# Data Analysis Lab

## Deep Learning Neural Networks

Fátima Leal

DCT DEPARTAMENTO CIÊNCIA E TECNOLOGIA

UPT UNIVERSIDADE PORTUCALENSE

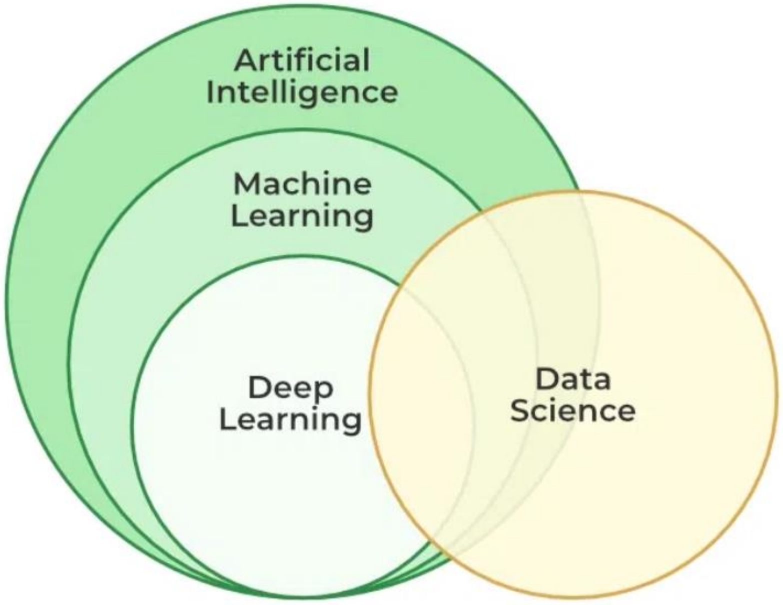
## Outline

- Deep learning
- Neural networks
- Example

## Deep Learning

- Deep Learning is a subfield of Machine Learning that involves the use of neural networks to model and solve complex problems

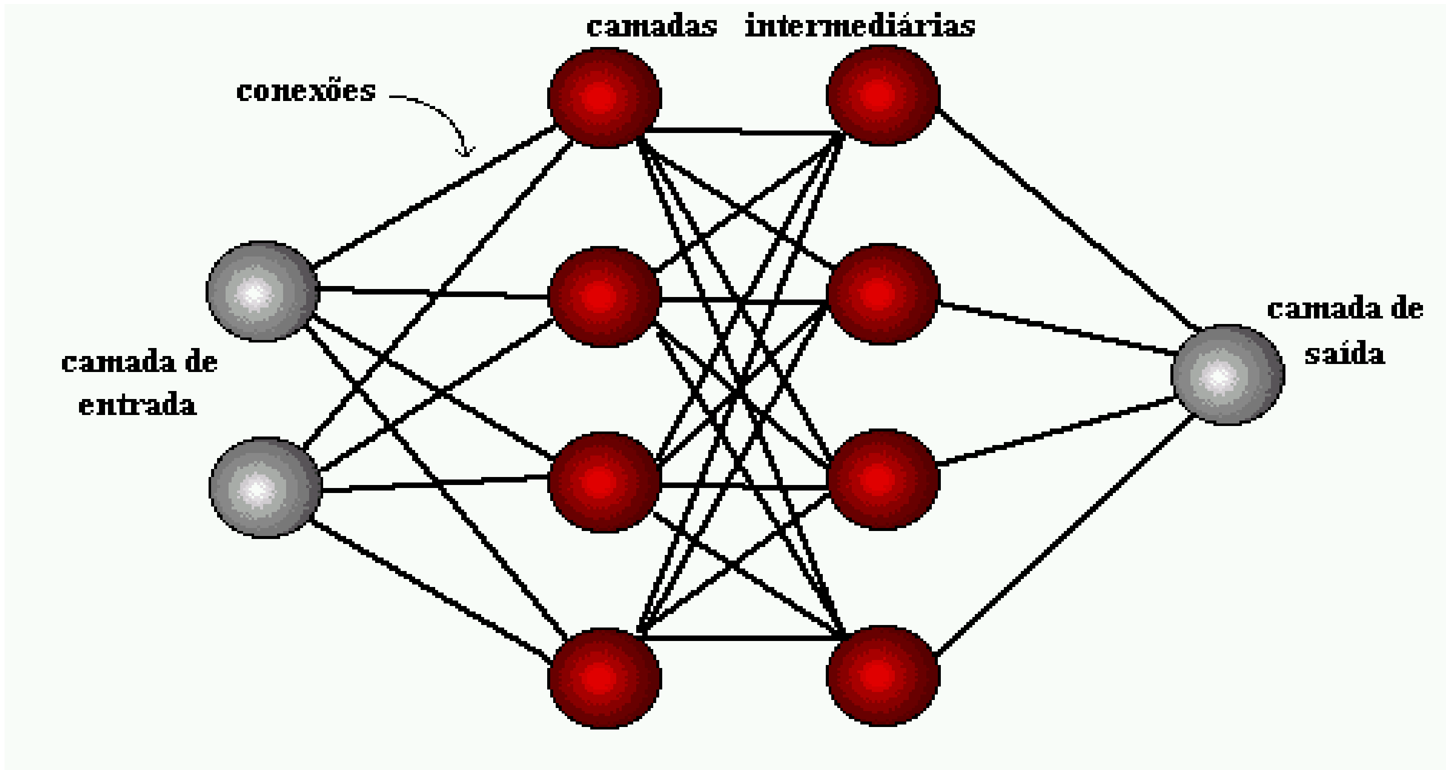
- It is a branch of machine learning which is based on artificial neural networks to model and solve complex problems.



# Artificial Neural Networks

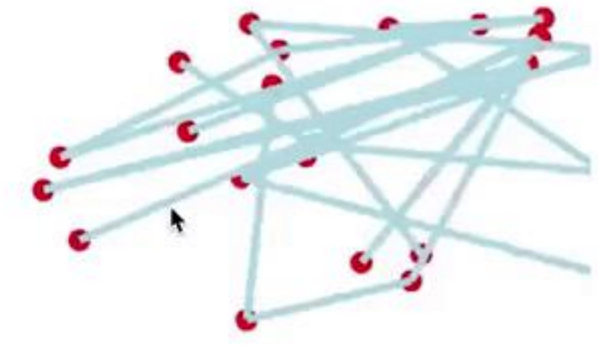
- Artificial neural networks are built on the principles of the structure and operation of human neurons.
- The input layer receives input from external sources and passes it on to the hidden layer
- Each neuron in the hidden layer gets information from the neurons in the previous layer, computes the weighted total, and then transfers it to the neurons in the next layer.
- These connections are weighted, which means that the impacts of the inputs from the preceding layer are more or less optimized by giving each input a distinct weight.
- These weights are then adjusted during the training process to enhance the performance of the model.

# Artificial Neural Networks

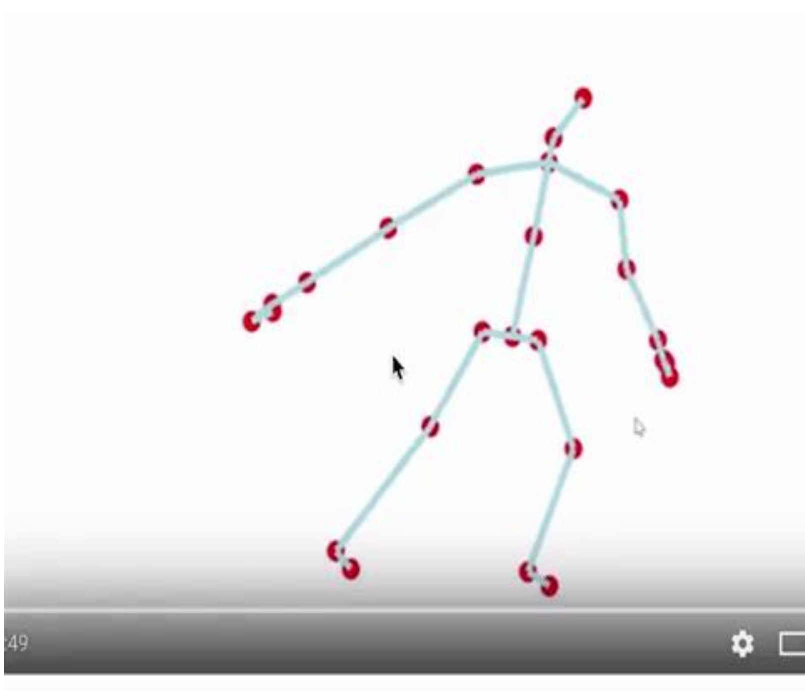


- Example:
  - An algorithm wants to learn how to dance

10 min training



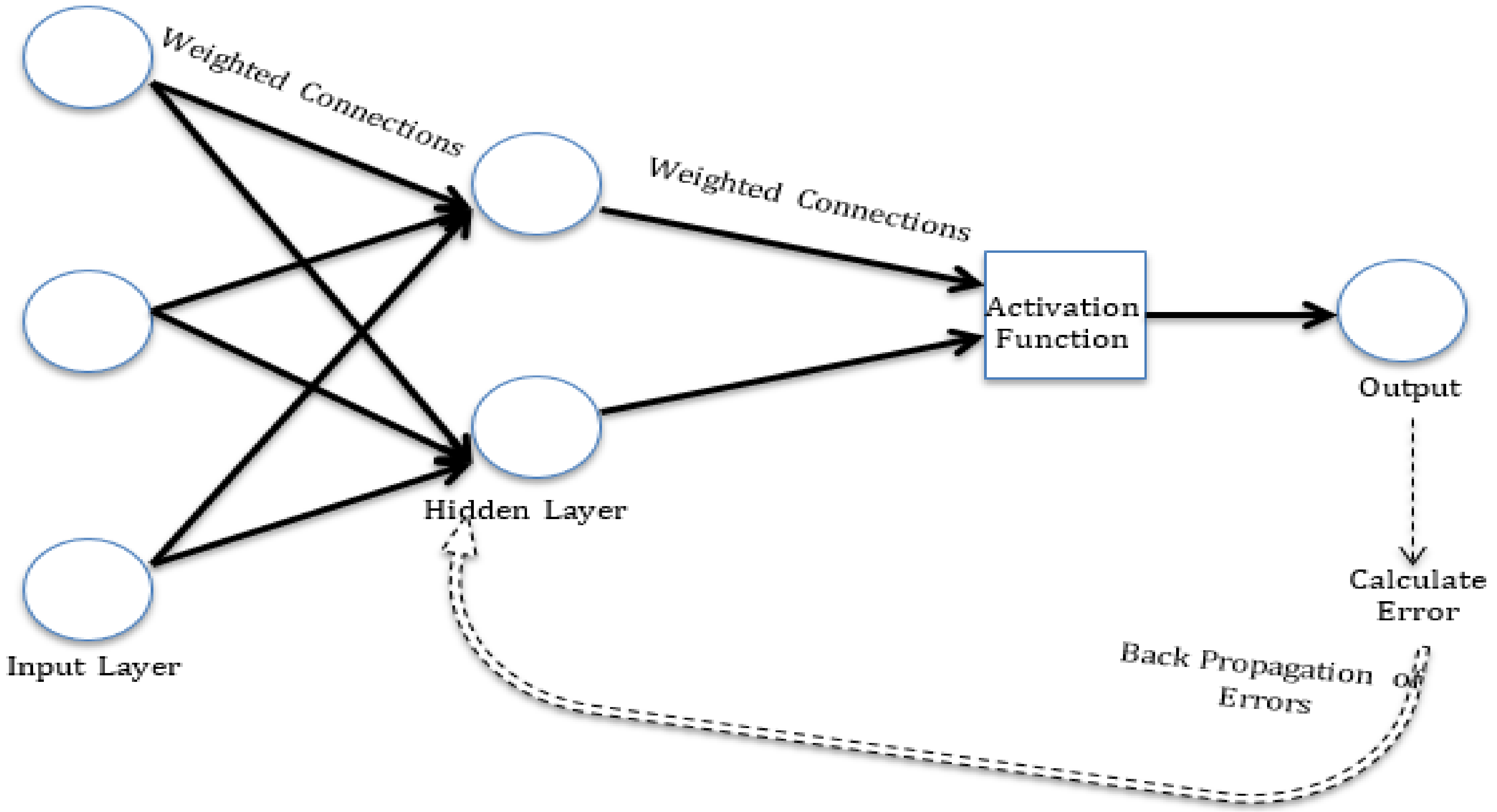
48h training



## The basic artificial neuron

- The building block of a neural network is an abstraction of a biological neuron, a quite simplistic but powerful computational unit that was proposed for the first time by F. Rosenblatt in 1957, called a perceptron.
- Contrary to Hebbian learning, which is more biologically plausible but has some strong limitations, the artificial neuron was designed with a pragmatic viewpoint and only its structure is based on a few of the elements that characterize a biological neuron.
- Recent deep learning research activities have unveiled the enormous power of this kind of architecture
- Even though there are more complex and specialized computational cells, the basic artificial neuron can be summarized as the conjunction of two blocks, which are clearly shown in the following diagram:

## The basic artificial neuron



## The basic artificial neuron

- The input of a neuron is a real-valued vector $\bar{x} \in \mathrm{R}^n$, while the output is a scalar $y \in \mathrm{R}$. The first operation is linear: $z = \bar{w}^T.\bar{x} + b$
- The vector $\bar{w} \in \mathrm{R}^n$ called a weight vector (or synaptic weight vector, because, analogously to a biological neuron, it reweights the input values), while the scalar term $b \in \mathrm{R}$ a constant called bias.
- In many cases, it's easier to consider only the weight vector. It's possible to get rid of the bias by adding an extra input feature equal to 1 and a corresponding weight: $\bar{x}^* = (x_1, x_2, ..., x_n, 1)$

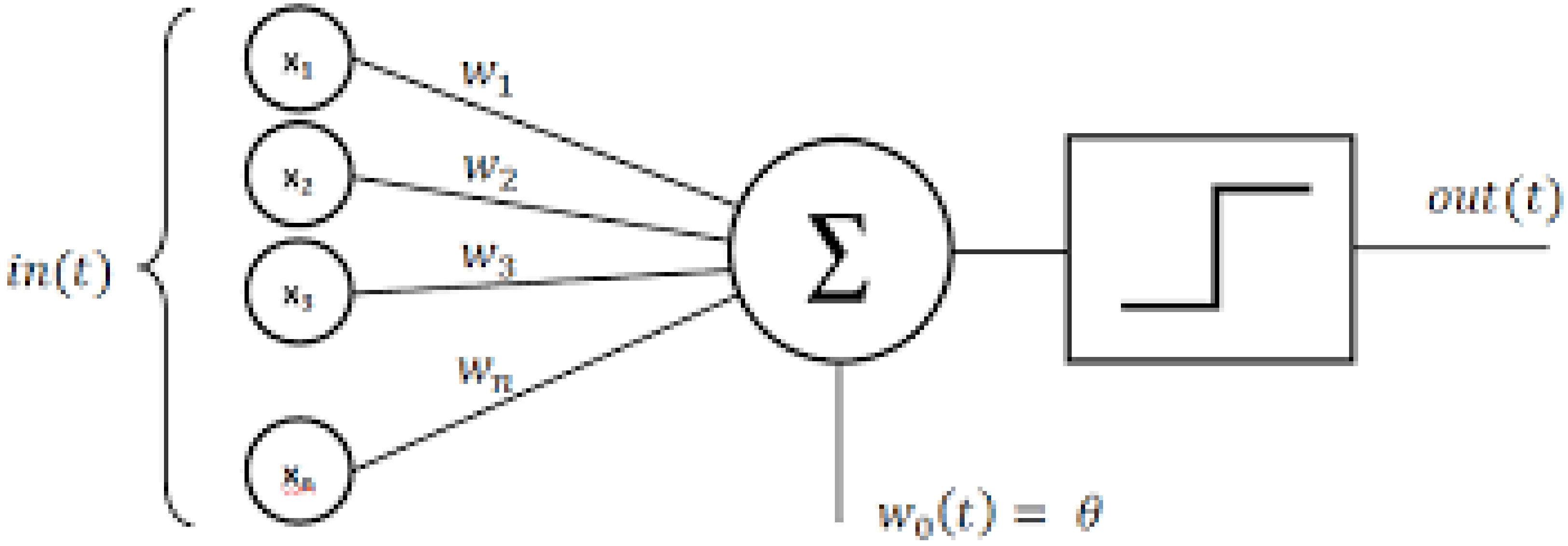
## The basic artificial neuron

- The only element that must be learned is the weight vector.
- The following block is called an activation function, and it's responsible for remapping the input into a different subset. If the function is $f_a(z) = z$, the neuron is called linear and the transformation can be omitted.
- The first experiments were based on linear neurons that are much less powerful than non-linear ones, this limitation opened the door for a new architecture instead, that had the chance to show its excellent abilities.

## Perceptron

- The perceptron was the name that Frank Rosenblatt gave to the first neural model in 1957.
- A perceptron is a neural network with a single layer of input linear neurons, followed by an output unit based on the $sign(x)$ function (alternatively, it's possible to consider a bipolar unit whose output is -1 and 1).
- The architecture of a perceptron is shown in the following diagram:

# Perceptron



UPT DCT DEPARTAMENTO CIÊNCIA E TECNOLOGIA

## Perceptron

- Even though the diagram might appear quite complex, a perceptron can be summarized by the following equation: $y_i = sign(\bar{w}^T.\bar{x}_i + b$ where $\bar{w}, \bar{x}_i \in \mathrm{R}^n$ and $y_i \in \{0, 1\}$
- All the vectors are conventionally column-vectors; therefore, the dot product $\bar{w}^T.\bar{x}_i$ transforms the input into a scalar, then the bias is added, and the binary output is obtained using the step function, which outputs 1 when $z > 0$ and 0 otherwise.
- We could object that the step function is non-linear; however, a non linearity applied to the output layer is only a filtering operation that has no effect on the actual computation.

## Perceptron

- A perceptron can be trained with an online algorithm but it's also possible to employ an offline approach that repeats for a fixed number of iterations or until the total error becomes smaller than a predefined threshold.
- The procedure is based on the squared error loss function (remember that, conventionally, the term loss is applied to single samples, while the term cost refers to the sum/average of every single loss): $L(\bar{x}_i, y_i; \bar{w}, b = \frac{1}{2}(\bar{w}^T.\bar{x}_i + b - y_i)^2$

## Perceptron

```
from sklearn import datasets
from sklearn.linear_model import Perceptron
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split

iris=datasets.load_iris()
X_train,X_test,y_train,y_test=train_test_split(iris.data,
iris.target,test_size=0.5,random_state=0)
clf=Perceptron()
clf=clf.fit(X_train,y_train)
y_pred=clf.predict(X_test)
print(clf.score(X_test,y_test))
print(confusion_matrix(y_test,y_pred))
```

# Multi layer Classifier

```
from sklearn import datasets
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier

iris=datasets.load_iris()
X_train,X_test,y_train,y_test=train_test_split(iris.data,
iris.target,test_size=0.5,random_state=0)
clf=MLPClassifier(max_iter=550)
clf=clf.fit(X_train,y_train)
y_pred=clf.predict(X_test)
print(clf.coefs_)
print(clf.n_layers_)
print(clf.n_outputs_)
print(clf.score(X_test,y_test))
print(confusion_matrix(y_test,y_pred))
```

## Multi layer Regressor

```
from sklearn.metrics import r2_score, mean_absolute_error
from sklearn.model_selection import train_test_split
import pandas as pd
from sklearn.neural_network import MLPRegressor

houses = pd.read_csv("USA_Housing.csv")

houses_target = houses['Price']
houses_data = houses.iloc[:,:5]

X_train, X_test, Y_train, Y_test = train_test_split(houses_data.values,
houses_target.values, test_size=0.3)

rf=MLPRegressor(max_iter=500)
clf=rf.fit(X_train,Y_train)

print("R2={:.2f}".format(r2_score(Y_test,clf.predict(X_test))))
print("MAE={:.2f}".format(mean_absolute_error(Y_test, clf.predict(X_test))))
```

## Other NN

- Supervised
  - Simple Neural Network
  - Convolutional Neural Network
  - Recurrent Neural Network (LSTM)
- Unsupervised
  - AutoEncoders
  - DCGAN (Deep Convolutional Generative Adversarial Networks)
- Libraries
  - TensorFlow
  - PyTorch

## Other NN

# References

- Bonaccorso, G. (2020). Mastering Machine Learning Algorithms: Expert techniques for implementing popular machine learning algorithms, fine-tuning your models, and understanding how they work. Packt Publishing Ltd.
- Bonaccorso, G. (2018). Machine Learning Algorithms: Popular algorithms for data science and machine learning. Packt Publishing Ltd.
- Lee, W. M. (2019). Python machine learning. John Wiley & Sons.
- Hauck, T. (2014). scikit-learn Cookbook. Packt Publishing Ltd.
- Montgomery, D. C., Peck, E. A., & Vining, G. G. (2021). Introduction to linear regression analysis. John Wiley & Sons.
- Hosmer Jr, D. W., Lemeshow, S., & Sturdivant, R. X. (2013). Applied logistic regression (Vol. 398). John Wiley & Sons.

UNIVERSIDADE PORTUCALENSE

Do conhecimento à prática.