Interactive Technologies Institute
LARSyS

arditi
agência regional para o desenvolvimento da investigação tecnologia e inovação

UNIVERSIDADE da MADEIRA

MADEIRA INTERNATIONAL WORKSHOP IN MACHINE LEARNING

IKnowD

startup madeira

Premium sponsor:

asseco
PST

Gold sponsor:

MADMAN TECH

Bronze sponsor:

CÂMARA DE LOBOS

MEO

2022

WHY PYTHON FOR DATA SCIENCE?

WHY PYTHON FOR DATA SCIENCE?

- Python is a **general-purpose programming language**.
- Known for its **simplicity and efficiency**.
- A wide **variety of open-source packages** and a **significant online community** are available.
- Effectively and efficiently **analyze and visualize large amounts of data**.
- Allows working with **multiprocessing** (multithreading and multiprocessing).

WHAT ARE THE MOST COMMON MODULES?

WHAT ARE THE MOST COMMON MODULES?

- **NumPy**—allows you to work with multidimensional arrays and has tools for numerical computation.
- **pandas**—allows you to create DataFrames for data manipulation and analysis.
- **Matplotlib**—allows you to get graphical representations of data through Python.
- **TensorFlow**—allows you to efficiently run the most popular machine-learning algorithms, focusing on deep neural networks.

All these libraries are interoperable!

AGENDA

- **Python Basics:** documentation, declaring a variable and a function; lists and dictionaries; operators and loops.
- **Introduction to NumPy:** creating an array, indexing, vectorized functions, and aggregation functions.
- **Introduction to Pandas:** creating dataframes, CSV reading/writing, data pre-processing, data grouping, and correlation;
- **Plotting** with Matplotlib and pandas;
- What is a Feed Forward Neural Network?
- *Extra*: Google Colab's specifics *(only if we have enought time)*.

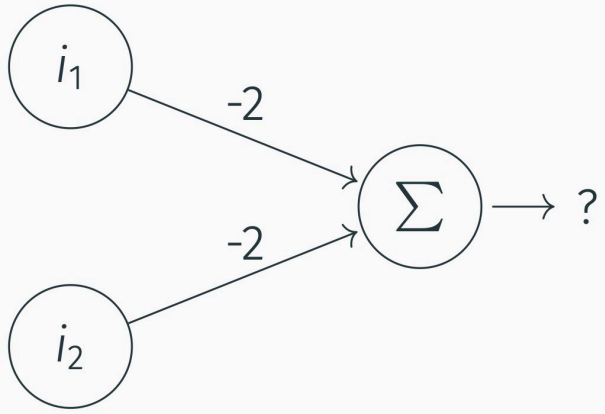
PERCEPTRON

PERCEPTRON

In the late ’50s and early ’60s, **Rosenblatt introduced the perceptron**.

In that model, neurons **also computed a weighted sum** of their inputs; however, this **weighted sum is compared with a threshold**, being the **output binary**, such as:

$$\hat{y} = \begin{cases} 0, \text{ if } \sum_j w_j i_j \leq \text{threshold}, \\ 1, \text{ otherwise.} \end{cases}$$



First case: $i_1 = i_2 = 1$

$$\sum_{j=1}^{2} w_j i_j = w_1 i_1 + w_2 i_2$$

$$= (-2) \times 1 + (-2) \times 1 = -4.$$

If we consider a threshold of -2, the outputs are:

Case	i_1	i_2	Output
1	1	1	1
2	1	0	0
3	0	1	0
4	0	0	0

Second case: $i_1 = 1$ e $i_2 = 0$, we obtain -2.

Third case: $i_1 = 0$ e $i_2 = 1$, we obtain -2.

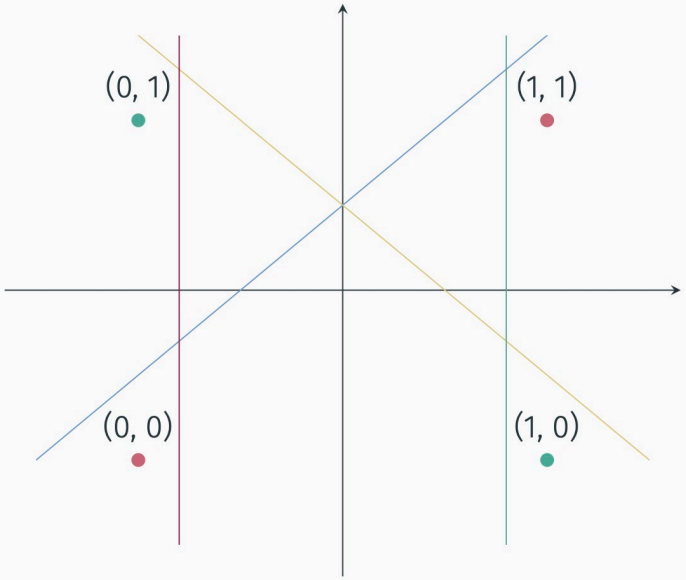
Fourth case: $i_1 = i_2 = 0$, we obtain 0.

EXAMPLE (CALCULATION OF THE OUTPUT OF A PERCEPTRON)

PERCEPTRON: LIMITATION

Perceptions can not compute the XOR operation:

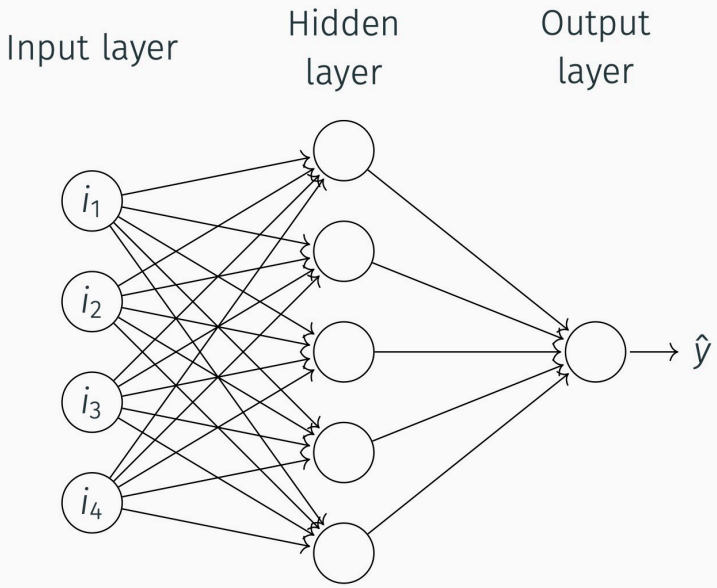
i_1	i_2	Output
1	1	0
1	0	1
0	1	1
0	0	0



For this reason, **ANNs** emerge as computational models that result from the **connection of simple neurons in a network fashion**.

ARTIFICIAL NEURAL NETWORKS

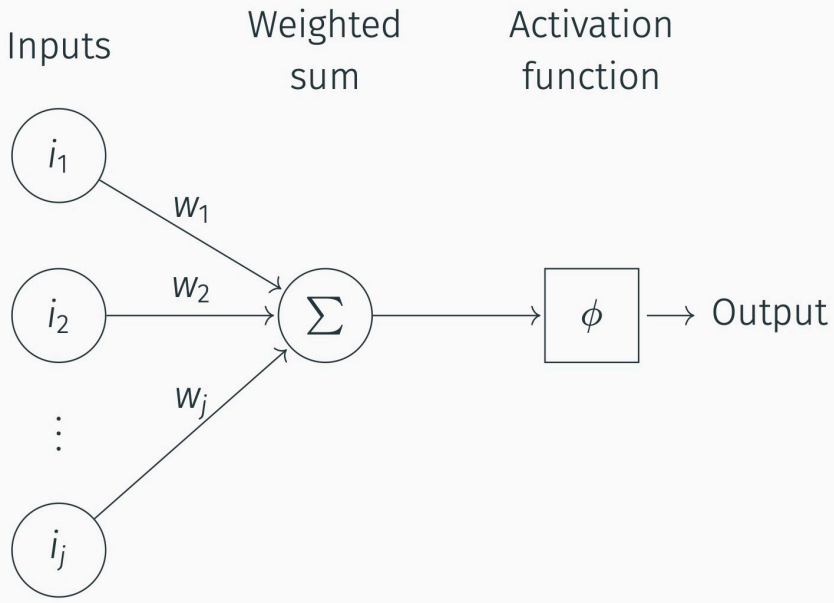
ARTIFICIAL NEURAL NETWORKS
(CONT.)



The **neurons do not have a hard threshold in an ANN**, but gradual.

That means that **small changes in the inputs or in the weights of the connections between units lead to small changes in the output**.

ARTIFICIAL NEURAL NETWORKS
(CONT.)



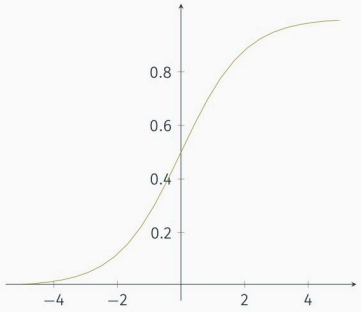
Neurons in an ANN **have two essential functions**:

- A **transfer function** ($\mathbf{T = W \cdot I}$);
- An **activation function** (ϕ).

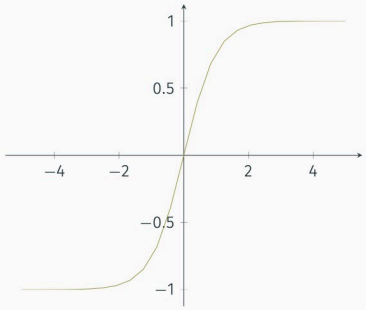
ARTIFICIAL NEURAL NETWORKS
(CONT.)

Thus, the output of a neuron is calculated as follows:

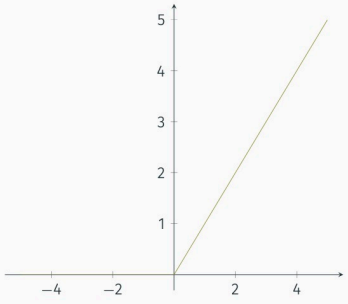
$$O_j = \phi(T_j).$$



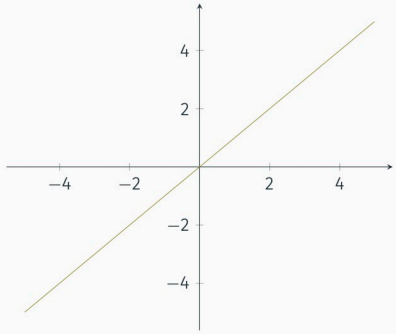
(a) Sigmoid.



(b) Hyperbolic tangent.



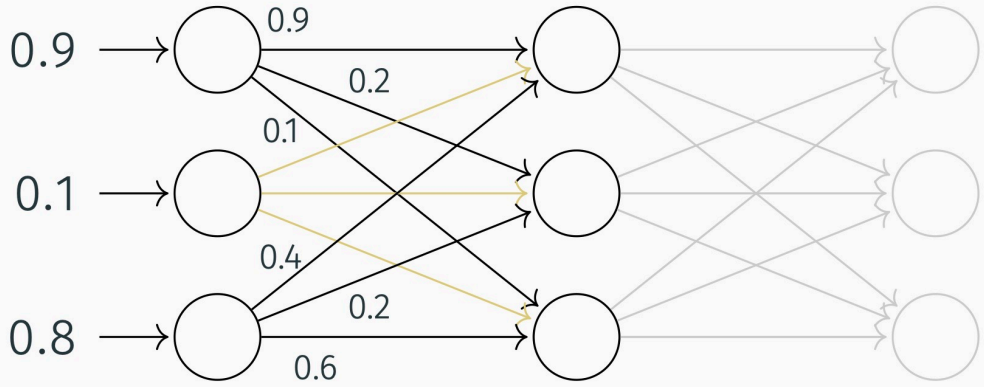
(c) Rectified linear unit (ReLU).



(d) Identity.

FORWARD PROPAGATION

Now that we know the function of a neuron in the network, let us introduce the concept of forward propagation.



$$\mathbf{I} = \begin{bmatrix} 0.9 \\ 0.1 \\ 0.8 \end{bmatrix}.$$

$$\mathbf{W}_{\text{in\_oc}} = \begin{bmatrix} 0.9 & 0.3 & 0.4 \\ 0.2 & 0.8 & 0.2 \\ 0.1 & 0.5 & 0.6 \end{bmatrix}.$$

FORWARD PROPAGATION (CONT.)

We proceed with the inner product between the matrices **W** and **I**. That is,

$$\mathbf{T}_{oc} = \mathbf{W}_{in\_oc} \cdot \mathbf{I} = \begin{bmatrix} 0.9 & 0.3 & 0.4 \\ 0.2 & 0.8 & 0.2 \\ 0.1 & 0.5 & 0.6 \end{bmatrix} \cdot \begin{bmatrix} 0.9 \\ 0.1 \\ 0.8 \end{bmatrix} = \begin{bmatrix} 1.16 \\ 0.42 \\ 0.62 \end{bmatrix},$$

which corresponds to the value of the **transfer function**.

FORWARD PROPAGATION (CONT.)

However, we cannot **forget to apply the activation function**.

For this example, **let us consider the sigmoid function**, given by:

$$\phi(z) = \frac{1}{1 + e^{-z}}.$$

Finally, the input values of the next layer are obtained:

$$\mathbf{O}_{oc} = \phi(\mathbf{T}_{oc}) = \frac{1}{1 + exp\left(-\begin{bmatrix} 1.16 \\ 0.42 \\ 0.62 \end{bmatrix}\right)} = \begin{bmatrix} 0.761 \\ 0.603 \\ 0.650 \end{bmatrix}.$$

BACKPROPAGATION

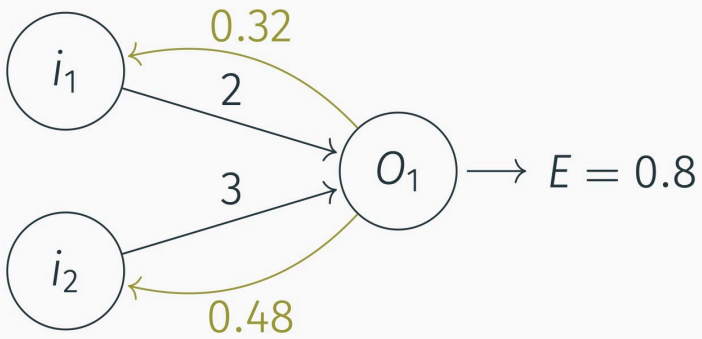
The backpropagation algorithm is made of **two fundamental parts**:

- Calculation of the **output error and backpropagation of it** until the input neurons;
- **Update the weights** using an optimisation algorithm.

This approach can be used to train an ANN for simple problems.

BACKPROPAGATION (CONT.)

How is the output error backpropagated?



The idea is to **proportionately distribute the error** according to the strength of each weight.

In this case,

$$e_{i_1} = \frac{w_1}{w_1 + w_2} \times E = \frac{2}{2+3} \times 0.8 = 0.32,$$

$$e_{i_2} = \frac{w_2}{w_1 + w_2} \times E = \frac{3}{2+3} \times 0.8 = 0.48.$$

BACKPROPAGATION (CONT.)

We need **to update the weights using an optimization algorithm**, such as the Gradient Descent.

The **gradient of the error function** for the last hidden layer and the output layer is given by:

$$\frac{\partial E}{\partial w_{jk}} = -E_k \times O_k \times (1 - O_k) \times o_j,$$

where $E_k = (y_k - \hat{y}_k)^2$, $O_k = \text{sigmoid}\,(\mathbf{w} \cdot \mathbf{i})$ and o_j is the output value of the last neuron.

The **weight w_{jk} is thus updated** as follows:

$$w'_{jk} = w_{jk} - \alpha \times \frac{\partial E}{\partial w_{jk}},$$

where α is the learning rate