
tensorflow / tensorflow
TensorFlow: Open sou
Clone this repo:
git clone https://tensor
Branches
master

5.3 数组运算

数组元素的切片—— 一维数组

可以使用切片来访问NumPy数组中的一部分，切片方法和Python序列数据结构的切片一样

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

>>>a[ :3]
array([ 0, 1, 2 ])

>>>a[0: ]
array([ 0, 1, 2 ,3])
```

数组元素的切片——二维数组

```
>>>b=np.array([[0,1,2,3],[4,5,6,7],[8,9,10,11]])
>>>b
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
>>>b[0]
array([0,1,2,3])

>>> b[0:2]
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])

>>> b[:2]
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])
```

```
>>>b[0:2, 0:2]
array([[0, 1],
       [4, 5]])

>>>b[0:2,1:3]
array([[1, 2],
       [5, 6]])

>>>b[:,0]
array([0, 4, 8])
```

数组元素的切片——三维数组

```
>>>t= np.array([[[0,1,2,3],[4,5,6,7],[8,9,10,11]],
                [[12,13,14,15],[16,17,18,19],[20,21,22,23]]])
>>> t
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],

       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]])
```

```
>>>t[:, :, 0]
array([[ 0,  4,  8],
       [12, 16, 20]])
```

```
>>> t[:, :, 1]
array([[ 1,  5,  9],
       [13, 17, 21]])
```

改变数组的形状

函 数	功能描述
np.reshape(shape)	**不改变当前数组**，按照shape创建新的数组
np.resize(shape)	**改变当前数组**，按照shape创建数组

```
>>>b = np.arange(12)
>>>b
array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 ])
```

```
>>> b.reshape(3,4)
array([[ 0, 1, 2, 3],
       [ 4, 5, 6, 7],
       [ 8, 9,10,11]])
>>> b
array([ 0,1,2,3,4,5,6,7,8,9,10,11 ])
```

```
>>>b.resize(3, 4)
>>>b
array([[ 0, 1, 2, 3],
       [ 4, 5, 6, 7],
       [ 8, 9,10,11]])
```

当改变形状时，应该考虑到数组中元素的个数，确保改变前后，**元素总个数相等**。

```
>>>b = np.arange(12)
>>>b
array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 ])
```

```
>>>b.reshape(2,5)
ValueError: cannot reshape array of size 12 into shape(2,5)
```

错误提示

创建数组并且改变数组形状

```
>>>b = np.arange(12)
>>>b.reshape(3,4)
```

```
>>>b = np.arange(12).reshape(3,4)
>>>b
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
>>>t = np.arange(24).reshape(2, 3, 4)
>>>t
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],

       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]])
```

```
>>>b.reshape(-1,1)
array([[ 0],
       [ 1],
       [ 2],
       [ 3],
       [ 4],
       [ 5],
       [ 6],
       [ 7],
       [ 8],
       [ 9],
       [10],
       [11]])
```

参数-1：根据数组中元素总个数、以及其他维度的取值，来**自动计算**出这个维度的取值。

```
>>>b.reshape(-1)
array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 ])
```

数组间的运算

```
>>> a = np.array([0,1,2,3])
>>> d = np.array([2,3,4,5])
>>> a+d
array([[ 2, 4, 6, 8]])
```

相加的2个数组的**形状**和**长度**应该一致，否则就会出现错误

```
>>> d = np.array([1,2])                    #创建一维数组
>>> a+d                                    #数组内的元素个数不同，无法相加
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-3-637545a26abd> in <module>()
----> 1 a+d

ValueError: operands could not be broadcast together with shapes (4,) (2,)
```

一维数组可以和**多维数组**相加，相加时会将一维数组**扩展**至多维。

```
>>>a = np.array([0,1,2,3])
>>>b = np.array([[0,1,2,3],[4,5,6,7],[8,9,10,11]])
>>> a+b
array([[ 0,  2,  4,  6],
       [ 4,  6,  8, 10],
       [ 8, 10, 12, 14]])
```

```
>>>b ** 2
array([[  0,   1,   4,   9],
       [ 16,  25,  36,  49],
       [ 64,  81, 100, 121]], dtype=int32)
```

- 数组之间的**减法**、**乘法**、**除法**运算，和加法运算规则相同。
- 当两个数组中元素的**数据类型不同**时，精度低的数据类型，会**自动转换**为精度更高的数据类型，然后再进行运算。

矩阵运算——矩阵乘法

乘号运算符：矩阵中对应的元素分别相乘

A B A*B

$$\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}\times\begin{bmatrix}2 & 0\\ 3 & 4\end{bmatrix}=\begin{bmatrix}2 & 0\\ 0 & 4\end{bmatrix}$$

```
>>>A = np.array([[1,1],[0,1]])
>>>B = np.array([[2,0],[3,4]])
>>>C = A * B
>>>C
array([[2, 0],
       [0, 4]])
```

矩阵相乘：按照**矩阵相乘**的规则运算

A B 矩阵相乘

$$\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}\times\begin{bmatrix}2 & 0\\ 3 & 4\end{bmatrix}=\begin{bmatrix}5 & 4\\ 3 & 4\end{bmatrix}$$

```
>>>np.matmul(A,B)
array([[5, 4],
       [3, 4]])
```

```
>>> np.dot(A,B)
array([[5, 4],
       [3, 4]])
```

矩阵运算——矩阵乘法

- @运算符：连续进行矩阵乘法

A B C 矩阵相乘

$$\begin{bmatrix}1 & 1\\0 & 1\end{bmatrix}\times\begin{bmatrix}2 & 0\\3 & 4\end{bmatrix}\times\begin{bmatrix}0 & 0\\1 & 1\end{bmatrix}=\begin{bmatrix}9 & 4\\7 & 4\end{bmatrix}$$

```
>>> A=np.array([[1,1],[0,1]])
>>> B=np.array([[2,0],[3,4]])
>>> C=np.array([[1,0],[1,1]])
>>> np.matmul(np.matmul(A,B),C)
array([[9, 4],
       [7, 4]])
>>> A@B@C
array([[9, 4],
       [7, 4]])
```

矩阵运算——转置和求逆

矩阵转置——np.transpose()

```
>>>np.transpose(A)
array([[1, 0],
       [1, 1]])

>>>np.transpose(B)
array([[2, 3],
       [0, 4]])
```

矩阵求逆——np.linalg.inv ()

```
>>>np.linalg.inv(A)
array([[ 1., -1.],
       [ 0.,  1.]])

>>>np.linalg.inv(B)
array([[ 0.5  ,  0.   ],
       [-0.375,  0.25 ]])
```

数组元素间的运算

函 数	功能描述
numpy.sum()	计算所有元素的和
numpy.prod()	计算所有元素的乘积
numpy.diff()	计算数组的相邻元素之间的差
np.sqrt()	计算各元素的平方根
np.exp()	计算各元素的指数值
np.abs()	取各元素的绝对值

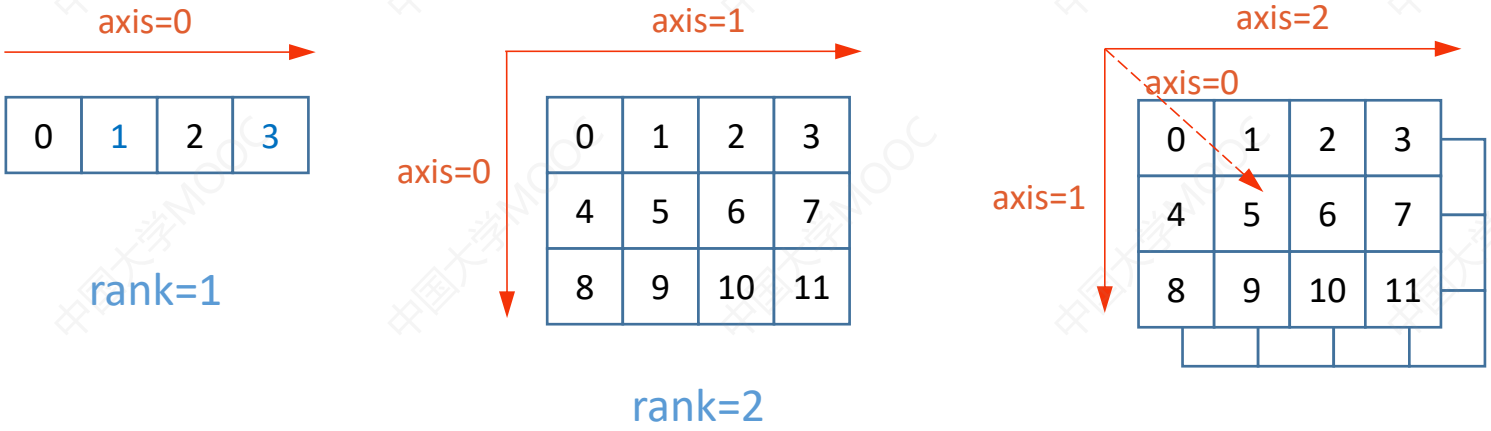
sum()——对数组中所有元素求和

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

>>>np.sum(a)
6
```

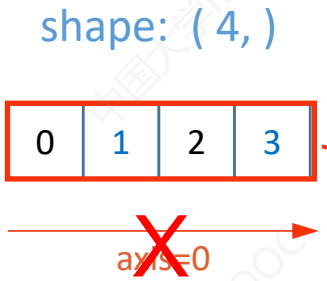
```
>>>b=np.arange(12).reshape(3,4)
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])

>>>np.sum(b)
66
```

按行求和&按列求和

- 轴（axes）：数组中的每一个维度被称为一个轴
- 秩（rank）：轴的个数

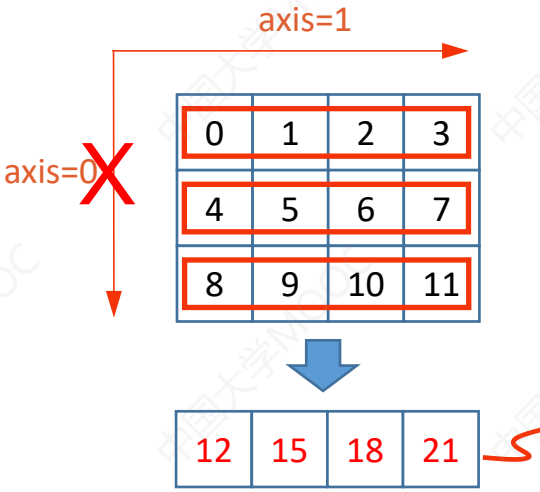


- 一维数组（rank=1）



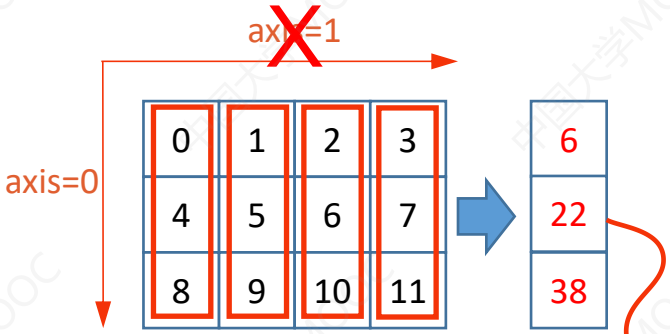

```
>>>a=np.arange(4)
>>>a
array([0, 1, 2, 3])

>>>np.sum(a)
6
```

二维数组（rank=2）




```
>>>b=np.arange(12).reshape(3,4)
>>>b
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])

>>>np.sum(b,axis=0)
array([12, 15, 18, 21])

>>>np.sum(b,axis=1)
array([ 6, 22, 38])
```

二维数组（rank=2）




```
>>>b=np.arange(12).reshape(3,4)
>>>b
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])

>>>np.sum(b,axis=0)
array([12, 15, 18, 21])

>>>np.sum(b,axis=1)
array([ 6, 22, 38])
```

- 三维数组（rank=3）

axis = 0，1，2

```
>>>t=np.arange(24).reshape(X,3,4)
>>>t
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],

       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]])
```

```
>>>np.sum(t,axis=0)
array([[12, 14, 16, 18],
       [20, 22, 24, 26],
       [28, 30, 32, 34]])

>>>print(np.sum(t,axis=1)
array([[12, 15, 18, 21],
       [48, 51, 54, 57]])

>>>np.sum(t,axis=2)
array([[ 6, 22, 38],
       [54, 70, 86]])
```

三维数组（rank=3）

axis = 0, 1, 2

```
>>>t=np.arange(24).reshape(2,3,4)
>>>t
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],
       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]])
```

12, 15, 18, 21

48, 51, 54, 57

```
>>>np.sum(t,axis=0)
array([[12, 14, 16, 18],
       [20, 22, 24, 26],
       [28, 30, 32, 34]])

>>>print(np.sum(t,axis=1)
array([[12, 15, 18, 21],
       [48, 51, 54, 57]])

>>>np.sum(t,axis=2)
array([[ 6, 22, 38],
       [54, 70, 86]])
```

- 三维数组（rank=3）

axis = 0, 1, 2

```
>>>t=np.arange(24).reshape(2,3,4)
>>>t
array([[[ 0,  1,  2,  3],      6
        [ 4,  5,  6,  7],      22
        [ 8,  9, 10, 11]],     38

       [[12, 13, 14, 15],      54
        [16, 17, 18, 19],      70
        [20, 21, 22, 23]]])    86
```

```
>>>np.sum(t,axis=0)
array([[12, 14, 16, 18],
       [20, 22, 24, 26],
       [28, 30, 32, 34]])

>>>print(np.sum(t,axis=1))
array([[12, 15, 18, 21],
       [48, 51, 54, 57]])

>>>np.sum(t,axis=2)
array([[ 6, 22, 38],
       [54, 70, 86]])
```

- 四维数组（rank=4）

axis = 0, 1, 2, 3

t.shape: (5,10,30, 5)

np.sum(t,axis=0) (10,30, 5)

np.sum(t,axis=1) (5,30, 5)

数组元素间的运算

函 数	功能描述
numpy.sum()	计算所有元素的和
numpy.prod()	计算所有元素的乘积
numpy.diff()	计算数组的相邻元素之间的差
np.sqrt()	计算各元素的平方根
np.exp()	计算各元素的指数值
np.abs()	取各元素的绝对值

sqrt()函数实例

```
>>> d = np.logspace(1,4,4,base=2)
>>> d
array([  2.,   4.,   8.,  16.])

>>> np.sqrt(d)
array([ 1.41421356,  2.        ,  2.82842712,  4.        ])
```

数组堆叠运算

np.stack((数组1, 数组2,...) , axis)

一维数组

x

1	2	3

y

4	5	6

shape=(3,)

axis=0

1	2	3
4	5	6

shape=(2, 3)

axis=1

1	4
2	5
3	6

shape=(3, 2)

一维数组堆叠运算

```
>>> x = np.array([1, 2, 3])          #创建一维数组x
>>> y = np.array([4, 5, 6])          #创建一维数组y

>>> np.stack((x, y), axis=0)         #在轴=0上堆叠
array([[1, 2, 3],
       [4, 5, 6]])

>>> np.stack((x, y), axis=1)         #在轴=1上堆叠
array([[1, 4],
       [2, 5],
       [3, 6]])
```

实例

```
area=[137.97,104.50,100.00,124.32,79.20,99.00,124.00,114.00,
      106.69,138.05,53.75,46.91,68.00,63.02,81.26,86.21]
room=[3,2,2,3,1,2,3,2,2,3,1,1,1,1,2,2]
b0 = np.ones(16)

np.stack((area,room,b0), axis = 1)
```

执行NumPy函数前，首先**自动把Python列表转换为NumPy数组，和数据类型转换**，然后再进行堆叠

a：Python列表，数据元素为浮点数
room：Python列表，数据元素为整数
b0：NumPy数组，全部元素为1

实例

```
area=[137.97,104.50,100.00,124.32,79.20,99.00,124.00,114.00,
      106.69,138.05,53.75,46.91,68.00,63.02,81.26,86.21]
room=[3,2,2,3,1,2,3,2,2,3,1,1,1,1,2,2]
b0 = np.ones(16)

np.stack((area,room,b0), axis = 1)
```

axis=1 堆叠 → (16,3)

```
array([[137.97,   3.  ,   1.  ],
       [104.5 ,   2.  ,   1.  ],
       [100.  ,   2.  ,   1.  ],
       [124.32,   3.  ,   1.  ],
       [ 79.2 ,   1.  ,   1.  ],
       [ 99.  ,   2.  ,   1.  ],
       [124.  ,   3.  ,   1.  ],
       [114.  ,   2.  ,   1.  ],
       [106.69,   2.  ,   1.  ],
       [138.05,   3.  ,   1.  ],
       [ 53.75,   1.  ,   1.  ],
       [ 46.91,   1.  ,   1.  ],
       [ 68.  ,   1.  ,   1.  ],
       [ 63.02,   1.  ,   1.  ],
       [ 81.26,   2.  ,   1.  ],
       [ 86.21,   2.  ,   1.  ]])
```

浮点数数组

二维数组堆叠

```
>>> m = np.array([[1, 2, 3], [4,5, 6], [7, 8, 9]])
>>> n = np.array([[10, 20, 30], [40, 50, 60], [70, 80, 90]])
>>> m.shape
(3, 3)
>>> n.shape
(3, 3)

>>> np.stack((m, n), axis=0).shape
(2, 3, 3)
>>> np.stack((m, n), axis=1).shape
(3, 2, 3)
>>> np.stack((m, n), axis=2).shape
(3, 3, 2)
```