## Université of Science et Technologies HOUARI BOUMEDIENE Département d'informatique



APPRENTISSAGE AUTOMATIQUE

ET RESEAUX DE NEURONNES

# **« RAPPORT TP 6 »**

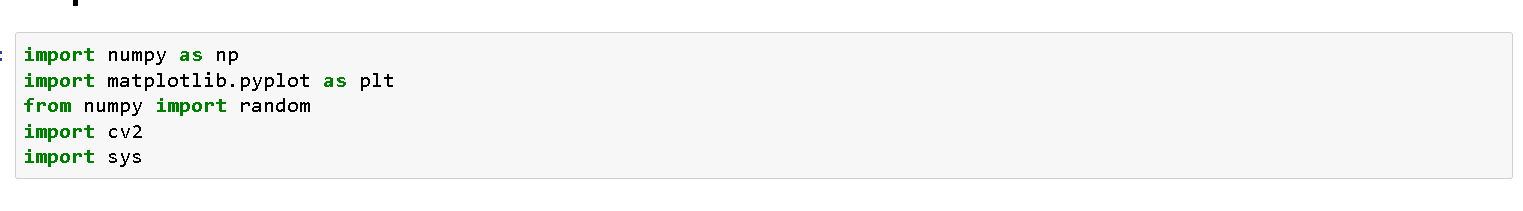
|  |  |  |  |
| --- | --- | --- | --- |
| *NOM* | *PRENOM* | *MATRICULLE* | *GROUPE* |
| ***BELIMI*** | ***Ibrahim Sabri*** | ***161631074255*** | ***A1 TP1*** |
| ***ZAIT*** | ***Fouad*** | ***181831072145*** | ***A1 TP1*** |
| ***ASMA*** | ***SRAOUIA*** | ***161631102642*** | ***A1 TP1*** |

***OBJECTIFS***

* Dans ce TP, nous aimerions entrainer un réseau de neurones pour la tâche de classification en utilisant l'algorithme d propagation en arrière (backpropagation).

L'ensemble de données que nous allons utiliser est le même que celui utilisé durant le TP1, i.e. les images de chiffres manuscrits.

**Importation des librairies nécessaires pour le travail**



***numpy :*** pour la manipulation des tableaux (array)

***matplotlib :*** pour la manipulation des graphs (2d, 3d)

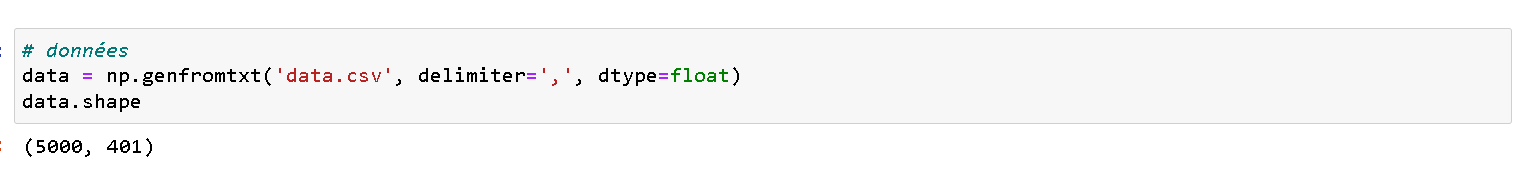
***CV2 :*** pour traiter les images

***SYS :*** pour interagir avec l’interpréteur Python

***SKLEARN :*** pour utiliser les outils de la machine learning

**Lecture des fichiers de données pour les classifier**

**Code :**



**Résultat :**

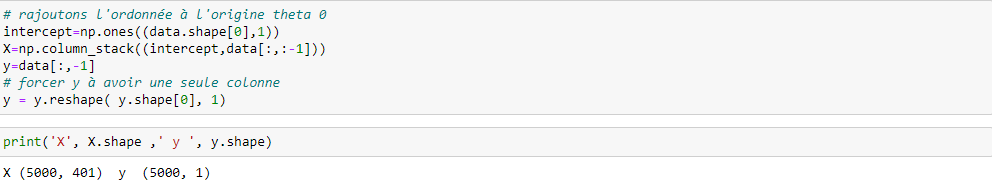
===> 

***5000 :*** signifie le nombre de données (ou exemples) dans le fichier (***data.csv***) c.à.d. il traite ***5000 images***

***401 :*** représente 401 colonnes dans le fichier

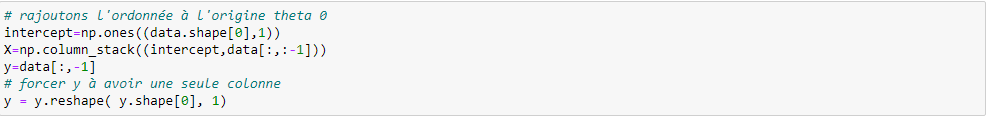
**(colonnes 1 à 400 X: Les pixels d’une image 20\*20 =400**

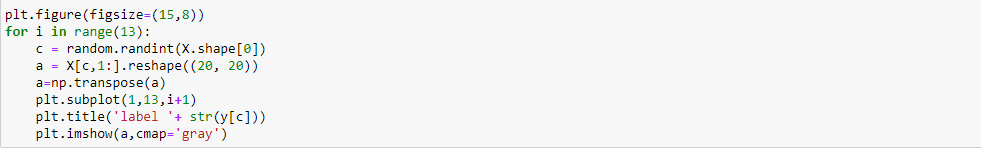
**colonne 401 Y: La classe d’une image de 0 à 9)**



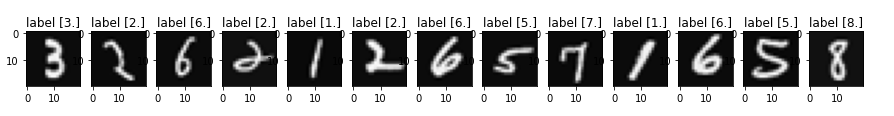
**Visualisation aléatoire de quelques données :**

**Code :**





**Résultat :**



**13** images sont représentées avec ses classes correspondantes (les labels).

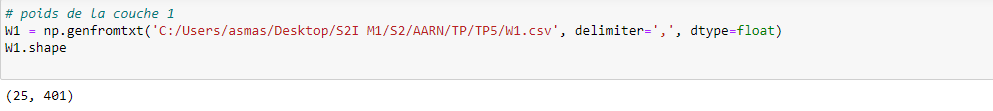
**Partie 1: Réseaux de neuronnes**

Dans cette partienous allons prédire la classe d’une image en utilisant un réseau de neurones.

Ce réseau contient 3 couches :

* Input layer : avec 401 neurones (taille d’image 20\*20 plus intercept).
* Hiden layer : avec 25 neurones.
* Output layer : avec 10 neurones (les classes de 0 à 9).

Nous allons charger les poids de chaque layer:

****

****

Initialiser nos paramètres à :

input\_layer\_size = 400

hidden\_layer\_size = 25

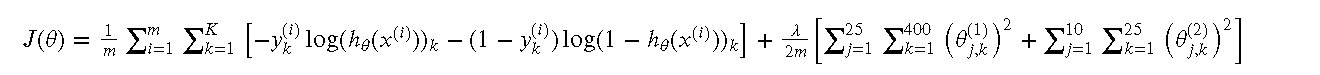
num\_labels = 10



**Calcul du cout:**

**Tâche 1**: Modifier la fonction computeCost afin d'obtenir un coût avec régularisation

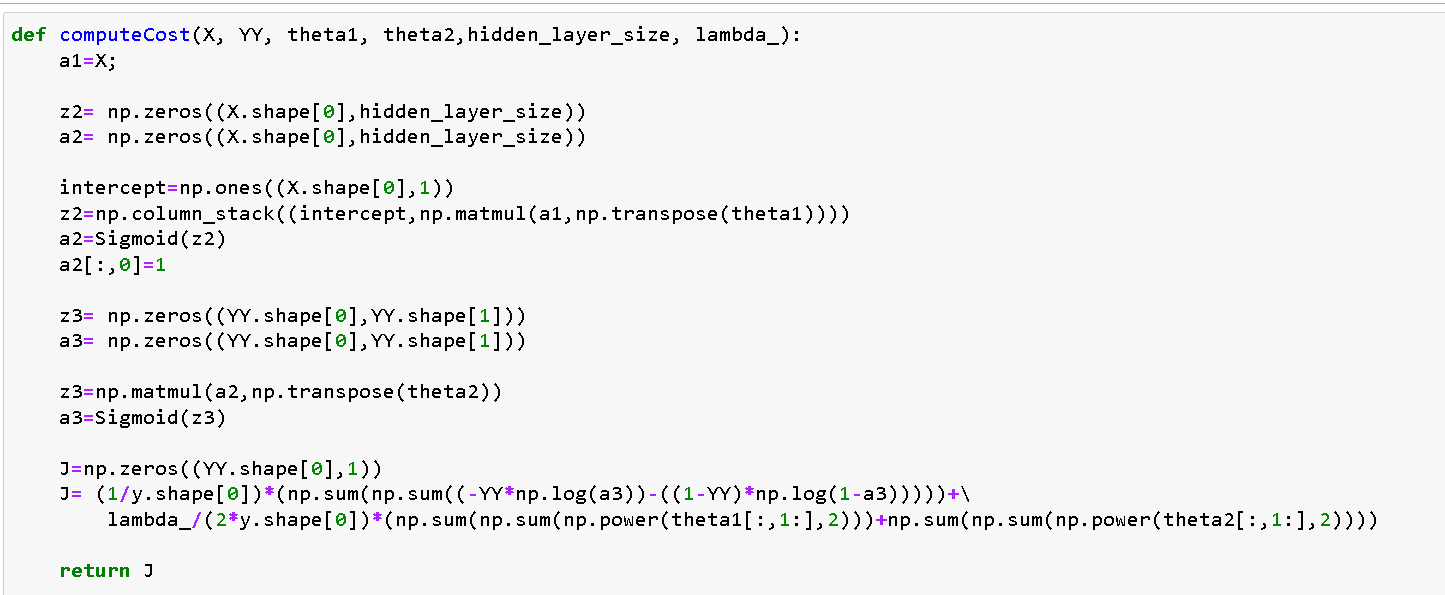
Rappelons que le coût avec régularisation est calculé comme suit:



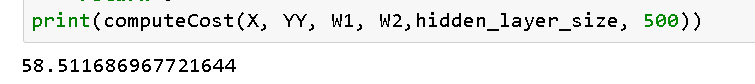
**Fonction Sigmoid:**



**Fonction ComputCost:**



**Resultat:**

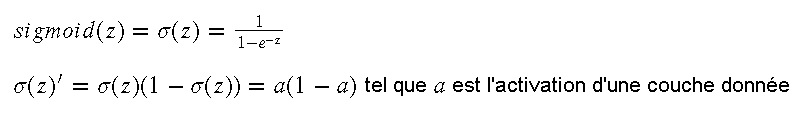


**Calcul du Gradient :**

**Tâche 2**: Implémenter la fonction NNCostFunction afin de retourner:

* Le coût avec régularisation
* Le gardient du coût par rapport à chaqu'un des paramètres du réseau

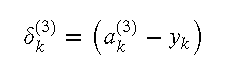
Rappelons le gradient de la sigmoid:



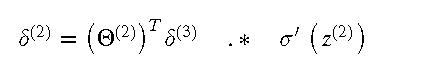
L'algorithme à implementer est comme suit:

Pour chaque exemple de l'ensemble d'apprentissage faire

Pour chaque noeud de la couche de sortie, calculer la dérivée (gradient):

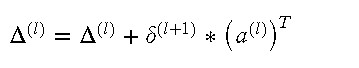


Pour chaque noeud de la couche caché calculer la dérivée (gradient):



Notons que l'opérateur .∗*.∗* represente la multiplication élement par élement et non pas la multiplication matricielle

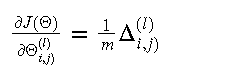
Le gradient de chaque noeud et de chaque couche sera finalement:



Fin pour

Fin pour

Diviser le gradient cumulé par le nombre d'exemples:

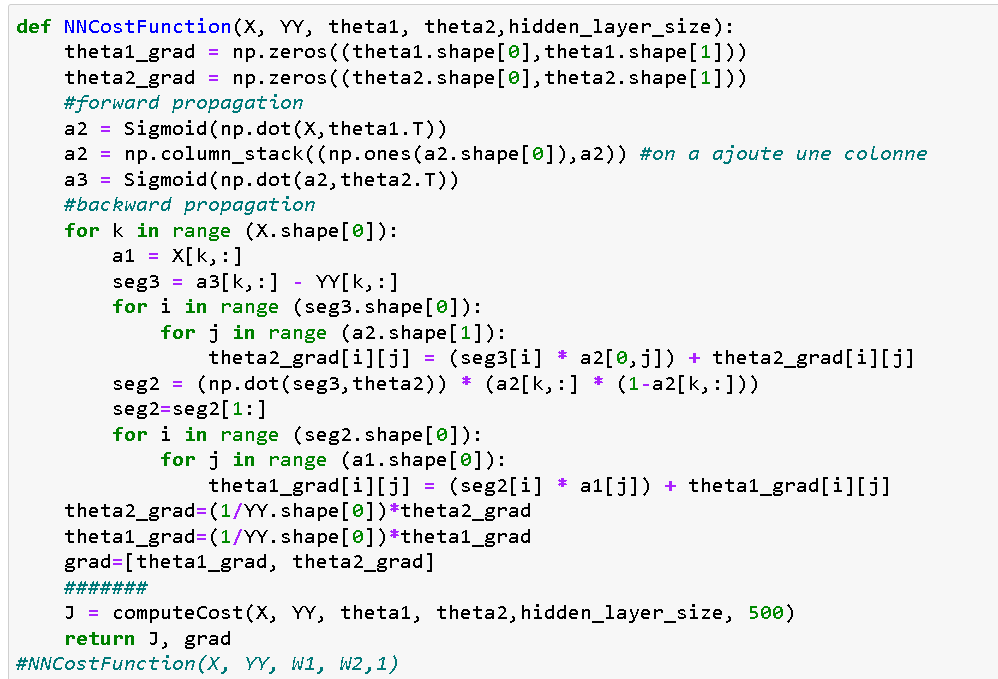


**Fonction NNCostFunction et GradientDescent :**

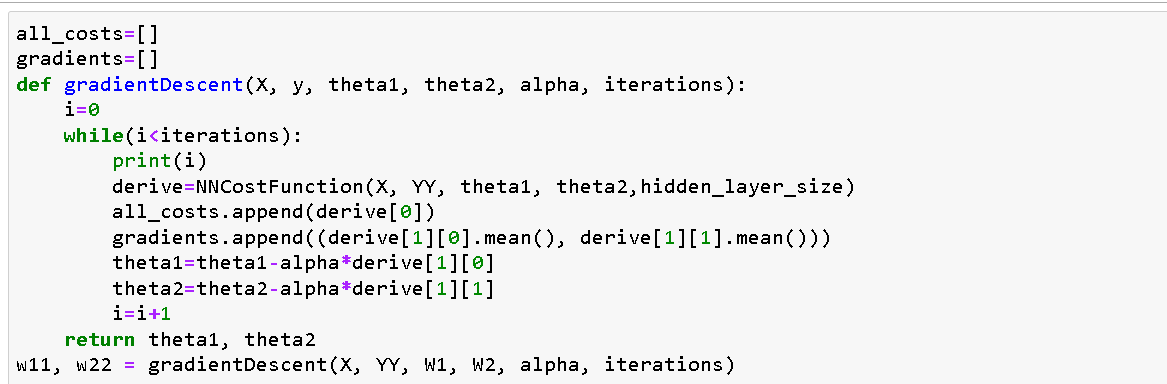
-on a implementer 2 fonctions chaqune par un membre de notre groupe on vous présente dans ce qui suit les 2 méthodes implémenté :

**Méthode 1 :**

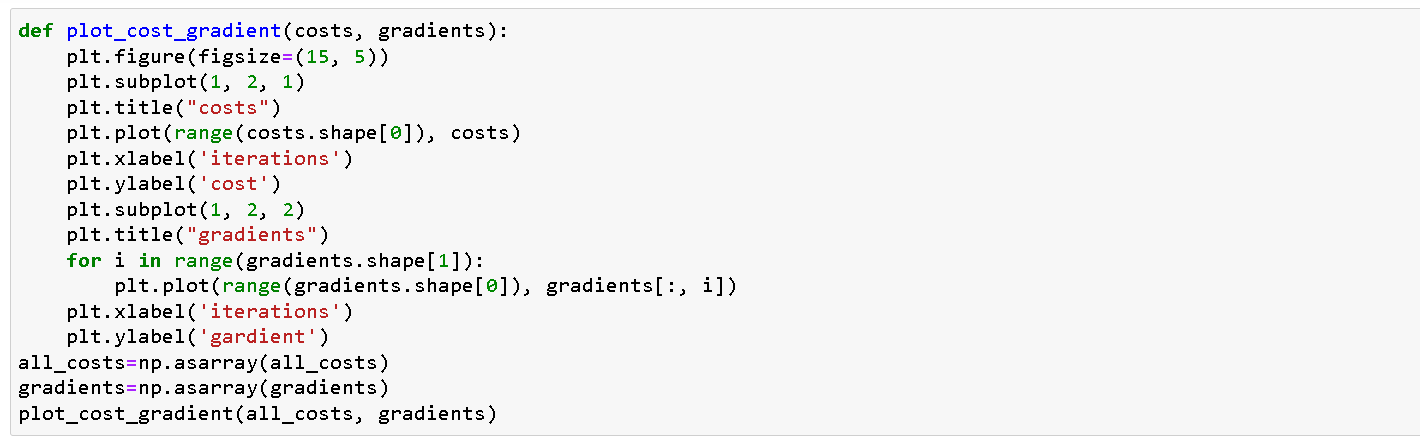
**NNCostFunction:**



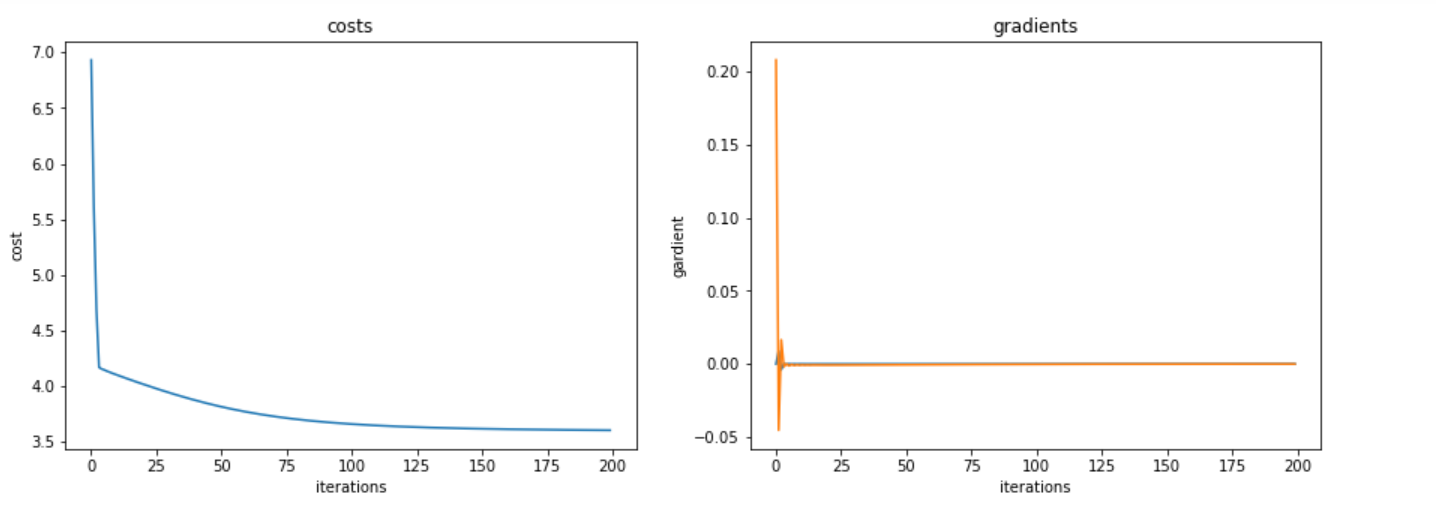
**GradientDescent :**



**Plot GradientDescent :**

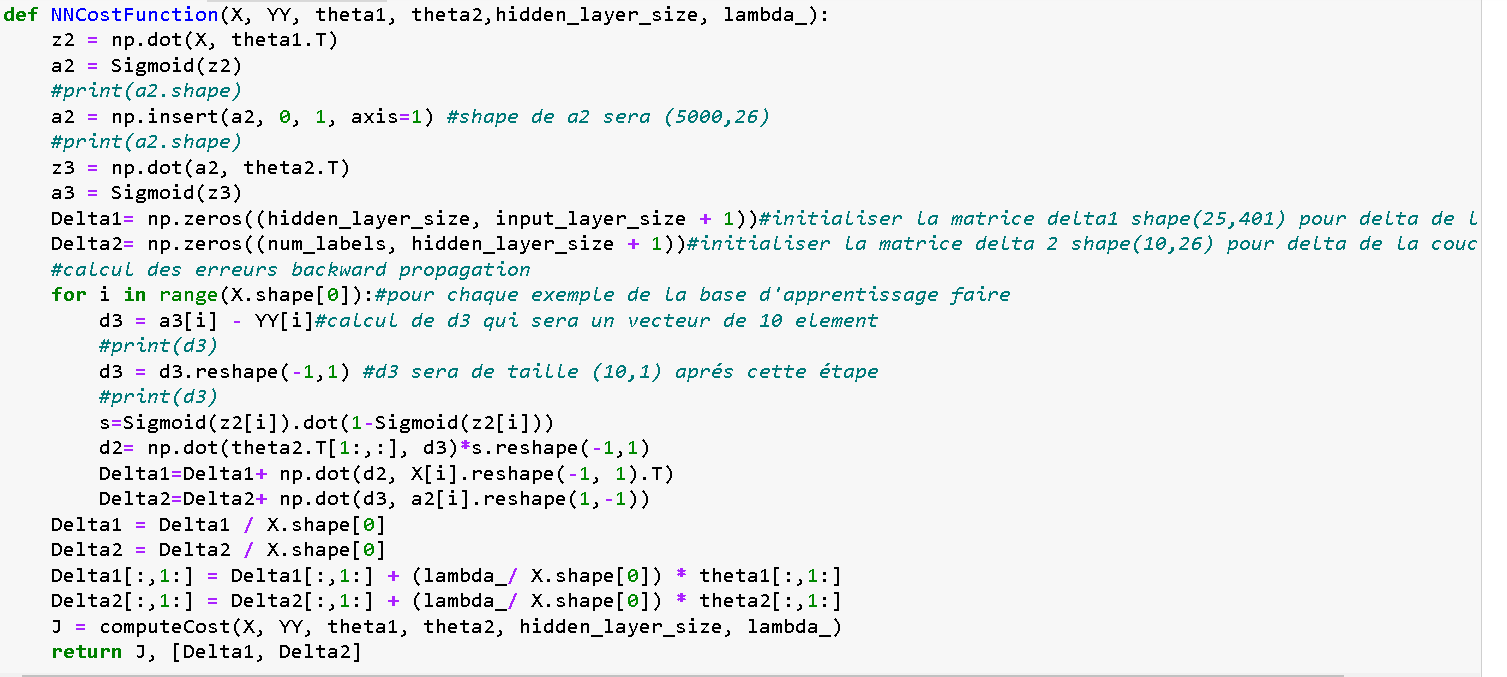


**Résultat:**

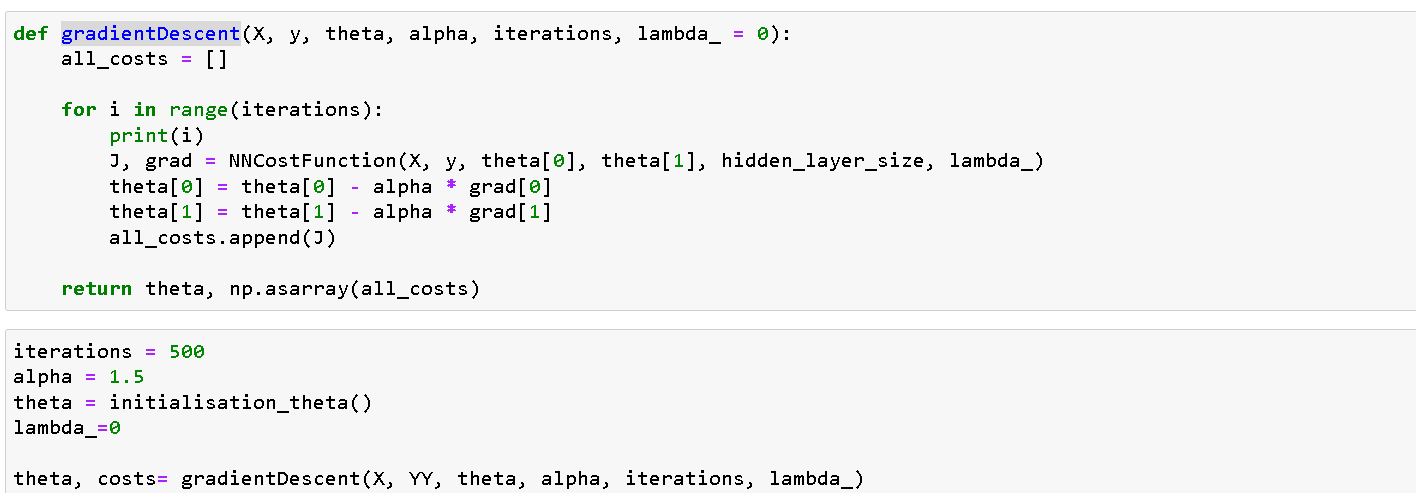


**-Méthode 2:**

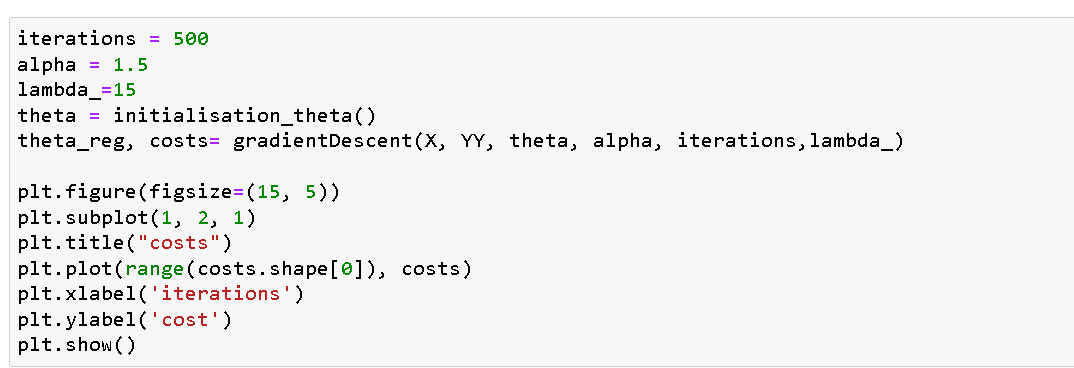
**NNCostFunction:**



**GradientDescent** :

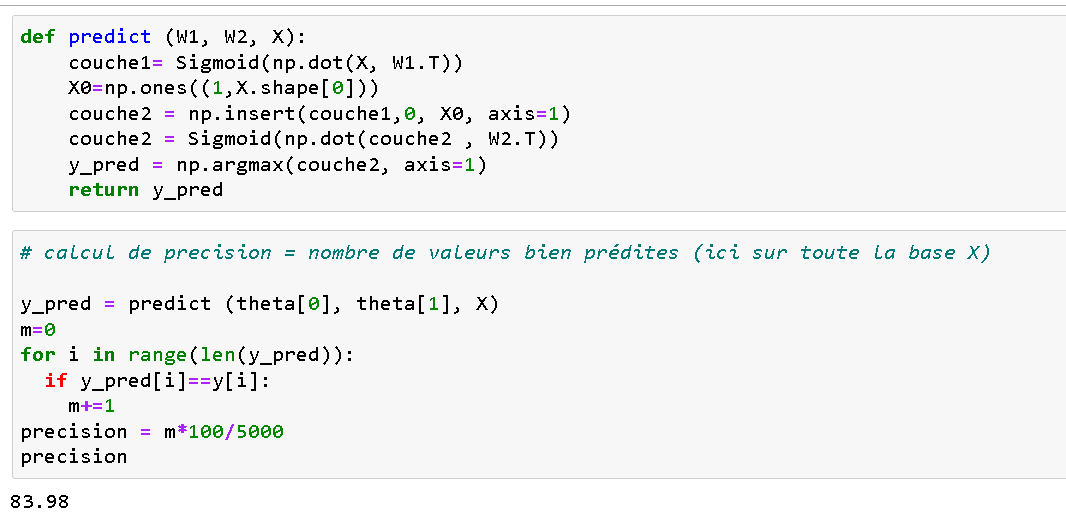


**Régularisation**:



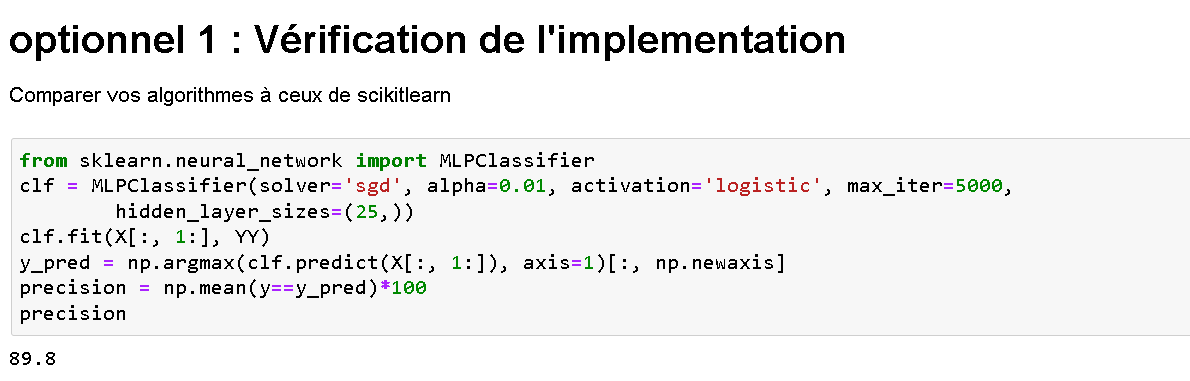
**2. Prédiction et précision :**

**Code et résultat :**



**Précision obtenu : 83.98 %**

**Vérification de l’implementation :**



**Précision obtenu : 89.8 %**

**FIN**