



# MATHEMATICAL TOOLS FOR DATA SCIENCE

Leila GHARSALLI (leila.gharsalli@ipsa.fr)

IPSA, AERO 4

2023-2024

## GOALS OF THE COURSE

- *Instructor : Leila GHARSALLI*
- *mail to : leila.gharsalli@ipsa.fr*
- Provide a general introduction to data mining and data science.
- Introduce some important tools for solving problems in data science.
- Grading : Participation in class (practical work) as well as a final exam will be graded.
- Main background needed: basic notions in probabilities and statistics, programming skill.

# CONTENT OF THE COURSE

1. Introduction
2. Linear regression
3. Sparse regression
4. Réseaux de neurones
5. Using trees for predictive analysis
6. Principal Component Analysis
7. Clustering
8. Density estimation

# INTRODUCTION

# INTRODUCTION



By "Neural Networks", we mean artificial Neural Networks (ANN). The idea of ANN is based on **biological neural networks like the brain of living being.**

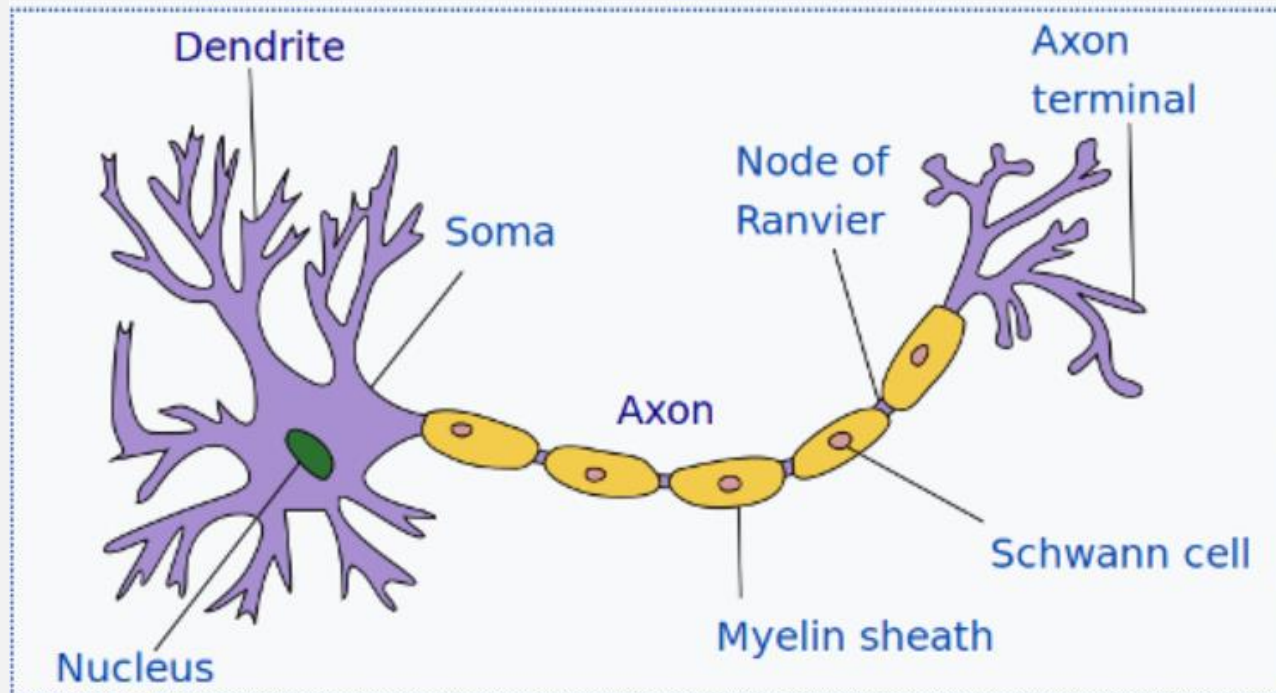
# INTRODUCTION



- Was very widely used in 80s and early 90s; popularity diminished in late 90s.
- Recent resurgence: State-of-the-art technique for many applications.
- ANN are the most popular machine learning algorithms today.

# INTRODUCTION

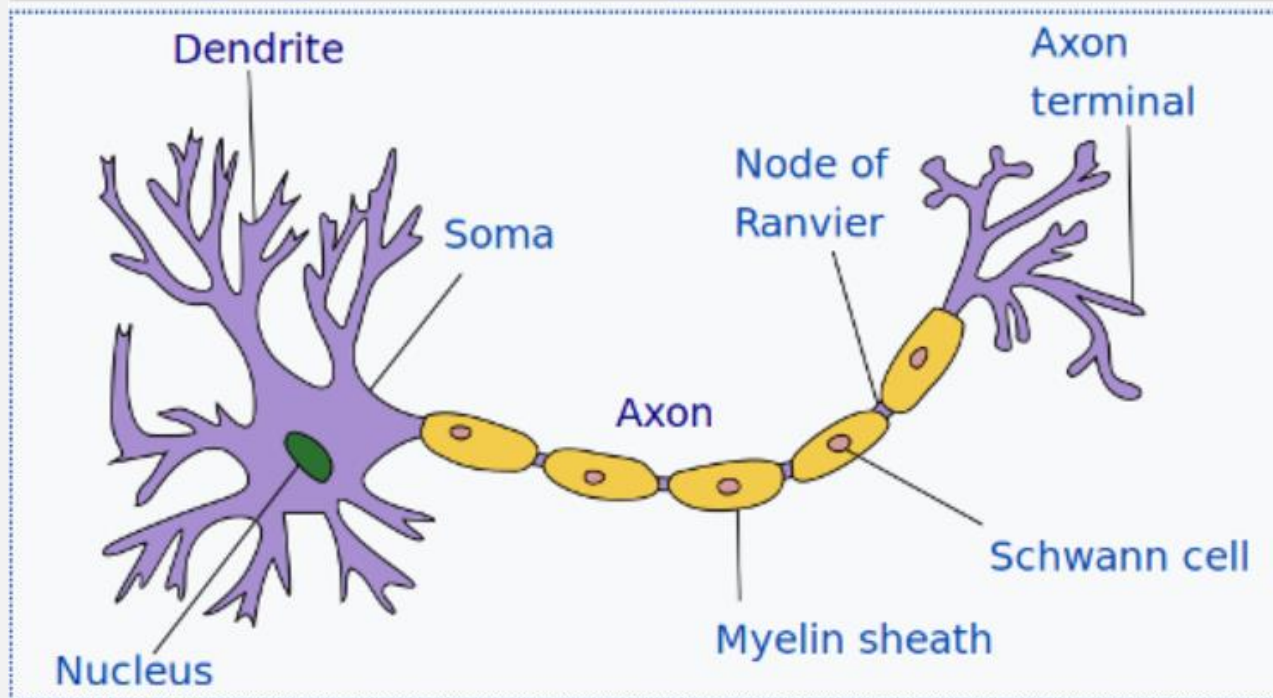
## Neuron (peripheral nervous system)



- The basic structure of a neural network - both an artificial and a living one - is **the neuron**.
- A neuron in biology consists of three major parts: **the soma (cell body)**, **the dendrites (inputs)** and **the axon (output)**.

# INTRODUCTION

**Neuron (peripheral nervous system)**

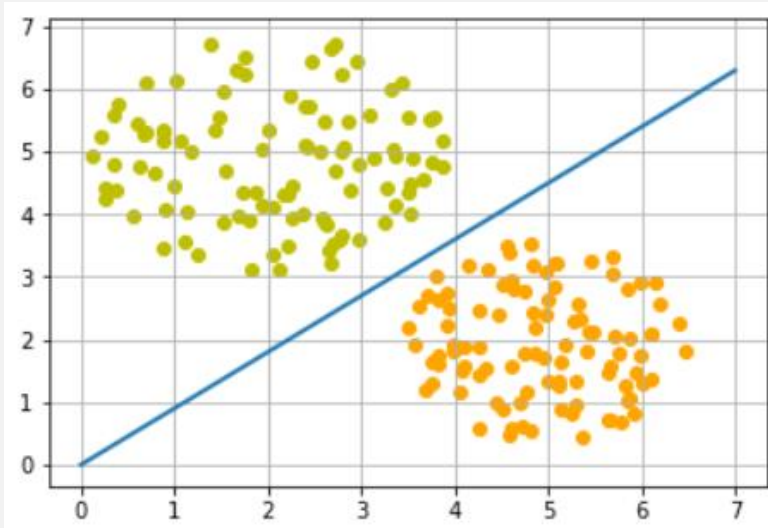


**Input** -> dendrites, **weight** -> synapses, **transfer function** -> Nucleus, **Output** -> Axon.

- The **dendrites branch** off from the soma in a tree-like way and become thinner with every branch.
- They receive signals (impulses) from other neurons at synapses.
- The axon - there is always only one - also leaves the soma and usually tend to extend for longer distances than the dendrites.
- The axon is used for sending the output of the neuron to other neurons or better to the synapses of other neurons.



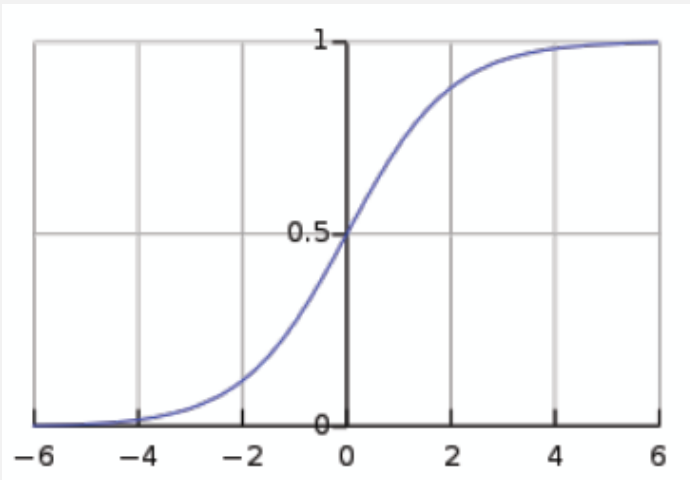
# INTRODUCTION



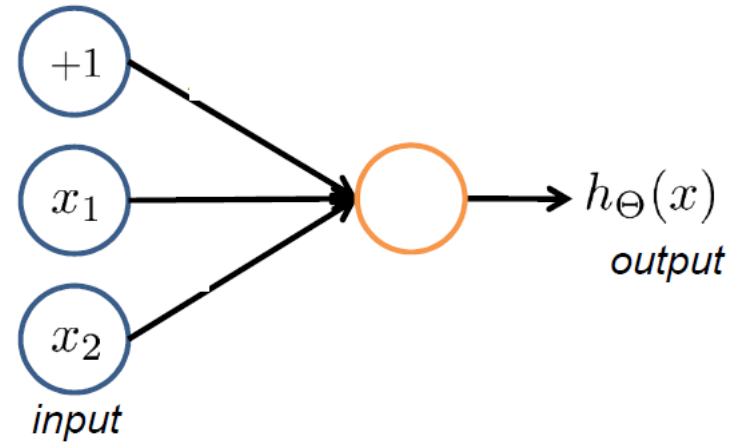
- If we are capable of separating the two classes with a straight line. One might wonder what this has to do with neural networks.
- We can define a neural network to classify this same dataset. Our neural network will only consist of one neuron with two input values.
- In general, the perceptron takes as input a vector with several dimensions (1 per neuron) and operates a separation between these data to provide an output. Thanks to this separation that he has constructed between the data, he knows, for a new example, what must be the answer.

# INTRODUCTION

$$X = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \quad \Theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}$$



bias unit



**Logistic activation function**

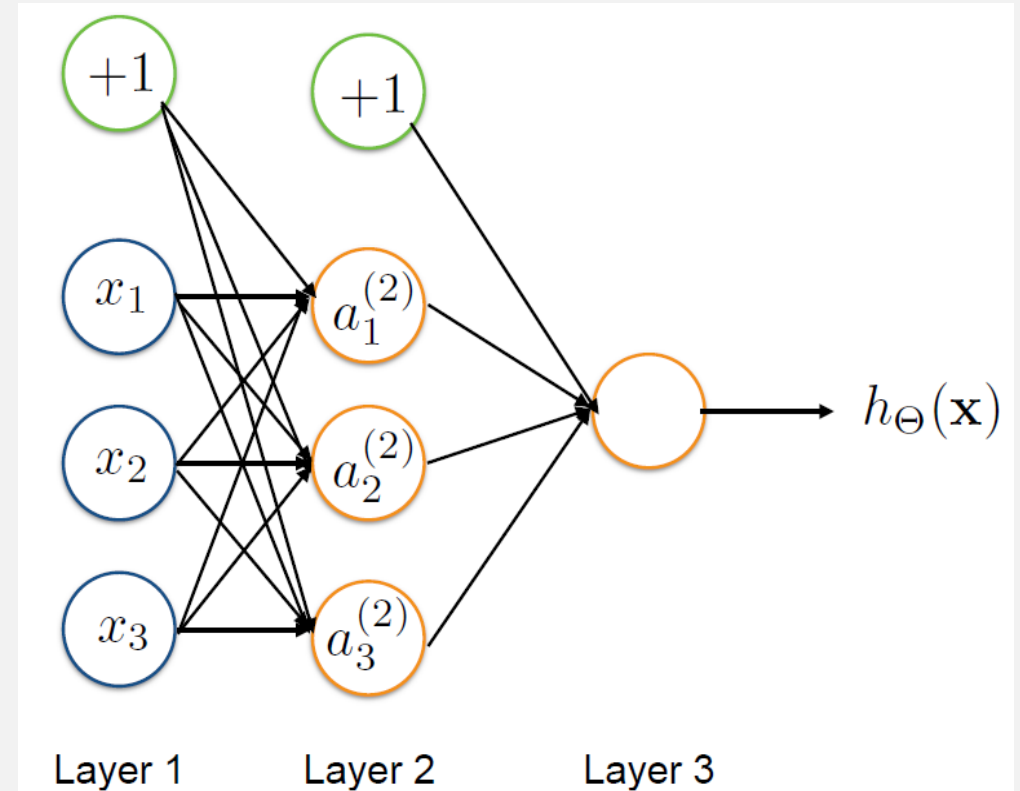
$$h_{\theta}(X) = \frac{1}{1 + e^{-\Theta X}}$$

# NEURAL NETWORKS

# NEURAL NETWORKS

$a_i^{(j)}$  = activation of unit  $i$  in layer  $j$

$\Theta^{(j)}$  = matrix of weights controlling function mapping from layer  $j$  to layer  $j + 1$



# FORWARD PROPAGATION

$$X = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

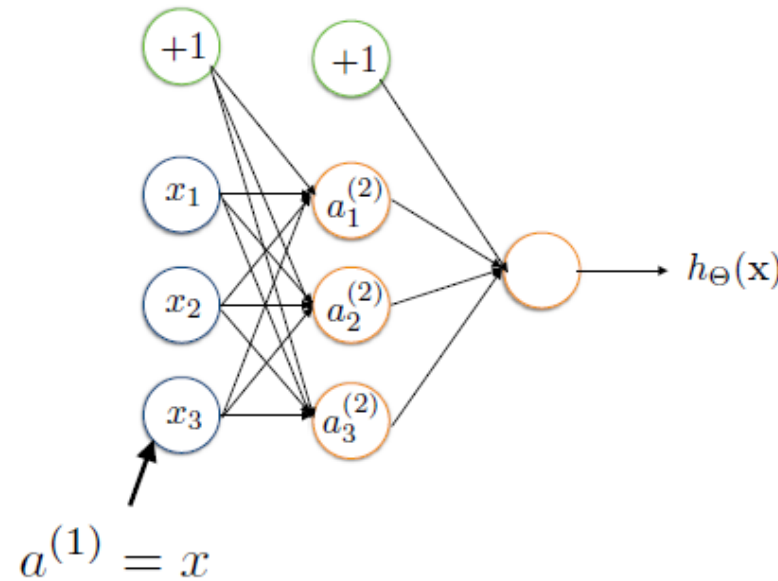
$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g(z^{(2)})$$

$$\text{Add } a_0^{(2)} = 1$$

$$z^{(3)} = \Theta^{(2)} a^{(2)}$$

$$h_{\Theta}(x) = a^{(3)} = g(z^{(3)})$$



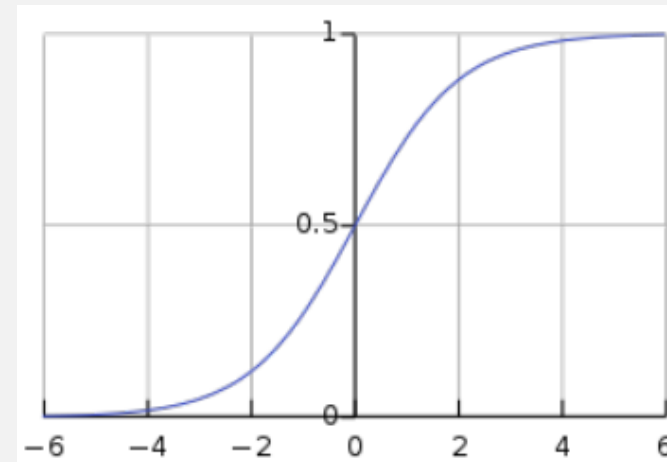
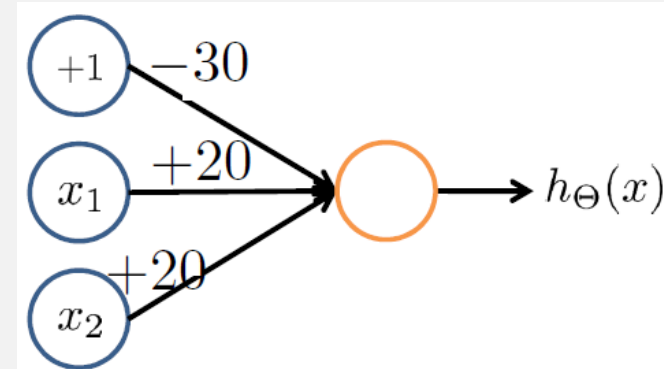
- The output  $z^{(h)}$  of the  $h^{th}$  neuron of the intermediate layer is obtained by applying the activation function of this neuron to a linear combination of the inputs.
- The output is then obtained by applying the activation function of the output neuron to the linear combination of the outputs of the intermediate layer.

# FORWARD PROPAGATION

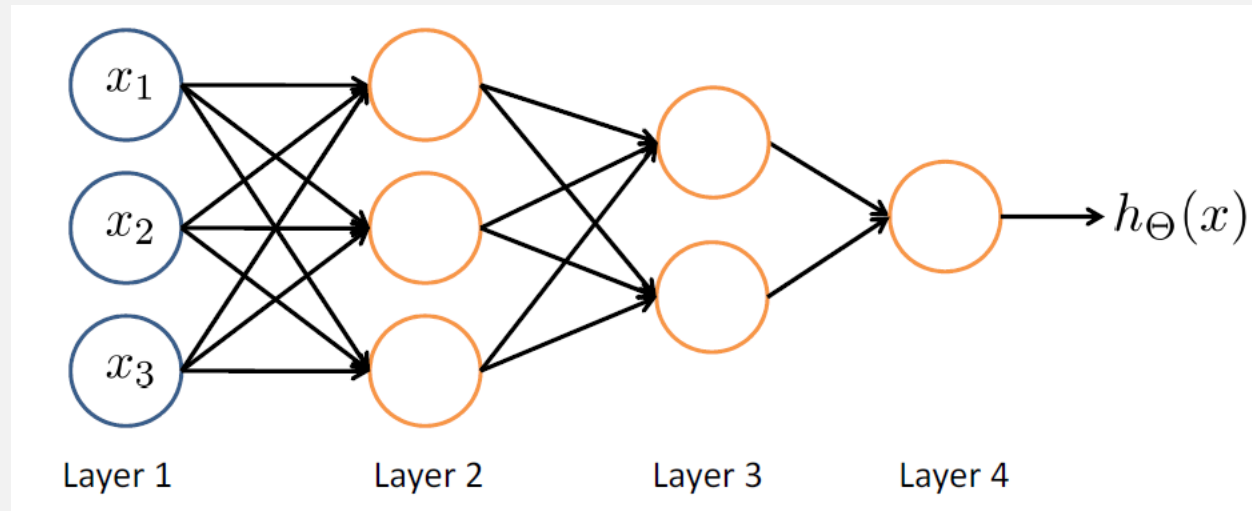
$$x_1, x_2 \in \{0, 1\}$$
$$y = x_1 \text{ AND } x_2$$

$$h_{\Theta}(x) = g(-30 + 20x_1 + 20x_2)$$

$x_1$	$x_2$	$h_{\Theta}(x)$
0	0	
0	1	
1	0	
1	1	



## OTHER NETWORK ARCHITECTURES



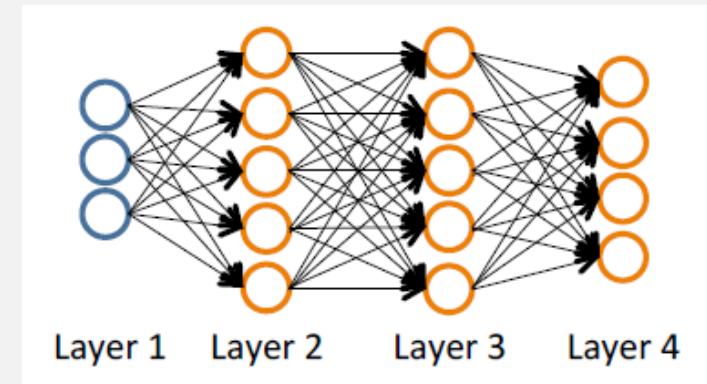
- No. of **input units**: Dimension of features ,
- No. **output units**: Number of classes ,
- Reasonable default: 1 hidden layer, or if >1 hidden layer, have same no. of hidden units in every layer (usually the more the better).

# CLASSIFICATION PROBLEM

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$$

$L$  = total number of layers in the network

$s_l$  = number of units(not counting bias unit)in layer  $l$



**Binary classification (1 output unit)**

$y = 0 \text{ or } 1$

**Multi-class classification (K output units)**

$$y \in \mathbb{R}^K; \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$



## LEARNING STEPS

- The steps below describe how Neural Networks learn, which is as follows;
1. Initialize weight values and bias
  2. **Forward** Propagate
  3. Check the error
  4. **Backpropagate** and Adjust weights and bias
  5. Repeat for all training examples

## LEARNING STEPS: BACK PROPAGATION

- We will use an iterative algorithm based on the **gradient algorithm**. To facilitate the calculations, we use an idea of **error back propagation** (or back propagation).
- **Cost function**

$$J(\Theta) = \frac{-1}{m} \left[ \sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log \left( h_{\Theta}(x^{(i)}) \right)_k + \left( 1 - y_k^{(i)} \right) \log \left( 1 - h_{\Theta}(x^{(i)}) \right)_k \right] + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{ji}^l)^2$$

**Nonconvex function.**

# BACK PROPAGATION

## Optimization

$\min_{\Theta} J(\Theta) \rightarrow \text{Need to compute } J(\Theta) \text{ and } \frac{\partial}{\partial \Theta_{ji}^l} J(\Theta)$

## Training example: forward propagation

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g(z^{(2)}) \text{ (Add } a_0^{(2)} = 1)$$

$$z^{(3)} = \Theta^{(2)} a^{(2)}$$

$$a^{(3)} = g(z^{(3)}) \text{ (Add } a_0^{(3)} = 1)$$

$$z^{(4)} = \Theta^{(3)} a^{(3)}$$

$$a^{(4)} = g(z^{(4)}) = h_{\Theta}(x)$$

# GRADIENT PROPAGATION

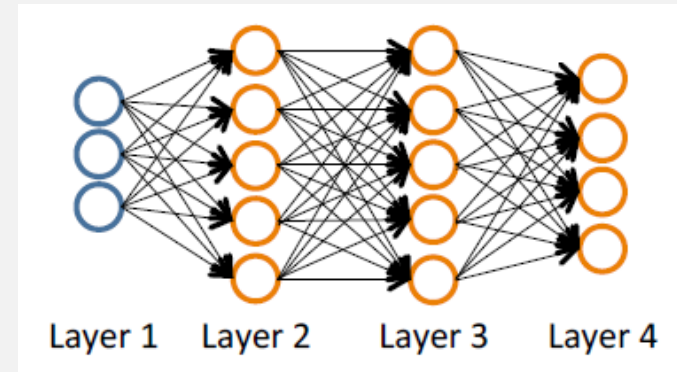
For each output unit of the last Layer L:

$$\delta_j^{(l)} = a_j^{(L)} - y_j \text{ and in vector form: } \delta^{(L)} = a^{(L)} - y$$

$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$$

$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$$

$$\frac{\partial}{\partial \Theta_{ji}^l} J(\Theta) = a_j^{(l)} \delta_i^{l+1}$$



The derivative illustrate how much we need to change the neural network's weights, in order to affect the intermediate values of the forward pass.

# TRAINING WITH BACKPROPAGATION

1. Initialize network with random weights (Symmetry breaking),
2. For all training samples in the training set :  $(x^{(i)}, y^{(i)})$ 
  - a) Present training inputs to the network and calculate output (Forward pass),  $\hat{y}(i)$
  - b) Compare network output with **correct output** (error function), for all layers (starting with output layer, back to input layer),
    - (i) **Backpropagate** gradient
    - (ii) **Adapt weights** in current layer.

# TOOLS

- Single neurone perceptron:



[sklearn.linear\\_model.Perceptron — scikit-learn 1.2.1 documentation](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html)

- Multi layer perceptron:

[https://scikit-learn.org/stable/modules/generated/sklearn.neural\\_network.MLPClassifier.html](https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html)

# SCIKIT LEARN HYPERPARAMETER OPTIMIZATION FOR MLP CLASSIFIER

- **GridSearchCV** method is responsible to fit() models for different combinations of the parameters and give the best combination based on the accuracies.

```
mlp_gs = MLPClassifier(max_iter=1)

parameter_space = {
    'hidden_layer_sizes': [(10,),(20,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}

from sklearn.model_selection import GridSearchCV

clf = GridSearchCV(mlp_gs, parameter_space, n_jobs=-1, cv=2)
clf.fit(X, y)
```

- cv=2 is for [cross validation](#), it means 2-folds Stratified K-fold cross validation.
- n\_jobs=-1 , -1 is for using all the CPU cores available.

# NEURAL NETWORKS

There are different types of neural networks. The two most popular neural networks are:

## 1. Recurrent Neural Network (RNN):

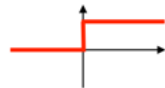
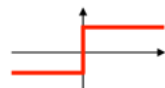


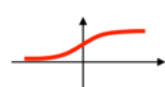
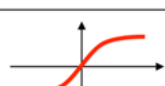


- These are specialized neural networks that use the context of the inputs when calculating the output. The output depends on the inputs and outputs calculated previously.
- RNNs are therefore suitable for applications where **historical information is important**. These networks help us predict **time series** in commercial applications and predict words in chatbot applications. They can operate with different lengths of input and output and require a large amount of data.

## 2. Convolution Neural Network (CNN):

- These networks are based on **convolution filters** (digital matrices). Filters are applied to the inputs before they are passed to the neurons.
- These neural networks are useful for **processing and predicting images**.



# ACTIVATION FUNCTIONS

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = \max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^z)$	Multi-layer Neural Networks	

# TOOLS

Class [MLPClassifier](#) implements a multi-layer perceptron (MLP) algorithm that trains using Backpropagation.

Class [MLPRegressor](#) implements a multi-layer perceptron (MLP) that trains using backpropagation with no activation function in the output layer



[https://scikit-learn.org/stable/modules/generated/sklearn.linear\\_model.Perceptron.html](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html)

[https://scikit-learn.org/stable/modules/generated/sklearn.neural\\_network.MLPClassifier.html](https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html)

[https://scikit-learn.org/stable/modules/generated/sklearn.neural\\_network.MLPRegressor.html](https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html)

## EXERCISE 1

In this exercise, you will use Multi-layer perceptron neural network to predict target variable in the [Boston Housing Price dataset](#).

1. Import the necessary *sklearn*, *pandas* and *numpy* libraries.
2. Load the Boston housing price dataset. The dataset has a sample size of 506 houses, 13 features and the regression target. You can explore your data by printing both features and targets.
3. Split the dataset into 80% for training and 20% for testing.
4. Implement the Multi-Layer Regressor algorithm then train the model on the training set. You can fix the parameter *max\_iter=400*.

## EXERCISE 1

5. Use the model for making prediction on the testing set. What do you remark?

- We can calculate the best parameters for the model using “GridSearchCV”. The input parameters for the [GridSearchCV method](#) are:
  - The MLP model
  - A parameter dictionary in which we define various hidden layers, activation units, learning rates.

6. Apply GridSearchCV to iterate through all the model parameters to be fed into the model to fit the training data. Apply a 3-fold cross validation on the training data. This means that the data is split into 3 groups and 1 set will be used as validation data, whereas the remaining 2 groups will be used as training data.

7. Once the model is fitted, print the best parameters found.

## EXERCISE 2

In this exercise, like in previous exercises, we will use the famous iris dataset that contains 150 instances of iris. The goal here is to classify each instance into one of three categories: Iris setosa, Iris virginica or Iris versicolor using the multi-layer classifier.

1. Start by importing the relevant python libraries.
2. Load the Iris dataset, then split the data between the observation matrix  $X$  and the variable  $Y$  containing the labels.
3. Split the dataset into 80% for training and 20% for testing.
4. Provide a neural network with 3 hidden layers and 10 neurons in each layer. What is the prediction score obtained by this neural network? You can use the `classification_report` and the `confusion_matrix`.
5. Could you make a comparison with other classification techniques? Conclude.