Rapport du TP1_chapitre1

Réalisé par le groupe 2 :

- Asma Charafi
- Hafssa Boujida
- Nossaiba El-Asri
- Nouhaila Ait jilali

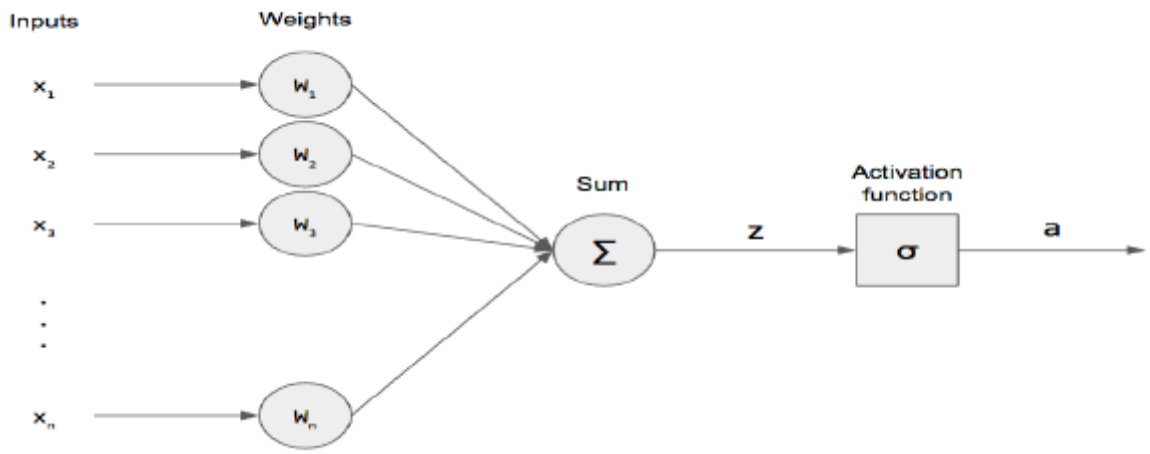
Objectif :

Dans ce TP, on s'intéresse à construire le **Perceptron** sous Python.

Introduction :

Deep learning est un ensemble de méthodes qui a pour objectif de faire apprendre à la machine d'imiter l'intelligence humaine. C'est pour cela que ce domaine se base sur un ensemble de réseaux de neurones qui est constitué d'une unité de base qu'on appelle le perceptron. Alors, comment ce **perceptron** permet-il à la machine d'apprendre ?

Schéma d'un perceptron :



Les composantes du modèle :

- Les entrées $x_1,x_2,\ldots,x_n$
- Les poids $W_1,W_2,\ldots,W_n$
- La pré-activation: la somme des produits $W_i x_i$
- La fonction d'activation : retourne 1 si l'entrée est positive, 0 sinon.

Algorithme d'apprentissage :

Algorithme :	Explication :
import numpy as np	L'importation de la bibliothèque numpy qui permet d'effectuer des calculs numériques.

class Perceptron(object):
"""Implements a perceptron network"""
def __init__(self, input_size):
self.W = np.zeros(input_size+1)
self.epochs = epochs
self.lr = lr
	Création de la classe Perceptron, qu'on peut utiliser ultérieurement, input_size est la taille de l'entrée à laquelle on ajoute 1 pour tenir compte du biais, et on construit le vecteur des poids de la même taille que les entrées tenant toujours compte du biais, epochs est le nombre de fois qu'on tourne l'algorithme.

def activation_fn(self, x): return 1 if x >= 0 else 0	C'est la fonction marche pied qui retourne 1 si la somme est supérieure à 0, et un 0 sinon.
def predict(self, x): z = self.W.T.dot(x) a = self.activation_fn(z) return a	C'est la fonction qui calcule la pré-activation et lui appliqué la fonction d'activation.

def fit(self, X, d):
for _ in range(self.epochs):
for i in range(d.shape[0]):
x = np.insert(X[i], 0, 1)
y = self.predict(x)
e = d[i] - y
self.W = self.W + self.lr * e * x
	La fonction nous donne une prédiction sur un nouveau set de données, elle tourner l'algorithme en essayant de minimiser l'erreur à travers la mise à jour des poids.

Code python :

```python
import numpy as np
class Perceptron(object):
 """Implements a perceptron network"""
 def __init__(self, input_size, lr=1, epochs=100):
  self.W = np.zeros(input_size+1)
  # add one for bias
  self.epochs = epochs
  self.lr = lr
 def activation_fn(self, x):
  #return (x >= 0).astype(np.float32)
  return 1 if x >= 0 else 0
 def predict(self, x):
  z = self.W.T.dot(x)
  a = self.activation_fn(z)
  return a
 def fit(self, X, d):
  for _ in range(self.epochs):
   for i in range(d.shape[0]):
    x = np.insert(X[i], 0, 1)
    y = self.predict(x)
    e = d[i] - y
    self.W = self.W + self.lr * e * x


X = np.array([
[0, 0],
[0, 1],
[1, 0],
[1, 1]
])
d = np.array([0, 0, 0, 1])
perceptron = Perceptron(input_size=2)
perceptron.fit(X, d)
print(perceptron.W)
```

Résultat de l'exécution :

```
[-3.  2.  1.]
```

Conclusion :

Dans ce TP on a vu le fonctionnement d'un perceptron monocouche, mais vu les applications qu'on souhaite réaliser à l'aide du deep learning, cette unité de base ne suffit pas toute seule, d'où la nécessité d'évaluer vers les réseaux de neurones.