INF8007 – Languages de scripts
Numpy

Antoine Lefebvre-Brossard

Hiver 2018

Contexte

- La `list` en Python est très puissante, mais est difficile à utilser lorsqu'on a des données plus facile représentables sur plus d'une dimension, par exemple les pixels d'une image, ou la grille d'un jeu
- **Exemple :** Pour déterminer la moyenne d'une colonne i dans une matrice

```
mean = sum([x[i] for x in matrix]) / len(matrix[i])
```

Contexte (suite)

- Pourtant, Python est un des languages les plus utilisés en science des données et en apprentissage-machine
- La raison principale pour cela est **numpy** (num*erical*-py*thon*)

Ce que c'est

- Librairie pour faire des calculs numériques en Python
- La majorité de l'éxécution se fait en C, rendant les opérations très rapides
- Possède une quantité toujours plus importante de fonctions d'algèbre linéaire et de calcul numérique en général
- Le projet est est fait en étroite collaboration avec **scipy** (sci*entific*-py*thon*) qui possède une énorme quantité de fonctions scientifiques

Bases

- La façon typique d'importer numpy est par

```
import numpy as np
```

- Un array (liste à plusieurs dimensions) est créé en transformant une `list` imbriquée

```
matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
matrix
>>> array([[1, 2, 3],
>>>        [4, 5, 6],
>>>        [7, 8, 9]])
```

- La moyenne d'une colonne i se fait maintenant simplement par

```
matrix.mean(1)[i]
```

Indexation

- À la place de devoir indexer une `list` imbriquée par
  `list_of_list[i][j]`

  un array peut être indexé par
  `matrix[i, j]`
- Des intervalles peuvent être choisis de la même façon qu'une liste
  `matrix[a:b, j]`
- Mais en plus, on peut utiliser une liste pour indexer
  `matrix[[1, 5, 1]]`

Exemples

Avec la matrice

```
m = np.array([[1, 2, 3, 4],
              [5, 6, 7, 8],
              [9, 10, 11, 12],
              [13, 14, 15, 16]])
```

que va retourner `m[1, 2]` ?

Réponse :

```
7
```

Exemples (suite)

Avec la matrice

```
m = np.array([[1, 2, 3, 4],
              [5, 6, 7, 8],
              [9, 10, 11, 12],
              [13, 14, 15, 16]])
```

que va retourner `m[1:3]`, `m[1:3, 1]`, `m[1:3,1:3]` et `m[[1,2],[1,2]]` ?

Réponse :

```
array([[5, 6, 7, 8],
       [9, 10, 11, 12]])

array([6, 10])

array([[6, 7],
       [10, 11]])

array([6, 11])
```

Exemples (suite)

Avec le vecteur `l = np.array([0, 1, 0, 2, 1, 3])`
et la fonction `np.zeros([height, width])`,
comment créer une matrice de vecteurs *one-hot* ?

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

Réponse :

```
one_hot = np.zeros([l.shape[0], l.max() + 1])
one_hot[np.arange(l.shape[0]), l] = 1
```

Algèbre linéaire

- La vraie force de numpy provient de ses opérations optimisées d'algèbre linéaire
- La multiplication vectorielle se fait par `np.dot(v1, v2)`
- La même opération peut être utilisé avec des matrices. Avec $A \in \mathbb{R}^{p \times k}$, $B \in \mathbb{R}^{k \times l}$, `C = np.dot(A, B)` va donner $C \in \mathbb{R}^{p \times l}$

Broadcasting

- Numpy supporte aussi ce qu'on appelle broadcasting
- Lorsqu'on effectue une opération par élément (*elementwise*) avec deux array de taille différente, numpy essaiera de déterminer si l'un d'eux peut être répété
- **Exemples :**

```
np.array([[1, 2], [3, 4]) + 1
>>> array([[2, 3],
           [4, 5]])
np.array([[1, 2], [3, 4]]) + np.array([1, 2])
>>> array([[2, 4],
           [4, 6]])
np.array([[1, 2, 3], [4, 5, 6]]) + np.array([1, 2])
>>> ValueError: operands could not be broadcast together
>>> with shapes (2,3) (2,)
```

Broadcasting (suite)

- En général, à moins que la façon de faire le broadcast est évidente, il est plus sécuritaire d'avoir une dimension de taille 1 sur la dimension sur laquelle on veut répéter
- **Exemples :**

```
np.array([[1, 2, 3], [4, 5, 6]]) + np.array([1, 2]).reshape(2, 1)
>>> array([[2, 3, 4],
           [6, 7, 8])
np.random.rand(2, 3, 4) * np.array([1, 2, 3]).reshape(1, 3, 1)
```

Exemples

Va-t-il y avoir une erreur avec ces fonctions ?

```
np.random.rand(2) * 3 + 5
```

Ok

```
np.random.rand(2) * np.array([3]) + np.array([5])
```

Ok

```
np.random.rand(3, 2) * np.array([3]) + np.array([5])
```

Ok

```
np.random.rand(3, 2) * np.array([3, 4]) + np.array([5, 6])
```

Ok

```
np.random.rand(2, 3) * np.array([3, 4]) + np.array([5, 6])
```

Error

Exemples (suite)

Va-t-il y avoir une erreur avec ces fonctions utilisant la matrice `m = np.random.rand(5, 8)` ?

```
m + m
```

Ok

```
m * m.T
```

Error

```
m * m[0]
```

Ok

```
m * m[:,0]
```

Error

```
m * m.sum(0)
```

Ok

```
m * m.sum(1).reshape(-1, 1)
```

Ok (peut être écrit `m.sum(1, keepdims=True)`)