02-MM-IA-Numpy-introduction

November 16, 2021

1 Introduction

Cette partie sera consacrée à la librairie numpy qui est largement utilisée pour la manipulation des tableaux de données. En gros, dès qu'il s'agit d'un tableau de dimension quelconque, on pense à numpy !!

Dans le domaine de l'apprentissage, les données peuvent avoir plusieurs formats numériques qui seront par la suite traduites en **tableaux** ou **dataframe** avant toute manipulation.

Par exemple, une image numérique en niveaux de gris est un tableau de deux dimensions (matrice), un texte peut être représenté par un tableau (word2vecor), un fichier .csv contenant des informations sur les cartes GPU et leurs performances sur 20 ans, etc.

Il est à noter que le codage ainsi que la manipulation des tableaux numériques est une partie fondamentale pour une application rapide et efficace.

Pour les petits tableaux on pourrait se contenter des listes mais, comme nous allons le voir par la suite, il est recommandé de se familiariser avec les tableaux avec l'aide de numpy

[ ]:
```
# comme n'importe quelle librarire, il faut commencer par la charger
# à l'aide de la commande import

import numpy
# maintenat que c'est fait on peut utiliser son contenu
# par exemple vérifier la version installer
numpy.__version__
```

[ ]:
```
# et si on lui donne un nom pour faciliter les appels
import numpy as np
np.__version__
```

[ ]:
```
# par la suite, si on ne se rappelle pas d'une commande, on peut se faire aider
#par tabulation (magique)

#np.TAB : on choisit dans la liste ou on commence par ...
a=np.array([-1, -3, -1, 1])
a=np.abs(a)
print(a)
#print(a.sum())
```

Avant de commencer à manipuler les tableaux, il est important de savoir ce qu'on manipule : type, structure, indices, etc.

Afin de remplir des tableaux aléatoirement, on va souvent utiliser les fonctions de la classe **random**

[ ]:
```
#numpy a été déjà chargé, la ligne suivante est juste pour nous rappeler
import numpy as np

np.random.seed(2) # important pour reproduire les mêmes data
a1 = np.random.randint(10,100, size=5)  # tableau : une seule dimension
print('a1=', a1)
```

[ ]:
```
print('-------------------------\n')
a2 = np.random.randint(10, size=(3, 4))  # tableau : 2 dimensions
print('a2=',a2)
```

[ ]:
```
#pour connaître les propriétées d'un tableau
print("a2 ndim: ", a2.ndim)
print("a2 shape:", a2.shape)


#remarquer que shape contient tous les autres informations : c'est le
#premier reflex dès qu'il s'agit d'un tableau...
```

[ ]:
```
print('-------------------------\n')
a3 = np.random.randint(10, size=(3, 4, 5))  # tableau : 3 dimensions
print('a3=', a3)
```

[ ]:
```
print("a3 size: ", a3.shape)
```

[ ]:
```
#pour connaître le type
print("le type de ce tableau est : ", a1.dtype)
```

[ ]:
```
#qui dit tableau dit indice
print(a1, 'a comme taille : ', a1.shape)
#il s'agit d'un tableau de trois lignes
```

[ ]:
```
#on affiche un élément si on connait un indice valide qui
#commence par ZERO et finit par taille-1 (équivalent -1)
print(a1)
print('Le premier élément est :',a1[0])
print('Le dernier élément est :',a1[-1])
```

[ ]:
```
print('Le nombre d\'éléments est :',a1.shape[0])
print('------')
#Afficher tous les éléments grâce à shape
for i in range(a1.shape[0]):
    print(a1[i])
```

```
    print('------')
```

[ ]:
```
#Afficher et vérifier le dernier élément
a1[-1]
```

1.1 Même principe pour des dimensions supérieures

[ ]:
```
#pour connaître le type
print("le type de ce tableau est : ", a2.dtype)
```

[ ]:
```
#qui dit tableau de 2 dimensions dit indice 2 indices
print(a2, '  \t a pour  taille : ', a2.shape)
#il s'agit d'un tableau de trois lignes
print(a2.shape[1])
```

[ ]:
```
#on affiche une ligne si on connait un indice valide
print('--------------------------')
print('Tableau : \n',a2)
print('--------------------------')
print('élément 3ème ligne et 2 ème colonne : ',a2[2,1])
# Ou une colonne
print('--------------------------')
print('La 2ème ligne du tableau : ',a2[1,:])
print('--------------------------')
print('La 3ème colonne du tableau : ',a2[:,2])
#il s'agit de la 3 ème colonne
```

[ ]:
```
## Juste des parties d'un tableau (slicing ou sous-tableau)
a = np.arange(5,10)
print('--------------------------')
print('Le contenur du tableau : ',a)
print('--------------------------')
print('Le type du tableau : ',type(a))
```

[ ]:
```
print('A partir du 2ème élément : ',a[1:]) #2 ème élément
```

[ ]:
```
print(a) #rien ne change dans le tablea
```

[ ]:
```
print(a[1:3]) #2 et 3 ème élément
print('--------------------------')
print(a[:5]) #début jusqu'au n-1 ème (n=5)
print('--------------------------')
print(a[0:5:2]) #par pas de deux
print('--------------------------')
print(a[::2]) #par pas de deux
print('--------------------------')
```

```
print(a[::-1])#order inversé
```

1.2 Changer le format (taille) d'un tableau

[ ]:
```
tab = np.random.randint(10,size=(3,4))#  3 lignes et 4 colonnes
print('nombre de lignes :',tab.shape[0])
print('--------------------------')
print('nombre de colonnes : ',tab.shape[1])
```

[ ]:
```
# si on veut transformer le tableau en une ligne :
# il nous faudra 3x4=12 éléments
a=tab
a=a.reshape(a.shape[0]*a.shape[1],1)#c'est  possible
print('nombre de lignes :',a.shape[0])
print('--------------------------')
print('nombre de colonnes : ',a.shape[1])
```

[ ]:
```
print('----------------------------')
a=a.reshape((3,4)) # retour
print('nombre de lignes :',a.shape[0])
print('--------------------------')
print('nombre de colonnes : ',a.shape[1])
```

1.3 On peut fusionner plusieurs tableaux si on fait attention aux dimensions

[ ]:
```
x = np.array([1, 2, 3]) # tableau 1 avec 3 elts
y = np.array([5, 6, 7, 8, 9, 10]) # tableau 2 avec 6 elts
z=np.array(np.concatenate([x, y])) #fusion de 1&2 en faisant attention aux␣
 ↪dimensions
print('nombre de lignes et de colonnes dans z :',z.shape)
print('--------------------------')
w=np.array(np.concatenate([x, y,z])) #fusion de 1&2&3
print('nombre de lignes et de colonnes dans w:',w.shape)
```

1.4 Les fonctions ou méthodes (pré-définies)

[ ]:
```
#l'avantage est de faire un seul appel pour tout le tableau sans boucle
x = np.array([-2, -1, 0, 1, 2])
print('--------------------------')
print('La valeur absolue appliquée à un tableau : \n',np.abs(x))
print('--------------------------')
print('La valeur absolue appliquée à un tableau plus grand : \n',np.abs(a3))
```

[ ]:
```
#Exemples de fonctions : fonctions trigonométriques
theta = np.linspace(0, np.pi, 3) #très pratique
y=np.cos(theta)
```

```
print(y) #cos, tan, arcsin, etc (à l'aide de TAB)
```

[ ]:
```
import matplotlib.pyplot as plt #la fameuse biblio pour l'affichage
# A quoi sert la ligne suivante ?
%matplotlib inline

theta = np.linspace(0, np.pi, 100)
y=theta**10*np.sin(theta)
plt.plot(theta,y)
```

[ ]:
```
# log et exp
a=np.array([1, 2, 4, 16])
print(np.log(a))#log_n
print(np.exp(a))#exp_n
#il y a aussi d'autres fonctions : max, min, sum, mean,std,  power, add.
 ↪accumulate, etc...
```

[ ]:
```
a=np.linspace(0,10,1000000)
y=1/(np.sqrt(2*np.pi))*np.exp(-.5*(a-5)**2)
plt.plot(a,y)
```

[ ]:
```
print('- la moyenne : ', np.mean(a))
print('- l\'écart type : ',np.std(a))
```

[ ]:
```
Tab = 5+np.random.random((3, 4))*(10-5)
print(Tab)

#?np.random.random
```

[ ]:
```
print('La somme des éléments du tableau est :', Tab.sum())
#print(Tab.cumsum())
```

[ ]:
```
print(Tab.min(axis=1))#max, mean, sum...  par ligne ou par colonne
```

1.5 Opérations booléennes

[ ]:
```
Tab1=np.zeros((3,4))#tableau de 0 utile pour initialiser

print(Tab1.any()) #test de tous les éléments
print('-------------------------')
Tab2=np.ones((2,3))#tableau de 1 utile pour initialiser
print(Tab2.any())
print(Tab2.all())
print('-------------------------')
Tab3=np.random.randint(5,size=(2,3))
print(Tab3.any())
```

```
print(Tab3.all())
```

1.6 Un petit mot sur l'affichage

il s'agit de la librairie matplotlib qui ressemble à celle de scilab mais encore plus puissante

En voilà un avant-goût

[ ]:
```
#charger la librairie comme plus haut (rappel)
import matplotlib.pyplot as plt

theta = np.linspace(-np.pi,np.pi,100)
y=np.cos(theta)
plt.plot(theta,y,'red',label='cos')

y=np.sin(theta)
plt.plot(theta,y,color='green', linestyle='dashed', marker='o',
     markerfacecolor='blue', markersize=2, label='sin')

plt.ylim([-1.1,1.4])
plt.legend(loc='upper right',ncol=2, mode="expand", shadow=True, fancybox=True)
plt.title('cos & sin')
```

[ ]:
```
# un sondage sur la taille
nb_simulations=1000000
mu=1.5 #mean
scale=0.15#std

#on génère un échantillion selon la loi normale
y=np.random.normal(mu,scale,nb_simulations)
#on affiche l'histogramme
plt.hist(y,200,density=True,color='m')
plt.title('Distribution de la taille')
plt.xlabel('Taille')
plt.ylabel('densité');
```

[ ]:
```
?plt.hist
```

2 Créer une fonction et l'appliquer sur tout le tableau (sans boucle)

3 Affichage avec subplot

[ ]:
```
def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.subplot(211) # 2 lignes et 1 colonnes : position 1
plt.plot(t1, f(t1), "bo")
plt.plot(t2, f(t2), "k")

plt.subplot(212) # 2 lignes et 1 colonnes : position 2
plt.plot(t2, np.cos(2*np.pi*t2), "r--")
plt.show()
```

3.0.1 pour avoir plus de détails ou besoin d'aide:

```
?np.random.normal
?plt.hist
```

4 Pratique

Exercice 1

1- Générer un tableau aléatoire d'entiers entre 0 et 15 et de dimension (1,100)

2- Calculer et afficher : moyenne, max, min, écart type

3- Générer un tableau aléatoire de réels entre -1 et 1 et de dimension (1,10000)

4- Calculer et afficher : moyenne, max, min, écart type

Exercice 2

1- Créer une matrice aléatoire de taille (5x6)

2- Remplacer une colonne sur deux, sauf la première colonne, par sa valeur moins le double de la colonne suivante

3- Remplacer les valeurs négatives par 0 en utilisant un masque binaire (une matrice de 0 et de 1)

Exercice 3

1- Choisir une image de niveaux de gris à récuoérer sur votre home ou en ligne

2- Charger cette image dans une variable I et Afficher I avec l'aide de ce code :

```
*import matplotlib.image as im
```

```
*I=im.imread('votre-image.format')

*import matplotlib.pyplot as plt

*plt.imshow(I)
```

4- Afficher la taille de I et vérifier qu'elle est en niveaux de gris

5- Transformer I en un tableau d'une seule colonne

6- Calculer et afficher l'histogramme de I

7- Soustraire la moyenne de I et afficher le résultat

6- Calculer le nouveau histogramme

8- Commenter

Exercice 4

1- Créer 9 matrices aléatoires de taille (128x128)

2- Afficher les matrices commes des images avec l'aide de subplot

3- Créer et tester une fonction qui permet de calculer l'image moyenne de toutes ces images

4- Lire l'image de Lena dans un tableau I (dispo sur l'ent et sur le net).

5- Créer 10 matrices M1, M2, .. M10 aléatoire de 0 et 255 et qui ont la même taille que I

6- Ajouter chaque image aléatoire à I tel que Ji=I+Mi, i=1:10

7- Calculer et afficher l'image moyenne de J1, J2, .., J10

8- Commenter

5 Calcul de l'intégrale

[ ]:
```
xmin = 0
xmax = np.pi
nbx = 30
nbi = nbx - 1 # nombre d'intervalles

x = np.linspace(xmin, xmax, nbx)
y = 1 + np.cos(x)
plt.plot(x,y,"bo-")

integrale = 0
for i in range(nbi):
    integrale = integrale + .5*(y[i+1]+y[i])*(x[i+1]-x[i])
    # dessin du rectangle
    x_rect = [x[i], x[i], x[i+1], x[i+1], x[i]] # abscisses des sommets
    y_rect = [0   , y[i], y[i]  , 0     , 0   ] # ordonnees des sommets
    plt.plot(x_rect, y_rect,"r")
```

```
print("integrale =", integrale ,'alors que la valeur exacte est ', np.pi)

plt.show()
```

6 Sauvegarde d'un tableau

[ ]:
```
# Pour des tableaux de dimension trois ou plus, il faut utiliser np.save

x = np.linspace(0, 1, 100)
a=np.exp(-20*(x-.5)**2)
np.savetxt('tableau.txt', a)# en fichier texte à vérifier avec un éditeur
np.save('tableau.npy', a) # en fichier numpy rapide et facile à charger

import os
#charger les fichier
os.path.getsize('tableau.npy')
b=np.load('tableau.npy')

plt.plot(b)
```

7 Application

A l'aide des fonction *sin* et *cos* créer une figure qui ressemble à celle donnée en dessous

[ ]:
```
import matplotlib.image as im
I=im.imread('Figure_diff.png')
plt.imshow(I)
```

[ ]:
```
I=im.imread('test.png')
plt.imshow(I)
```

8 Surprise me !!