Quadrinôme :

- ABDESSAMED BOULARIACHE
- KEMOUM Meroua
- MAHIDDINE Mohamed Amine
- TAZIR REDA

TP 5 Réseaux de neurones - Multi classification et Propagation en avant

Dans ce TP, nous aimerions faire une classification multiclasse. Pour ce faire, nous allons comparer entre une classification utilisant la regression logistique, et une classification en utilisant les réseaux de neurones.

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 necessaires au travail

Entrée [1]:

```
import numpy as np
import matplotlib.pyplot as plt
from numpy import random
import cv2
```

Lecture des fichiers de données

Pour ce TP, nous allons lire les données à partir d'un fichier csv.

Entrée [2]:

```
# données
data = np.genfromtxt('data.csv', delimiter=',', dtype=float)
data.shape
```

Out[2]:

```
(5000, 401)
```

Dans ces données (data), les 400 premières colonnes representent les pixels de l'image (20x20), la dernière colonne represente la classe de l'image (chiffres de 0 à 9). (http://yann.lecun.com/exdb/mnist/ (http://yann.lecun.com/exdb/mnist/))

Chaque ligne represente un exemple de notre ensemble de données.

Mettons ces données dans leus vecteurs correspondants.

Entrée [3]:

```
# rajoutons l'ordonnée à l'origine theta 0
intercept=np.ones((data.shape[0],1))
X=np.column_stack((intercept,data[:,:-1]))
y=data[:,-1]
# forcer y à avoir une seule colonne
y = y.reshape( y.shape[0], 1)
```

Entrée [4]:

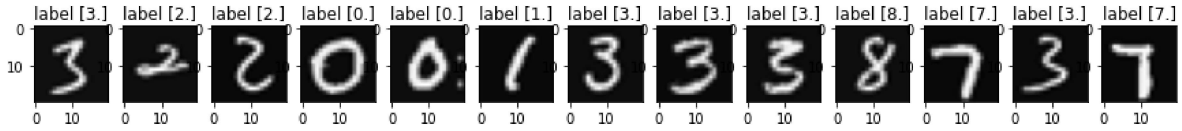
```
print('X', X.shape ,' y ', y.shape)
```

```
X (5000, 401)  y  (5000, 1)
```

Visualisation aléatoire de quelques données

Entrée [5]:

```
plt.figure(figsize=(15,8))
for i in range(13):
    c = random.randint(X.shape[0])
    a = X[c,1:].reshape((20, 20))
    a=np.transpose(a)
    plt.subplot(1,13,i+1)
    plt.title('label '+ str(y[c]))
    plt.imshow(a,cmap='gray')
```



Partie 1, régression logistique

Dans cette partie, nous aimerions utiliser la régression logistique pour classifier nos images

Rappelons que la régression logistique nous donne la probabilité d'appartenance (oui ou non) à la classe 1 (elle permet une classification binaire).

Pour étendre la régression logistique à une multi-classification, nous allons utiliser une stratégie 1 contre tous.

Nous paramètres θ seront donc une matrice avec un nombre de lignes égale au nombre de classes, et avec un nombre de colones égale au nombre decaractéristiques (chaque ligne de la matrice θ correspond aux paramètres d'un classifieur.

Entrée [6]:

```
def Sigmoid(z):
    # pour une valeur donnée, cette fonction calculera sa sigmoid
    return 1/(1+np.exp(-z));
```

Entrée [34]:

```
def lrCostFunction (X, y, initial_theta, alpha, MaxIter,lambda_):
    # dans cette fonction vous devez appliquer la regression logistique avec tout ce que ce
    # calcul du coût, minimisation du coût avec descente du gradient, et retour des paramèt

    m = len(X)
    X = np.matrix(X)
    y = np.matrix(y)
    theta = np.matrix(initial_theta)

    parameters = int(theta.shape[0])
    grad = np.zeros(parameters)

    for j in range(MaxIter):
        error = Sigmoid(X @ theta) - y

        for i in range(parameters):
            term = np.multiply(error, X[:,i])

            if (i == 0):
                grad[i] = np.sum(term) / m
            else:
                grad[i] = (np.sum(term) / m) + ((lambda_ * alpha / m) * theta[i,:])

        g =grad.reshape(theta.shape[0], 1)
        theta[0] = theta[0] - g[0]
        theta[1:,0] = theta[1:,0] - g[1:]

    return theta.ravel()
```

Entrée [35]:

```
def predictOneVsAll (all_theta, X):
    # ici en utilisant les paramètres calculés par la régression logisitique,
    # nous aiemrions retourner les etiquettes predites

    # Ici chaque classifieur retournera une de probabilité, il faudra choisir la probabilit
    # de tous les classifieurs d'une exemple donné
    # répeter pour tous les exmemples

    y_pred = np.argmax(Sigmoid(X @ all_theta.T), axis=1) # choix de la probabilité maximale
    y_pred = y_pred[:, np.newaxis] # redimensionner "y_pred" pour avoir les même dimensions

    return y_pred
```

Entrée [36]:

```
classes= np.unique(y)
number_classes=classes.shape[0]
all_theta = np.zeros((number_classes, X.shape[1]));
all_theta.shape
```

Out[36]:

```
(10, 401)
```

Entrée [39]:

```
MaxIter= 1000 #10000
lambda_= 0.1
alpha = 0.01
# initial_theta pour chaque classifieur
initial_theta=np.zeros((X.shape[1], 1));
for i in range (number_classes):
     # appel pour chaque classifieur
     theta = lrCostFunction(X,(y==classes[i]).astype(int),initial_theta,alpha, MaxIter,lamb
     all_theta[i,:]=theta;
```

Entrée [40]:

```
import sys
np.set_printoptions(threshold=sys.maxsize)
print(all_theta)
np.set_printoptions(threshold = False)
```

```
[[-2.52159875e+00  0.00000000e+00  0.00000000e+00 -1.63036534e-07
  -5.93937017e-06  8.40060686e-05  7.92283727e-04  2.14564918e-04
  -5.76323099e-04 -9.81131814e-04 -9.21530386e-04 -1.29137437e-04
   2.29962015e-04  4.55553408e-04  1.98769280e-03  1.91081187e-03
   8.53554501e-05  1.67687959e-05  1.86545434e-07 -2.73947614e-07
   0.00000000e+00 -1.44192481e-07 -3.95333261e-06  1.22419867e-05
   5.81661625e-04  1.47300661e-04 -9.16248245e-03 -4.17273467e-03
  -4.24613074e-03 -6.23091891e-03  2.95962645e-03  2.98943400e-03
   5.00684606e-04 -1.77643656e-03 -1.69662290e-02 -1.85506932e-02
  -1.28188056e-03  1.39571432e-03  1.05032972e-03 -2.23685223e-05
   4.44949075e-06  1.26658852e-06  1.06614339e-05 -3.32499915e-05
  -1.93879364e-03 -9.10801144e-03 -3.75372858e-02 -5.50980428e-02
  -8.46079484e-02 -1.11169289e-01 -1.09767059e-01 -1.19362255e-01
  -1.27437166e-01 -1.18941623e-01 -2.14065977e-01 -1.86836072e-01
  -4.90736441e-02 -2.62097889e-02 -1.14755765e-02  2.35494802e-03
   4.48598817e-04  4.85214662e-06  3.39261355e-04 -6.01998582e-04
  -3.48654380e-02 -7.49922857e-02 -1.16906063e-01 -2.02350939e-01
  -2.55595223e-01 -2.29956340e-01 -8.93066515e-02  1.25171400e-01
   6.15074596e-02 -2.11510878e-02 -1.41340300e-01 -1.50091827e-01
```

Entrée [41]:

```
all_theta.shape
```

Out[41]:

```
(10, 401)
```

Entrée [42]:

```
y_pred = predictOneVsAll(all_theta, X);
```

Qualité du classifieur RL

Prédire des valeurs de y

Ici il serait interessant de calculer la précision de notre classifieur

Essayons de calculer ça avec

moyenne(y==y-pred) * 100

Ceci donnera un pourcentage de precision

Entrée [43]:

```
precision = np.mean(y==y_pred)*100
precision
```

Out[43]:

```
93.24
```

Partie 2: Réseaux de neurones

Pour cette partie, nous choisissons une réseau simple:

- une couche d'entrée avec 400 noeuds (20 x 20 pixels) + le biais
- une couche cachée avec 25 noeuds
- une couche de sortie avec 10 noeuds (nombre de classes)

Entrée [44]:

```
# poids de la couche 1
W1 = np.genfromtxt('W1.csv', delimiter=',', dtype=float)
W1.shape
```

Out[44]:

```
(25, 401)
```

Entrée [45]:

```
# poids de la couche 2
W2 = np.genfromtxt('W2.csv', delimiter=',', dtype=float)
W2.shape
```

Out[45]:

```
(10, 26)
```

Entrée [46]:

```
input_layer_size  = 400;
hidden_layer_size = 25;
num_labels = 10;
```

Prédiction

Appliquer une propagation en avant en utilisant les paramètres données pour prédir les classes de l'ensemble d'apprentissage.

Entrée [47]:

```
def predict (W1, W2, X):

    # appliquer une propagation en avant
    # !--- n'oubliez pas d'appliquer la sigmoid à chaque couche afin d'avoir les probabilit

    # prédire la classe en choisissant la probabilité maximale parmi les 10 noeuds de sorti

    hiddenlayer1 = Sigmoid(X @ W1.T)

    intercept = np.ones((X.shape[0],1))

    hiddenlayer2 = np.column_stack((intercept, hiddenlayer1))
    hiddenlayer2 = Sigmoid(hiddenlayer2 @ W2.T)

    y_pred = np.argmax(hiddenlayer2, axis=1)[:, np.newaxis]

    y_pred = y_pred + 1
    y_pred[y_pred==10] = 0

    return y_pred
```

Entrée [48]:

```
# calcul de precision = nombre de valeurs bien prédites (ici sur toute la base X)
y_pred=predict(W1, W2, X)
precision = np.mean(y==y_pred)*100
precision
```

Out[48]:

```
97.52
```

Vérification de l'implementation

Comparer vos algorithmes à ceux de scikitlearn

Entrée [49]:

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(penalty="l2", max_iter=1000)

model.fit(X[:, 1:], np.squeeze(y, axis=1))

y_prob = model.predict_proba(X[:, 1:])
y_pred = np.argmax(y_prob, axis=1)[:, np.newaxis]

np.mean(y_pred==y)*100
```

Out[49]:

```
96.24000000000001
```

Renforcement d'apprentissage

Mettre ici toute idée qui pourrait renforcer votre apprentissage

Entrée [ ]:

```
```

Consignes

Le travail est à remettre par groupe de 4 au maximum [1..4].

Le délai est le vendredi 22 Avril 2022 à 22h

Entrée [ ]:

```
# bonne chance
```