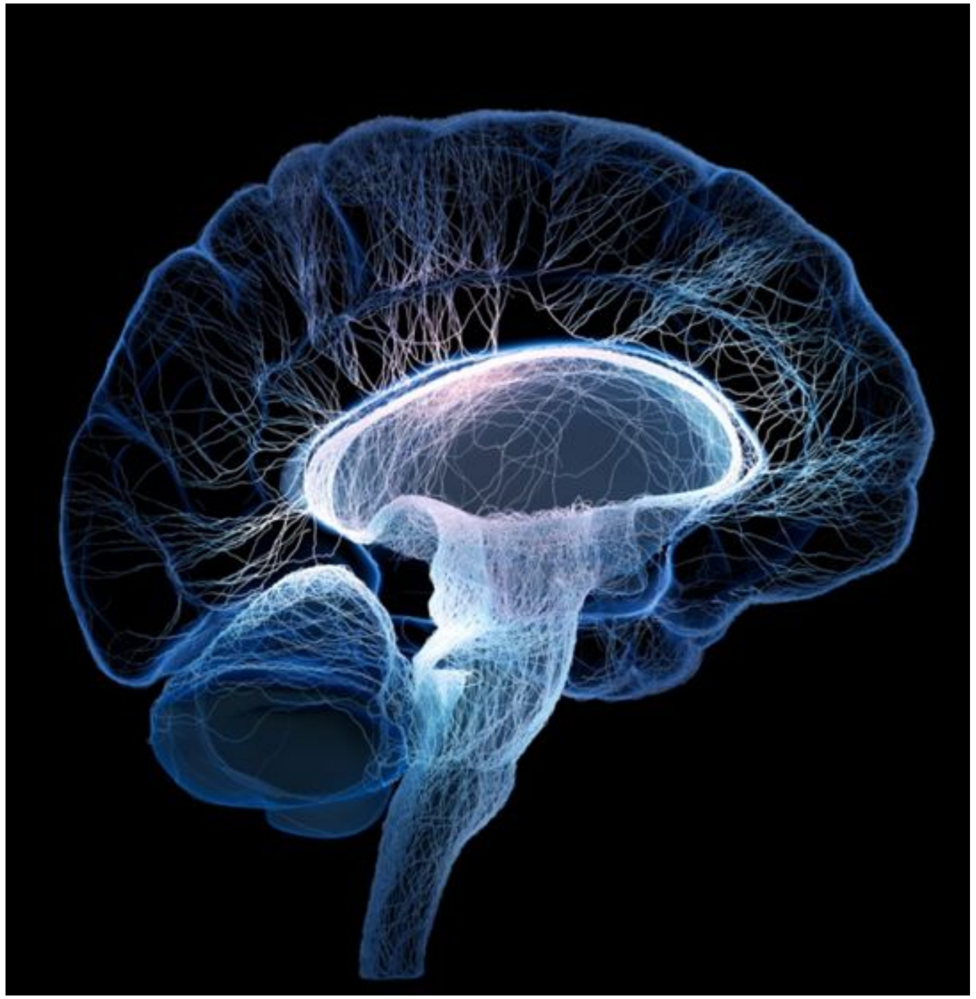
AI Course

Dr. Mürsel Taşgın

Neural Networks

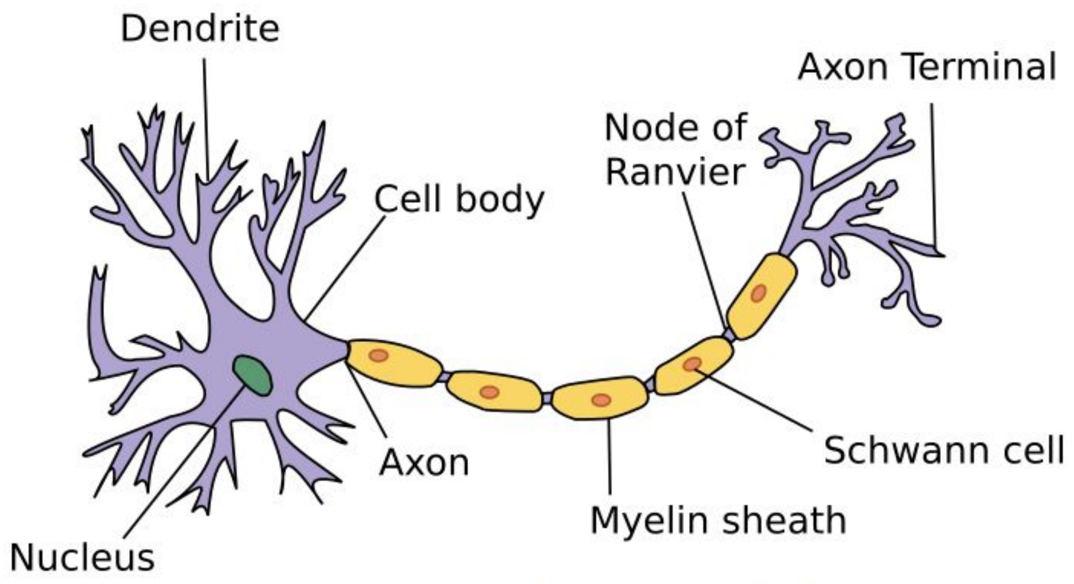
Neural Networks

How our brain works?



Neural Networks

How our brain works?



Schematic representation of **biological neuron**. From Wikipedia.

Dendrites, also known as *dendrons*, are branched protoplasmic extensions of a nerve cell that propagate electrochemical stimulation received from other neural cells to the cell body (or **soma**).

Soma is where the signals received from the dendrites are joined and passed on. It contains many organelles as well as the cell nucleus.

Axon hillock is a specialized part of the soma that connects to the **axon**. It is it that controls the firing of the neuron. If the total strength of the signal it receives exceeds its threshold limit, the neuron will fire a signal (known as an action potential) down the axon.

Axon is the elongated fiber extending from the soma down to the terminal endings. Its role is to transmit the neural signal to other neurons through its **synapses**.

Synapses are small gaps located at the very end of the axon terminal connecting the neuron to other nerve cells. There, neurotransmitters are used to carry the signal across the synapse to other neurons.

Neural Networks *– Threshold Logic Unit (MCP)*

Threshold Logic Unit - *McCulloch & Pitts 1943*

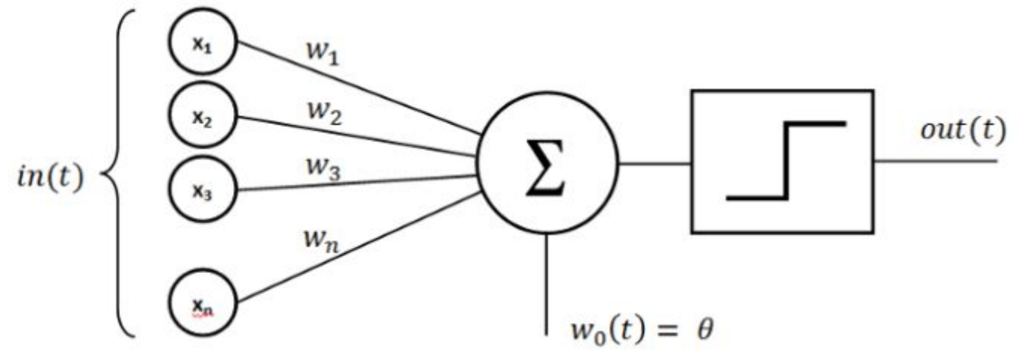
- First mathematical model of an artificial neuron
- McCulloch & Pitts' neuron model, hereafter denoted simply as *MCP neuron*, can be defined by the following rules :
    - It has a binary output y ∈ {0, 1}, where y=1 indicates that the neuron fires and y=0 that it is at rest.
    - It has a number N of excitatory binary inputs x□ ∈ {0, 1}.
    - It has a single inhibitory input *i*. If it is on, the neuron cannot fire.
    - It has a threshold value Θ. If the sum of its inputs is larger than this critical value, the neuron fires. Otherwise, it stays at rest.

$$\sigma(\mathbf{x}) = \begin{cases} 1 \text{ if } \sum_{k=1}^{n} x_k > \Theta \text{ and } i = 0, \\ 0 \text{ otherwise.} \end{cases}$$

Neural Networks *- Perceptron model*

Single-layer perceptron - *Rosenblatt 1957*

- Synaptic plasticity of the brain
- An improvement over MCP model
  - Instead of absolute inhibition, equal contribution of all inputs
  - Artificial neurons can learn from data



Artificial neuron used by the perceptron. From Wikipedia.

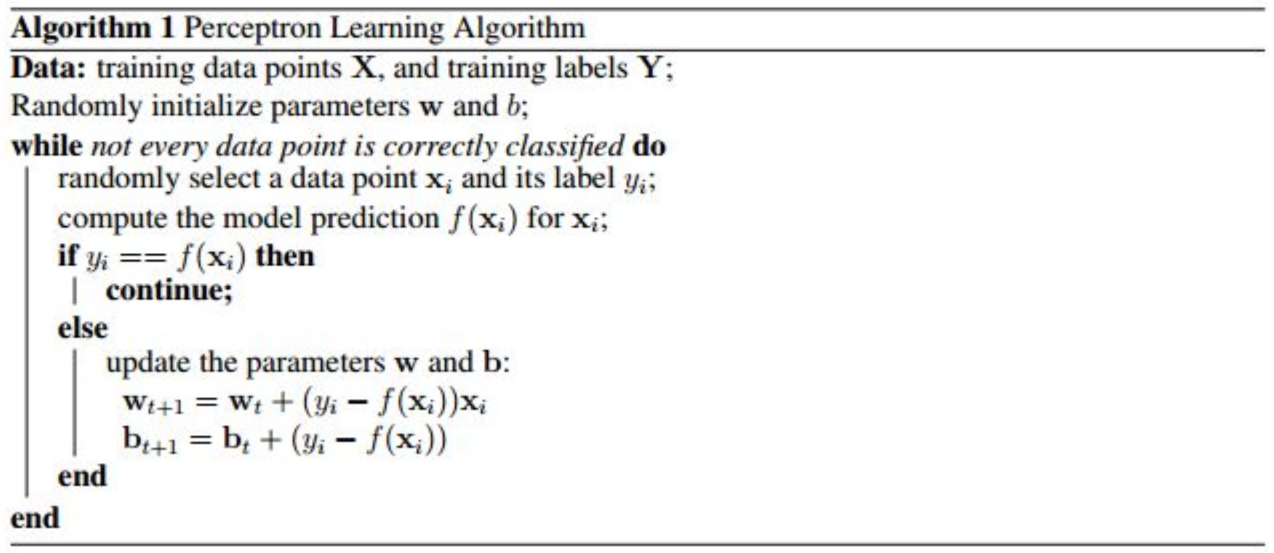
Differences from MCP

- The neuron takes an extra constant input associated with a synaptic weight *b* (denoted Θ in the figure above), also known as the bias. Concerning the MCP neuron, the bias *b* is simply the negative of the activation threshold.
- The synaptic weights w_k are not restricted to unity, thus allowing some inputs to have more influence on the neuron's output than others.
- They are not restricted to be strictly positive either. Some of the inputs can hence have an inhibitory influence.
- The absolute inhibition rule no longer applies.

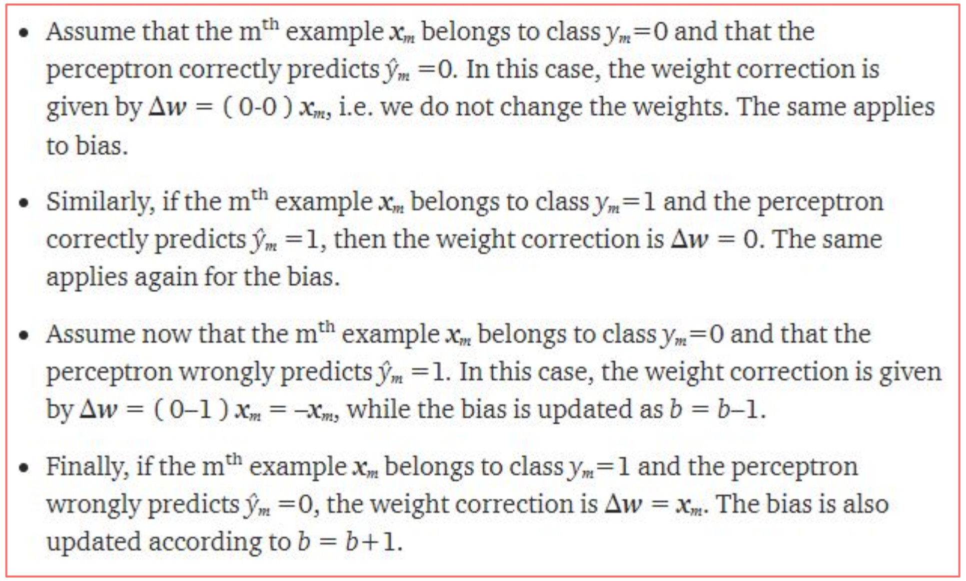
Neural Networks *- Perceptron model*

Perceptron learning algorithm

*Given a set of M examples (**x**, y), how can the perceptron learn the correct synaptic weights **w** and bias b to correctly separate the two classes?*



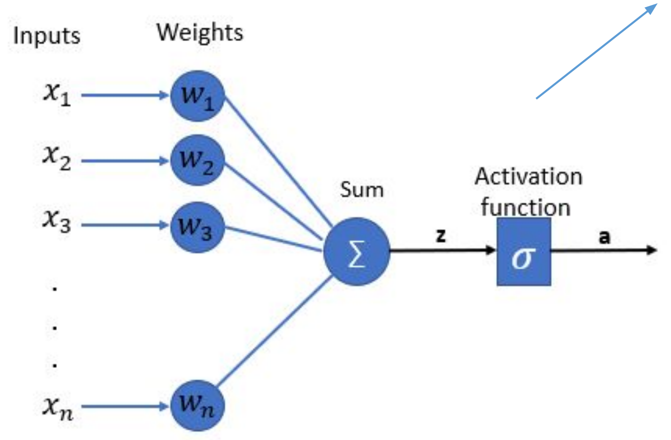
```
Algorithm 1 Perceptron Learning Algorithm
Data: training data points $\mathbf{X}$, and training labels $\mathbf{Y}$;
Randomly initialize parameters $\mathbf{w}$ and $b$;
while not every data point is correctly classified do
    randomly select a data point $\mathbf{x}_i$ and its label $y_i$;
    compute the model prediction $f(\mathbf{x}_i)$ for $\mathbf{x}_i$;
    if $y_i == f(\mathbf{x}_i)$ then
        continue;
    else
        update the parameters w and b:
          $\mathbf{w}_{t+1} = \mathbf{w}_t + (y_i - f(\mathbf{x}_i))\mathbf{x}_i$
          $\mathbf{b}_{t+1} = \mathbf{b}_t + (y_i - f(\mathbf{x}_i))$
    end
end
```



- Assume that the mth example x_m belongs to class y_m=0 and that the perceptron correctly predicts $\hat{y}_m$ =0. In this case, the weight correction is given by Δw = (0-0) x_m, i.e. we do not change the weights. The same applies to bias.
- Similarly, if the mth example x_m belongs to class y_m=1 and the perceptron correctly predicts $\hat{y}_m$ =1, then the weight correction is Δw = 0. The same applies again for the bias.
- Assume now that the mth example x_m belongs to class y_m=0 and that the perceptron wrongly predicts $\hat{y}_m$ =1. In this case, the weight correction is given by Δw = (0–1) x_m = –x_m, while the bias is updated as b = b–1.
- Finally, if the mth example x_m belongs to class y_m=1 and the perceptron wrongly predicts $\hat{y}_m$ =0, the weight correction is Δw = x_m. The bias is also updated according to b = b+1.

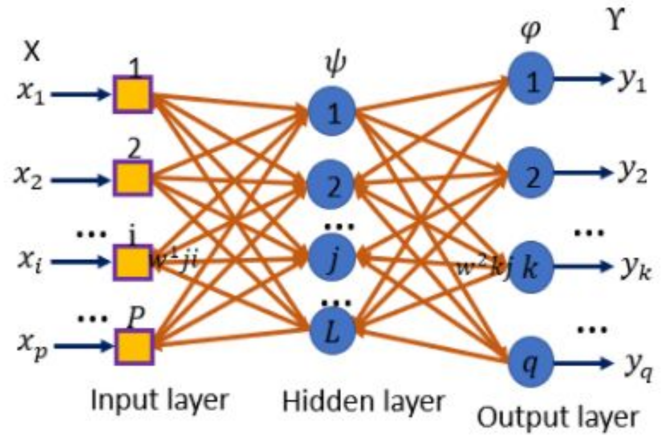
Neural Networks *- Perceptron model*

Can be regarded as a single neuron



Single-layer perceptron

Can be used on linearly separable data!

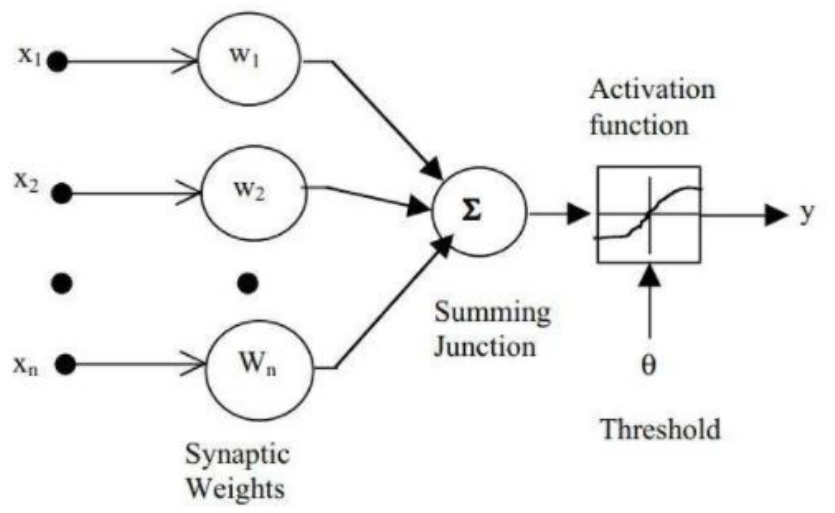


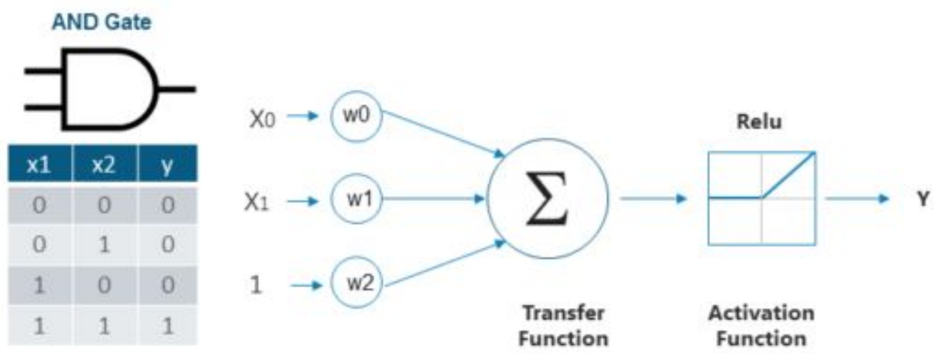
Multi-layer perceptron

Neural Networks *- Artificial Neural Networks*

Artificial Neural Networks

- ANNs are composed of artificial neurons which are conceptually derived from biological neurons.
- Each artificial neuron has inputs and produces a single output which can be sent to multiple other neurons.
- Connections have weights and a weighted sum of inputs arrive at a neuron
- We add a bias term to the weighted sum and decide on activation

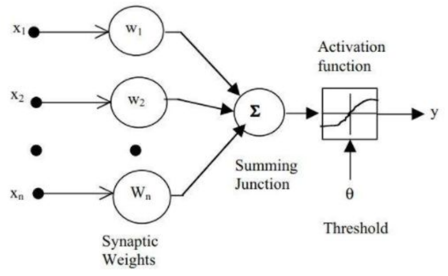




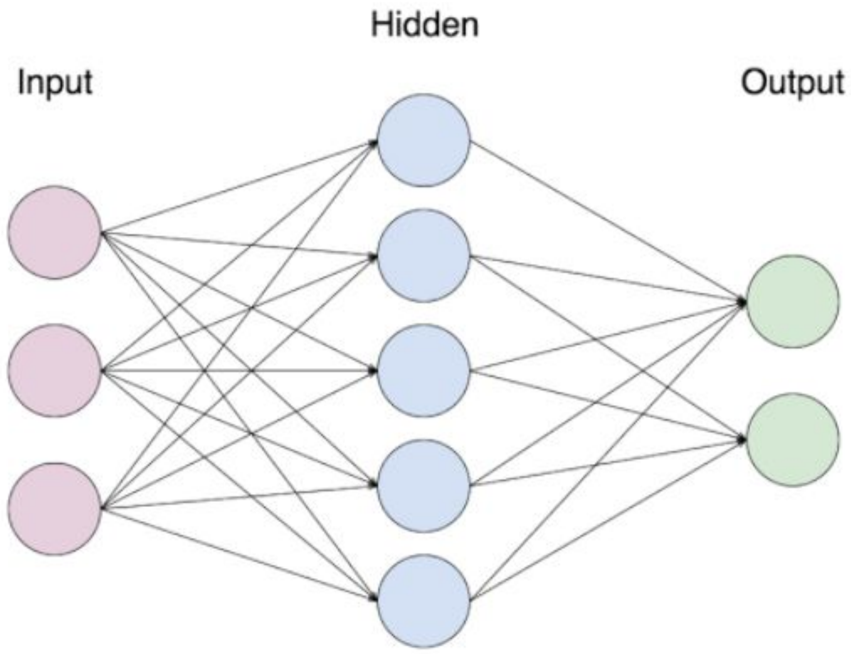
x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1

Neural Networks *- Artificial Neural Networks*

Artificial Neural Networks

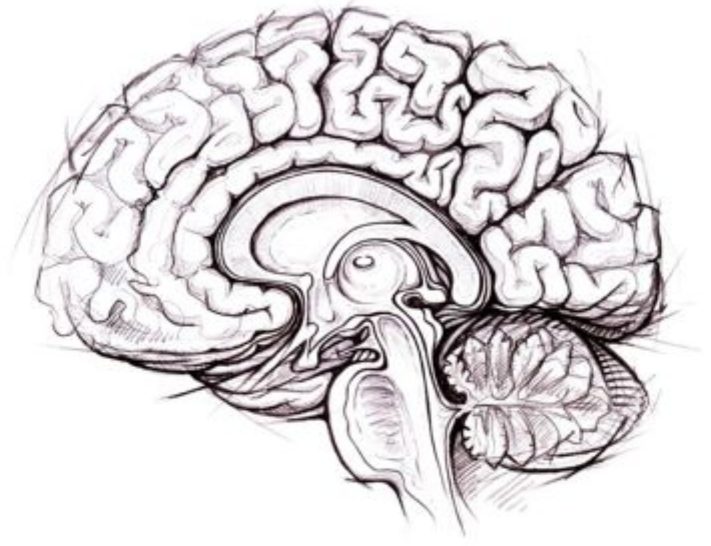


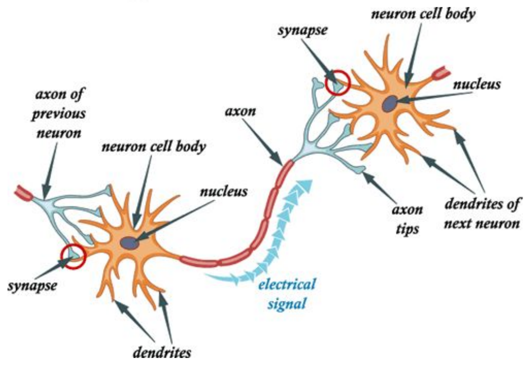
Use single perceptrons to build multi-layer neural networks



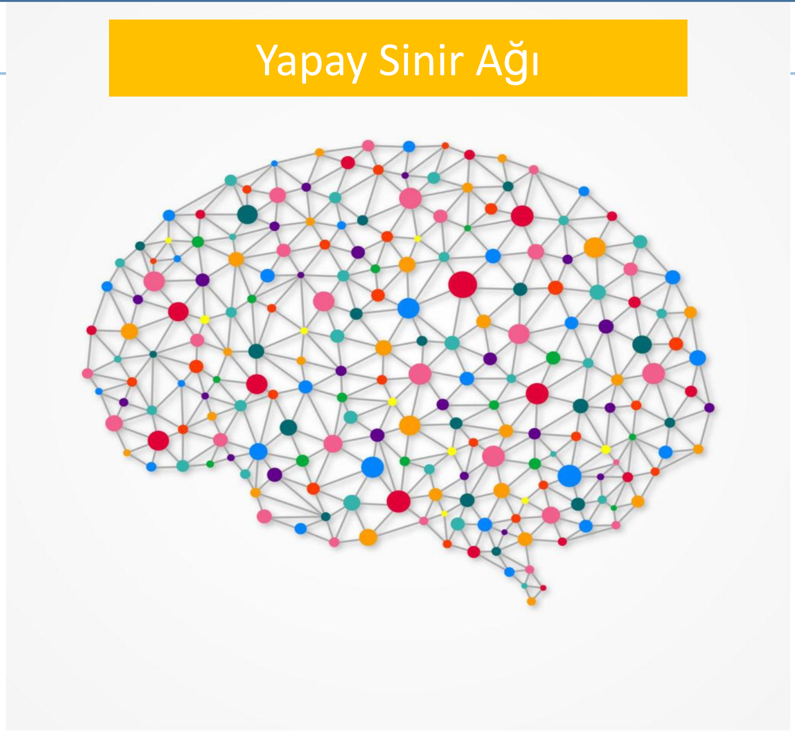
Yapay sinir ağı modeli

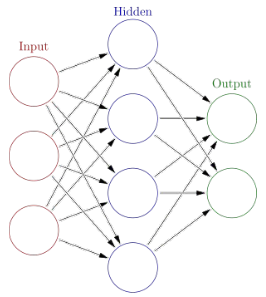
Biyolojik Beyin

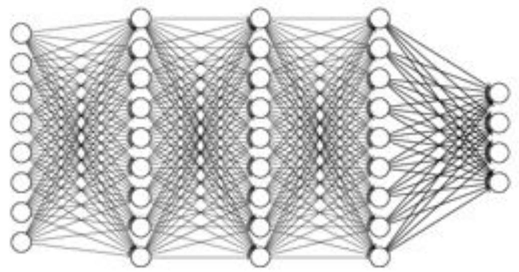




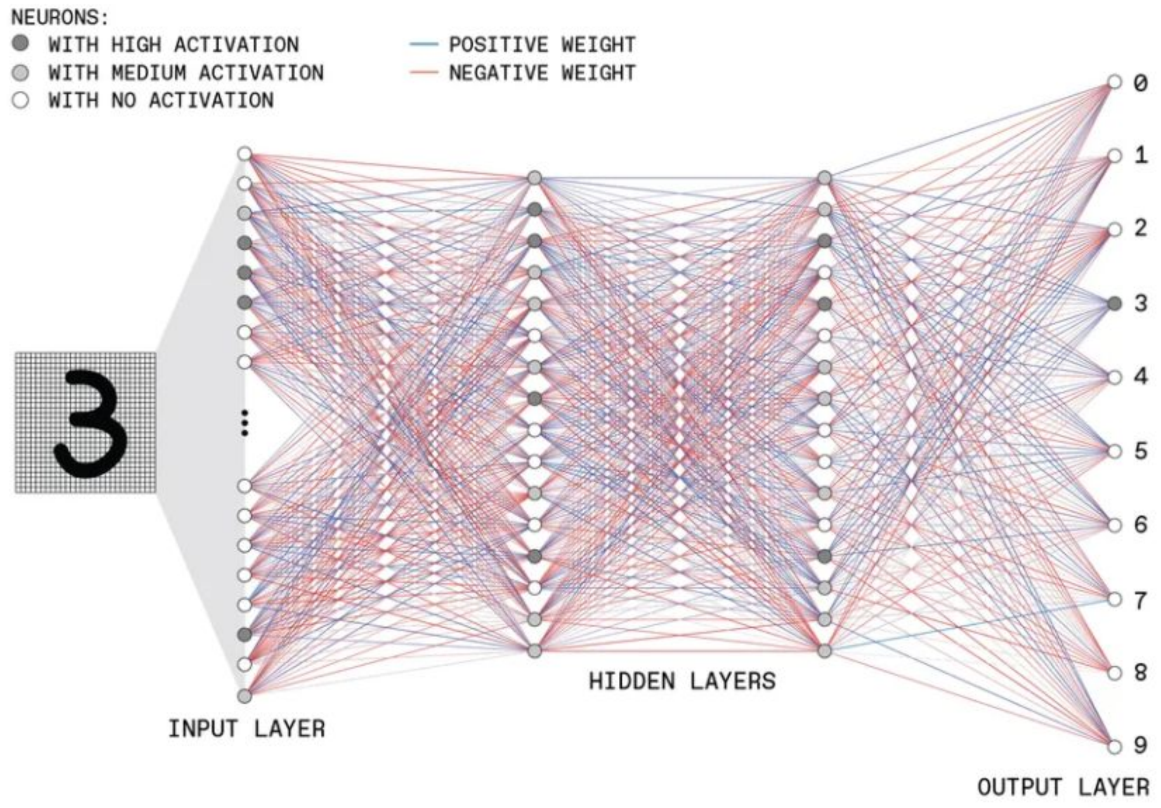
Yapay Sinir Ağı





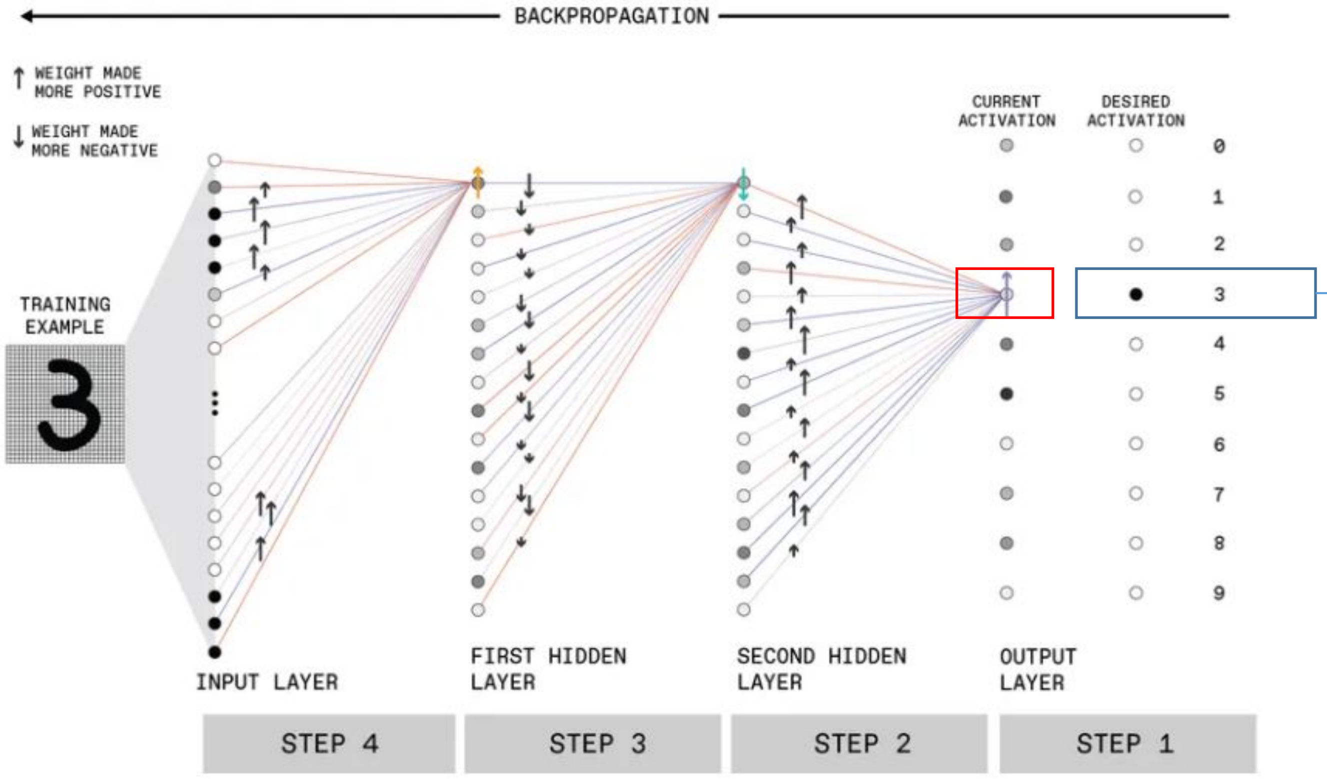


Neural Networks *- Artificial Neural Networks*



Neural Networks *- Artificial Neural Networks*

How learning happens in ANN?



Predicted value equals to desired (target) value ✓

Neural Networks *- Artificial Neural Networks*

How learning happens in ANN?

- Weights and biases are initialized in ANN *(i.e., random weights and biases)*
- During training, predicted values are compared with target (actual) values and *prediction errors* are calculated
- Prediction errors (i.e., `actual - predicted`) are used for learning the weights and biases so that the NN will make less error for next inputs
- Backpropagation: Compute the gradient of loss function with respect to each weight by chain rule (one layer at a time), iterating backward from the last layer □ **auto-differentiation !**

Loss function

$$E = L(t, y)$$

L is the loss for the output y and target value t,

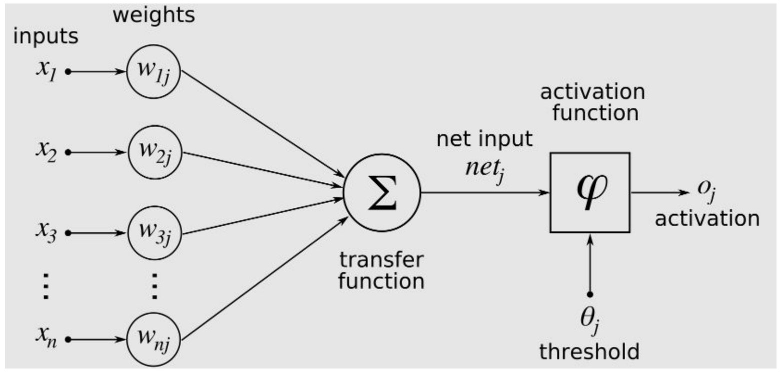
t is the target output for a training sample, and

y is the actual output of the output neuron.

For each neuron j, its output o_j is defined as

$$o_j = \varphi(\text{net}_j) = \varphi\left(\sum_{k=1}^{n} w_{kj} o_k\right)$$

Neural Networks *- Artificial Neural Networks*



Chain rule

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial \mathrm{net}_j}\frac{\partial \mathrm{net}_j}{\partial w_{ij}}$$

Let activation function be

$$\varphi(z) = \frac{1}{1+e^{-z}} \xrightarrow{\text{derivative}} \frac{d\varphi(z)}{dz} = \varphi(z)(1-\varphi(z))$$

The derivative of the output of neuron j with respect to its input is partial derivative of activation function

$$\frac{\partial o_j}{\partial \mathrm{net}_j} = \frac{\partial \varphi(\mathrm{net}_j)}{\partial \mathrm{net}_j} \longrightarrow \frac{\partial o_j}{\partial \mathrm{net}_j} = \frac{\partial}{\partial \mathrm{net}_j}\varphi(\mathrm{net}_j) = \varphi(\mathrm{net}_j)(1-\varphi(\mathrm{net}_j)) = \boxed{o_j(1-o_j)}$$

Updates the weights

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = -\eta o_i \delta_j \longrightarrow w'_{ij} = w_{ij} - \Delta w_{ij}$$

Learning rate

Update the biases too!

$$b' = b - \eta \frac{\partial E}{\partial b}$$

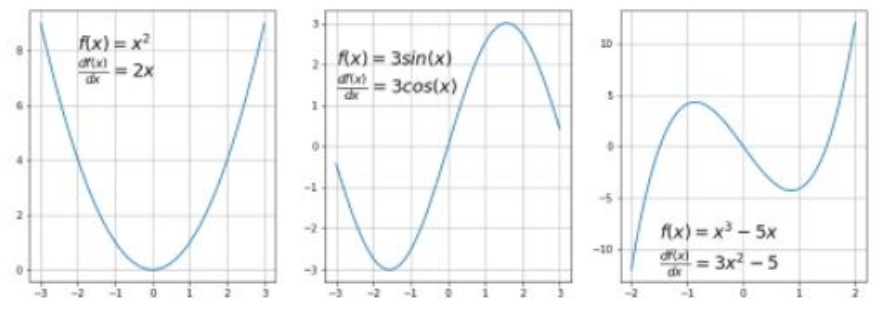
Neural Networks *- Gradient Descent*

Gradient descent (GD) is an iterative first-order optimization algorithm used to find a local minimum/maximum of a given function. This method is commonly used in *machine learning* (ML) and *deep learning*(DL) to minimize a cost/loss function (i.e. *in a linear regression*)

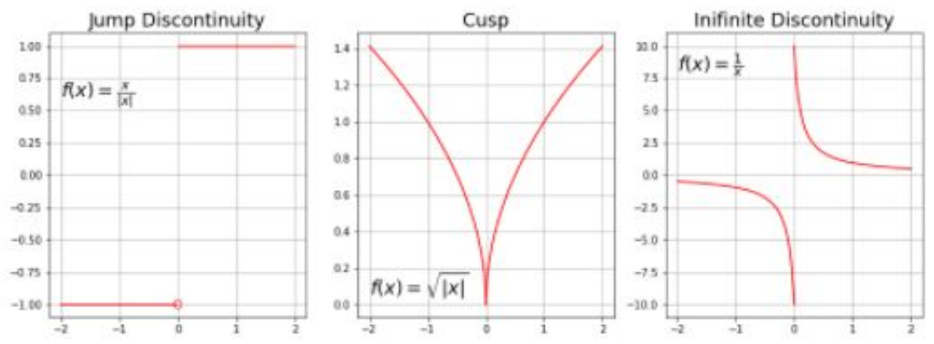
Gradient descent algorithm does not work for all functions.

There are two specific requirements; afunction has to be:

- differentiable
- convex



Examples of differentiable functions



Examples of non-differentiable functions;

If a function is differentiable it has a derivative for each point in its domain

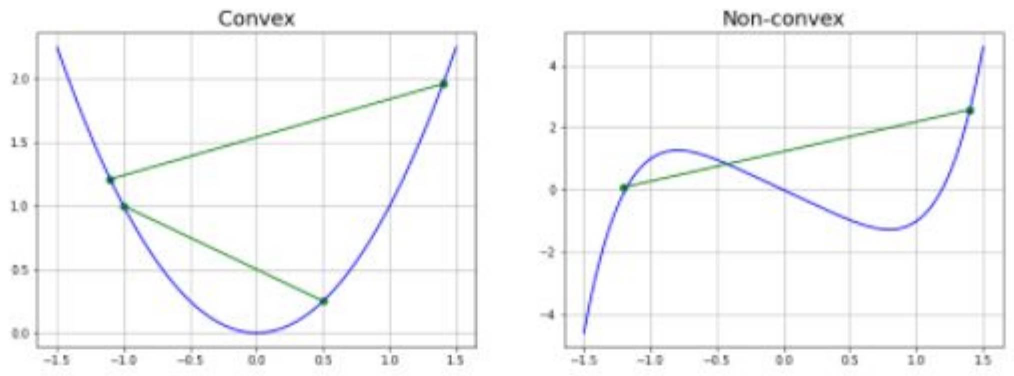
Typical non-differentiable functions have a step a cusp or a discontinuity

Neural Networks *- Gradient Descent*

Gradient descent algorithm does not work for all functions.

There are two specific requirements; a function has to be:

- differentiable
- convex



Exemplary convex and non-convex functions.

For a univariate function, this means that the line segment connecting two function's points lays on or above its curve (it does not cross it). If it does it means that it has a local minimum which is not a global one.

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda) f(x_2)$$

Another way to check mathematically if a univariate function is convex is to calculate the second derivative and check if its value is always bigger than 0.

$$\frac{d^2 f(x)}{dx^2} > 0$$

Neural Networks *- Gradient Descent*

What is a gradient?

It is a slope of a curve at a given point in a specified direction.

In the case of **a univariate function**, it is simply the **first derivative at a selected point**.

In the case of **a multivariate function**, it is a **vector of derivatives** in each main direction (along variable axes). Because we are interested only in a slope along one axis and we don't care about others these derivatives are called **partial derivatives**.

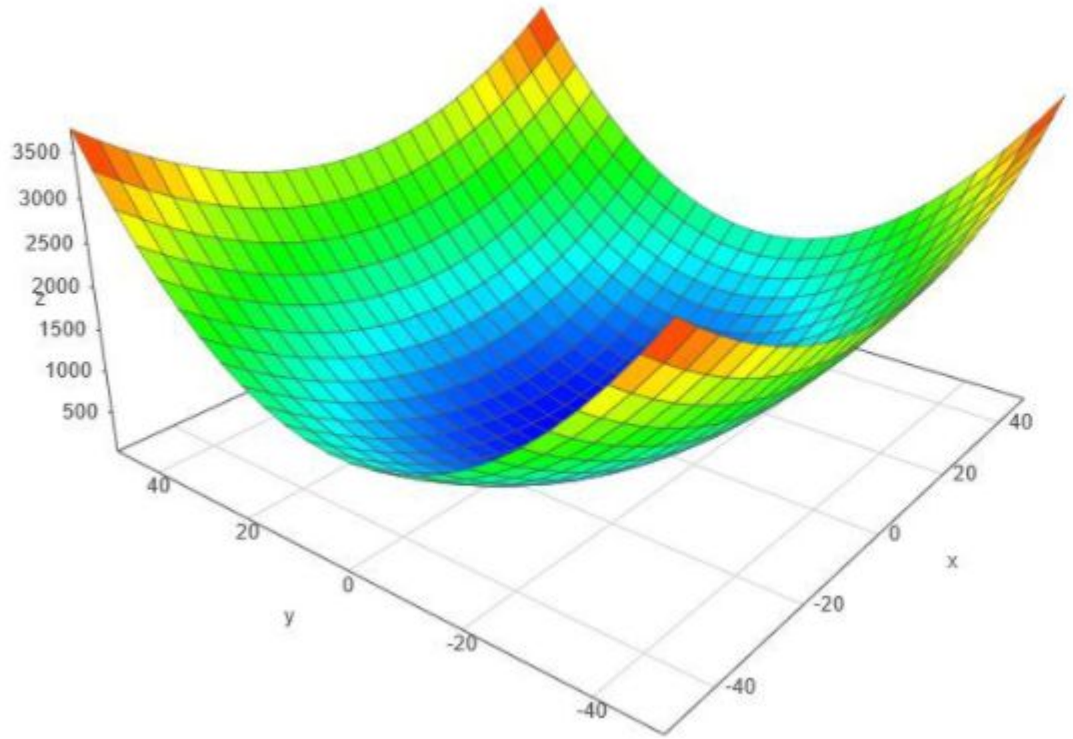
$$\nabla f(p) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(p) \\ \vdots \\ \frac{\partial f}{\partial x_n}(p) \end{bmatrix}$$

Neural Networks *- Gradient Descent*

What is a gradient?

It is a slope of a curve at a given point in a specified direction.

$$f(x) = 0.5x^2 + y^2$$



Let's assume we are interested in a gradient at point p(10,10):

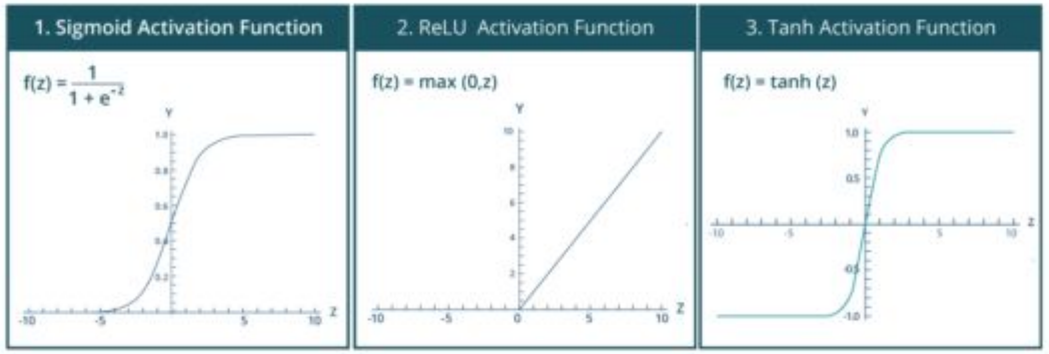
$$\frac{\partial f(x,y)}{\partial x} = x, \quad \frac{\partial f(x,y)}{\partial y} = 2y$$

$$\nabla f(x,y) = \begin{bmatrix} x \\ 2y \end{bmatrix}$$

$$\nabla f(10,10) = \begin{bmatrix} 10 \\ 20 \end{bmatrix}$$

Neural Networks *- Artificial Neural Networks*

Activation Functions


1. Sigmoid Activation Function
$f(z) = \frac{1}{1+e^{-z}}$
2. ReLU Activation Function
f(z) = max (0,z)
3. Tanh Activation Function
f(z) = tanh (z)