Numpy基本運用

1. 陣列運算：

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

[[1 2 3]

[4 5 6]

[7 8 9]]

b=np.array([[10, 20, 30], [40, 50, 60], [70, 80, 90]])=

[[10 20 30]

[40 50 60]

[70 80 90]]

#相加：a + b =

[[11 22 33]

[44 55 66]

[77 88 99]]

#相減：a - b =

[[ -9 -18 -27]

[-36 -45 -54]

[-63 -72 -81]]

#相除：a / b =

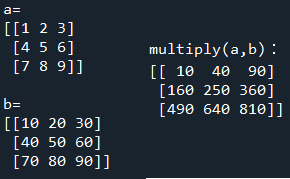
[[0.1 0.1 0.1]

[0.1 0.1 0.1]

[0.1 0.1 0.1]]

#元素乘積(點乘運算：**對應位置的乘積**)----------

#1. 元素乘積：a \* b =

****[[ 10 40 90]

[160 250 360]

[490 640 810]]

#2. 點乘運算：

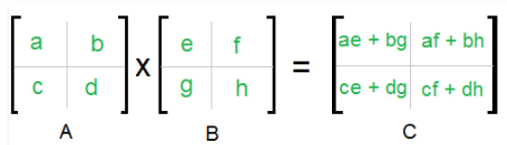
multiply：np.multiply(a,b) =

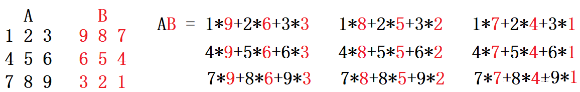
[[ 10 40 90]

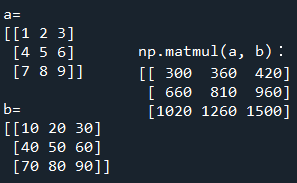
[160 250 360]

[490 640 810]]

#矩陣相乘----------**矩陣乘法，列\*行後相加**





****

**matmul**：Matrix Multiplication

#(1).matmul(a, b)：np.matmul(a, b) =

#(2).a @ b = 同np.matmul(a, b)

#(3).點積：a.dot(b) =

#matmul與 dot 都是矩陣乘法 (Matrix Multiplication)，兩者非常類似，其相同及不同點如下：

(1).如果2個都是二維陣列的話，matmul與 dot 相同。

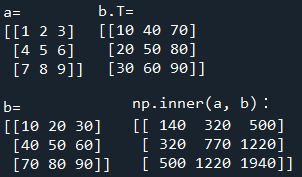
(2).在 matmul 中，多維的矩陣，將前 n-2 維視為後2维的元素後，進行乘法運算。

(3).matmul **不允許矩陣與純量相乘**。

在 Python 3.5版本之後提供@做為矩陣相乘運算子，提供更簡潔的語法。

#矩陣內外積

#內積----------：**將第2矩陣轉置後再矩陣相乘**



np.inner(a, b) = a @ b.T

[[ 140 320 500]

[ 320 770 1220]

[ 500 1220 1940]]

#外積：np.outer(a, b) =

a=[5, 6, 8, 6, 1] b=[6, 4, 8, 1, 8, 5]

|  |  |  |
| --- | --- | --- |
| [ 5  6  a.T= 8  6  1 ] | [6, 4, 8, 1, 8, 5]) = | [[30, 20, 40, 5, 40, 25],  [36, 24, 48, 6, 48, 30],  [48, 32, 64, 8, 64, 40],  [36, 24, 48, 6, 48, 30],  [ 6, 4, 8, 1, 8, 5]] |

#外積：np.outer(a, b) =

1. a轉置 => a.T
2. b的10分別乘於a.T的[1,2,…,9](垂直), 再20,…90分別乘於的[1,2,…,9](垂直)

|  |  |  |
| --- | --- | --- |
| [ [1 4 7]  a.T = [2 5 8]    [3 6 9] ] | [  [10 20 30]    b = [40 50 60] =    [70 80 90]  ] | [[10 20 30 40 50 60 70 80 90]  [20 40 60 80 100 120 140 160 180]  [30 60 90 120 150 180 210 240 270]  [40 80 120 160 200 240 280 320 360]  [50 100 150 200 250 300 350 400 450]  [60 120 180 240 300 360 420 480 540]  [70 140 210 280 350 420 490 560 630]  [80 160 240 320 400 480 560 640 720]  [90 180 270 360 450 540 630 720 810]] |

其他參考：

※二階陣列乘積

|  |  |
| --- | --- |
| 陣列 a：  [[1, 2]  [3, 4]]  陣列 b：  [[11, 12]  [13 ,14]] | **元素乘積**：**(對應位置的乘積)**  **a \* b = np.multiply(a,b)**  [[11 24]  [39 56]]  矩陣乘法(Matrix Multiplication)  **a@b = a.dot(b)**  1\*11+2\*13, 1\*12+2\*14 [ [37 40]  3\*11+4\*13, 3\*12+4\*14 [85 92] ] |
| a = [ [1,0],  [0,1]]  b = [ [4,1],  [2,2]] | 矩陣乘法(Matrix Multiplication)  a.dot(b) = a @ b = np.matmul(a,b)  [[4 1]  [2 2]]  **1\*4+0\*2, 1\*1+0\*2 [ [4, 1]**  **0\*4+1\*2, 0\*1+1\*2 [2, 2]** |

※二階矩陣乘法：

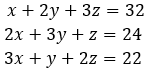
|  |  |
| --- | --- |
| a=np.arange(8).reshape(2,2,2)  b=np.arange(4).reshape(2,2)  print(f'\na={a}') a=[[[0 1]  [2 3]]  [[4 5]  [6 7]]]  print(f'\nb={b}') b=[[0 1]  [2 3]] | 矩陣乘法(Matrix Multiplication)  a.dot(b) = a @ b = np.matmul(a,b)  print(f'\nnp.matmul(a,b)={np.matmul(a,b)}')  [[[ 2 3] #0\*0+1\*2, 0\*1+1\*3 => 2, 3  [ 6 11]] #2\*0+3\*2, 2\*1+3\*3 => 6, 11  [[10 19] #4\*0+5\*2, 4\*1+5\*3 => 10, 19  [14 27]]] #6\*0+7\*2, 6\*1+7\*3 => 14, 27 |

※二元一次方程組求解

|  |
| --- |
| ※linear algebra　/ˈæl.dʒə.brə/  ※二元一次方程組求解：  3 \* x + y = 9 and x + 2 \* y = 8  a = np.array([[3,1], [1,2]])  **>>>** b = np.array([9,8])  **>>>** x = np.linalg.solve(a, b)  **>>>** x  array([ 2., 3.])  Check that the solution is correct:  >>>  **>>>** np.allclose(np.dot(a, x), b)  True |

※三元一次方程組求解：

考慮下麵提到的線性方程組:



可以用矩陣形式表示如下。



上面的解方程為:x=1，y=5，z=7。這可以使用 numpy.linalg.solve() 函數來解決，如下例所示。

A = np.array( [[1, 2, 3],

[2, 3, 1],

[3, 1, 2]])

B = np.array( [32, 24, 22])

sol = np.linalg.solve(A, B)

print(sol) #[ 1. 5. 7.]

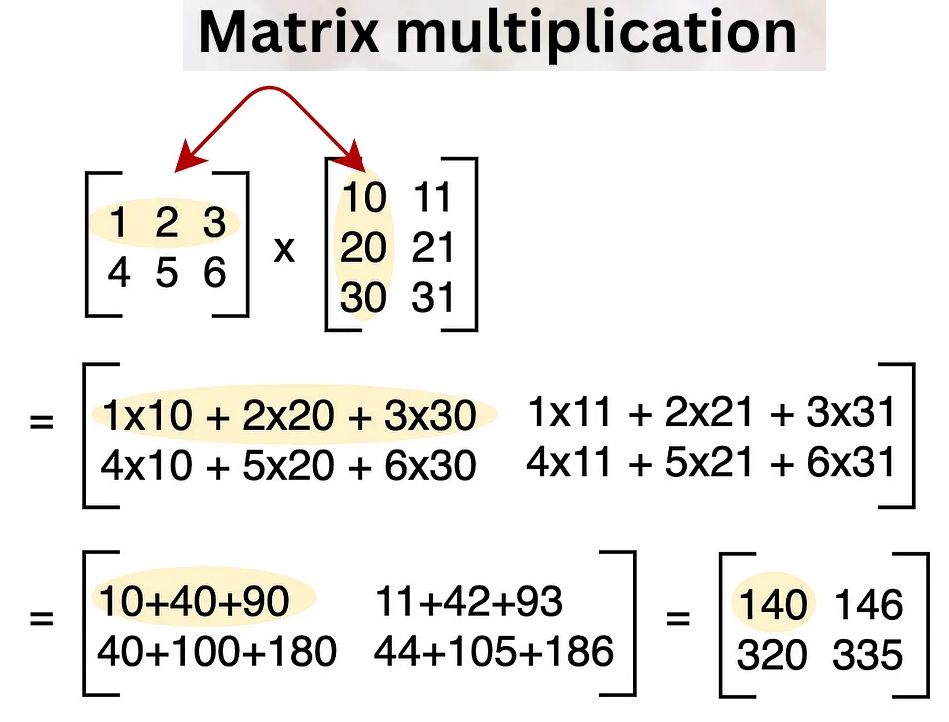
From：https://www.yxjc123.com/post/tmtdl6

※a.dot(b)與 [ a @ b = np.matmul(a, b) ] 是一般的矩陣乘法，dot 在一維和二維矩陣結果與 matmul 相同，三維則不一樣；a @ b = np.matmul(a, b)。

※注意矩陣乘法中前一個矩陣的**列**必須和後一個矩陣的**行**相等

※注意：矩陣積計算不遵循交換律,np.dot(x,y) 和 np.dot(y,x) 得到的結果是不一樣的。

1. **陣列**使用**\***運算子時，其計算方式為對應的位置相乘，當想使用陣列進行矩陣乘法時，可以使用dot實現，
2. **矩陣**使用\*運算子時，其計算方式矩陣相乘，當想使用矩陣進行對應位置相乘時，可以使用multiply實現。可見 dot運算子負責陣列/矩陣的矩陣乘法，multiply負責矩陣/陣列的對應位置相乘。



※a為2x3，b為 3x2 結果為 2x2

當進行向量的內積運算時，可以使用np.dot()

當進行矩陣的乘法運算時，可以使用np.matmul()或者 @

當進行標量的乘法運算時，可以使用np.multiply()或者 \*

numpy.dot()：numpy.dot() #兩陣列的點積

numpy.vdot：numpy.vdot() #兩向量的點積

numpy.inner()：numpy.inner() #兩陣列的內積

numpy.determinant()：numpy.determinant() #陣列的行列式

numpy.matmul()：numpy.matmul() #兩陣列的矩陣積

numpy.inv()：numpy.inv() #求矩陣的逆

numpy.solve()：numpy.solve() #求解線性矩陣方程

From：https://juejin.cn/post/7089770686922620942

1. 陣列取值
2. 一維陣列取值

na = np.arange(0,6)

na=[0 1 2 3 4 5]

na[0]=0

na[5]=5

na[1:5]=[1 2 3 4]

na[1:5:2]=[1 3]

na[5:1:-1]=[5 4 3 2]

na[:]=[0 1 2 3 4 5]

na[:3]=[0 1 2]

na[::3]=[0 3]

na[3:]=[3 4 5]

1. 多維陣列取值

na = np.arange(1, 17).reshape(4, 4)

[[ 1 2 3 4]

[ 5 6 7 8]

[ 9 10 11 12]

[13 14 15 16]]

na[2, 3]=12

na[1, 1:3]=[6 7]

na[1:3, 2]=[ 7 11]

na[1:3, 1:3]=

[[ 6 7]

[10 11]]

na[::2, ::2]=

[[ 1 3]

[ 9 11]]

na[:, 2]=[ 3 7 11 15]

na[1, :]=[ 3 7 11 15]

na[:, :]=

[[ 1 2 3 4]

[ 5 6 7 8]

[ 9 10 11 12]

[13 14 15 16] ]

ary=np.hstack([a,b]) #水平串接

ary = np.vstack((a,b)) #垂直堆叠

ary = np.stack((a,b)) #增加一個维度

NumPy ufuncs

NumPy ufuncs 實作出向量化，因此會比正常的運算還要更快。

當然原先Python提供的加減乘除也可以用，但用 ufuncs 會跑得更快!

基礎運算

加法：newarr = np.add(arr1, arr2)

減法：newarr = np.subtract(arr1, arr2)

乘法：newarr = np.multiply(arr1, arr2)

除法：newarr = np.divide(arr1, arr2)

https://hackmd.io/@AndyChiang/PythonHardNote