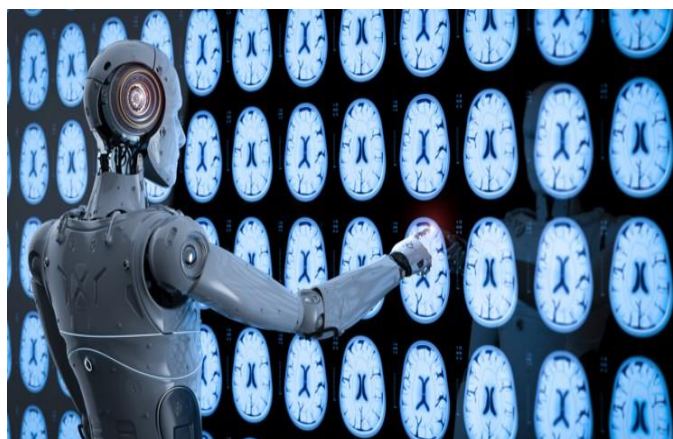




ÉCOLE CENTRALE CASABLANCA

LIVRABLE_4

UNE PREMIÈRE COMPRÉHENSION DE VGG16



Réalisé par :

Amine NAIT CHARIF
Doha FEDDOUL
Florian BOUBOUSSI DONGMO
Youssef CHOUHAIDI

Tuteur Entreprise :

M. Fayçal NOUSHI

Tuteur école :

M. Mohamed-Hassan KHALILI

SOMMAIRE

<u>INTRODUCTION</u>	3
 <u>PREMIÈRE PARTIE : LES RÉSEAUX DE NEURONES</u>	4
<u>I-LE NEURONE ARTIFICIEL</u>	4
<u>1-Le neurone biologique</u>	4
<u>2-Le neurone formel</u>	5
<u>3-Équation d'un neurone formel</u>	6
<u>4-Apprentissage d'un neurone</u>	6
<u>II-LES COUCHES DE NEURONES</u>	6
 <u>DEUXIÈME PARTIE : LA MÉTHODE DE DESCENTE DE GRADIENT</u>	10
<u>I-LE PROBLÈME DE LA DROITE OPTIMALE :</u>	10
1-Fonction objectif	11
2-Minimisation de la fonction objective.....	11
3-Programmation Python.....	12
4-Application.....	12

INTRODUCTION

Le Machine Learning, compte tenu des nombreuses possibilités qu'il offre et pourrait offrir, suscite beaucoup d'intérêts de la part de la communauté scientifique. En effet, les recherches dans ce domaine ne cessent de croître, notamment dans les techniques de mise en place de ce dernier. L'une des méthodes les plus efficaces aujourd'hui est le réseau de neurones. Ce système, se basant sur une reproduction du mode de fonctionnement des neurones biologiques, est beaucoup employé dans les logiciels de programmation à l'instar de Python. À travers la bibliothèque libre TensorFlow, on peut utiliser sous Python le réseau de neurones VGG16. VGG16 est un réseau de neurones **convolutif** performant.

Ce rapport, visant à mieux comprendre pour un début l'architecture et le fonctionnement de VGG16, présentera le concept de neurone artificiel, de couche de neurone et du procédé d'apprentissage dit apprentissage par descente du gradient.

PREMIÈRE PARTIE : LES RÉSEAUX DE NEURONES

I- LE NEURONE ARTIFICIEL

Le neurone artificiel est conçu sur la base ou l'architecture du neurone biologique.

1- Le neurone biologique :

Le neurone biologique est la plus petite entité constituant le système nerveux. Elle est à la base de la conscience observée chez les êtres vivants et intervient dans le processus d'apprentissage et de mémorisation.

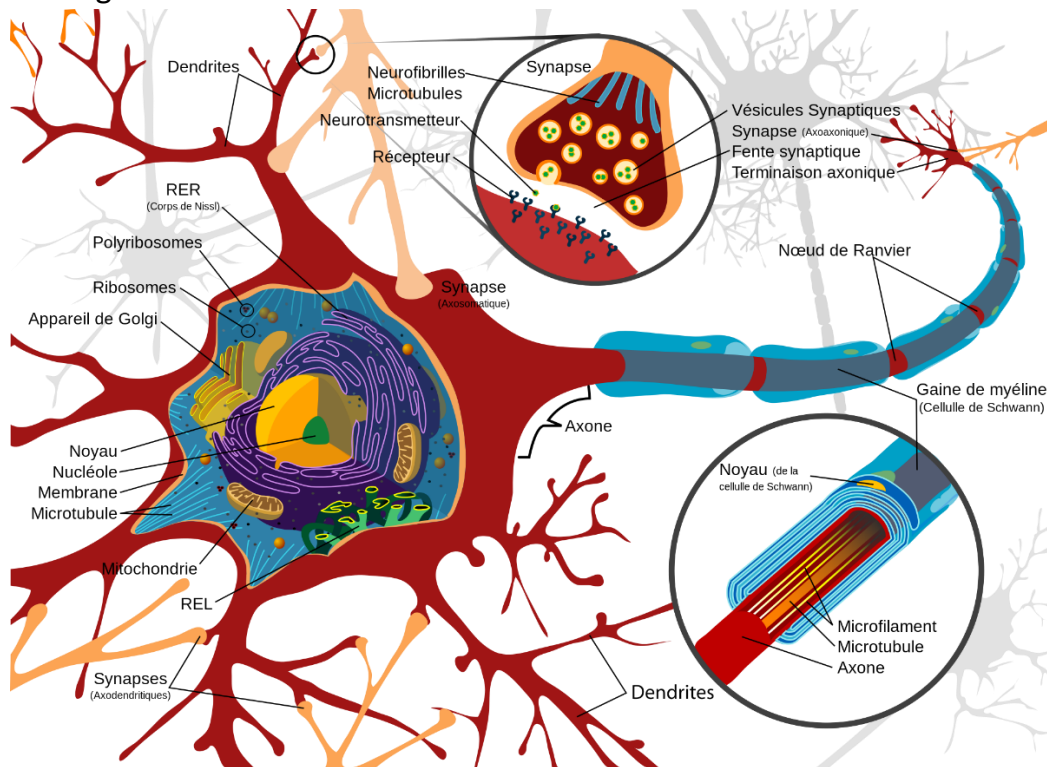


Figure 1- Un neurone biologique [domaine public - source :

https://commons.wikimedia.org/wiki/File:Complete_neuron_cell_diagram_fr.svg]

Bien qu'étant une structure assez complexe, une représentation simple du neurone présente ce dernier comme étant constitué de 3 grandes parties essentielles :

- Un corps cellulaire : appelé le péricaryon.
- Un ensemble de dendrites (environ 7000).
- Un axone.

Les dendrites sont les éléments d'entrée du neurone. Ce sont elles qui captent l'influx nerveux (l'information) généré par des stimulus (réactions produisant la création d'un signal nerveux dans le cerveau). Ces stimulus ont lieu au niveau de leurs extrémités et sont envoyés vers le péricaryon : l'influx nerveux est donc ici centripète (de la périphérie vers le centre).

L'influx nerveux étant arrivé au niveau du péricaryon, s'il dépasse un certain seuil, alors un nouveau influx nerveux est renvoyé en direction cette fois-ci de l'axone : cet influx nerveux est donc centrifuge. La fréquence du signal nerveux en direction de l'axone est d'autant plus grande que le signal initial l'était.

L'extrémité de l'axone peut être en contact avec une dendrite d'un autre neurone et dans ce cas le signal refait le même parcours et il y a transmission du signal nerveux au sein du système nerveux.

2- Le neurone formel

Le neurone artificiel est encore appelé neurone formel ou perceptron. Il existe différentes façons de le représenter selon les similitudes que l'on souhaite préserver entre celui-ci et le neurone biologique.

Une version assez simplifiée le présente comme suit :

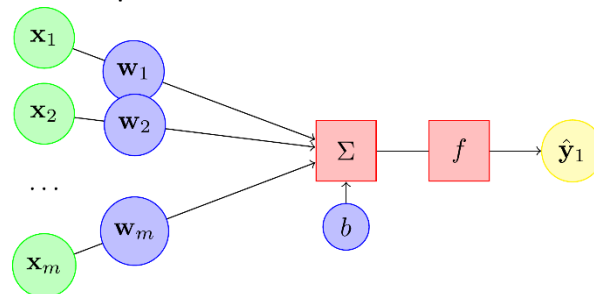


Figure 2 : Un neurone biologique [domaine public - source :

https://commons.wikimedia.org/wiki/File:Complete_neuron_cell_diagram_fr.svg]

Il est constitué de trois parties (en accord avec le modèle du neurone biologique présenté plus haut) :

- Des entrées, notées X_i , sous forme de vecteurs modélisant les dendrites.
- Une sortie, notée \hat{Y}_1 , représentant l'axone.
- Des paramètres W_i, b et des fonctions Σ et f qui représentent le corps du neurone.

Les entrées sont fournies au neurone en début d'exécution et sont donc variables. Les paramètres quant à eux sont fixés par le modèle et la sortie est bien sûr calculée en fonction des entrées et des paramètres.

3- Équation d'un neurone formel

Pour calculer la sortie Y d'un réseau de neurone, on procède comme suit :

- Chaque entrée X_i est multipliée par le paramètre W_i .
- On effectue par la suite une sommation de toutes les entrées (multipliées au préalable par leurs paramètres respectifs).
- La somme obtenue est ajoutée au paramètre b .
- Le tout est passé au sein d'une fonction f non nécessairement linéaire (dans des cas véritablement pratique la fonction f n'est pas linéaire).

Cette méthode de calcul de la sortie d'un neurone n'est tout de même pas entièrement conforme avec le principe de fonctionnement du neurone biologique : en effet, ici il y'a modulation d'amplitude et non de fréquence. La sortie sera d'autant plus grande (en valeur absolue) que les entrées le seront ; hors, pour le cas du neurone biologique, la fréquence du signal de sortie est d'autant plus grande que les signaux d'entrées sont grands (ici grand c.-à-d. dépasse largement le seuil d'excitation).

4- Apprentissage d'un neurone

L'apprentissage d'un réseau de neurone consiste à déterminer, pour un ensemble de valeurs d'entrées, les bons paramètres W_i et b .

L'apprentissage peut se faire comme suit :

- Détermination d'une fonction perte en fonction des données (paramètres et entrées).
- Optimisation de la fonction perte, c.-à-d. détermination des paramètres W_i et b minimisant cette fonction.

Cette méthode d'apprentissage se nomme apprentissage par descente de gradient.

II- LES COUCHES DE NEURONES

Un neurone à lui seul ne saurait apprendre à résoudre des problèmes complexes (comme le simple problème de séparation des points de différentes couleurs par une courbe sur un graphe). Pour pallier à ces limitations, les neurones, comme c'est le cas au sein du cerveau humain, sont mis ensembles : on parle alors de réseau de neurones.

Plusieurs typologies (assemblage de neurones) peuvent être mises en place :

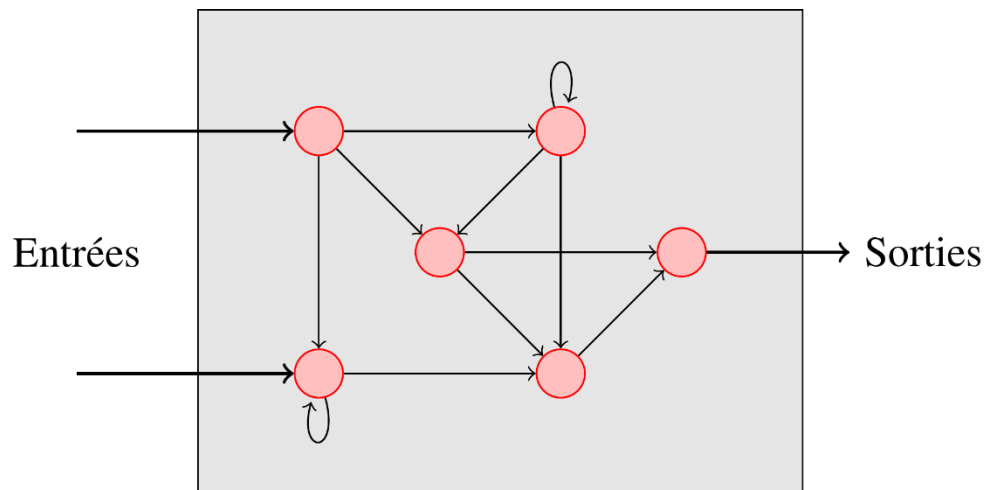


Figure 3 : Une typologie quelconque de neurones [Source : <https://openclassrooms.com/fr/courses/5801891-initiez-vous-au-deep-learning/5814616-explorez-les-reseaux-de-neurones-en-couches#/id/r-5989691>]

Ces typologies aléatoires sont très difficiles à entraîner.

La typologie de neurone par excellence est la typologie en couche :

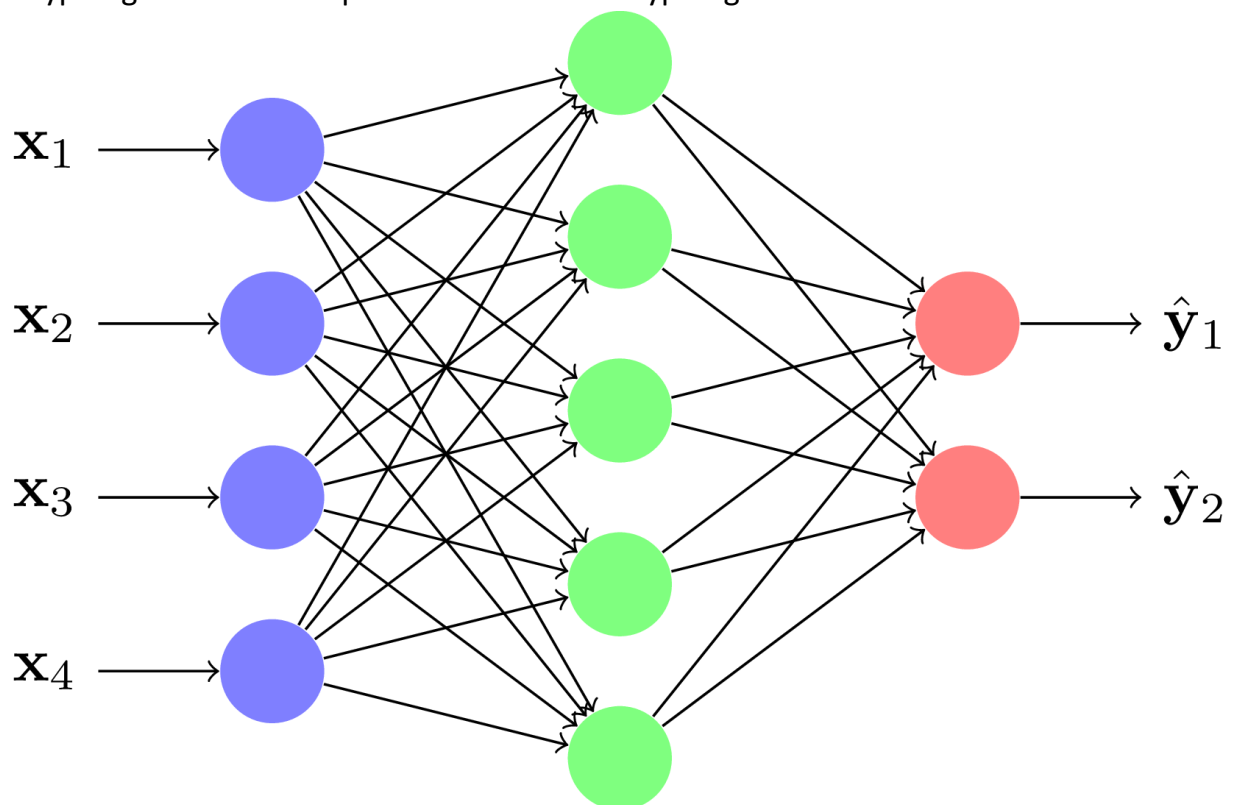


Figure 4 : Un réseau de neurones multicouche
[Source- <https://openclassrooms.com/fr/courses/5801891-initiez-vous-au-deep-learning/5814616-explorez-les-reseaux-de-neurones-en-couches#/id/r-5989693>]

La structure couche (ou multicouches) de neurones se présente comme suit :

- Un premier ensemble constitué d'un nombre n_1 de neurones tous en parallèles est aligné devant les entrées : c'est la première couche ou couche d'entrée du réseau de neurones.

- Chacun de ces neurones est relié à toutes les entrées (qui sont au préalable multipliées par leurs paramètres W_j) et à leurs paramètres respectifs b_{1i} .

- S'il existe des couches de neurones par la suite, au sein de chacune d'elles les neurones sont toujours en parallèles et chacun d'entre eux est reliés (en entrée) à tous les neurones de la couche précédente en plus d'être relié à son paramètre b_{ij} (i pour le numéro de la couche de neurone et j pour la position du neurone dans la couche i).

- Les neurones de la dernières couches produisent chacun une valeur Y_j et l'ensemble est inséré au sein d'un vecteur pour constituer le vecteur de sortie du réseau de neurone.

La méthode d'apprentissage par descente de gradient peut être appliquer à cette structure de neurones. Les données dans un réseau de neurones multicouches circulent toujours de la couches d'entrées vers les couches de sortie.

Si un neurone reboucle sur lui-même ou vers un neurone de sa couche ou d'une couche précédente (renvoie le signal vers son entrée ou vers l'entrée d'un autre neurone lui précédent), alors le réseau est dit récurrent.

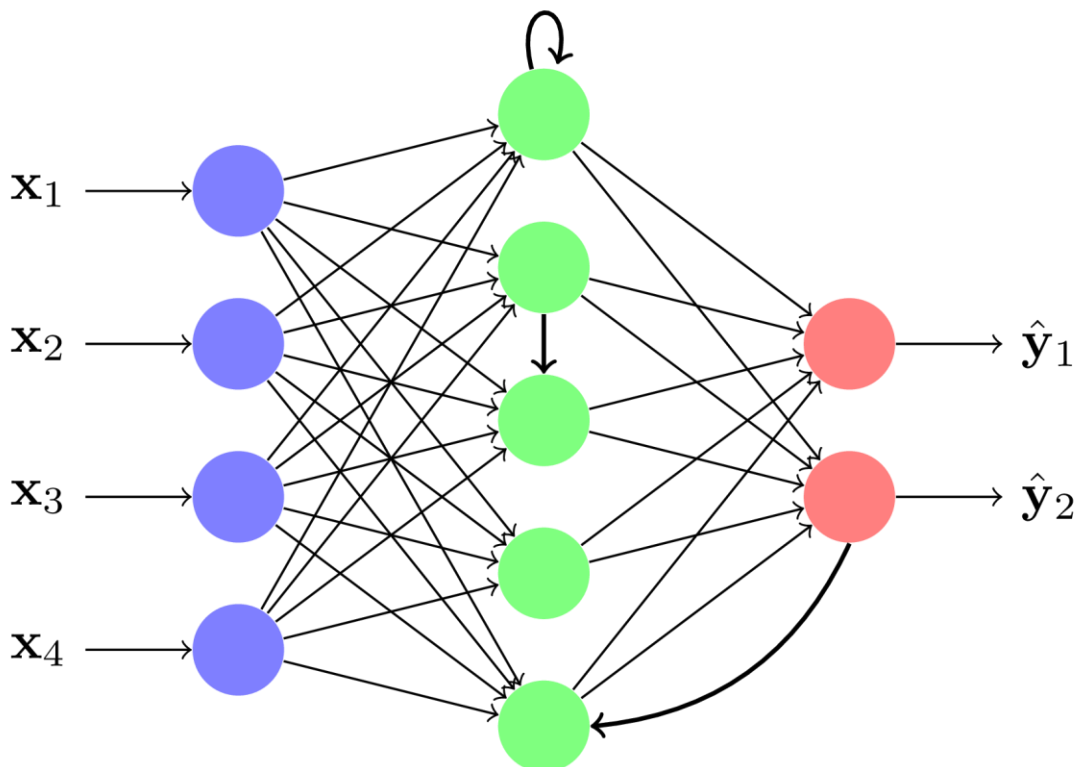


Figure 5 : Réseau de neurone récurrent [Source-

<https://openclassrooms.com/fr/courses/5801891-initiez-vous-au-deep-learning/5814616-explorez-les-reseaux-de-neurones-en-couches#/id/r-5989695>]

Les réseaux de neurones récurrents sont appropriés pour des données de tailles variables tandis que les neurones multicouches sont appropriés pour les données de taille fixe telles que les images. Elles sont donc à cet effet employés pour la conception du réseau de neurone VGG16.

DEUXIÈME PARTIE : LA MÉTHODE DE DESCENTE DE GRADIENT

Comme mentionné plus haut, pratiquer l'apprentissage d'un réseau de neurone c'est déterminer des valeurs optimales pour les paramètres W_i et b_i . Pour ce faire, on établit une fonction (fonction objectif) faisant intervenir ces paramètres et on essaye de déterminer les valeurs de ces derniers pour lesquelles la fonction est minimale.

La méthode de descente de gradient est donc l'une des méthodes (assez simple en comparaison à d'autres méthodes) pouvant permettre de déterminer ces valeurs.

Pour présenter cette méthode, nous présenterons un problème linéaire où l'on ne recherche qu'un seul paramètre b .

I- LE PROBLÈME DE LA DROITE OPTIMALE :

Considérons un plan avec n points repartis aléatoirement.

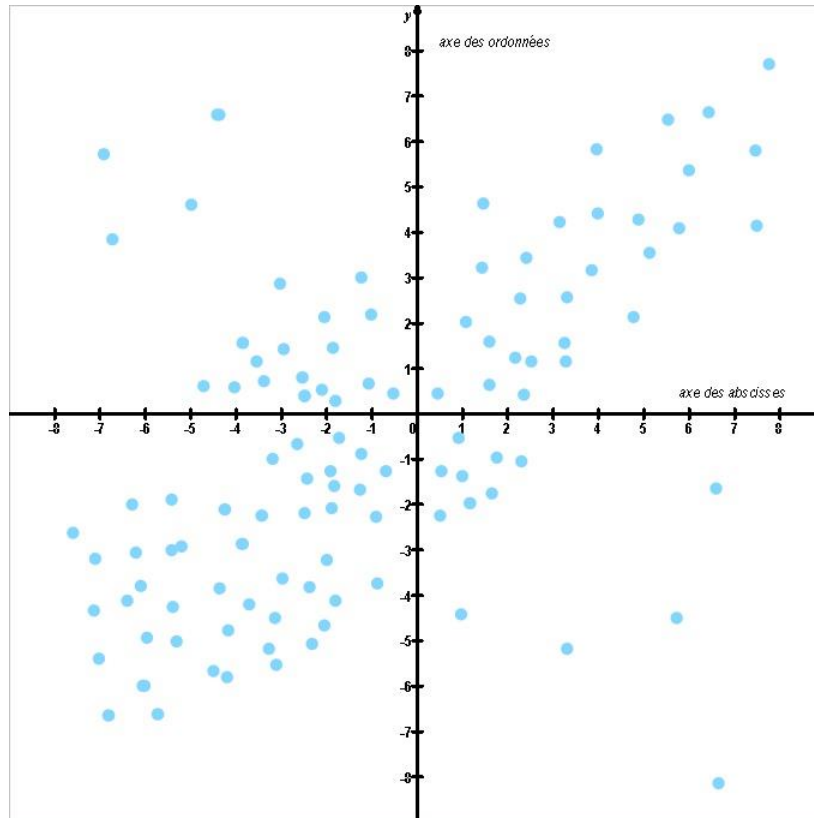


Figure 6 : Repère cartésien avec n points répartis aléatoirement [Source modifiée https://www.google.com/url?sa=i&url=https%3A%2F%2Fmiysaintbar.be%2Fmiysaintbar.be%2Findex.php%2Fpremiere-annee-3%2Frepere-cartesien&psig=AOvVaw18qPi_EXxkNvAGfdCDON45&ust=1592694355149000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCMDL5oP_juoCFQAAAAAdAAAAABAD]

Le but du problème est de trouver la valeur de b telles que la droite d'équation $y = bx$ passe le plus proche possible des n points.

1- Fonction objectif :

Le choix de la fonction objectif (fonction à minimiser) est très importante. En effet plus elle sera pertinente mieux précis le résultat le sera.

On choisit comme fonction objectif la moyenne de la somme des écarts au carré entre les valeurs observés (y_i ordonnées des n points) et les valeurs calculées par la droite d'équation $y = bx$:

$$f(b) = \frac{1}{n} \sum_{i=1}^n (y_i - bx_i)^2$$

On remarque bien que si tous les points sont alignés suivant une droite $y = \alpha x$ en prenant $b = \alpha$ la fonction objectif est bien nulle : le choix a donc été assez bien fait.

2- Minimisation de la fonction objective :

Le choix de la fonction objectif ayant été fait, il faut maintenant rechercher la valeur de b pour laquelle f est minimale.

La procédure adoptée dans la méthode de descente en gradient consiste à prendre une valeur de « b » au hasard et à la faire varier plus ou moins fortement par rapport à la pente de la fonction objective.

En effet, on sait que en b optimale la dérivé de la fonction f est nulle (et plus on s'en rapproche plus $|f'|$ diminue). Donc pour chaque valeur de b_i s'approchant de $b_{optimal}$, on détermine b_{i+1} par ajout à b_i d'une valeur b'_i qui est d'autant plus petite en valeur absolue que $|f'(b_i)|$ l'est.

La valeur b'_i est égale à $-f'(b_i)/t$.

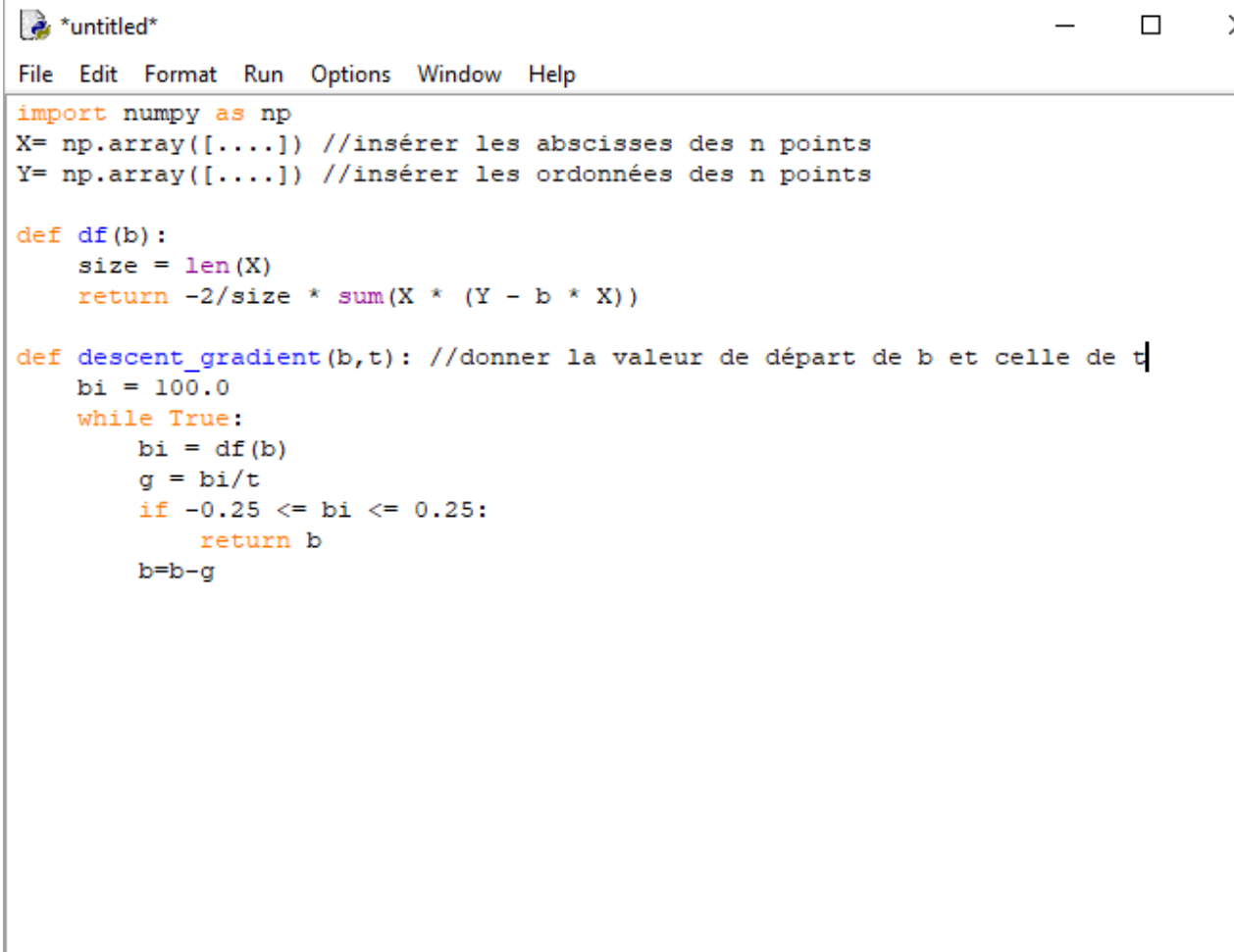
t est une constante appelée taux d'apprentissage permettant d'ajuster la taille du pas. Avec un taux d'apprentissage grand, on obtiendra (par résolution numérique) une bonne précision mais avec un temps de calcul long ; Au contraire avec un taux d'apprentissage petit, le temps de calcul est assez court mais le résultat pourrait être moins précis.

On a :

$$f'(b) = \frac{-2}{n} \sum_{i=1}^n x_i (y_i - bx_i)$$

3- Programmation Python :

L'écriture sous Python de la fonction f' et de la fonction de calcul de b donne :



```
*untitled*
File Edit Format Run Options Window Help

import numpy as np
X= np.array([....]) //insérer les abscisses des n points
Y= np.array([....]) //insérer les ordonnées des n points

def df(b):
    size = len(X)
    return -2/size * sum(X * (Y - b * X))

def descent_gradient(b,t): //donner la valeur de départ de b et celle de t
    bi = 100.0
    while True:
        bi = df(b)
        g = bi/t
        if -0.25 <= bi <= 0.25:
            return b
        b=b-g
```

Figure 7 : programmation Python de la méthode de descente de gradient pour le problème de la droite optimale

La condition d'arrêt $f'(b_i) = 0$ est représentée numériquement par $-0.25 \leq f'(b_i) \leq 0.25$.

4- Application :

On considère les cinq points $A(0,0)$, $B(10,5)$, $C(5,10)$, $D(33,-5)$ et $E(-5,33)$. On essaye de déterminer b par la méthode de descente de gradient.

On prend comme valeur initiale de b 100 et $t = 1000000$. On obtient après lancement du programme :

$b_{optimal} = -0.18512921473895982$.

Pour cette valeur la fonction objectif donne : $f(-0.18512921473895982) = 598.1522964098799$.

On est encore loin du zéro.