Neural Network & Deep Learning

Keras & TensorFlow

1

Hichem Felouat

hichemfel@gmail.com

https://www.linkedin.com/in/hichemfelouat/

Getting Data Into the Right Shape

Features

Label

Examples (Training sample + Test sample) = Dataset

	rank	discipline	yrs.since.phd	yrs.service	sex	salary
1	Prof	B	19	18	Male	139750
2	Prof	B	20	16	Male	173200
3	AsstProf	B	4	3	Male	79750
4	Prof	B	45	39	Male	115000
5	Prof	B	40	41	Male	141500
6	AssocProf	B	6	6	Male	97000
7	Prof	B	30	23	Male	175000
8	Prof	B	45	45	Male	147765
9	Prof	B	21	20	Male	119250
10	Prof	B	18	18	Female	129000

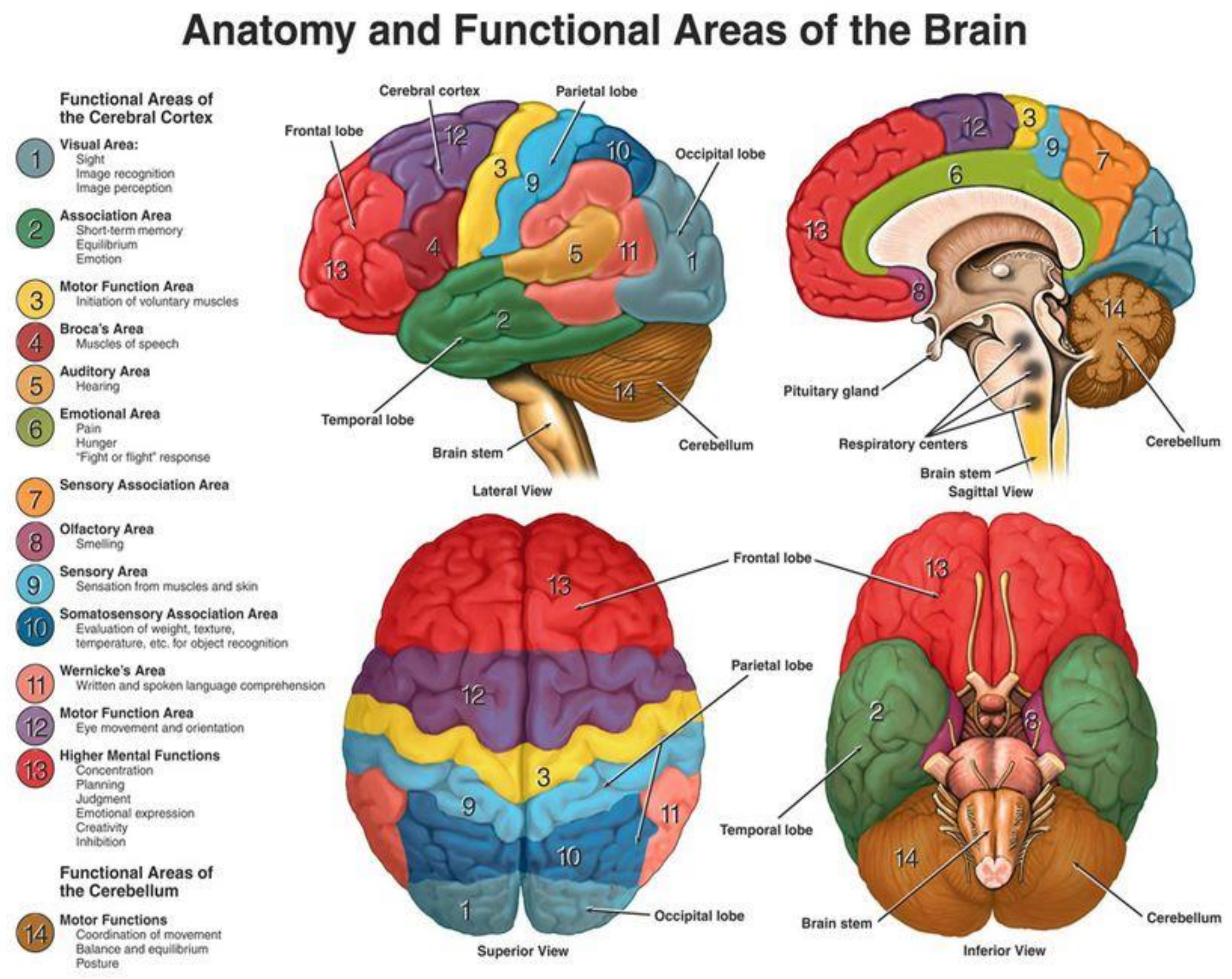
One Example

Steps to Build a NN

1. Data collection.
2. Improving data quality (data preprocessing: drop duplicate rows, handle missing values and outliers).
3. Feature engineering (feature extraction and selection, dimensionality reduction).
4. Splitting data into training (and evaluation) and testing sets.
5. Algorithm selection (Regression, Classification,).
6. Training.
7. Evaluation + Hyperparameter tuning.
8. Testing.
9. Deployment

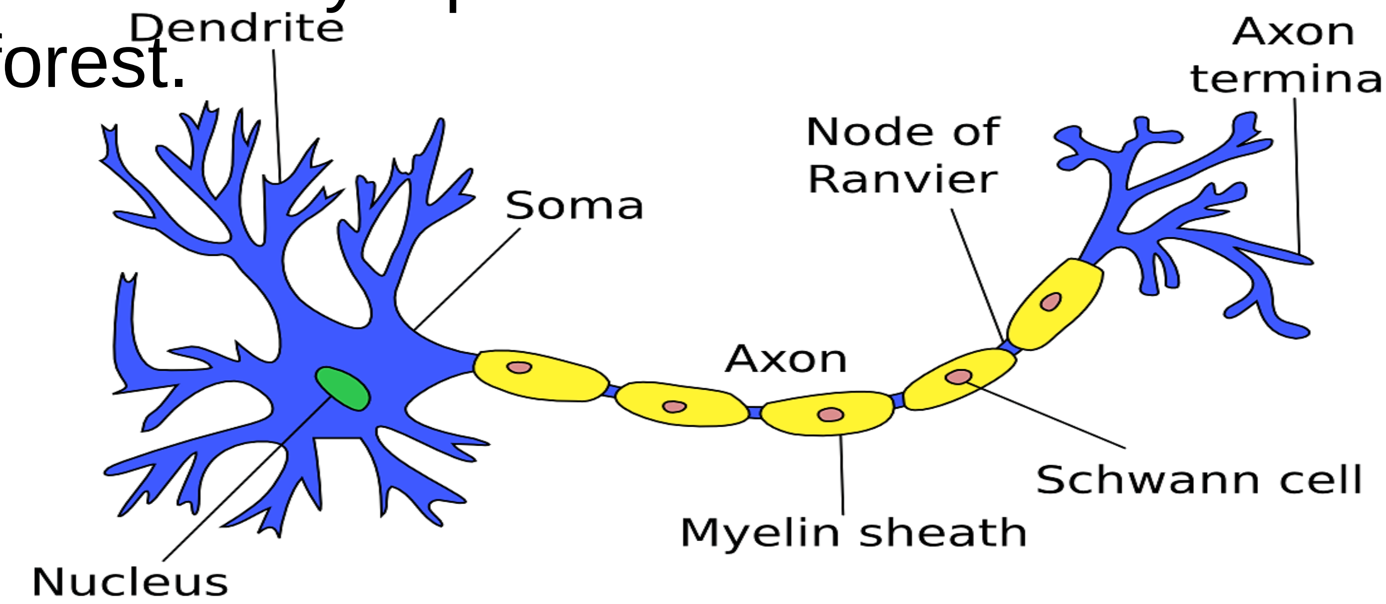
Artificial Intelligence

- **For a long time, scientists have been interested to understand how the human brain works.**
- **The brain has huge capabilities to perform complex tasks (e.g.,vision & speech recognition, language processing, etc.).**



Human Brain Network

- The brain is composed of **neurons** linked together by **synapses**.
- Human brain has **~10^11 neurons ≈** number of trees in the amazon forest.
- The number of synapses **≈** number of **tree leaves** in the amazon forest.

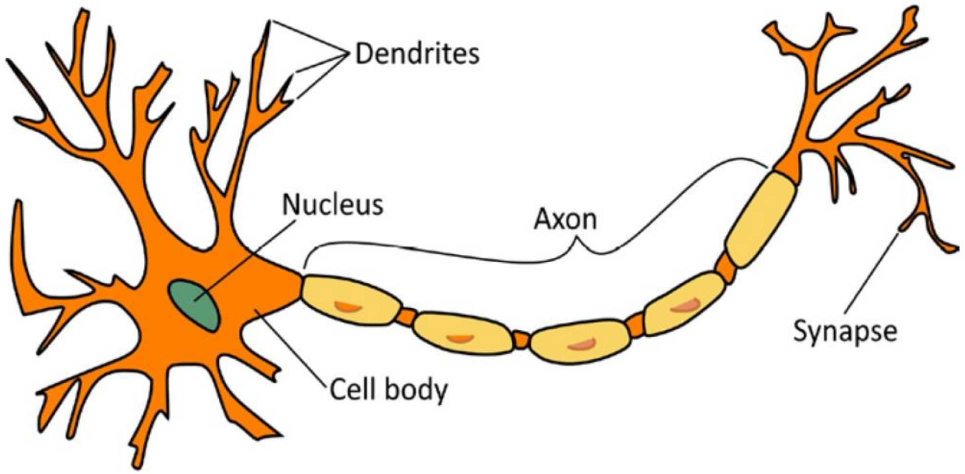


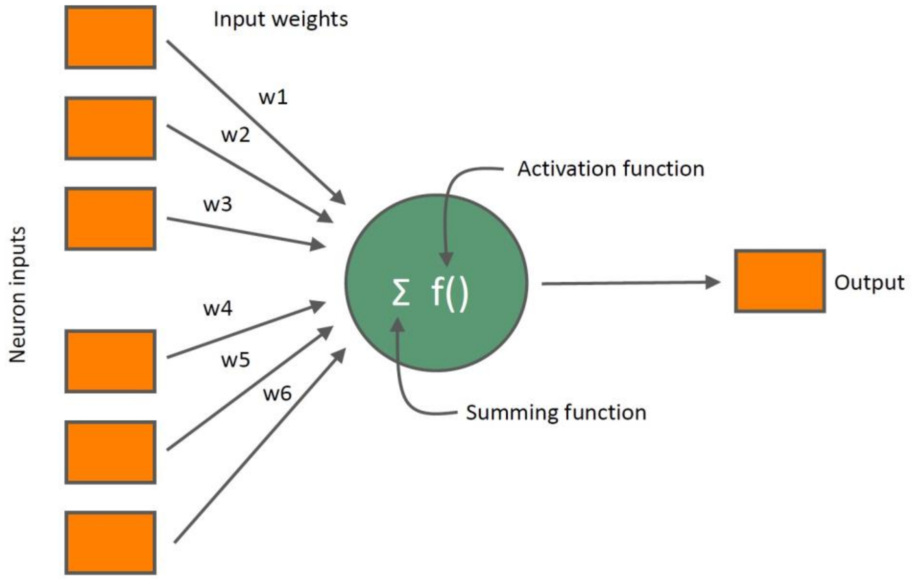
From Biological to Artificial Neurons

- Biological neurons are **fired** based on the intensity of the entering signals.

- Can be simulated by **activation functions**.

- **Artificial neurons** are arranged in layers and linked by **weights** in a similar way to **synapses** linking biological neurons.





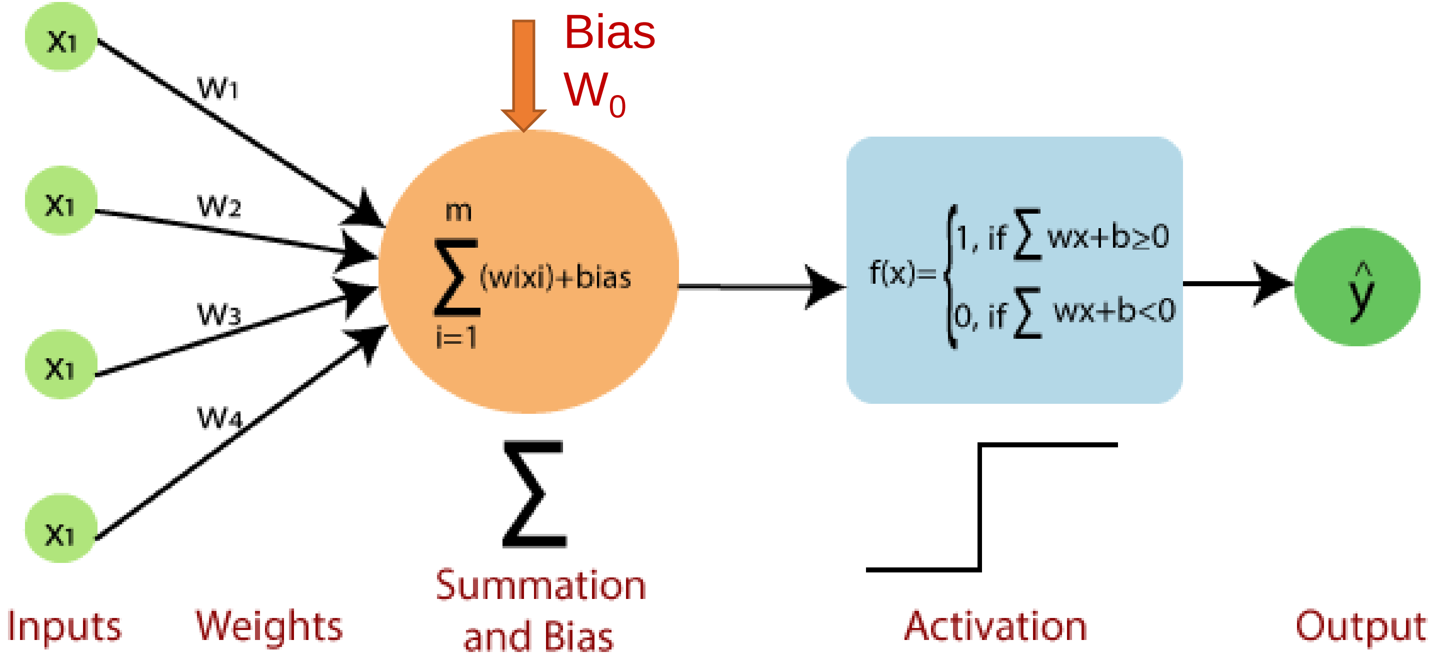
Perceptron

Perceptron

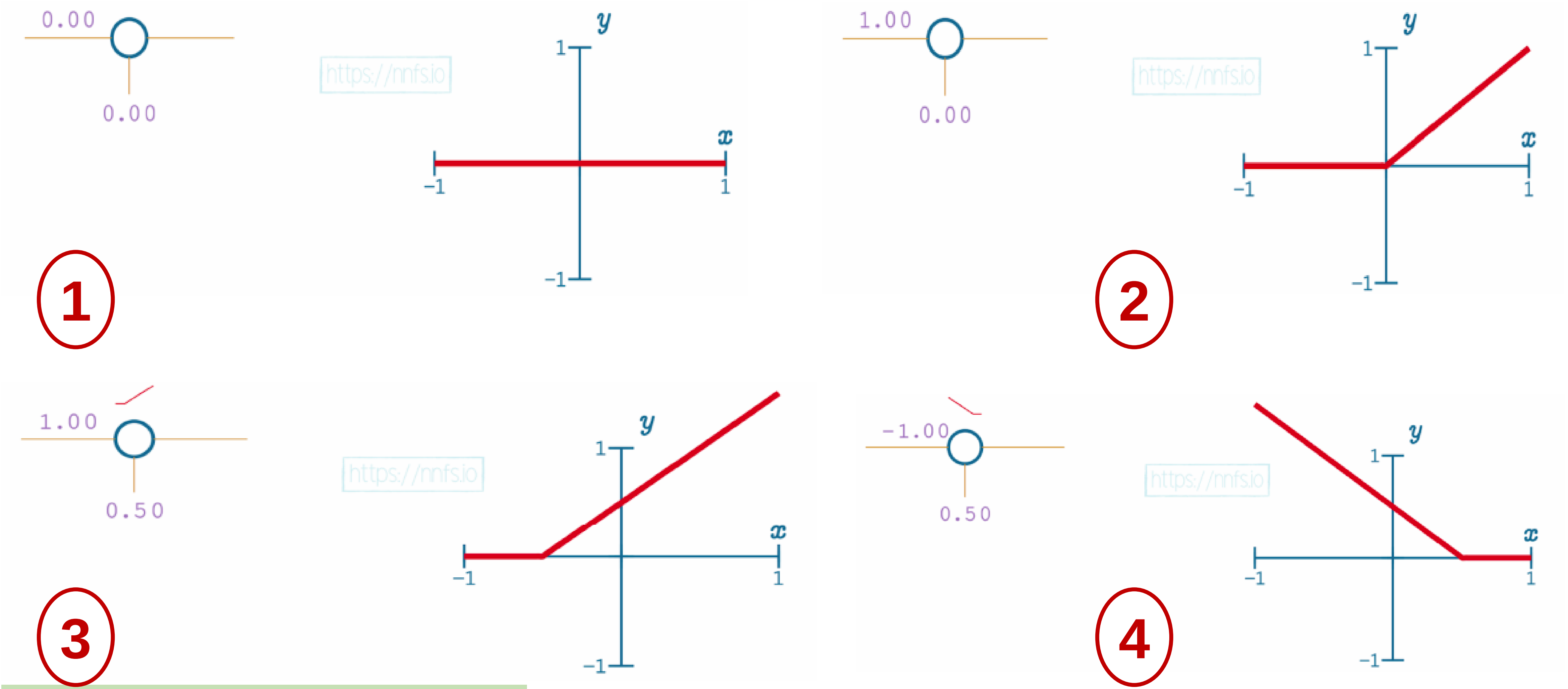
- Has been invented in **1957** par Frank Rosenblatt at the Aerospatiale labs at Cornell University.
- Used **analog hardware** to ensure **connections** between neurons.

Perceptron

- The inputs and output are numbers and each input connection is associated with a weight.
- Compute the **weighted sum of its inputs** then applies an **activation function** to that sum and **outputs the result**.



Perceptron - Activation Function and Bias



ReLU Activation = max(0,x)

Algorithm for Training a Perceptron

Input :($\mathcal{D}$, $\mathbf{w}^{(0)}$).

Output :$\mathbf{w}$.

for each data point $\mathbf{x}^{(j)}, j = 1, \dots n$, **do**

    if $\left(y^{(j)} = +1 \text{ and } f(\mathbf{x}^{(j)}) \leq 0\right)$ **then**

        $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \alpha\, \mathbf{x}^{(j)}$

    else   **if** $\left(y^{(j)} = -1 \text{ and } f(\mathbf{x}^{(j)}) \geq 0\right)$ **then**

            $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \alpha\, \mathbf{x}^{(j)}$

        else

            $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)}$.

        end

    end

end

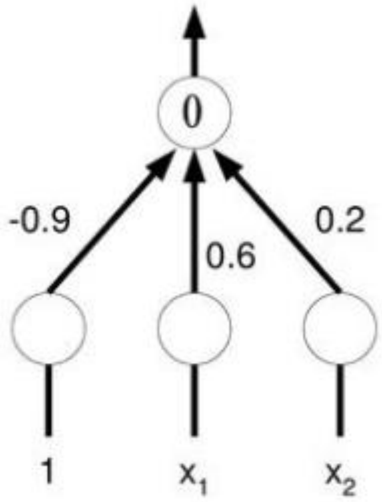
Algorithm for Training a Perceptron

Our Dataset

X1	X2	R
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

training set:

$x_1 = 1, x_2 = 1 \rightarrow 1$
$x_1 = 1, x_2 = -1 \rightarrow -1$
$x_1 = -1, x_2 = 1 \rightarrow -1$
$x_1 = -1, x_2 = -1 \rightarrow -1$



randomly, let: $w_0 = -0.9$, $w_1 = 0.6$, $w_2 = 0.2$

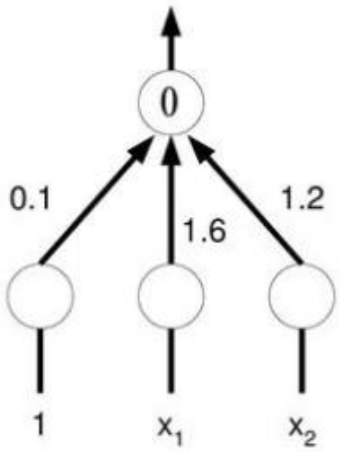
using these weights:

$x_1 = 1, x_2 = 1$: $-0.9*1 + 0.6*1 + 0.2*1 = -0.1 \rightarrow -1$ WRONG
$x_1 = 1, x_2 = -1$: $-0.9*1 + 0.6*1 + 0.2*-1 = -0.5 \rightarrow -1$ OK
$x_1 = -1, x_2 = 1$: $-0.9*1 + 0.6*-1 + 0.2*1 = -1.3 \rightarrow -1$ OK
$x_1 = -1, x_2 = -1$: $-0.9*1 + 0.6*-1 + 0.2*-1 = -1.7 \rightarrow -1$ OK

new weights: $w_0 = -0.9 + 1 = 0.1$
$w_1 = 0.6 + 1 = 1.6$
$w_2 = 0.2 + 1 = 1.2$

$\alpha = 1$

Algorithm for Training a Perceptron



using these updated weights:

$x_1 = 1, x_2 = 1: 0.1*1 + 1.6*1 + 1.2*1 = 2.9 \rightarrow 1$ OK

$x_1 = 1, x_2 = -1: 0.1*1 + 1.6*1 + 1.2*-1 = 0.5 \rightarrow 1$ WRONG

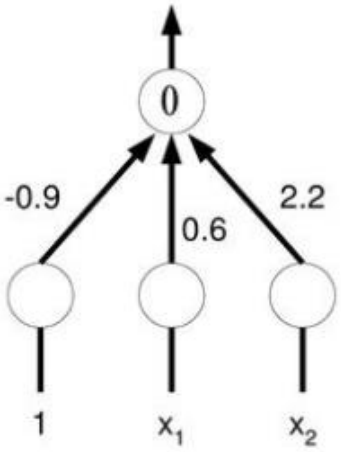
$x_1 = -1, x_2 = 1: 0.1*1 + 1.6*-1 + 1.2*1 = -0.3 \rightarrow -1$ OK

$x_1 = -1, x_2 = -1: 0.1*1 + 1.6*-1 + 1.2*-1 = -2.7 \rightarrow -1$ OK

new weights: $w_0 = 0.1 - 1 = -0.9$

$w_1 = 1.6 - 1 = 0.6$

$w_2 = 1.2 + 1 = 2.2$



using these updated weights:

$x_1 = 1, x_2 = 1: -0.9*1 + 0.6*1 + 2.2*1 = 1.9 \rightarrow 1$ OK

$x_1 = 1, x_2 = -1: -0.9*1 + 0.6*1 + 2.2*-1 = -2.5 \rightarrow -1$ OK

$x_1 = -1, x_2 = 1: -0.9*1 + 0.6*-1 + 2.2*1 = 0.7 \rightarrow 1$ WRONG

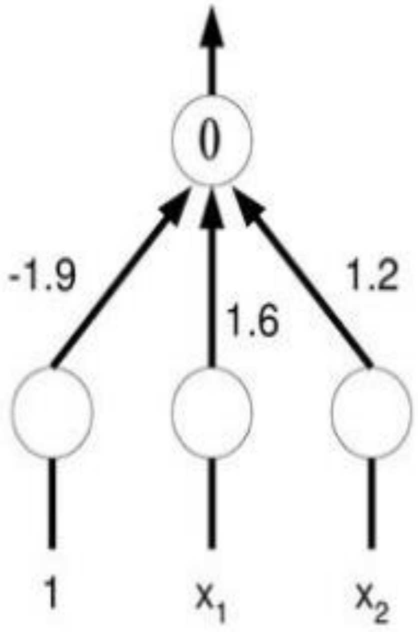
$x_1 = -1, x_2 = -1: -0.9*1 + 0.6*-1 + 2.2*-1 = -3.7 \rightarrow -1$ OK

new weights: $w_0 = -0.9 - 1 = -1.9$

$w_1 = 0.6 + 1 = 1.6$

$w_2 = 2.2 - 1 = 1.2$

Algorithm for Training a Perceptron



using these updated weights:

$x_1 = 1, x_2 = 1: -1.9*1 + 1.6*1 + 1.2*1 = 0.9 \rightarrow 1$ OK

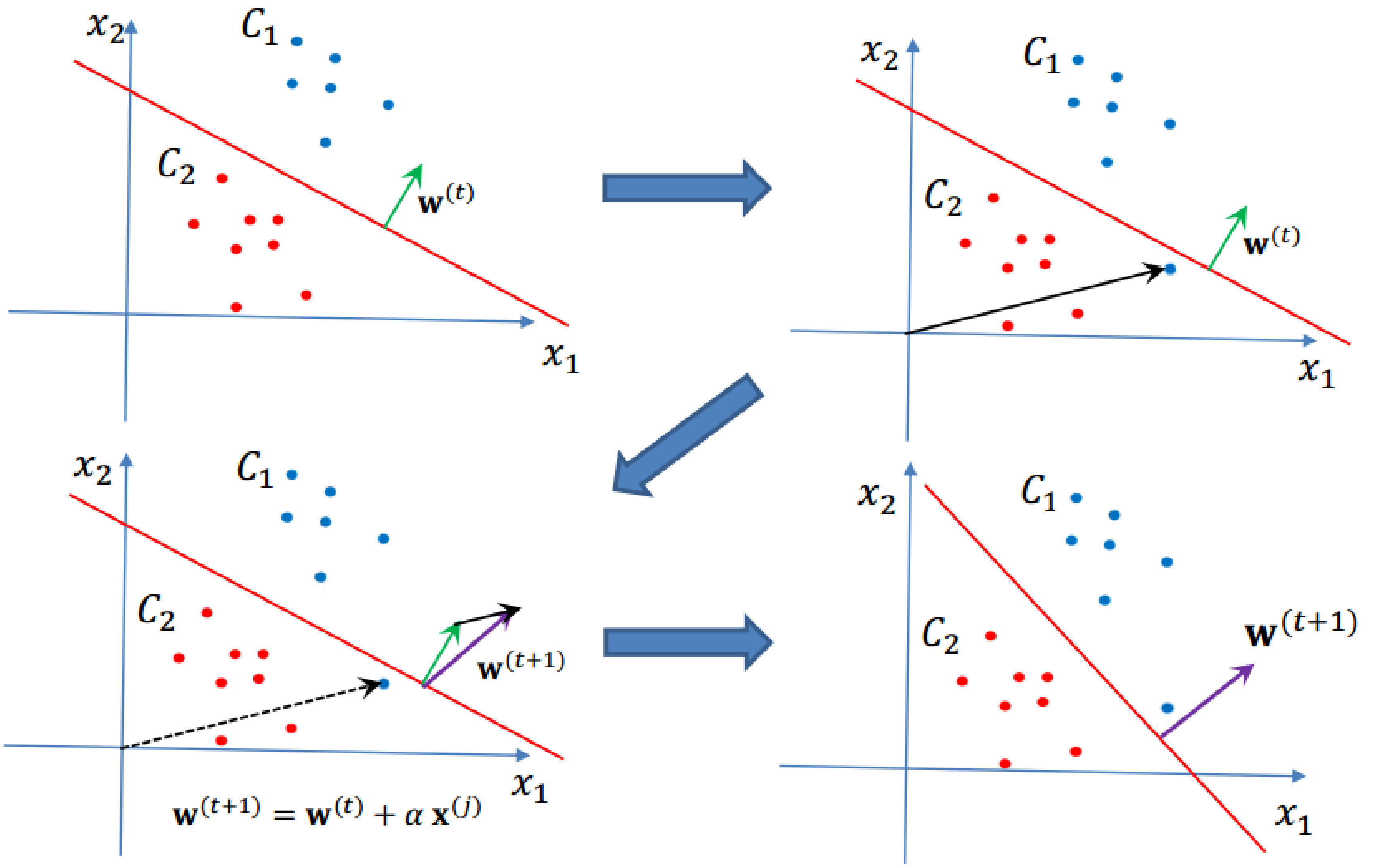
$x_1 = 1, x_2 = -1: -1.9*1 + 1.6*1 + 1.2*-1 = -1.5 \rightarrow -1$ OK

$x_1 = -1, x_2 = 1: -1.9*1 + 1.6*-1 + 1.2*1 = -2.3 \rightarrow -1$ OK

$x_1 = -1, x_2 = -1: -1.9*1 + 1.6*-1 + 1.2*-1 = -4.7 \rightarrow -1$ OK

DONE!

Algorithm for Training a Perceptron

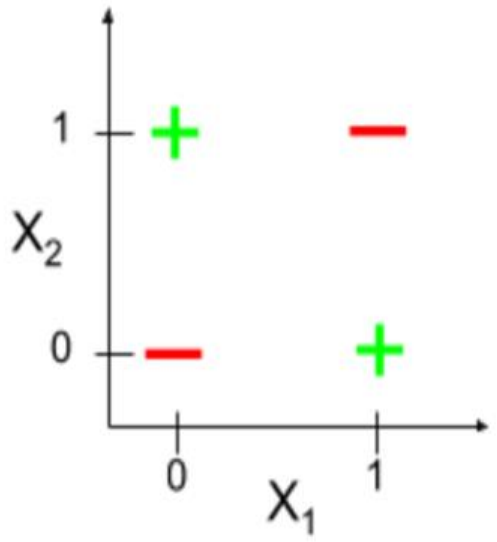


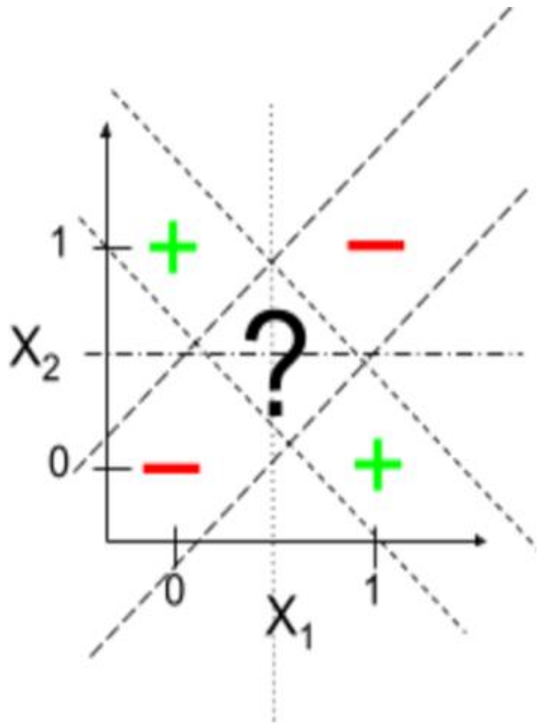
Limitations of Perceptron

- A perceptron can only separate **linearly separable classes**, but it is unable to separate **non-linear class boundaries**.

Example: Let the following problem of binary classification(problem of the **XOR**) (1969).

- **Clearly, no line can separate the two classes!**





Solution :

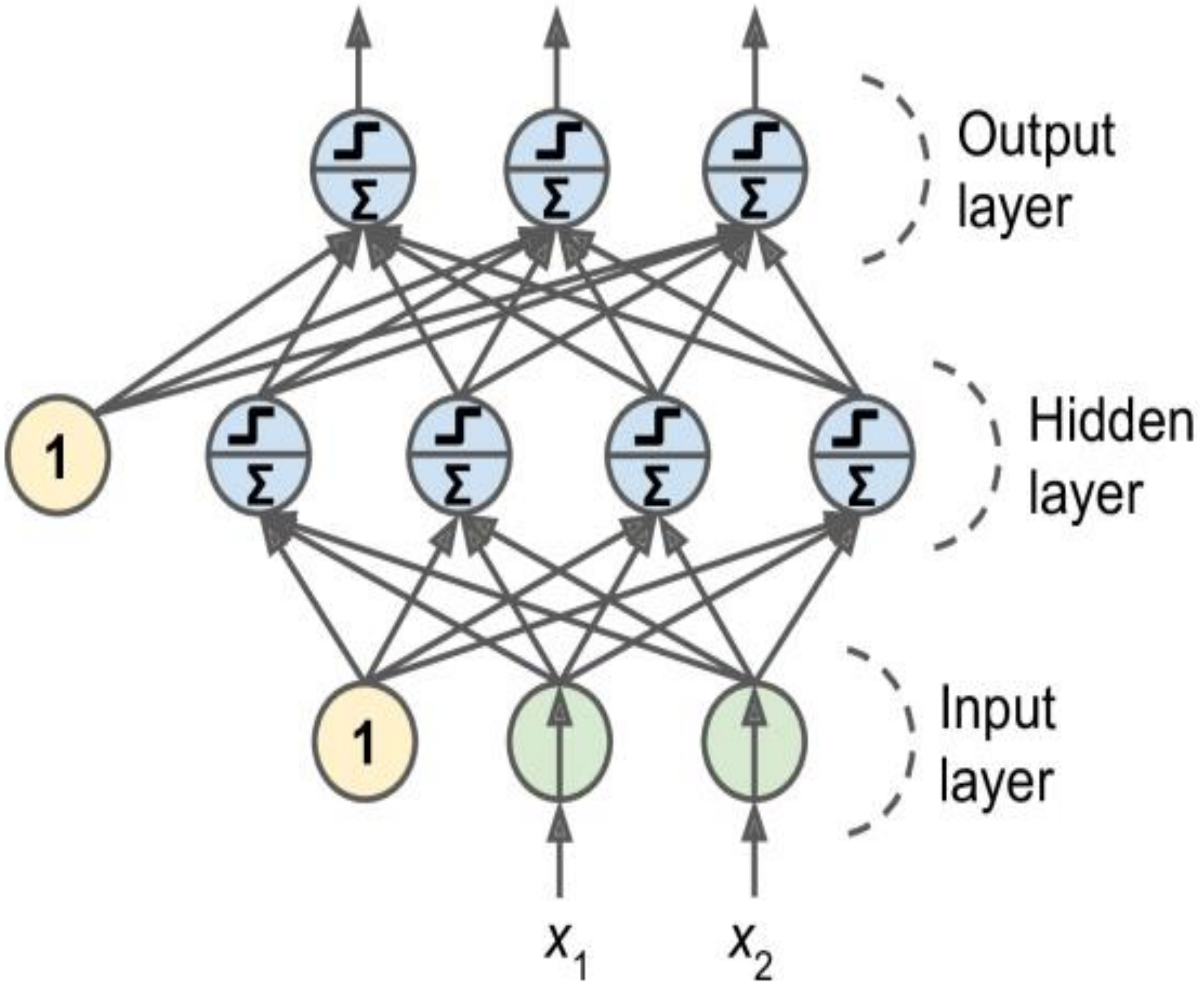
- **Use two lines instead of one!**
- **Use an intermediary layer of neurons in the NN.**

The Multilayer Perceptron MLP

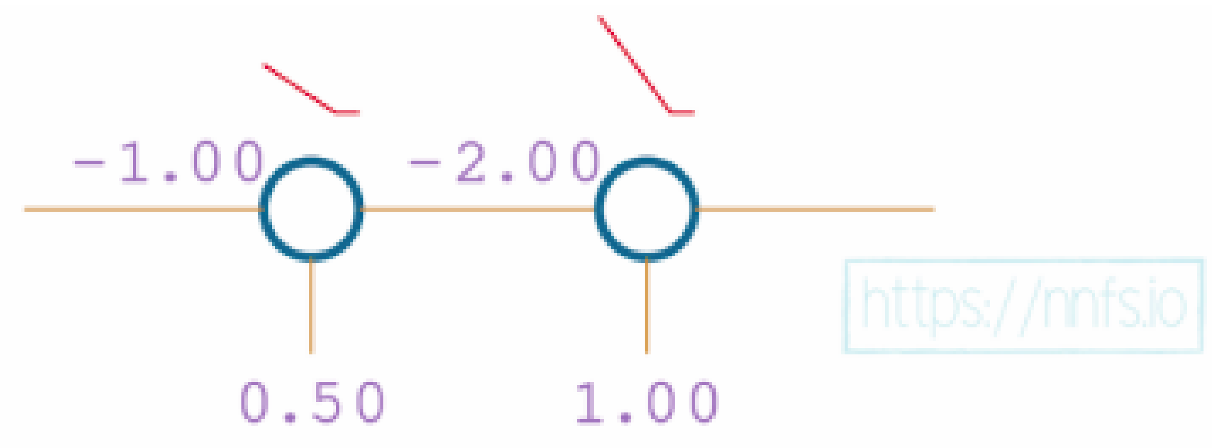
The Multilayer Perceptron MLP

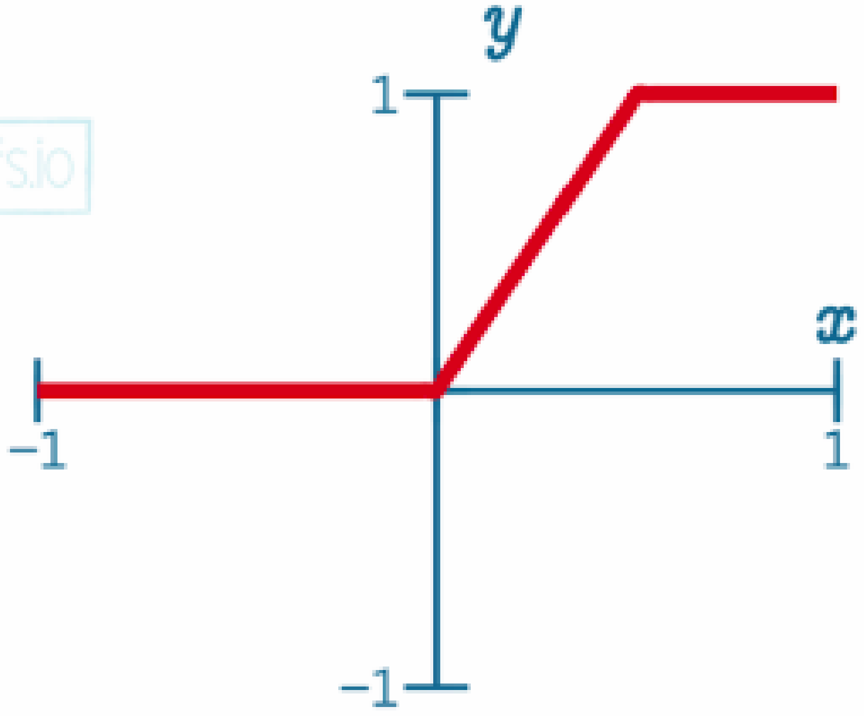
- The signal flows only in one direction (from the inputs to the outputs), so this architecture is an example of a **feedforward neural network (FNN)**.

- When an ANN contains a deep stack of hidden layers, it is called a **deep neural network (DNN)**.



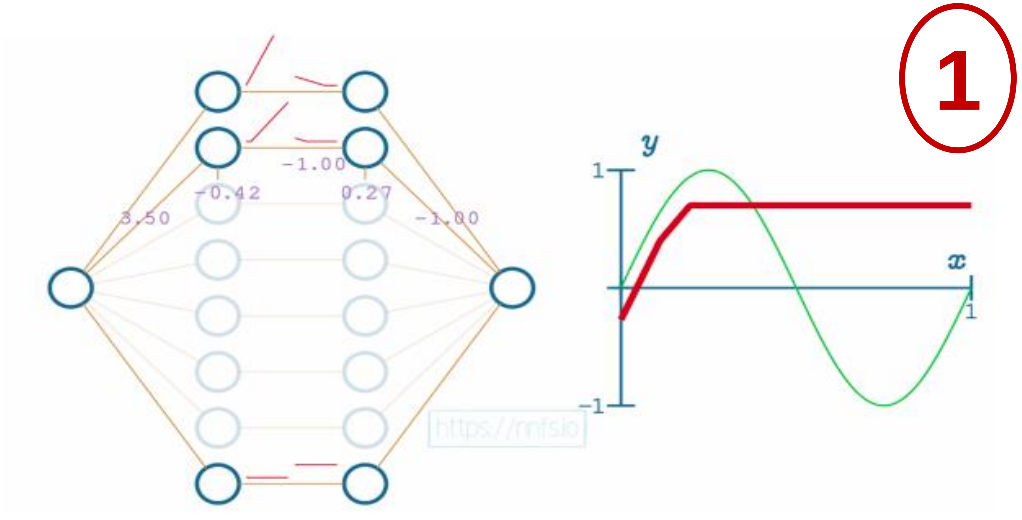
MLP - Activation Function in The Hidden Layers

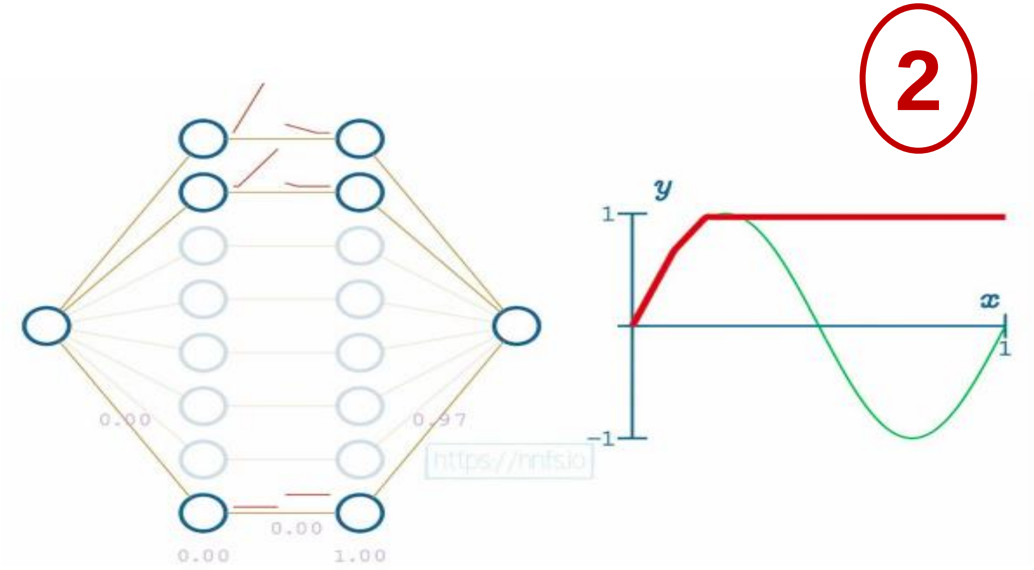


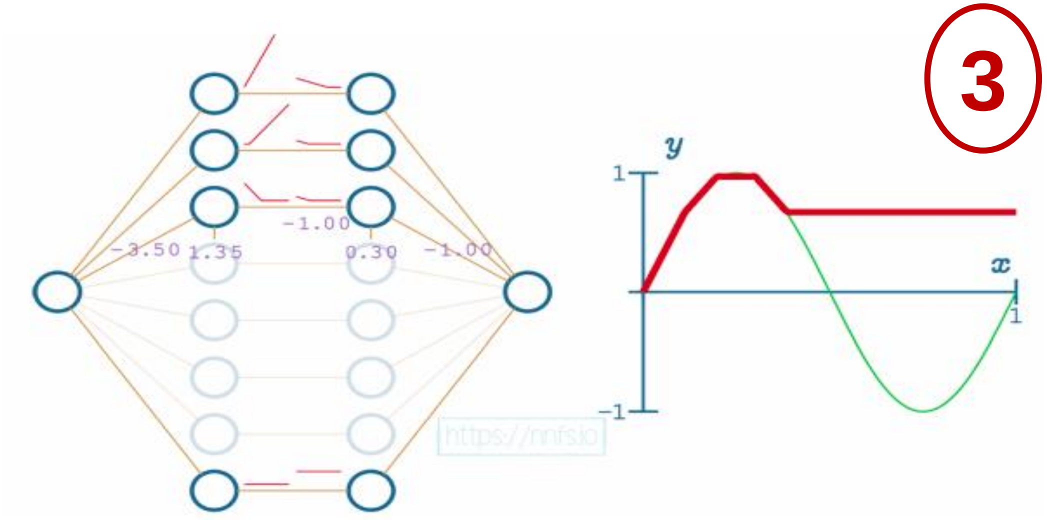


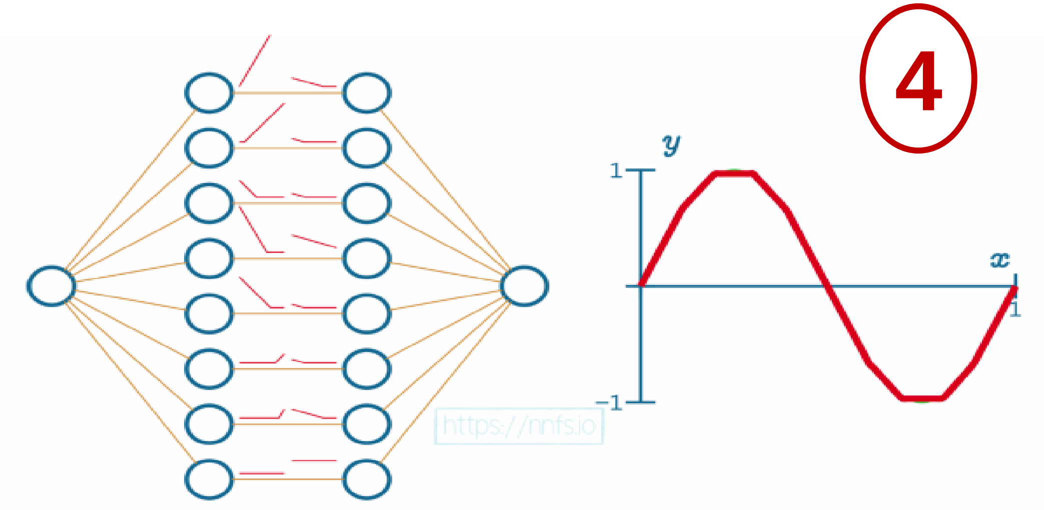
ReLU Activation = max(0,x)

MLP - Activation Function in The Hidden Layers



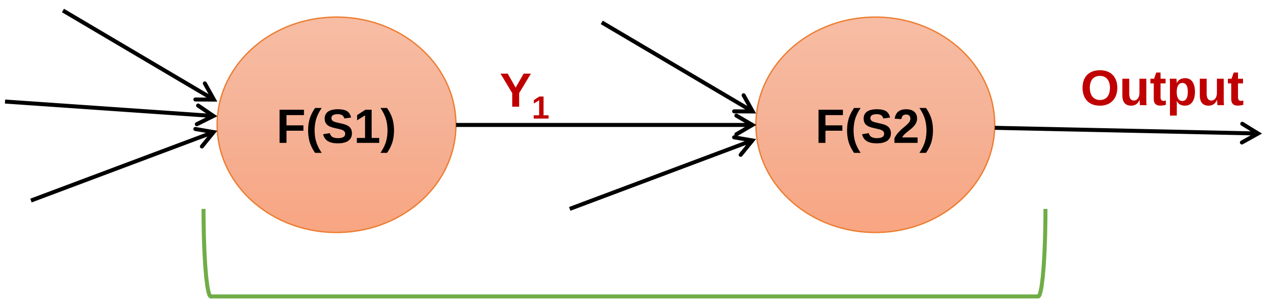






ReLU Activation = max(0,x)

MLP - Activation Function in The Hidden Layers



Output ≈ F(F(S1))

- **The activation function must be nonlinear**

$F(x) = 3x - 1$
$F(F(x)) = 9x - 4$
$F(F(F(x))) = 81x - 37$

$G(x) = 1/\ 1+\exp(-x)$
$G(G(x)) = 1/\ 1+\exp(-(1/\ 1+\exp(-x)))$

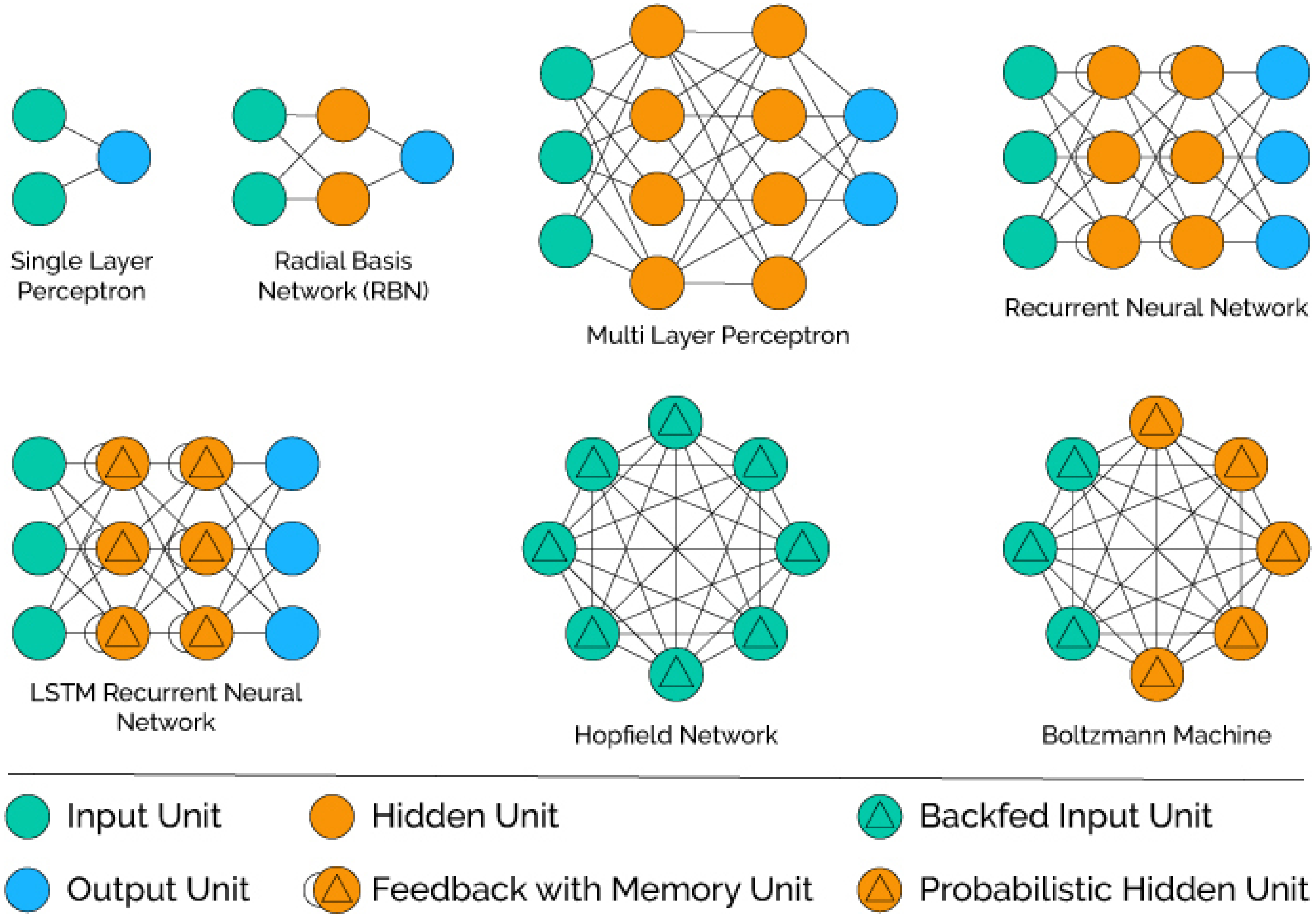
The Multilayer Perceptron MLP

- In 1986, the **backpropagation** training algorithm was introduced, which is still used today.

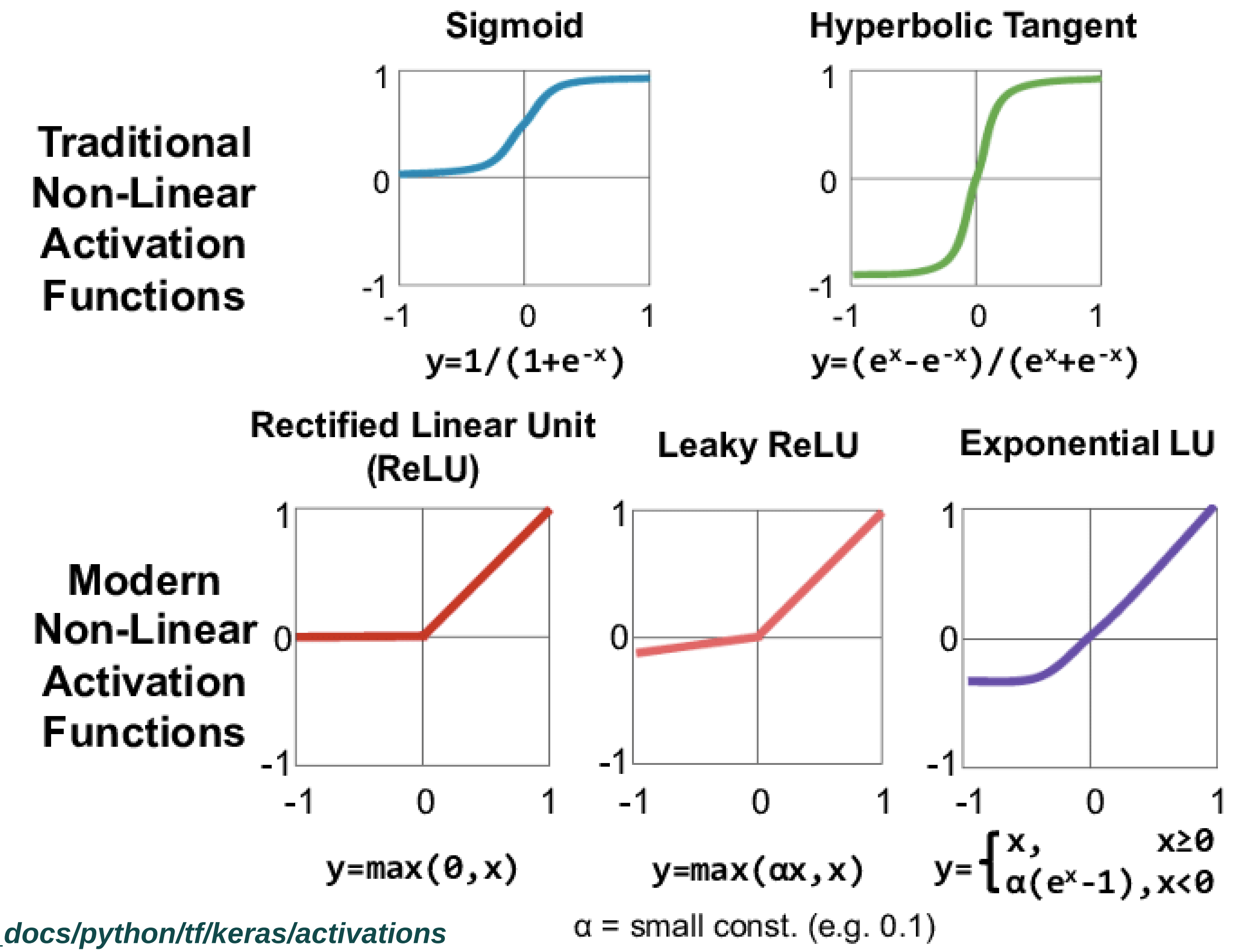
- **The backpropagation** consists of only two passes through the network **(one forward, one backward)**, the backpropagation algorithm is able to compute the gradient of the **network's error** with regard to every single model parameter. In other words, it can find out how each **connection weight** and each **bias** term should be tweaked in order to reduce the error.

David Rumelhart et al. "Learning Internal Representations by Error Propagation," (Defense Technical Information Center technical report, September 1985).

Popular NN Architecture



Popular Activation functions for MLP



https://www.tensorflow.org/api_docs/python/tf/keras/activations

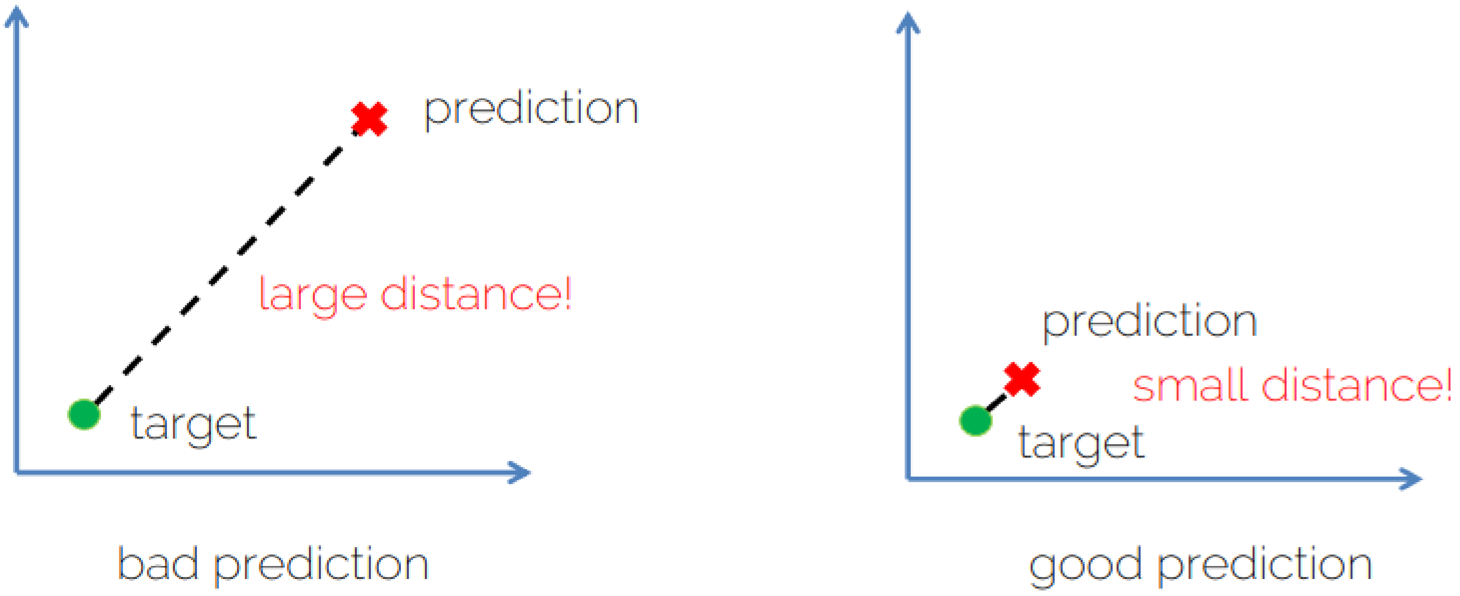
Neural Network vocabulary

1) Cost Function
2) Gradient Descent
3) Learning Rate
4) Backpropagation
5) Batches
6) Epochs

Neural Network vocabulary

Cost Function: When we build a network, the network tries to predict the output as close as possible to the actual value. We measure this accuracy of the network using the **cost/loss function**. **Idea: calculate a "distance" between prediction and target!**

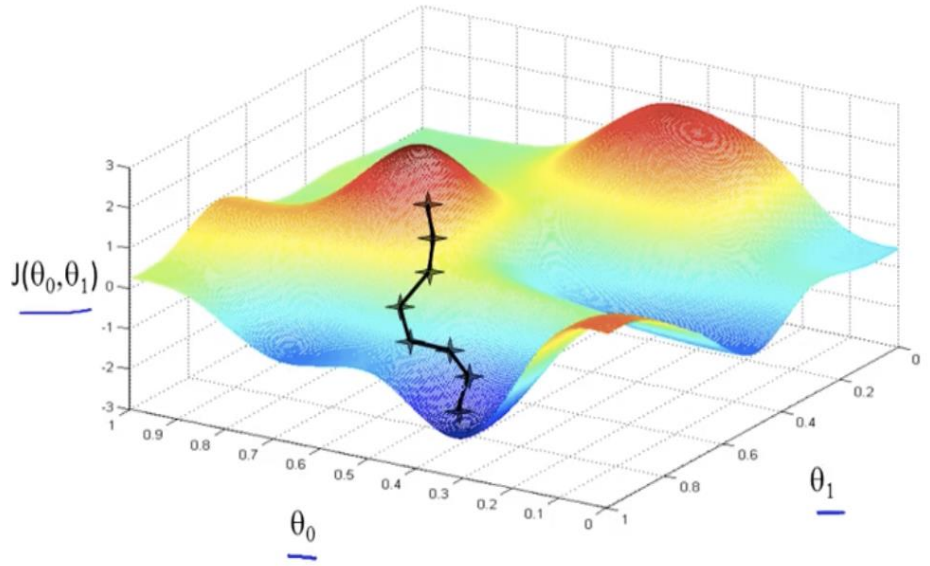
The choice of the loss function depends on the concrete problem or the distribution of the target variable.

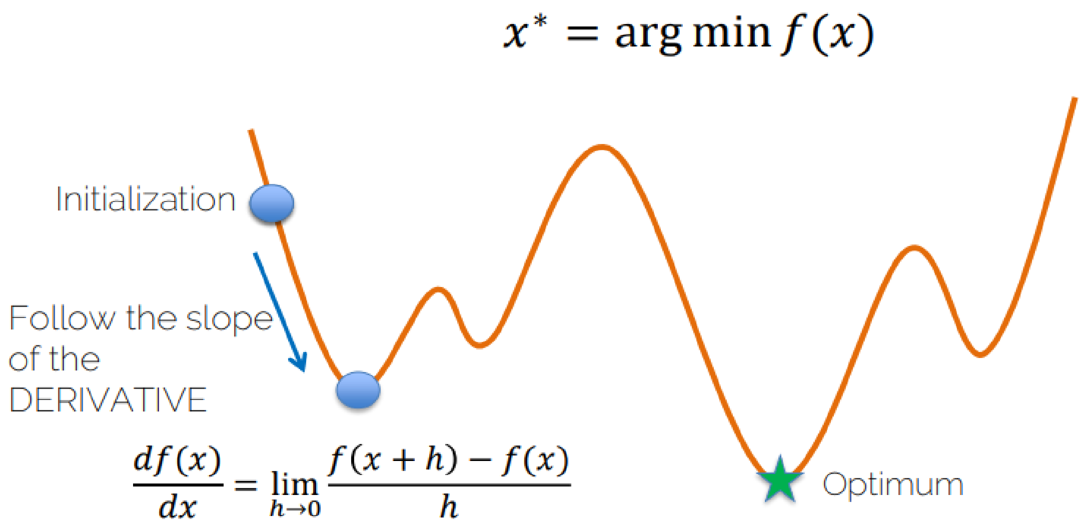


Neural Network vocabulary

Gradient descent: is an optimization algorithm used to minimize some function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient. In machine learning, we use gradient descent to **update the parameters of our model**. Parameters refer to **weights** in neural networks.

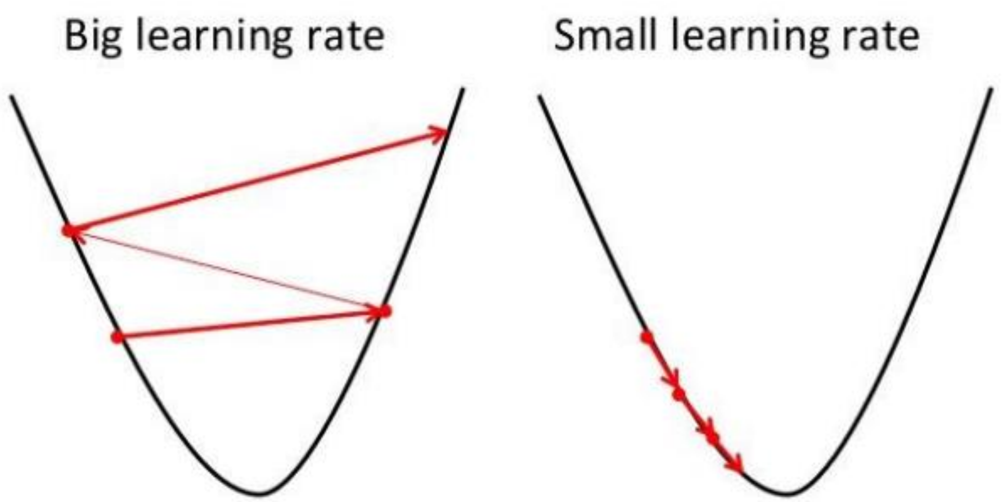
To put it simply, we use gradient descent to minimize the cost function, **J(w)**.

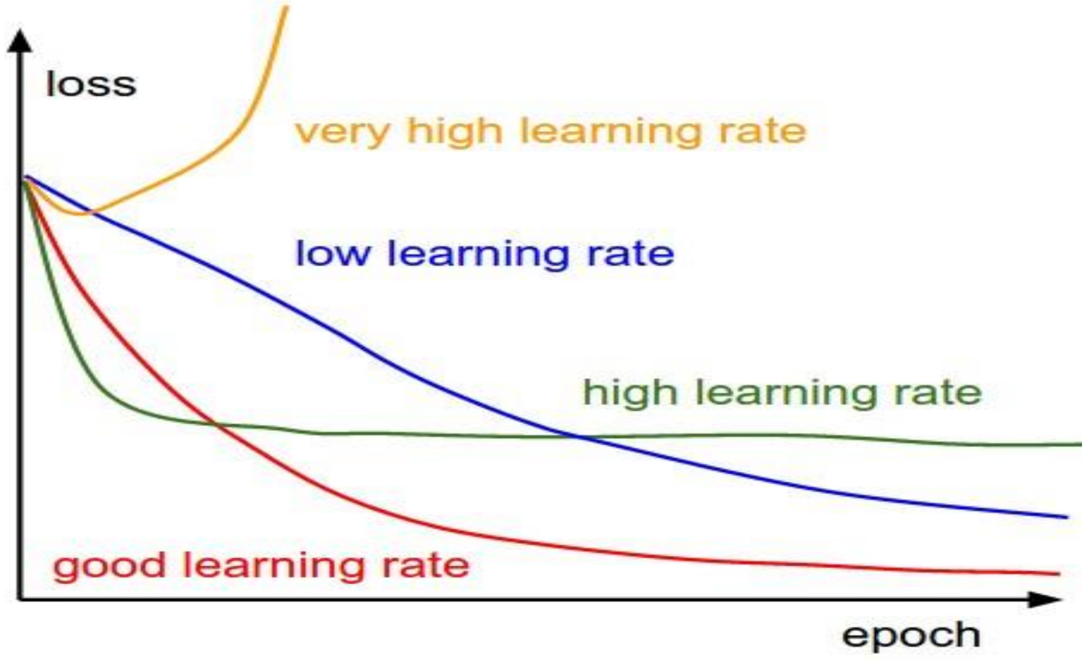




Neural Network vocabulary

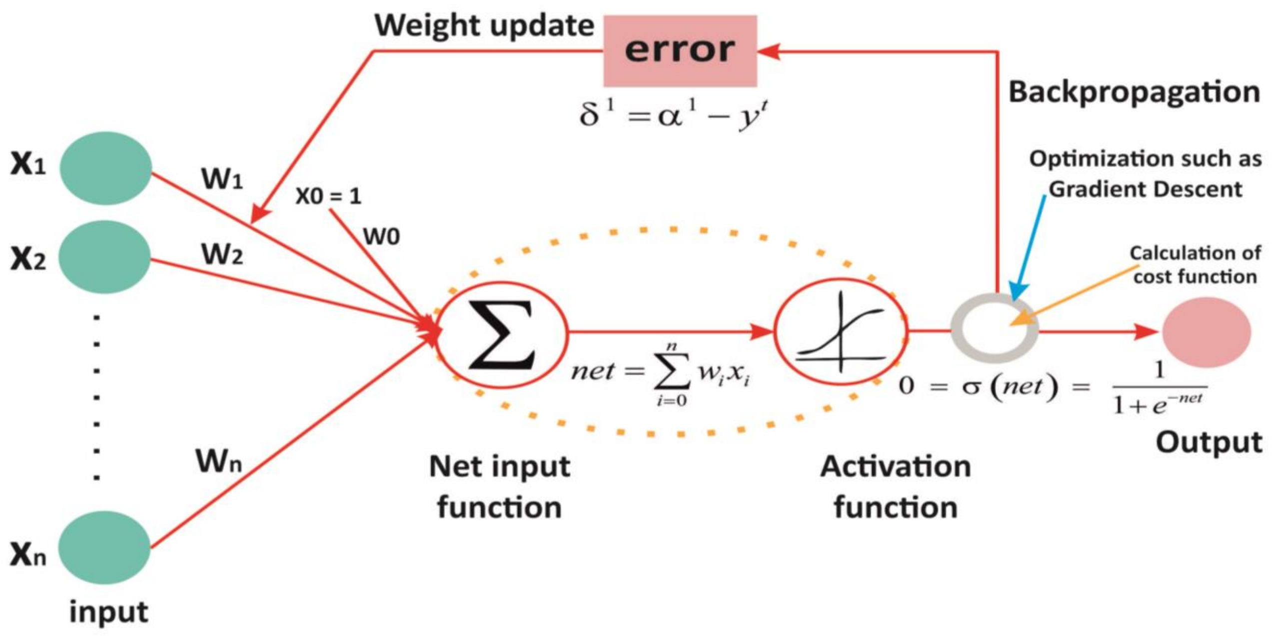
Learning Rate: is defined as the amount of minimization in the cost function in each iteration. In simple terms, the rate at which we descend towards the minima of the cost function is the learning rate. We should choose the learning rate very carefully since it should neither be very large that the optimal solution is missed and nor should be very low that it takes forever for the network to converge.





Neural Network vocabulary

Backpropagation: after we calculated the error of the network, this error is then fed back to the network along with the gradient of the cost function to update the weights of the network. These weights are then updated so that the errors in the subsequent iterations is reduced. This updating of weights using the gradient of the cost function is known as backpropagation.



Neural Network vocabulary

Batches: while training a neural network, instead of sending the entire input in one go, **we divide in input into several chunks of equal size randomly**. Training the data on batches makes the model **more generalized** as compared to the model built when the entire data set is fed to the network in one go.

- **It requires less memory.**
- **Typically networks train faster with mini-batches.**

- **Batch Gradient Descent : Batch Size = Size of Training Set**
- **Stochastic Gradient Descent: Batch Size = 1**
- **Mini-Batch Gradient Descent: 1 < Batch Size < Size of Training Set**

Neural Network vocabulary

Epochs: an epoch is defined as **a single training iteration** of all batches in both **forward and backpropagation**. This means **1 epoch** is a single forward and backward pass of the entire input data.

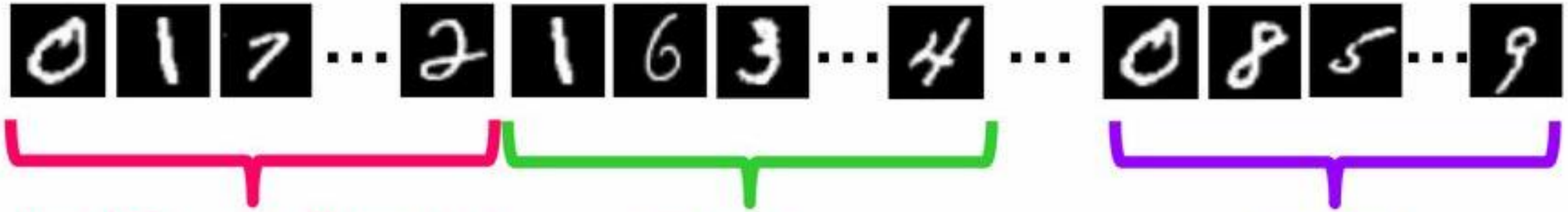
The number of epochs you would use to train your network can be **chosen by you**. It is highly likely that **more number of epochs would show higher accuracy** of the network, however, it would also take longer for the network to converge. Also, you must take care that if the number of epochs are too high, the network might be **over-fit**.

Neural Network vocabulary

Epoch / Iteration

Example: MNIST data

- number of training data: **N=55,000**
- Let's take batch size of **B=100**



First 100 images (=1st iteration) — 2nd iteration — Last iteration

- How many iteration in each epoch? **55000/100 = 550** $\boxed{\textbf{1 epoch = 550 iteration}}$

Neural Network vocabulary

Updating weights - In a neural network, weights are updated as follows:

- Step 1: Take a batch of training data.
- Step 2: Perform forward propagation to obtain the corresponding loss.
- Step 3: Backpropagate the loss to get the gradients.
- Step 4: Use the gradients to update the weights of the network.

Thank you for your attention

Hichem Felouat …