TP: Travailler avec des vecteurs, des matrices et des tableaux dans NumPy

NumPy est un élément fondamental dans l'ensemble des outils Python pour l 'apprentissage profond. Il permet d'effectuer des opérations efficaces sur les structures des données souvent utilisées dans l'apprentissage automatique, qu'il s'agisse de vecteurs, de matrices ou de tenseurs.

1. Créer un vecteur

```
# Creer un vecteur

# Utiliser NumPy pour creer un tableau a une seule dimension
# Chargement de la bibliotheque
import numpy as np

# Creation d'un vecteur ligne
vector_row = np.array([1, 2, 3])

# Creation d'un vecteur colonne
vector_column = np.array([[1], [2], [3]])
```

```
vector_row
```

```
array([1, 2, 3])
```

```
vector_column
```

```
array([[1],
       [2],
       [3]])
```

```
print(vector_row)
```

```
[1 2 3]
```

```
print(vector_column)
```

```
[[1]
 [2]
 [3]]
```

2. Créer une matrice

```
# Creer une matrice

# Utiliser NumPy pour creer un tableau a deux dimensions
# Chargement de la bibliotheque
import numpy as np


# creation d'une matrice
matrix= np.array([[1, 2], [1, 2], [1, 2]])
```

```
matrix
```

```
array([[1, 2],
       [1, 2],
       [1, 2]])
```

```
print(matrix)
```

```
[[1 2]
 [1 2]
 [1 2]]
```

3. Créer une matrice creuse

```
# Creer une matrice creuse

# Chargement des bibliotheques
import numpy as np
from scipy import sparse

# creation d'une matrice
matrix = np.array([[0, 0], [0, 1], [3, 0]])

# creation d'une matrice au format CSR (compressed sparse row, ligne creuse compressee)
matrix_sparse = sparse.csr_matrix(matrix)

# visualisation de la matrice creuse
print(matrix_sparse)
```

```
  (1, 1)        1
  (2, 0)        3
```

Il existe un certain nombre de types de matrices creuses. Toutefois, dans les matrices au format CSR, (1, 1) et (2, 0) représentent respectivement les indices (qui démarrent à partir de zéro) des valeurs non nulles 1 et 3. Nous pouvons voir l'avantage des matrices creuses si nous creons une matrice beaucoup plus grande avec beaucoup plus d'éléments nuls et que nous comparons ensuite cette matrice plus grande avec notre matrice creuse d'origine :

```
# creation d'une matrice plus grande
matrix_large = np.array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                         [0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
                         [3, 0, 0, 0, 0, 0, 0, 0, 0, 0]])

# creation d'une matrice au format CSR
matrix_large_sparse = sparse.csr_matrix(matrix_large)

# visualisation de la matrice creuse d'origine
print(matrix_sparse)

# visualisation de la matrice creuse d'origine
print(matrix_large_sparse)
```

```
  (1, 1)        1
  (2, 0)        3
  (1, 1)        1
  (2, 0)        3
```

L'ajout d'éléments nuls n'a pas modifie la taille de la matrice creuse.

4. Preallouer des tableaux NumPy

Préallouer des tableaux d'une taille donnée avec une certaine valeur.
NumPy dispose de fonctions permettant de générer des vecteurs et des matrices de n'importe quelle taille en utilisant des 0, des 1 ou des valeurs de votre choix :

```
# Chargement de la bibliotheque
import numpy as np

# Genere un vecteur de forme (1,5) ne contenant que des 0
vector = np.zeros(shape=5)

#Visualisation du vecteur
print (vector)

# Genere une matrice de forme (3,3) ne contenant que des 1
matrix = np.full(shape=(3,3), fill_value=1)

#Visualisation de la matrice
print(matrix)
```

```
[0. 0. 0. 0. 0.]
[[1 1 1]
 [1 1 1]
 [1 1 1]]
```

5. Sélectionner des éléments :

La sélection de un ou plusieurs éléments dans un vecteur ou une matrice.

NumPy permet de sélectionner facilement des éléments dans des vecteurs ou des matrices

```
# Chargement de la bibliotheque
import numpy as np

# Creation d'un vecteur ligne
vector = np.array([1, 2, 3, 4, 5, 6])

# Creation d'une matrice
matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# selection du troisieme element du vecteur
vector [2]
```

```
3
```

```
# Selection de la deuxieme ligne de la deuxieme colonne
matrix [1,1]
```

```
5
```

NumPy offre une grande variété de méthodes pour sélectionner (indicer et découper) des éléments ou des groupes d'éléments dans les tableaux :

```
# Selection de tous les elements d'un vecteur
vector[ :]
```

```
array([1, 2, 3, 4, 5, 6])
```

```
# Selection de la totalite jusqu'au troisieme element inclus
vector[ :3]
```

```
array([1, 2, 3])
```

```
# Selection de tout ce qui trouve apres le troisieme element
vector[3 :]
```

```
array([4, 5, 6])
```

```
# Selection du dernier element
vector[-1]
```

```
6
```

```
# Inversion du vecteur
vector[ ::-1]
```

```
array([6, 5, 4, 3, 2, 1])
```

```
# Selection des deux premieres lignes et toutes les colonnes d'une matrice
matrix[ :2, :]
```

```
array([[1, 2, 3],
       [4, 5, 6]])
```

```
# Selection de toutes les lignes et la deuxieme colonne
matrix[ :,1:2]
```

```
array([[2],
       [5],
       [8]])
```

6. Décrire une matrice

Décrire la forme, la taille et les dimensions d'une matrice

Utiliser les attributs shape, size et ndim d'un objet NumPy

```
# Chargement de la bibliotheque
import numpy as np

# Creation d'une matrice
matrix = np.array([[1, 2, 3, 4], [5, 6, 7 ,8], [9, 10, 11, 12]])

# Affichage du nombre de lignes et de colonnes
matrix.shape
```

```
(3, 4)
```

```
# Affichage du nombre d'elements (lignes * colonnes)
matrix.size
```

```
12
```

```
# Affichage du nombre de dimensions
matrix.ndim
```

```
2
```

7. **Appliquer des fonctions à chaque élément**

   On souhaite appliquer une fonction a tous les éléments d'un tableau.

```
# Chargement de la bibliotheque
import numpy as np

# Creation d'une matrice
matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

#Ajouter 100 a tous les elements
matrix + 100
```

```
array([[101, 102, 103],
       [104, 105, 106],
       [107, 108, 109]])
```

8. **Rechercher les valeurs maximales et minimales**

   Trouver la valeur maximale ou minimale d'un tableau

   Utiliser les méthodes max et min de NumPy

```
# Chargement de la bibliotheque
import numpy as np

# Creation d'une matrice
matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# On renvoie le plus grand element
np.max(matrix)
```

```
9
```

```
# On renvoie le plus petit element
np.min(matrix)
```

```
1
```

```
# On trouve l'element maximum dans chaque colonne
np.max(matrix, axis=0)
```

```
array([7, 8, 9])
```

```
# On trouve l'element maximum dans chaque ligne
np.max(matrix, axis=1)
```

```
array([3, 6, 9])
```

9. **Calculer la moyenne, la variance et l'écart type**

   Problème : Calculer des statistiques descriptives sur un tableau.

   Solution : Utiliser les fonctions mean, var et std de NumPy

```
# Calculer la moyenne, la variance et l'ecart type
np.mean(matrix)
```

```
5.0
```

```
np.var(matrix)
```

```
6.666666666666667
```

```
np.std(matrix)
```

```
2.581988897471611
```

Tout comme max et min nous pouvons facilement obtenir des statistiques descriptives sur l'ensemble de la matrice ou effectuer des calculs le long d'un seul axe :

```
np.mean(matrix, axis=0)
```

```
array([4., 5., 6.])
```

10. **Modifier la forme des tableaux :**

On souhaite modifier la forme (nombre de lignes et de colonnes) d'un tableau sans modifier les valeurs des éléments.

Utilisez la fonction reshape de NumPy

```
import numpy as np
matrix = np.array([[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9],
        [10, 11, 12]])
#transformation de la matrice en une matrice 2*6
matrix.reshape(2, 6)
```

```
array([[ 1,  2,  3,  4,  5,  6],
       [ 7,  8,  9, 10, 11, 12]])
```

Reshape permet de restructurer un tableau de manière a conserver les mêmes données, mais en les organisant dans un nombre différent de lignes et colonnes. La seule conditio est que la forme de la matrice originale et celle de la nouvelle matrice contiennent le même nombre d'éléments (c'est à dire qu'elles aient la même taille). La taille d une matrice peut être visualisée a l'aide de la fonction size (matrix.size).

Reshape possède un argument utile : -1, qui signifie en fait " autant que nécessaire ". Ainsi reshape (1, -1) signifie une ligne et autant de colonnes que nécessaire :

```
matrix.reshape(1, -1)
```

```
array([[ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12]])
```

Enfin, si nous fournissons un entier, reshape renverra un tableau unidimentionel de cette longueur :

```
matrix.reshape(12)
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

11. **Transposer un vecteur ou une matrice :**

Transposer un vecteur ou une matrice avec matrix.T

```
matrix.T
```

```
array([[ 1,  4,  7, 10],
       [ 2,  5,  8, 11],
       [ 3,  6,  9, 12]])
```

La transposition est une opération courante dans l'algèbre linéaire qui consiste a intervenir les indices de colonne et de ligne de chaque élément.

```
# Transposition d'un vecteur
np.array([[1, 2, 3, 4, 5, 6]]).T
```

```
array([[1],
       [2],
       [3],
       [4],
       [5],
       [6]])
```

12. **Aplatir une matrice**

Transformer une matrice en un tableau a une seule dimension

Utiliser la méthode flatten

```
# Aplatissement de la matrice
matrix.flatten()
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

Flatten est une methode simple pour transformer une matrice en un tableau unidimensionnel. Nous pouvons utiliser reshape pour créer un vecteur ligne.

La méthode ravel est une autre façon courante d'aplatir les tableaux. Contrairement a la methode flatten, qui renvoie une copie du tableau original, la méthode ravel opère sur l'objet original, ce qui la rend donc légèrement plus rapide. Elle permet également d'aplatir des listes de tableaux, ce que ne permet pas la méthode flatten. Cette opération est utile pour aplatir de très grands tableaux et accélérer le code :

```
import numpy as np
# Creation d'une matrice
matrix_a = np.array([[1, 2], [3, 4]])

# Creation de la deuxieme matrice
matrix_b = np.array([[5, 6], [7, 8]])

# Creation d'une liste de matrices
matrix_list = [matrix_a, matrix_b]

#Aplatissement de la liste complete des matrices
np.ravel(matrix_list)
```

```
array([1, 2, 3, 4, 5, 6, 7, 8])
```

13. **Obtenir la diagonale d'une matrice**

Obtenir les éléments diagonaux d'une matrice.

```
# Creation d'une matrice
matrix = np.array([[1, 2, 3], [2, 4, 6], [3, 8, 9]])

# Retourne les elements diagonaux
matrix.diagonal()
```

```
array([1, 4, 9])
```

14. **Calculer la trace d'une matrice**

    Calculer la trace d'une matrice.

```
# Creation d'une matrice
matrix = np.array([[1, 2, 3], [2, 4, 6], [3, 8, 9]])

# Retourne la trace
matrix.trace()
```

```
14
```

15. **Effectuer le produit scalaire**

    Le produit scalaire de deux vecteurs .

```
# Chargement de la bibliotheque
import numpy as np

# Creation de deux vecteurs
vector_a = np.array([1, 2, 3])
vector_b = np.array([4, 5, 6])

# Calcul du produit scalaire
np.dot(vector_a, vector_b)
```

```
32
```

Le produit scalaire de deux vecteurs, a et b, est défini comme suit :

$$\sum_{i=1}^{n} a_i b_i$$

16. **Additionner et soustraire des matrices**

    Additionner et soustraire deux matrices.
    Utilisez la fonction add et subtract de NumPy

```
# Creation d'une matrice
matrix_a = np.array([[1, 1, 1], [1, 1, 1], [1, 1, 2]])

matrix_b = np.array([[1, 3, 1], [1, 3, 1], [1, 3, 8]])
# Ajout de deux matrices
np.add(matrix_a, matrix_b)
```

```
array([[ 2,  4,  2],
       [ 2,  4,  2],
       [ 2,  4, 10]])
```

```
matrix_a + matrix_b
```

```
array([[ 2,  4,  2],
       [ 2,  4,  2],
       [ 2,  4, 10]])
```

```
# Soustraction d'une matrice
np.subtract(matrix_a, matrix_b)
```

```
array([[ 0, -2,  0],
       [ 0, -2,  0],
       [ 0, -2, -6]])
```

```
matrix_a-matrix_b
```

```
array([[ 0, -2,  0],
       [ 0, -2,  0],
       [ 0, -2, -6]])
```

17. **Multiplier des matrices :**

Pour multiplier les matrices utiliser la fonction dot de NumPy.

```
# Creation d'une matrice
matrix_a=np.array([[1, 1], [1, 2]])

# Creation d'une matrice
matrix_b=np.array([[1, 3], [1, 2]])

# Multiplication de deux matrices
np.dot(matrix_a, matrix_b)
```

```
array([[2, 5],
       [3, 7]])
```

L'operateur * effectue une multiplication au niveau des éléments :

```
matrix_a * matrix_b
```

```
array([[1, 3],
       [1, 4]])
```

18. **Inverser une matrice :**

Calculer l'inverse d'une matrice carrée.

```
# Creation d'une matrice
matrix = np.array([[1, 4], [2, 5]])

# calcul l'inverse de la matrice
np.linalg.inv(matrix)
```

```
array([[-1.66666667,  1.33333333],
       [ 0.66666667, -0.33333333]])
```

L'inverse d'une matrice carrée, A, est une seconde matrice A^{-1}, telle que :

$AA^{-1} = I$

Ou I est la matrice identité.

```
matrix @ np.linalg.inv(matrix)
```

```
array([[1., 0.],
       [0., 1.]])
```

19. **Générer de valeurs aléatoires :**

Générer des valeurs pseudo-aléatoires.

Utiliser la fonction random de NumPy

```
# Chargement de la bibliotheque
import numpy as np

# Initialise le generateur de nombres aleatoires
np.random.seed(0)

# Genere trois valeurs en virgule flottante aleatoires entre 0,0 et 1,0
np.random.random(3)
```

```
array([0.5488135 , 0.71518937, 0.60276338])
```

Dans notre solution, nous avons généré des nombres en virgule flottante, mais il est également courant de générer des nombres entiers :

```
# Generation de trois entiers entre 0 et 10
np.random.randint(0, 11, 3)
```

```
array([3, 5, 2])
```