Numpy

Hadrien Bodin

Mines

9 septembre 2025

Table des matières

Introduction

- Numpy est une bibliothèque python
- Est optimisé pour le traitement de matrices numériques

Example

Quels types de données peut on stocker sur une matrice ?

Données matricielles

- Une matrice ou un vecteur numérique classique
- Une image
- Une série temporelle
- Un enregistrement sonore
- etc...

Definition

Toutes ces données peuvent être stockées dans un tableau multidimensionnel. On l'appellera un array.

Les array

```
import numpy as np

A = np.array()
type(A)
    -> <class 'numpy.ndarray'>
```

Les types d'éléments

Un array stocke des éléments qui partagent tous le même type. Ce type peut être imposé ou non par l'utilisateur

```
A = np.array([1,2])
A.dtype
    -> dtype('int64')

B = np.array([1,2], dtype = np.uint8)
B.dtype
    -> dtype('uint8')
```

Question

Quelle valeurs peut on stocker dans un array de dtype uint8 ?

Question

Que se passe-t-il si on veut stocker une valeur supérieure au maximum ?

Les types d'éléments

```
A = np.array([-1, 2, 3, 255, 256], dtype=np.uint8)
Traceback (most recent call last):
  File "<console>", line 1, in <module>
OverflowError: Python integer -1 out of bounds for uint8
```

```
B = np.array([-2, -1, 0, 2, 3, 255, 256, 257])
B = B.astype(np.uint8)
B
    -> array([254, 255,   0,   2,   3, 255,   0,   1], dtype=uint8)
```

```
C = np.array([1,2,3], dtype=np.uint8)
C*100
    -> array([100, 200,  44], dtype=uint8)
```

Les types d'éléments

- Le cast ne se fait pas automatiquement.
- Une fois un tableau créé, le dtype n'est pas modifié
- On évitera donc de spécifier le type d'élément, et on laissera numpy gérer (sauf besoin précis)

Taille mémoire d'un tableau

Question

Comment avoir une idée de la taille en mémoire d'un tableau array ?

Taille mémoire d'un tableau

- Taille mémoire = nombre d'éléments x octets par élément
- Attention ne pas confondre bits et octets

```
A = np.array([1,2,3])
A.dtype
    -> dtype('int64')
print(A.itemsize)
    -> 8
print(A.size)
    -> 3
print(A.nbytes)
    -> 24
```

Les autres constructeurs

- numpy.zeros
- numpy.ones
- numpy.empty
- numpy.arange
- numpy.linspace
- numpy.random.randint
- numpy.random.randn

Exercice

Exercice

- Créer un tableau contenant des 1, stockés sur des entiers 8 bits non signés, avec 3 lignes et 5 colonnes
- Créer un tableau de la même dimension, comprenant des entiers tirés aléatoirement entre le min et le max possible pour ce type

Mémoire

```
help(np.ndarray)
    -> An array object represents a multidimensional,
    homogeneous array of fixed-size items.
```

Mémoire

homogène

- toutes les cases du tableau ont le même type
- donc elles occupent la même taille en mémoire

Mémoire

taille fixe

- une fois un tableau créé, on ne peut plus modifier la taille de ses éléments i.e. le nombre d'octets sur lequel chaque élément est stocké en mémoire est fixe
- si on manipule et que la taille des éléments ne suffit plus, numpy convertit la valeur mais ne modifie pas la taille de ses éléments
- pour modifier la taille des éléments on n'a pas le choix, il faut allouer un nouveau tableau, et recopier l'ancien dedans (et c'est à éviter...)

Mémoire

pourquoi ces contraintes ?

- pour que numpy soit le plus rapide possible dans ses manipulations de tableaux
- grâce à ces contraintes, passer d'une case du tableau à une autre est très rapide

Mémoire

Il n'y a pas d'indirection mémoire
Les blocs mémoire d'un même tableau sont contigus.
Chaque case mémoire contient donc la valeur que l'on cherche en allant la visiter

Cela s'oppose fortement à la liste en python qui contient les adresses mémoires des différents éléments

Mémoire

```
l = ['un', 'deux', 'trois', 'cinq']
# creer un tableau a partir de l et l'afficher
tab[0] = 'quatre'
#afficher tab
```

Exercice

Pourquoi ce résultat ?

Mémoire

Pour toutes ces raisons, ne pas mélanger les types d'objets

```
l = [127, 128, 17.4, np.pi, True, False, 'test']
a = np.array(l)
a
      -> array(['127', '128', '17.4', '3.141592653589793',
       'True', 'False', 'test'],
        dtype='<U32')
a[0] + 1
      -> Traceback (most recent call last):
TypeError: can only concatenate str (not "int") to str
```

Indexation

- Notion de shape d'un tableau (équivalent lignes, colonnes, épaisseur, ...)
- Equivalent à une liste de liste de liste de liste...

```
a = np.array([[1,2],[3,4]])
a[0]
    -> array([1,  2])
a[0][1]
    -> np.int64(2)
```

Indexation

Possibilité de modifier la forme d'un tableau

- np.reshape renvoit un tableau
- np.resize agit inplace

si aucune mémoire n'est créée, c'est que les différentes indexations prises sur un tableau partagent l'objet sous-jacent

Indexation

Exercice

- créez un tableau tab de 6 ones de forme (6) et affichez-le
- mettez dans tab1 le reshape de tab avec la forme (3, 2) et affichez-le
- modifiez le premier élément de tab
- affichez tab1

Indexation

Les deux objets sont des objets différents (leurs index sont différents) mais ils ont le même segment sous-jacent de données. Toucher l'un a pour effet de modifier l'autre.

Vectorisation

Le but de la vectorisation est de faire une opération sur tous les éléments d'un tableau sans utiliser de boucle for

La vectorisation est la seule manière d'écrire du code en numpy pour avoir des temps d'exécution acceptables

Exercice

Vérifier cela en utilisant %timeit

Il faut donc utiliser des ufunc

Slicing

Il faut retenir que :

`tab[i,j,k,l]` == `tab[i][j][k][l]`

Revenir à une vision de liste de liste de liste de liste... permet donc d'avoir l'intuition du slicing
Cependant numpy permet plus de choses que ces simples opérations

Slicing

- créez un tableau des 30 valeurs paires à partir de 2
- donnez lui la forme de 2 matrices de 5 lignes et 3 colonnes
- accédez à l'élément qui est à la 3ème colonne de la 2ème ligne de la 1ère matrice
- obtenez-vous 12 ?

Slicing

Le slicing numpy est syntaxiquement équivalent à celui des listes Python
La grande différence est que

- quand vous slicez un tableau numpy vous obtenez une vue sur le tableau initial
- quand vous slicez une liste python vous obtenez une copie de la liste initiale

Le slicing numpy va

- regrouper des éléments du tableau initial dans un sous-tableau numpy.ndarray avec l'indexation adéquate
- la mémoire sous-jacente reste la même la seule structure informatique qui sera créée est l'indexation
- vous pourrez ensuite, par exemple, modifier ces éléments et donc ils seront modifiés dans le tableau initial

Slicing

Rappel du slicing Python

`tab[from:to-excluded:step]`

- paramètres tous optionnels
- par défaut : from = 0 to-excluded = len(l)et step=1
- indices négatifs ok -1 est le dernier élément, -2 l'avant dernier...

Slicing

```
tab = np.arange(120).reshape(2, 3, 4, 5)
```

Exercice

Extrayez du tableau tab précédent la sous-matrice au milieu (i.e. garder deux lignes et 3 colonnes, au centre) des premières matrices de tous les groupes
[[6 7 8
,[11 12 13]],[[66 67 68],[71 72 73]]]

Slicing

Le slicing calcule une nouvelle indexation sur le segment mémoire du tableau existant
Si à chaque slicing, numpy faisait une copie du tableau sous-jacent, les codes seraient inutilisables parce que coûteux (pénalisés) en place mémoire

Donc lors d'un slicing un nouvel objet np.ndarray est bien créé, son indexation est différente de celle de l'objet np.ndarray initial mais ils partagent la mémoire (le segment unidimensionnel sous-jacent)
si un utilisateur veut une copie, il la fait avec la méthode copy

Slicing

Un tableau numpy.ndarray peut être

- un tableau original (on vient de le créer éventuellement par copie)
- une vue sur un tableau (il a été créé par slicing ou indexation)

Il partage son segment de mémoire avec au moins un autre tableau.
l'attribut numpy.ndarray.base vaut alors

- None si le tableau est un tableau original
- le tableau original qui a servi à créer la vue quand le tableau est une vue

Slicing

Exercice

- Créez un nouveau tableau formé des deux matrice [[[2 4 6] [8 10 12]]
  [[14 16 18] [20 22 24]]]
- affichez sa base
- slicez le tableau pour obtenir [[[24 22 20] [18 16 14]]
  [[12 10 8] [6 4 2]]]
- vérifier que les deux bases sont le même objet

Slicing

Utilisation de np.indices

Exercice

- créer des rayures
- créer un damier
- créer un damier par bloc
- créer un escalier

Images

- dans une image en couleur, les pixels sont représentés par leurs dosages dans les 3 couleurs primaires : red, green, blue (RGB)
- si le pixel vaut (r, g, b) = (255, 0, 0), il ne contient que de l'information rouge, il est affiché comme du rouge
- l'affichage à l'écran, d'une image couleur rgb, utilise les règles de la synthèse additive
- (r, g, b) = (255, 255, 255) donne la couleur blanche
- (r, g, b) = (0, 0, 0) donne la couleur noire
- (r, g, b) = (255, 255, 0) donne la couleur jaune ...
- pour afficher le tableau im comme une image, utilisez : plt.imshow(im)
- pour afficher plusieurs images dans une même cellule de notebook faire plt.show() après chaque plt.imshow(...)

Exercice

- Créez un tableau de 91 pixels de côté, d'entiers non-signés 8 bits et affichez-le. Le tableau n'est pas forcément initialisé à ce stade et il vous faut pouvoir stocker 3 uint8 par pixel pour ranger les 3 couleurs.
- Transformez le en tableau blanc (en un seul slicing) et affichez-le
- Transformez le en tableau vert (en un seul slicing) et affichez-le
- Affichez les valeurs RGB du premier pixel de l'image, et du dernier
- Faites un quadrillage d'une ligne bleue, toutes les 10 lignes et colonnes et affichez-le

Images

Exercice

- Avec la fonction plt.imread lisez le fichier data/les-mines.jpg ou toute autre image - faites juste attention à la taille
- Vérifiez si l'objet est modifiable avec im.flags.writeable si il ne l'est pas copiez-le
- Affichez l'image
- Quel est le type de l'objet créé ?
- Quelle est la dimension de l'image ?
- Quelle est la taille de l'image en hauteur et largeur ?
- Quel est le nombre d'octets utilisé par pixel ?
- Quel est le type des pixels ? (deux types pour les pixels : entiers non-signés 8 bits ou flottants sur 64 bits)
- Quelles sont ses valeurs maximale et minimale des pixels ?
- Affichez le rectangle de 10 x 10 pixels en haut de l'image

Image slicings

Exercice

- Relire l'image
- Slicer et afficher l'image en ne gardant qu'une ligne et qu'une colonne sur 2, 5, 10 et 20
- Isoler le rectangle de l lignes et c colonnes en milieu d'image affichez-le pour (l, c) = (10, 20)) puis (l, c) = (100, 200))
- Affichez le dernier pixel de l'image

Image canaux

Exercice

- Relire l'image
- Slicer et afficher l'image en ne gardant qu'une ligne et qu'une colonne sur 2, 5, 10 et 20
- Isoler le rectangle de l lignes et c colonnes en milieu d'image affichez-le pour (l, c) = (10, 20)) puis (l, c) = (100, 200))
- Affichez le dernier pixel de l'image

Agrégation

Il peut être utile de réduire la dimension d'un tableau, en appliquant une fonction selon un axe :

- Faire la moyenne des 3 intensités R,G,B
- Faire la somme des valeurs de rouge dans l'image
- ....

Faire ceci s'appelle faire de l'agrégation
Le tableau retourné aura en général une taille inférieure au tableau d'entrée.

Agrégation

```
tab = np.random.randint(0, 2, size=(10), dtype=bool)
print(np.all(tab), np.any(tab))
print(tab.all(), tab.any())
```

Il n'y a pas besoin de préciser l'axe ici car il n'y a qu'une dimension. On garde donc axis = 0.

Exercice

Créez une fonction manuelle (sans utiliser np.all et np.any) qui prend un tableau numpy de booléens en paramètre et détermine si tous les éléments du tableau sont vrais.
Créez une fonction manuelle (sans utiliser np.all et np.any) qui prend un tableau numpy de booléens en paramètre et détermine si tous les éléments du tableau sont faux.

Agrégation

En dimension plus grande que 1, par défaut, l'agrégation sera appliquée à tous les axes et la fonction ne renverra qu'une valeur

```
tab = np.arange(120).reshape(2, 3, 4, 5)
tab.sum()
-> 7140
tab.sum(axis=0).shape # on rend la forme obtenue
-> (3, 4, 5)
tab.sum(axis=1).shape
-> (2, 4, 5)
tab.sum(axis=2).shape
-> (2, 3, 5)
tab.sum(axis=3).shape
-> (2, 3, 4)
```

Agrégation

Fonction argmax Il peut être pertinent de retrouver la position d'un max dans le tableau

```
tab = np.arange(120).reshape(2, 3, 4, 5)
```

Exercice

Expliquer le retour de la fonction argmax appelée sur les différents axes

Agrégation

Quand argmax est appelée sans axe, la position du maximum dans un tableau applati est renvoyée. La fonction numpy.unravel_index re-calcule les coordonnées à partir de l'indice absolu et de la forme du tableau

```
tab = np.arange(120).reshape(2, 3, 4, 5)
np.unravel_index(tab.argmax(), tab.shape)
-> (np.int64(1), np.int64(2), np.int64(3), np.int64(4))
```

Exercice

Proposer une implémentation de unravel index