Master’s Degree in Intelligent Systems

Deep Learning

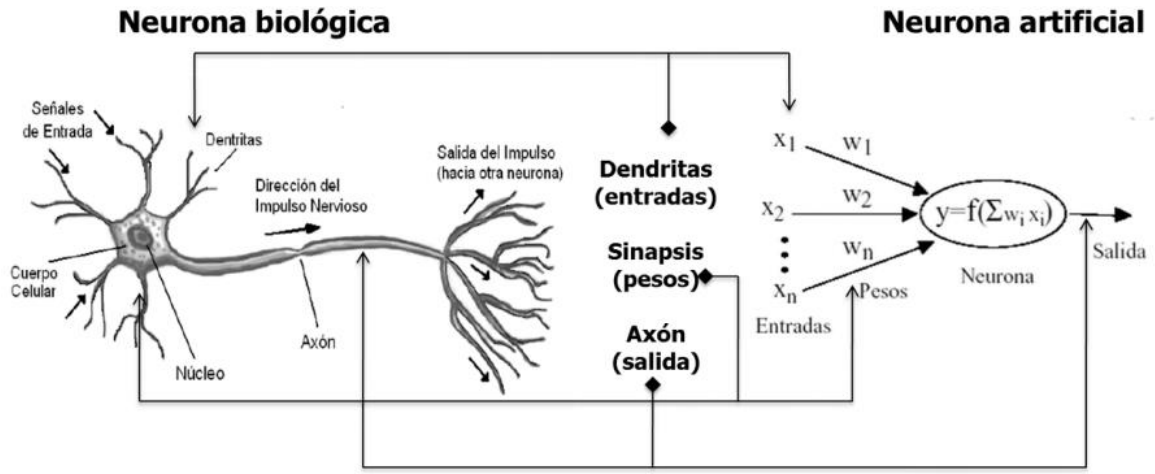
Manuel Piñar Molina & Miguel Ángel Calafat Torrens

Intro to Deep Learning and deep neural networks (DNN)

Images from udacity.com

Perceptron

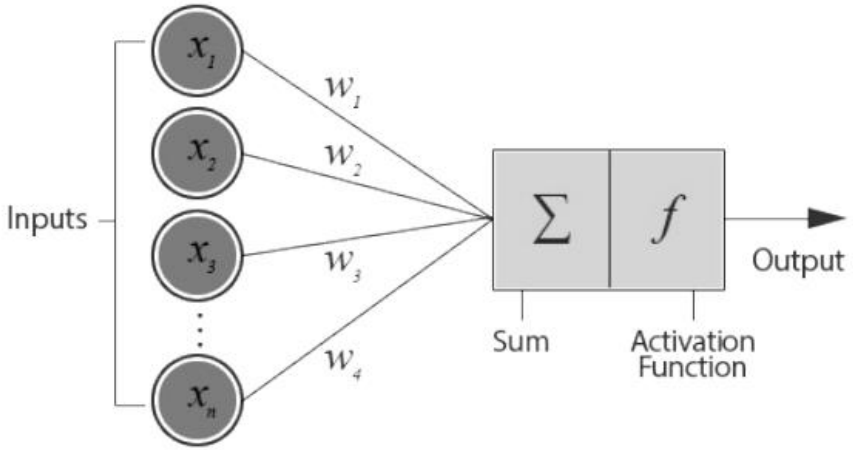
- What's a Perceptron?



A perceptron in the simplest neural network there is, is just a neuron. The perceptron works exactly like a biological neuron.

Perceptron

- How works a Perceptron?

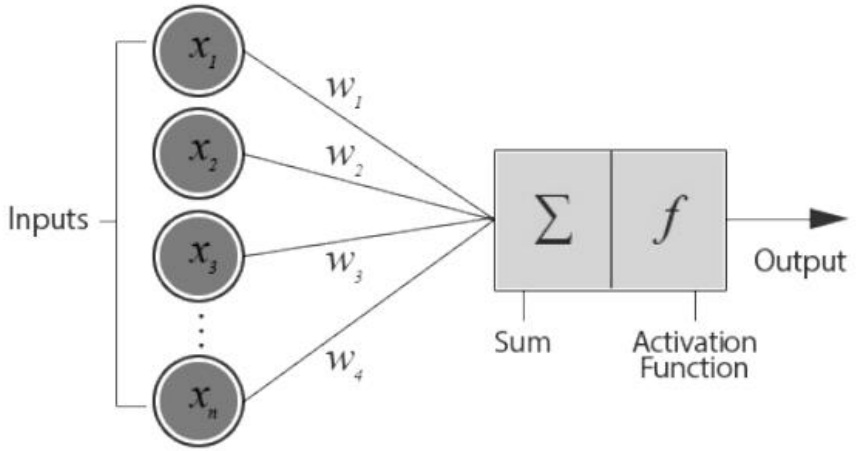


If we focus on artificial neurons, to model the behavior of the biological neuron, it will carry out two actions consecutively:

1) Will calculate the weighted sum of the inputs $[x_1, x_2, \ldots, x_n]$ to represent the total weight of the inputs
2) It will apply an activation function [f] on the output which will give us a result within certain values

Perceptron

- Who play the game?

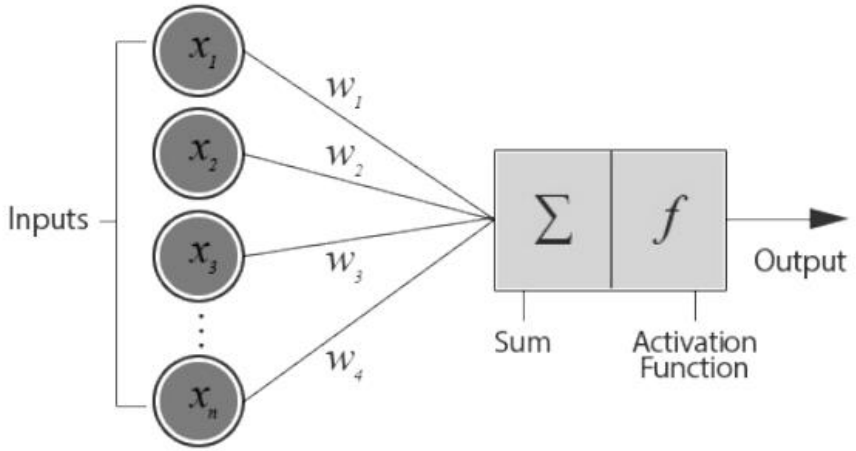


Elements that come into play:

1) ***Input vector***: is the vector $[x_1, x_2, \ldots, x_n]$, normally represented by X. Is the feature vector that feeds the neuron.
2) ***Weights vector***: is the vector $[w_1, w_2, \ldots, w_n]$, normally represented by W. Is the vector composed of each of the weights assigned to the inputs.
3) ***Neuron function***: the calculations performed within the neuron to modulate input.
4) ***Output***: the output is determined by the type of activation function. You can find different types of outputs depending on the chosen activation function.

Perceptron

- Weighted sum

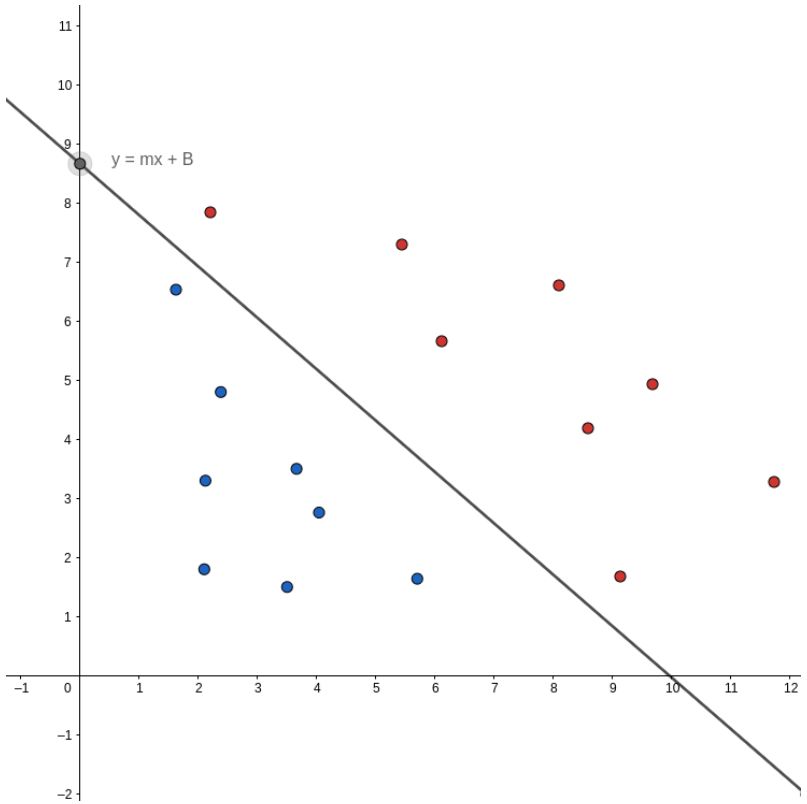


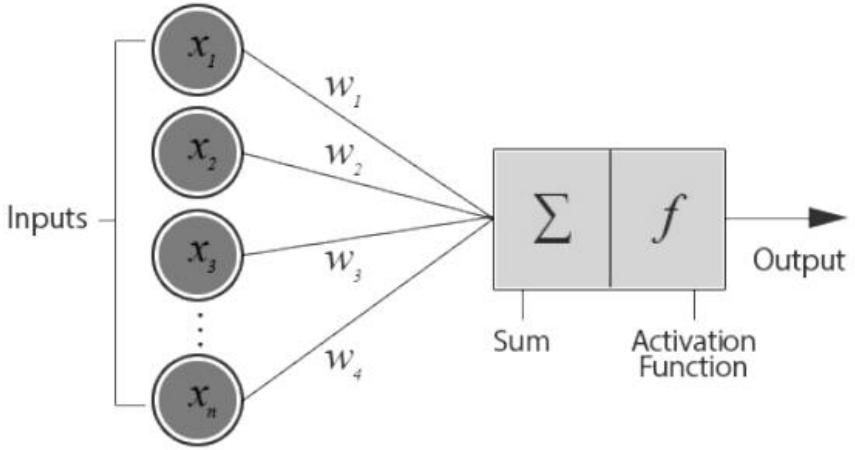
$$z = \sum x_i \ .w_i \ + b \ (bias)$$

$$z = x_1 \ .w_1 + x_2.w_2 + x_3.w_3 + ... + x_n \ .w_n \ + b$$

Perceptron

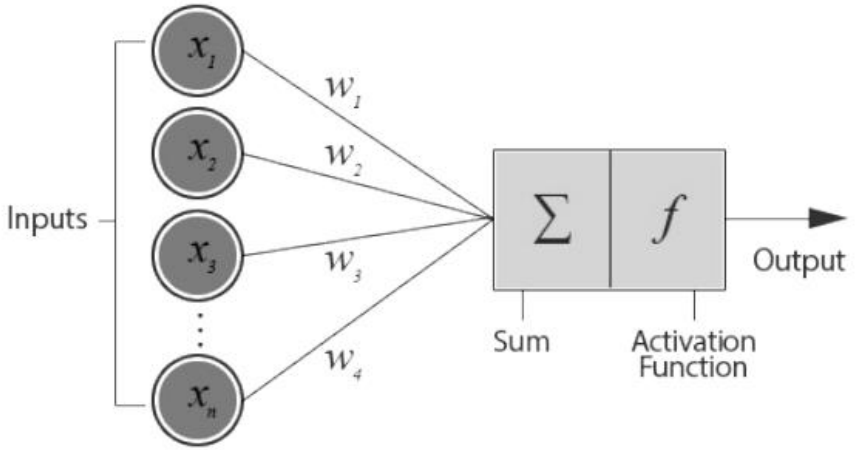
- Bias





Perceptron

- How the perceptron learn?



1) Passforward, compute the weighted sum and multiply it by the activation, so we get a prediction of the output ỹ:

$$\hat{y} = activation\ (\sum x_i . w_i + b)$$

2) We calculate the error (with the chosen error function, MSE, MAE, CROSS ENTROPY...)

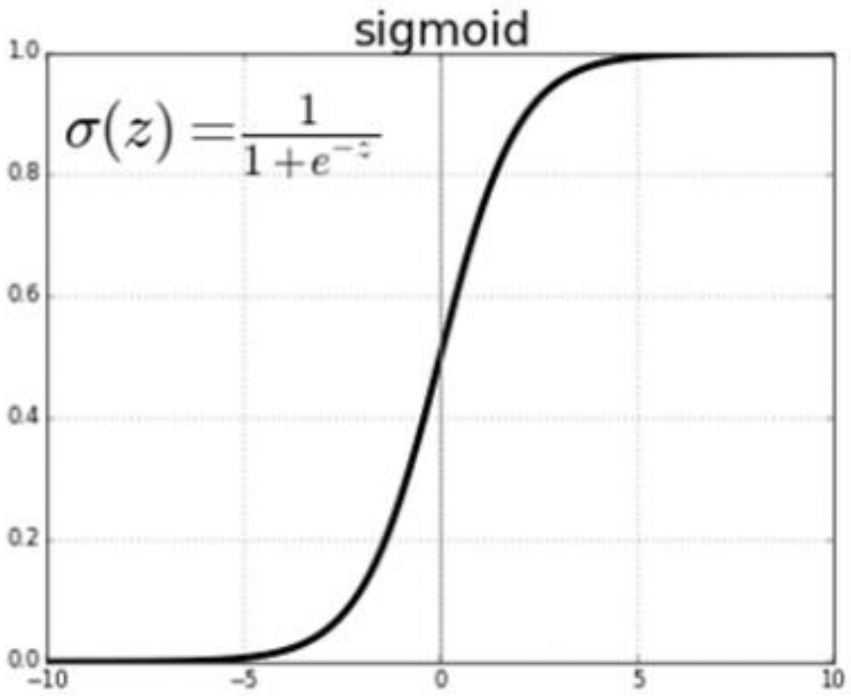
$$error = y - \tilde{y}\ (general)$$

3) We recalculate the weights W: if the prediction is very high or very low, the weights will be adjusted to bring the prediction closer to the expected output, thus minimizing the error term.

4) We repeat the previous points until the error is minimal or very close to zero, this means that our prediction will be very close to reality.

Activation functions

- **Sigmoid**

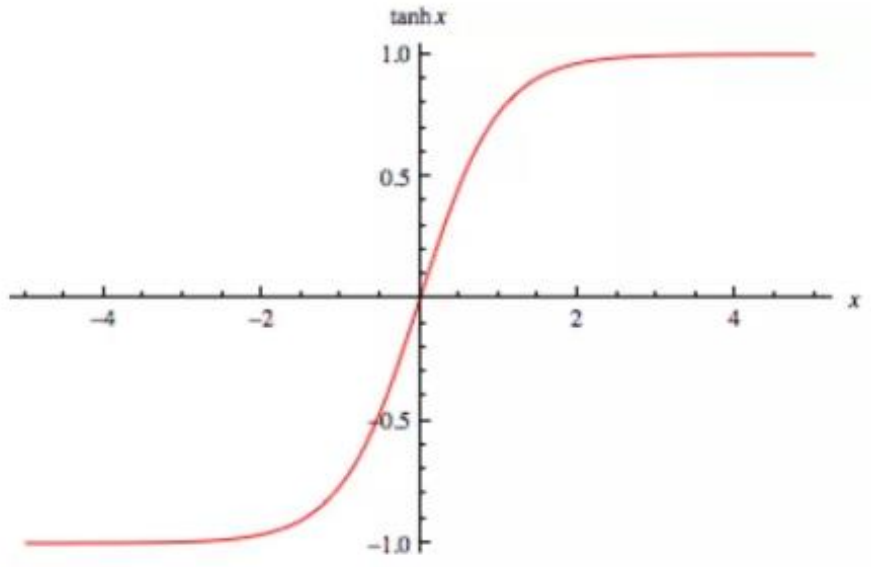


- Saturate and kill the gradient.
- Slow convergence.
- It is not centered at zero.
- It is bounded between 0 and 1.
- Good performance in the last layer.

Activation functions

- **Tanh Function (Hyperbolic Tangent)**

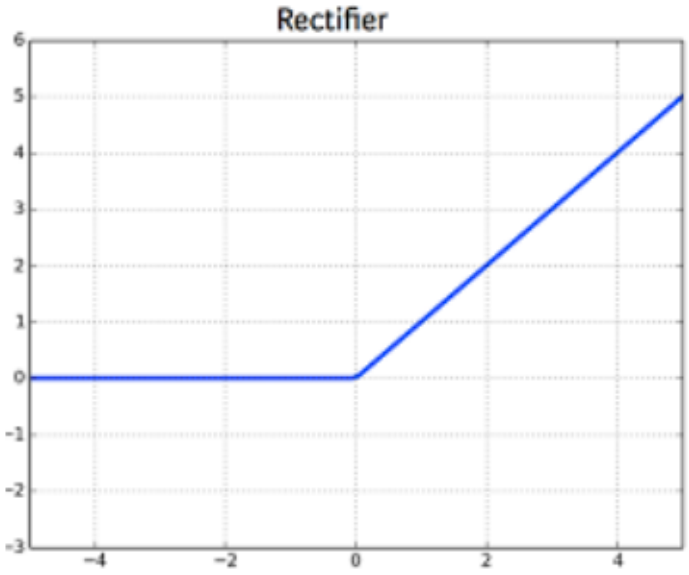
$$f(x) = \frac{\left(e^{x} - e^{-x}\right)}{\left(e^{x} + e^{-x}\right)}$$



- Very similar to the sigmoid.
- Saturate and kill the gradient..
- Slow convergence..
- It is bounded between -1 and 1.
- Centered to 0
- Good performance in recurring networks.
- It is used to decide between one option and the opposite

Activation functions

- **ReLU – Rectified Linear Unit**

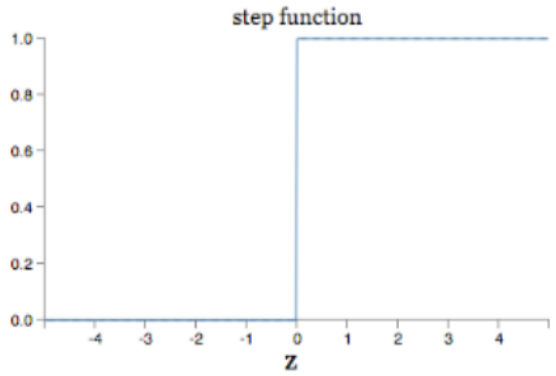


$$RELU(x) = \begin{cases} 0 \text{ if } x < 0 \\ x \text{ if } x >= 0 \end{cases}$$

- Sparse activation : only activates if they are positive.
- It is not bounded.
- Too many neurons can die.
- It behaves well with images.
- Good performance in convolutional networks..

Activation functions

- **Step function**



$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

- The input fed to the activation function is compared to a certain threshold; if the input is greater than it, then the neuron is activated, else it is deactivated, meaning that its output is not passed on to the next hidden layer.

Neural networks

- **Artificial Neural Networks**