LẬP TRÌNH PYTHON
NumPy & Pandas

NGUYỄN HẢI TRIỀU[1]

[1]Bộ môn Kỹ thuật phần mềm,

Khoa Công nghệ thông tin, Trường ĐH Nha Trang

NhaTrang, February 2022

Nội dung

Numerical Operations on Numpy Arrays

Các toán tử thông thường cũng được nạp chồng trong NumPy, vậy nên chúng ta có thể sử dụng các toán tử một cách bình thường. Các phép tính với ndarray được hỗ trợ trên numpy. Ví dụ để cộng (+), trừ (-), nhân (*), chia (/) trên Numpy

```
1  import numpy as np
2  v = np.array([2,3, 7.9, 3.3, 6.9, 0.11, 10.3, 12.9])
3  v_add = v + 2 #[ 4.    5.    9.9   5.3   8.9   2.11 12.3  14.9 ]
4  v_sub = v - 2 #[ 0.    1.    5.9   1.3   4.9  -1.89  8.3  10.9 ]
5  v_mul = v * 2 #[ 4.    6.   15.8   6.6  13.8   0.22 20.6  25.8 ]
6  v_div = v / 2 #[1.    1.5   3.95  1.65  3.45  0.055 5.15  6.45 ]
7  ######
8  p=np.array([[1,2],[3,4]])
9  q=np.array([[2,2],[1,4]])
10 print(p+q) #[[3 4] [4 8]]
11 print(p*q) #[[ 2  4] [ 3 16]]
12 print(p.transpose()) #[[1 3]  [2 4]]
```

Lưu ý: đừng nhầm lần $p * q$ (thường được gọi là element-wise) là phép nhân hai ma trận.

Nhân hai ndarray: **dot Product**

dot Product

Trong không gian 2 chiều, *dot Product* chính là nhân hai ma trận: $np.dot(A, B)$. Ví dụ:

```
1  import numpy as np
2  x = np.array([3, -2])
3  y = np.array([-4, 1])
4  print(np.dot(x, y))#-14 trong 1D nhu nhan vo huong 2 vector
5  A = np.array([ [1, 2, 3], [3, 2, 1] ])
6  B = np.array([ [2, 3, 4, -2],
7                 [1, -1, 2, 3],
8                 [1, 2, 3, 0] ])
9  print(np.dot(A,B))
10 #[[ 7  7 17  4]
11 # [ 9  9 19  0]] ket qua tra ra la 1 ndarray
```

Hỏi *dot Product* của 2 ndarray trong không gian 3 chiều sẽ như thế nào? Ví dụ: A.shape=(4, 2, 3); B.shape=(2, 3, 5); R=np.dot(A,B) thì R.shape=?

Xét ví dụ trong không gian ba chiều

```
1  import numpy as np
2  X = np.array( [[[3, 1, 2],
3                  [4, 2, 2],
4                  [2, 4, 1]], [[3, 2, 2],
5                               [4, 4, 3],
6                               [4, 1, 1]], [[2, 2, 1],
7                                            [3, 1, 3],
8                                            [3, 2, 3]]])
9  Y = np.array( [[[2, 3, 1],
10                 [2, 2, 4],
11                 [3, 4, 4]], [[1, 4, 1],
12                              [4, 1, 2],
13                              [4, 1, 2]], [[1, 2, 3],
14                                           [4, 1, 1],
15                                           [3, 1, 4]]])
16 R = np.dot(X, Y)
17 print("X.shape: ", X.shape, "   X.ndim: ", X.ndim)
18 print("Y.shape: ", Y.shape, "   Y.ndim: ", Y.ndim)
19 print("R.shape: ",      R.shape, "R.ndim: ", R.ndim)
20 #X.shape:  (3, 3, 3)     X.ndim:  3
21 #Y.shape:  (3, 3, 3)     Y.ndim:  3
22 #R.shape:  (3, 3, 3, 3) R.ndim:  4
```

Để hiểu rõ bản chất vấn đề, ta có thể tính R[0] như sau:

```
i = 0
for j in range(X.shape[1]):
    for k in range(Y.shape[0]):
        for m in range(Y.shape[2]):
            fmt = "    sum(X[{}, {}, :] * Y[{}, :, {}] :  {}"
            arguments = (i, j, k, m, sum(X[i, j, :] * Y[k, :, m])
                )
            print(fmt.format(*arguments))
    sum(X[0, 0, :] * Y[0, :, 0] :  14
    sum(X[0, 0, :] * Y[0, :, 1] :  19
    sum(X[0, 0, :] * Y[0, :, 2] :  15
    sum(X[0, 0, :] * Y[1, :, 0] :  15
    sum(X[0, 0, :] * Y[1, :, 1] :  15
    sum(X[0, 0, :] * Y[1, :, 2] :  9
    sum(X[0, 0, :] * Y[2, :, 0] :  13
    sum(X[0, 0, :] * Y[2, :, 1] :  9
    sum(X[0, 0, :] * Y[2, :, 2] :  18
    sum(X[0, 1, :] * Y[0, :, 0] :  18
    sum(X[0, 1, :] * Y[0, :, 1] :  24
    sum(X[0, 1, :] * Y[0, :, 2] :  20
    sum(X[0, 1, :] * Y[1, :, 0] :  20
    sum(X[0, 1, :] * Y[1, :, 1] :  20
    sum(X[0, 1, :] * Y[1, :, 2] :  12
    sum(X[0, 1, :] * Y[2, :, 0] :  18
```

Phương thức np.mat()

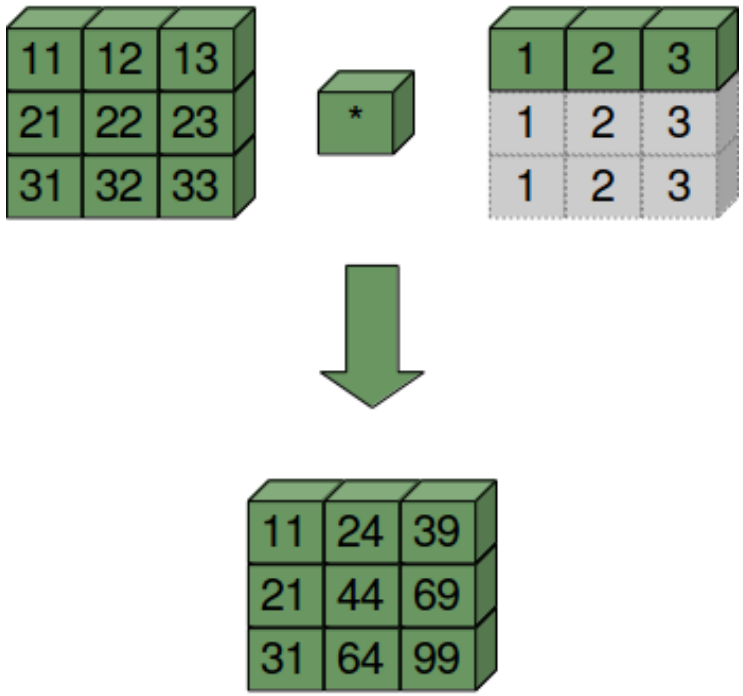
np.mat()

Để tính tích của 2 ma trận, thay vì sử dụng phương thức *np.dot()* ta có thể sử dụng phương thức *np.mat()* sau đó sử dụng toán tử nhân như bình thường. Ví dụ

```
1 import numpy as np
2 A = np.array([ [1, 2, 3], [3, 2, 1] ])
3 B = np.array([ [2, 3, 4, -2],
4                [1, -1, 2, 3],
5                [1, 2, 3, 0] ])
6 print(np.mat(A)*np.mat(B))
```

Broadcasting

Broadcasting là một kĩ thuật cho phép Numpy làm việc với các ndarray có shape khác nhau khi thực hiện các phép tính số học. Cụ thể, mảng nhỏ hơn được biến đổi lặp lại nhiều lần để nó có cùng shape với mảng lớn hơn.



Broadcasting

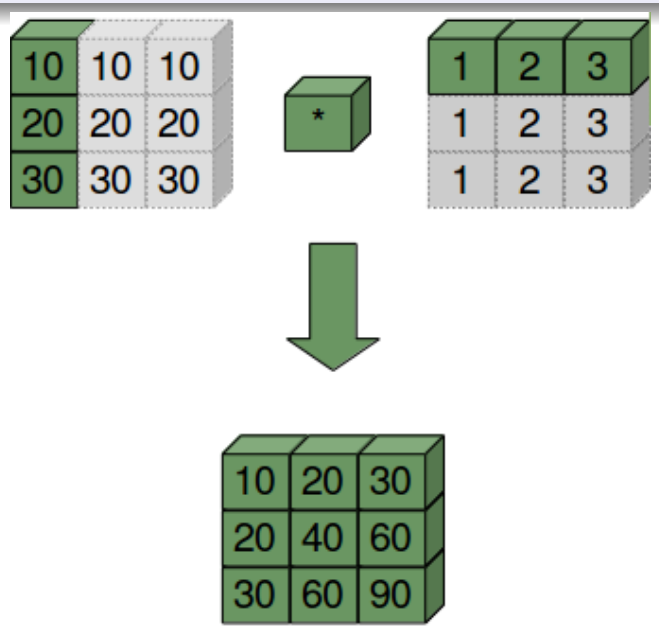
Ví dụ 2.1 (về Broadcasting)

```
1  import numpy as np
2  A = np.array([ [11, 12, 13], [21, 22, 23], [31, 32, 33] ])
3  B = np.array([1, 2, 3])
4  print("Multiplication with broadcasting: ")
5  print(A * B)
6  print("... and now addition with broadcasting: ")
7  print(A + B)
8  #Multiplication with broadcasting:
9  [[11 24 39]
10  [21 44 69]
11  [31 64 99]]
12 ... and now addition with broadcasting:
13 [[12 14 16]
14  [22 24 26]
15  [32 34 36]]
```

Broadcasting

Ví dụ 2.2 (về Broadcasting)

```
1 import numpy as np
2 A = np.array([10, 20, 30])
3 B = np.array([1, 2, 3])
4 A[:, np.newaxis] # np.newaxis: Adding New Dimensions
5 print(A[:, np.newaxis]) #turn a row vector into a column vector
6 print(A[:, np.newaxis] * B)
```



Ứng dụng của Broadcasting

Distance Matrix

Trong lĩnh vực toán học, khoa học máy tính và đặc biệt là lý thuyết đồ thị, ma trận khoảng cách là một ndarray 2D, chứa khoảng cách giữa các phần tử trong mảng. Ví dụ:

```
1 cities = ["Barcelona", "Berlin", "Brussels", "Bucharest", "
      Budapest", "Copenhagen", "Dublin", "Hamburg", "Istanbul", "
      Kiev", "London", "Madrid", "Milan", "Moscow", "Munich", "
      Paris", "Prague", "Rome", "Saint Petersburg", "Stockholm", "
      Vienna", "Warsaw"]
2
3 dist2barcelona = [0,  1498, 1063, 1968, 1498, 1758, 1469, 1472, 2
      230, 2391, 1138, 505, 725, 3007, 1055, 833, 1354, 857, 2813,
      2277, 1347, 1862]
4 dists =  np.array(dist2barcelona[:4])
5 print(dists)
6 print(np.abs(dists - dists[:, np.newaxis]))
```

Concatenating Arrays

Chúng ta có thể nối các mảng lại với nhau sử dụng phương thức *np.concatenate()*

```
1 x = np.array([11,22])
2 y = np.array([18,7,6])
3 z = np.array([1,3,5])
4 c = np.concatenate((x,y,z))
5 print(c)#[11 22 18  7  6  1  3  5]
```

Sum

Tính tổng các phần tử trong array theo các chiều khác nhau sử dụng *np.sum()*

```
1 import numpy as np
2 x = np.array([[1,2],[3,4]])
3 print(np.sum(x)) #10
4 print(np.sum(x, axis=0)) # [4 6]
5 print(np.sum(x, axis=1)) # [3 7]
```

Tài liệu tham khảo

Trung tâm tin học , Đại Học KHTN Tp.HCM
Lập trình Python nâng cao. *03/2017*.

Bernd Klein
Data Analysis: Numpy, Matplotlib and Pandas.
bernd.klein@python-course.eu, 2021.

Luciano Ramalho
Fluent Python (2nd Edition). *O'Reilly Media, Inc, 2021*.

Python Software Foundation
`https://docs.python.org/3/tutorial/`