TP 2 Artificial intelligence and machine learning October 2025 #
Dr. Samir KENOUCHE

Problem 1. *A perceptron with a sigmoid activation function is a modified version of the classic perceptron, where the binary activation function (such as the Heaviside threshold) is replaced by a sigmoid (continuous and differentiable function). The typical sigmoid function is defined as:*

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{1}$$

This function converts any real value into a value between 0 and 1. Let's take an input vector $\vec{x} = [x_1, \cdots, x_n]$ *and a weight vector* $\vec{w} = [w_1, \cdots, w_n]$ *with a bias* b*. The output of the perceptron is simply given by:*

$$\sigma(z) = \frac{1}{1 + e^{-(\vec{w}\cdot\vec{x}+b)}} \tag{2}$$

Solution. An example of an application is provided as a Phyton program:

```
import numpy as np

### Dr. SAMIR KENOUCHE - PERCEPTRON
### WITH A SIGMOID ACTIVATION FUNCTUN

def activation_sigmoid(z):
    return 1 / (1 + np.exp(-z))

def the_perceptron(x_input, weight, b):
    z = np.dot(weight, x_input) + b
    return activation_sigmoid(z)

### EXAMPLE ###
x_input = np.array([0.1, 1])   # INPUT
weight = np.array([0.1, -0.2]) # WEIGHT
b = 0.3                        # BIAIS

### RESULT ###
yhat = the_perceptron(x_input, weight, b)
print("PERCEPTRON'S OUTPUT:", yhat)
```

A more complicated example than the previous one is presented below:

```
import numpy as np

""" Dr. SAMIR KENOUCHE - EXAMPLE OF A NEURAL NETWORK
WITH THE SIGMOID ACTIVATION FUNCTION """
```

```python
### ACTIVATION FONCTION ###
def activation_sigmoid(x):
    return 1 / (1 + np.exp(-x))
### SIGMOID DERIVATIVE ###
def deriv_sigmoid(x):
    return x * (1 - x)

### INPUTS ###
X = np.array([
    [1, 0],
    [1, 1],
    [0, 0],
    [1, 0]
])

### TARGET ###
y = np.array([
    [1],
    [1],
    [0],
    [1]
])

### WEIGHT INITIALISATION ###
weights = np.array([[0.0], [0.1]])

### LEARNING ###
Nepoch = 1800
for epoch in range(Nepoch):
    yhat= activation_sigmoid(np.dot(X, weights)) # OUTPUT
    delta = y - yhat                             # ERROR
    myfitting = delta * deriv_sigmoid(yhat)
    weights = weights + np.dot(X.T, myfitting)

### RESULT ###
print("OUTPUT AFTER LEARNING:")
print(yhat)
```

□

This kind of perceptron is commonly used in multilayer neural networks (MLP), since the sigmoid is differentiable, which is crucial for backpropagation learning. More explanation will be provided during the practical session.

1 Going further ...

The sigmoid function is an advanced concept in data science and machine learning, powering algorithms such as logistic regression and neural networks. It converts complex numerical data into probabilities that are easier to interpret. More specifically, this function transforms a real-valued input into a probability-type output between 0 and 1. The sigmoid is therefore essential for tasks such as predicting binary outcomes (yes or no) and making informed predictions in machine learning classification models. The sigmoid function is widely used in data science in two main ways:

- Binary classification: The sigmoid function transforms the output of a model into a probability score
- Activation function: In neural networks, the sigmoid function adds nonlinearity, allowing the model to learn complex patterns in the data.

The sigmoid function plays a central role in neural networks as an activation function. The main role of the sigmoid function as an activation function is to take the weighted sum of the inputs from the previous layer and transform it into an output value between 0 and 1. This transformation is useful for introducing nonlinearity into the model, allowing the hidden layers of a deep neural network to learn complex relationships and solve problems that cannot be separated by straight lines, such as image recognition or natural language processing. However, the sigmoid function has limitations, the main one being the problem of gradient vanishing. For very large or very small input values, the output of the function saturates near 1 or 0, and its gradient becomes almost zero. This slows down the learning process in dense neural networks, as the weights are updated too slowly during training. The sigmoid function is based on the exponential operation, which is computationally expensive compared to simpler activation functions such as ReLU (Rectified Linear Unit).