LAB : Perceptron Learning

In machine learning, **the perceptron** is an algorithm for supervised learning of **binary** classifiers. A binary classifier is a function which can decide whether an input, represented by a vector of numbers, belongs to some specific class. It is a type of **linear** classifier, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector.

The task is to implement a simple **perceptron** to compute logical operations like **AND**, **OR**, and **XOR**.

- Input: x_1 and x_2
- Bias: $b = -1$ for AND; $b = 0$ for OR
- Weights: $w = [1, 1]$

with the following **activation** function:

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

#We can define this threshold function in Python as:

```
def activation(z):
    if z > 0:
        return 1
    return 0
```

#For AND we could implement a perceptron as:

```
w = np.array([1, 1])
b = -1
x = np.array([0, 0])
print("0 AND 0:", activation(w.dot(x) + b))
x = np.array([1, 0])
print("1 AND 0:", activation(w.dot(x) + b))
x = np.array([0, 1])
print("0 AND 1:", activation(w.dot(x) + b))
x = np.array([1, 1])
print("1 AND 1:", activation(w.dot(x) + b))
```

#For OR we could implement a perceptron as:

```
w = np.array([1, 1])
b = 0
x = np.array([0, 0])
print("0 OR 0:", activation(w.dot(x) + b))
x = np.array([1, 0])
print("1 OR 0:", activation(w.dot(x) + b))
x = np.array([0, 1])
print("0 OR 1:", activation(w.dot(x) + b))
x = np.array([1, 1])
print("1 OR 1:", activation(w.dot(x) + b))
```

#With a different activation function and a different weight vector we can also solve the XOR problem using Perceptron learning

If we assume the **weights** to be set to **0.5** and the **bias** to **0**.

- Input: x_1 and x_2
- Bias: $b = 0$ for XOR
- Weights: $w = [0.5, 0.5]$

And the **activation function** can be defined as:

$$y = \begin{cases} 0 & \text{if } w \cdot x + b \neq 0.5 \\ 1 & \text{if } w \cdot x + b = 0.5 \end{cases}$$

#We can define this threshold function in Python as:

```
def bactivation(z):
    if z == 0.5:
        return 1
    else: return 0
```

Then, the XOR can be implemented a perceptron as:

```
w = np.array([0.5, 0.5])
b = 0
x = np.array([0, 0])
print("0 OR 0:", bactivation(w.dot(x) + b))
x = np.array([1, 0])
print("1 OR 0:", bactivation(w.dot(x) + b))
x = np.array([0, 1])
print("0 OR 1:", bactivation(w.dot(x) + b))
x = np.array([1, 1])
print("1 OR 1:", bactivation(w.dot(x) + b))
```