Introduction à `numpy`

INF8460 - TP2

Polytechnique Montréal

Automne 2020

numpy

```
import numpy as np
```

Présentation

Numpy : une librairie pour la manipulation efficace de listes et de matrices

Liens utiles :

- **Numpy Quickstart Tutorial** : lire notamment les sections
  - Array Creation pour la création d'arrays et de matrices
  - Basic Operations pour les opérations matricielles de base
  - Universal Functions pour l'application de fonctions mathématiques (somme, log, exponentielle) sur des matrices
  - Functions and Methods Overview qui réunit les méthodes les plus utiles, toutes catégories confondues
- documentation complète de Numpy : numpy.org/devdocs/reference/index.html

L'array Numpy

Un `array` est l'objet de base de Numpy. C'est un tableau multidimensionnel dont tous les éléments ont le même type. Il peut représenter un vecteur (1D), une matrice (2D) ou un tenseur (3D ou plus).

```
>>> a = np.array([[0.5, 1.0], [1.0, 0.5]])
>>> print(a)
[[0.5 1. ]
 [1.  0.5]]
```

Il a un attribut `shape` qui contient ses dimensions et un attribut `dtype` pour le type de ses éléments.

```
>>> a.shape
(2, 2)
>>> a.dtype
dtype('float64')
```

Création d'un array

Créer un array à partir d'une liste :

```
>>> a = np.array([1, 2, 3])
```

Une liste de liste donnera une matrice :

```
>>> b = np.array([[1., 0.],[0., 1.]])
>>> b.shape
(2, 2)
>>> print(b)
[[1. 0.]
 [0. 1.]]
```

Création d'un array (suite)

Initialiser un array avec des zéros ou des uns :

```
# On précise les dimensions de l'array
>>> c = np.zeros((3, 2))
# On peut aussi spécifier un type
>>> d = np.ones((2, 2), dtype="int32")
>>> print(c)
[[0. 0.]
 [0. 0.]
 [0. 0.]]
>>> print(d)
[[1 1 1]
 [1 1 1]]
```

On peut initialiser un array vide avec `np.empty(shape)`, ou un array rempli d'une constante avec `np.full(shape, fill_value)`.

Manipulations basiques

La plupart des opérations se font terme à terme :

```
# Ajouter ou soustraire une constante
>>> b + 1
array([[2., 1.],
       [1., 2.]])
# Multiplier ou diviser par une constante
>>> 2 * b / 3
array([[0.66666667, 0.        ],
       [0.        , 0.66666667]])
# Puissance
>>> (b + 1)**3
array([[8., 1.],
       [1., 8.]])
```

Indexation

On peut accéder à un élément, à une ligne ou à une colonne d'une matrice :

```
>>> X = np.array([[1, 2, 3], [4, 5, 6]])
>>> print(X)
[[1 2 3]
 [4 5 6]]
# Accéder à un élément
>>> X[0, 0], X[-1, -1], X[0, 1]
(1, 6, 2)
# Accéder à la ligne i=1
>>> X[1, :]
array([4, 5, 6])
# Accéder à la colonne j=1
>>> X[:, 1]
array([2, 5])
```

Opérations matricielles : addition

Addition de matrices de mêmes dimensions:

```
>>> A = np.array([[1, 2],[2, 3], [3, 4]]) # 3x2
>>> B = np.array([[0, 0],[1, -1], [0.5, 2]])
>>> A + B
array([[1.  , 2.  ],
       [3.  , 2.  ],
       [3.5, 6.  ]])
```

Addition ligne par ligne : une matrice (m, n) et une matrice-ligne de dimensions $(1, n)$ ou $(n,)$:

```
>>> A + np.array([1, 10])
array([[ 2, 12],
       [ 3, 13],
       [ 4, 14]])
```

Opérations matricielles : addition (suite)

Addition colonne par colonne : une matrice (m, n) et une matrice-colonne de dimensions $(m, 1)$:

```
>>> A + np.array([[1], [0], [-1]])
array([[2, 3],
       [2, 3],
       [2, 3]])
```

Même principe pour la soustraction (−), la division terme à terme (/) ou la multiplication terme à terme (∗).

Opérations matricielles : produit

Produit matriciel avec @ (attention aux dimensions !) :

```
>>> C = np.array([[0.5],[2]]) # 2x1
>>> A @ C
array([[4.5],
       [7. ],
       [9.5]])
```

Fonctionne aussi pour la multiplication matrice-vecteur :

```
>>> A @ np.array([2, 1])
array([ 4,  7, 10])
```

Et, comme attendu, le produit vecteur-vecteur donne le produit scalaire.

Opérations matricielles : produit

le produit dyadique entre deux vecteurs avec `np.outer` (attention aux dimensions !) :

```
>>> M = np.outer(array1, array2)
```

Qui calcule une matrice de la façon suivante :

$$\mathbf{u} \otimes \mathbf{v} = \mathbf{u}\mathbf{v}^{\mathsf{T}} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix} = \begin{bmatrix} u_1 v_1 & u_1 v_2 & u_1 v_3 \\ u_2 v_1 & u_2 v_2 & u_2 v_3 \\ u_3 v_1 & u_3 v_2 & u_3 v_3 \\ u_4 v_1 & u_4 v_2 & u_4 v_3 \end{bmatrix}$$

(Tiré de la page Wikipédia Outer Product)

Opérations matricielles : transposée, norme

Transposée avec `array.T` :

```
>>> A.T
array([[1, 2, 3],
       [2, 3, 4]])
```

Norme d'un vecteur :

```
>>> u = np.array([0, 1, -2])
>>> np.linalg.norm(u) # Par défaut, norme L2
2.23606797749979
>>> np.linalg.norm(u, 1) # Norme L1
3.0
```

Même principe pour la norme d'une matrice.

Opérations matricielles : somme

La fonction `np.sum` peut s'appliquer soit sur l'array entier, soit sur une seule dimension (ligne ou colonne).

Array entier :

```
>>> np.sum(A)
15
```

Somme par colonne :

```
>>> np.sum(A, axis=0)
array([6, 9])
```

Somme par ligne :

```
>>> np.sum(A, axis=1)
array([3, 5, 7])
```

Fonctions mathématiques

Numpy fournit de nombreuses fonctions mathématiques, qui s'appliquent à chaque élément de l'array : `log`, `exp`, `sin` et `cos`, `sqrt`, `abs`, `ceil`, etc.

```
# Logarithme (voir aussi log2, log10)
>>> np.log([0, 1, np.e, np.e**2, 100])
array([-inf, 0. , 1., 2., 4.60517019])
# Et on peut tout combiner
>>> np.exp(np.sin(1/(1+ (A @ A.T))))
array([[1.18045049, 1.11726376, 1.08679926],
       [1.11726376, 1.07397621, 1.05401563],
       [1.08679926, 1.05401563, 1.0392009 ]])
```

Il existe aussi des fonctions comme np.isinf(n) ou np.isnan(n) qui indique respectivement si n est infinie ou si c'est un Not a Number (dans le cas de calcul comme 0/0, inf/inf . . .)
Voir *Mathematical Functions* dans la doc Numpy pour une liste exhaustive.

Manipulation de dimensions

Passage d'un vecteur de dimension $(n,)$ à une matrice-ligne $(1, n)$ ou à une matrice-colonne $(n, 1)$:

```
>>> u = np.array([0, 1, 2])
>>> u.shape
(3,)
>>> v = u.reshape((1, -1)) # --> matrice ligne
>>> w = u.reshape((-1, 1)) # --> matrice colonne
>>> print(v); print(w)
[[0 1 2]]
[[0]
 [1]
 [2]]
>>> print(v.shape, w.shape)
(1, 3) (3, 1)
```

plus d'explications ici sur l'utilisation de `-1` avec `reshape`