

Numpy :



简介

标准安装的 Python 中用列表(list)保存一组值，可以用来当作数组使用，不过由于列表的元素可以是任何对象，因此列表中所保存的是对象的指针。这样为了保存一个简单的[1,2,3]，需要有 3 个指针和三个整数对象。对于数值运算来说这种结构显然比较浪费内存和 CPU 计算时间。

虽然 Python 还提供了一个 array 模块，array 对象和列表不同，它直接保存数值，和 C 语言的一维数组比较类似。但是由于它不支持多维，也没有各种运算函数，因此也不适合做数值运算。

NumPy 的诞生弥补了这些不足，NumPy 提供了两种基本的对象：`ndarray` (N-dimensional array object) 和 `ufunc` (universal function object)。

`ndarray` 是存储单一数据类型的多维数组，而 `ufunc` 则是能够对数组进行处理的函数。

Numpy 是 Python 下的一个 library。numpy 最主要的是支持矩阵操作与运算，非常高效是 numpy 的优势，core 为 C 编写。提升了 python 的处理效率，同时 numpy 也是一些与比较流行的机器学习框架的基础。

名词解释：`ndarray` 是 numpy 的核心数据类型，即 (n-dimensional array) 多维数组

，tensorflow 中的 tensor (张量)，它本质上也多维数组，但这个名字很高大上。因此，理解多维数组对之后的机器学习会有很大帮助。

安装与导入

```
pip install numpy
```

```
#或者直接安装 Anaconda 环境，自带 numpy
```

导入

```
import numpy as np
```

```
In [1]: import numpy as np
```

查看版本

```
np.version.version
```

使用帮助

```
dir(np)
```

基本数据类型：

类型	类型代码	说明
int8、uint8	i1、u1	有符号和无符号 8 位整型（1 字节）
int16、uint16	i2、u2	有符号和无符号 16 位整型（2 字节）
int32、uint32	i4、u4	有符号和无符号 32 位整型（4 字节）
int64、uint64	i8、u8	有符号和无符号 64 位整型（8 字节）
float16	f2	半精度浮点数
float32	f4、f	单精度浮点数
float64	f8、d	双精度浮点数
float128	f16、g	扩展精度浮点数
complex64	c8	分别用两个 32 位表示的复数

complex128	c16	分别用两个 64 位表示的复数
complex256	c32	分别用两个 128 位表示的复数
bool	?	布尔型
object	O	python 对象
string	Sn	固定长度字符串，每个字符 1 字节，如 S10
unicode	Un	固定长度 Unicode，字节数由系统决定，如 U10

使用 Numpy

构建 ndarray

构建一维数组

```
n1=np.array([1,2,3])
n1.shape
```

```
n1=np.array([1,2,3])
n1.shape
```

```
(3,)
```

构建二维数组

```
n2= np.array([[1,2,3],[4,5,6]])
n2.shape
```

```
np.array([[1,2,3],[4,5,6]])
array([[1, 2, 3],
       [4, 5, 6]])
```

从 Python 数组构建

```
l1=[1,2,3,4,5]
print (type(l1))
```

```
l2=np.array(l1)
print (type(l2))
```

```
l1=[1,2,3,4,5]
print (type(l1))
l2=np.array(l1)
print (type(l2))
```

```
<class 'list'>
<class 'numpy.ndarray'>
```

ndarray 转为 list

```
l3=list(l2) # ndarray 转为 Python list
print (type(l3))
```

```
l3=list(l2)
type(l3)
```

```
list
```

快速构建 ndarray

序列创建：

np.arange(15)#类似于 python 中的 range，创建一个第一个维度为 15 的 ndarray 对象。

```
np.arange(15)
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

np.arange(2,3,0.1) #起点，终点，步长值。含起点值，不含终点值。

np.linspace(1,10,10) #起点，终点，区间内点数。起点终点均包括在内。

np.arange(0,1,0.1) #0 到 1 之间步长为 0.1 的数组，数组中不包含 1

np.linspace(0, 1, 5) # 开始：0，结束 1，元素数 5。

```
a11= np.arange(99)
array([ 0. ,  0.25,  0.5 ,  0.75,  1.  ])
```

```
list2=np.array((range(5,10),range(5,10)))
```

```
t_x=np.linspace(-1,1,100)
t_x
```

空矩阵

`np.empty((6,6))` #空的 ndarray, 指定其 shape 即可, 注意: 空不意味着值为 0, 而是任何的 value, 内存中没有被初始化的。

```
np.empty((6,6))  
  
array([[ 2.31276869e-152,  2.64625024e-260,  7.70858946e+218,  
        6.01334668e-154,  4.47593816e-091,  3.29750046e-028],  
       [ 6.01347002e-154,  8.74369854e+140,  3.32231080e+257,  
        2.44514198e-154,  6.01347002e-154,  6.05142438e-154],  
       [ 4.59541097e-072,  6.01347002e-154,  1.90465047e-028,  
        8.82085571e+199,  5.30282495e+180,  5.26757287e+170],  
       [ 4.73443029e-120,  2.52895809e-086,  2.09222269e-110,  
        6.01347002e-154,  3.03426177e-086,  6.01347002e-154],  
       [ 2.63765253e-061,  3.94527705e-114,  1.94895206e-110,  
        6.01347002e-154,  1.04991723e-153,  1.94895206e-110],  
       [ 1.04991780e-153,  6.01346953e-154,  1.30452181e+243,  
        1.03765594e-311,  4.49006859e-319,  5.56268465e-309]])
```

对角线矩阵

`np.eye(3)` # 对角线矩阵

```
np.eye(3)  
  
array([[ 1.,  0.,  0.],  
       [ 0.,  1.,  0.],  
       [ 0.,  0.,  1.]])
```

随机数矩阵

`np.random.rand(3,2)` #随机数矩阵:

```
np.random.rand(3,2)  
  
array([[ 0.53089721,  0.06054321],  
       [ 0.98713323,  0.23285739],  
       [ 0.86890877,  0.77251641]])
```

```
a = np.random.rand(5,5) #指定数字矩阵 fill()函数  
a.fill(7)
```

关于随机数方法:

rand : 返回均匀分布随机数

randn: 返回服从正态分布的随机数

随机打乱矩阵

```
a11= np.array(range(3))
```

```
>>> ar=np.arange(10)
>>> np.random.shuffle(ar)
>>> ar
array([8, 5, 9, 1, 6, 0, 3, 2, 4, 7])
```

0, 1 矩阵

```
z= np.zeros((2,3)) #建立 2*3 的 0 值矩阵, 注意: 传入参数是一个 tuple, 因此别忘了()
```

```
z= np.ones((5,7)) #建立 5*7 的 1 值矩阵, 注意: 传入参数是一个 tuple, 因此别忘了()
```

ones_like#根据某个矩阵的形态创建一个矩阵

```
>>> a1
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
>>> np.ones_like(a1)
array([[1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1]])
```

指定数字矩阵 fill

```
a = np.random.rand(5,5)
a.fill(7)
```

numpy 数据类型及转换

```
d1=np.array([1,2,3,4,5])
print(d1)

print(d1.dtype) # Numpy 会自动根据 ndarray 对象中的值判定数据类型, 这里为整型。
```

```
d1=np.array([1,2,3,4,5])
print(d1)
print(d1.dtype) # Numpy会自动根据ndarray对象中的值判定数据类型，这里为整型。

[1 2 3 4 5]
int32
```

`d1.astype(np.float32)` # 如果想把它强制转为浮点型，可以用 `astype` 函数转换。

```
d1.astype(np.float32)
array([ 1.,  2.,  3.,  4.,  5.], dtype=float32)
```

`np.random.rand((3,2), dtype=np.float32)` # 也可以在创建 `ndarray` 时，即指定其数据类型

ndarray 基本属性

ndarray.ndim

查看 `ndarray` 的 dimension 维度数。

```
n1=np.identity(5)
n1.ndim
```

```
n1=np.identity(5)
n1.ndim
```

2

ndarray.shape

查看 `ndarray` 的 `shape` 形状,返回值 是一个 `tuple`，当维数为 2 维时，返回的是行数、列数、.....数组的各个维

(注意和维和维数要区分开)。它是一个数组各个维的长度构成的整数元组。对 `n` 行 `m` 列矩阵而言，`shape` 将是

(`n,m`)。因此，**shape 元组的长度也就是 rank，也是维数 `ndim`。**

ndarray.dtype

查看 `ndarray` 对象的数据类型

```
n1.dtype
dtype('float64')
```

ndarray.size

查看 ndarray 对象中元素的数量

NDArray 取值赋值与维度操作

我们首先创建一个矩阵。

```
import numpy as np
a=np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
```

```
import numpy as np
```

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

```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

我们手工创建一个二维矩阵名为 a , shape 为(3,4)

接下来我们开始对这个矩阵进行读取

读取

切片读取

还记得 Python 中对列表进行切片访问吗？numpy 对矩阵进行切片访问时也是遵循着同样的规则，只不过 python 的列表中是按一行（一维）方向进行访问，而在 numpy 中是按行，列两个维度进行切片。（如果 ndarray 的 ndim 超过 2，也是遵循同时的方式进行访问）

a[:, :]#第一个冒号代表切出所有的行，第二个冒号代表切出所有的列。这样会比较容易理解。

```
a[:, :]
```

```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

a[2:, 2:]#这里的红框代表选中的行，绿框代表，选中的列。


```
a[:, :]
```

```
array([[ 1,  2,  3,  4],  
       [ 5,  6,  7,  8],  
       [ 9, 10, 11, 12]])
```

```
a[2:, 2:]
```

```
array([[11, 12]])
```

`a[:, 1:3]` #第一个冒号代表，选中所有行，第二个参数 1:3 代表从第二列到第 3 列。同样遵循 python 的左闭右

开的切片访问规则。

```
a[:, 1:3]
```

```
array([[ 2,  3],  
       [ 6,  7],  
       [10, 11]])
```

索引访问

也可以对 ndarray 在指定维度上的进行索引式访问

```
a[[2], [0, 2]]
```

#红色的行代表第一个参数[2]，代表先选中第 3 行。绿色的列代表第二个参数[0, 2]，选中了第一个和第三个元素 9, 11。

#注意，使用索引访问时，索引值要放进[]中，代表传入一个列表。

```
a
```

```
array([[ 1,  2,  3,  4],  
       [ 5,  6,  7,  8],  
       [ 9, 10, 11, 12]])
```

```
a[[2], [0, 2]]
```

```
array([ 9, 11])
```

#除了可以指定数字为索引，还可以用布尔类型的数组，做为索引传入，对下标进行过滤。

```
bool_a = a > 2
```

```
bool_a
```

```
a[bool_a]
```

```
bool_a=a>2
bool_a

array([[False, False,  True,  True],
       [ True,  True,  True,  True],
       [ True,  True,  True,  True]], dtype=bool)
```

```
a[bool_a]

array([ 3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

切片与索引混合访问

访问 ndarray 的方式非常灵活，可以使用切片与索引可以混合进行访问。

```
a[[1,2],:]
```

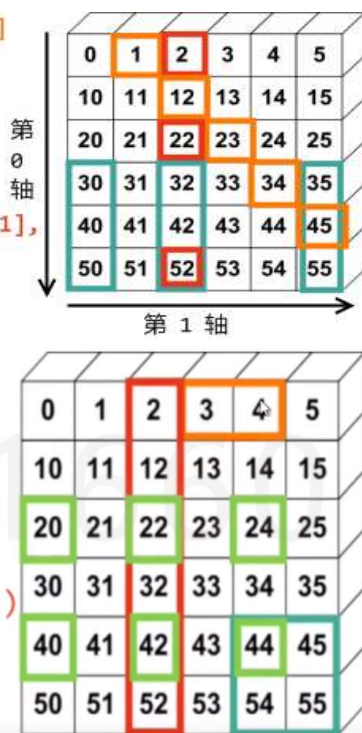
```
a[[1,2],:]

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

ndarray 读取的经典图片，更直观地去理解

```
>>> a[(0,1,2,3,4),(1,2,3,4,5)]
array([1,12,23,34,45])
>>> a[3:,[0,2,5]]
array([[30,32,35],
       [40,42,45],
       [50,52,55]])
>>> mask=np.array([1,0,1,0,0,1],
                   dtype=np.bool)
>>> a[mask,2]
array([2,22,52])

>>> a[0,3:5]
array([3,4])
>>> a[4:,4:]
array([[44,45],[54,55]])
>>> a[:,2]
array([2,12,22,32,42,52])
>>> a[2::2,::2]
array([[20,22,24],
       [40,42,44]])
```



遍历读取

对于 ndarray，也可以像对待 Python 中的可迭代对象一样，对它进行遍历。

```
for i in a:
```

```
print(i)
```

```
for i in a:  
    print(i)
```

```
[1 2 3 4]  
[5 6 7 8]  
[ 9 10 11 12]
```

赋值

既然能取出来，那赋值就很简单了，直接=号 就可以。

ndarray 维度操作

ndarray 对象 shape 变换

```
np.arange(16).reshape(2,8).shape
```

```
np.arange(16).reshape(2,8).shape
```

```
(2, 8)
```

这里先用 arange 生成了从 0 到 15 的一维数组，然后使用 reshape，将这个数组的形态改变为有行与列两个维度了（2 行 8 列）#注意这个形状的乘积必须和总的元素个数相同，否则将会报错。

可以把这个数组变更 shape 为(1, 16)，这也是有两个维度，

也可以用 reshape 转为(2,2,4)，这代表有 3 个维度，也可以这样去理解：两行两列，每个元素有 4 个维度来表示。

增维操作 np.newaxis

```
import numpy as np  
x=np.linspace(-1,1,100)  
x.shape  
x[:,np.newaxis].shape  
x[np.newaxis,:].shape
```

```
x=np.linspace(-1,1,100)
x.shape
(100,)
```

```
x[:,np.newaxis].shape
(100, 1)
```

```
x[np.newaxis,:].shape
(1, 100)
```

维度转置 np.transpose

shape 为(100,)这种 ndarray 是无法进行转维，因为它的 ndim 为 1 `x=x[np.newaxis,:]` #先增维
`np.transpose(x).shape` #然后可以维度转置了
#一维的 vector 转置后还是自己

全降维为一维数组：

```
x.flatten()
```

Numpy 运算

逐元素运算

numpy ufunc 运算

ufunc 是 universal function 的缩写，它是一种能对数组的每个元素进行操作的函数。numpy 内置的许多 ufunc 函数都在 C 语言级别实现的，因此它们的计算速度非常快。以下讲到的 add, subtract 等都是 numpy 提供的 ufunc 函数。

逐元素运算：就是两个 shape 一致的矩阵，相同位置上的元素的运算。

求和

我们可以像在 python 里进行数值运算一样，直接运行： $x+y$

该图用颜色标识出了，**逐元素求和的过程**。这个很容易理解。

```
[[1, 2],  
 [3, 4],  
 [5, 6],  
 [7, 8]]
```

```
print(x+y)
```

```
[[6, 8],  
 [10, 12]]
```

也可以使用 `np.add()` 函数，传入 x,y 后得到求和结果。

求差

与求和类似，两个矩阵直接减，或使用 `np.subtract()` 函数

```
np.subtract(x,y)
```

```
array([[ -4., -4.],  
       [ -4., -4.]])
```

乘与除

乘法： $x*y$ 或者 `np.multiply(x,y)`

```
a=np.arange(1,5)
```

```
np.multiply(a,a)
```

```
array([ 1,  4,  9, 16])
```

除法： x/y 或者 `np.divide(x,y)`

```
a=np.arange(1,5)
```

```
np.divide(a,a)
```

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

Matrix Multiplication

Numpy 可以轻松进行矩阵的乘法，矩阵的乘积使用 `dot` 函数进行计算。而对于 **一维数组**，它计算的是其点积。

向量的内积 inner product :

```
v=np.array([9,10])  
w=np.array([10,11])  
v.dot(w)
```

200

即 $9*10+10*11$, 最后的结果是 200

也可以这样写 `np.dot(v,w)`

矩阵乘积

```
x=np.array([[1,2],[3,4]])  
y=np.array([[5,6],[7,8]])  
print(x,y)
```

```
x=np.array([[1,2],[3,4]])  
y=np.array([[5,6],[7,8]])  
print(x)  
print(y)
```

```
[[1 2]  
 [3 4]]  
[[5 6]  
 [7 8]]
```

`x.dot(y)` #或者是 `np.dot(x,y)`

```
x.dot(y)
```

```
array([[19, 22],  
       [43, 50]])
```

#重点：矩阵运算是机器学习中核心的核心，必须多练习熟悉。

ndarray 对象内部运算

求和 sum

```
a=np.arange(16).reshape(2,8)
np.sum(a)
```

120

其它的运算如 `np.mean()` , `max()` , `min()` 等运算。

按矩阵的维度进行运算

```
x=np.array([[1,2],[3,4]])
print(x)
print('-----')
print('axis=0:',np.sum(x,axis=0))
print('axis=1:',np.sum(x,axis=1))
```

```
x=np.array([[1,2],[3,4]])
print(x)
print('-----')
print('axis=0:',np.sum(x,axis=0))
print('axis=1:',np.sum(x,axis=1))
```

```
[[1 2]
 [3 4]]
-----
axis=0: [4 6]
axis=1: [3 7]
```

#axis=0 时指定是第一个维度，在维度数为 2 的时候（即一个矩阵），第一个维度就是行，第二个维度就是列。将

axis 设置为 0，即表示消除了行个维度，保留列的维度。所以结果是[4,6]（可以认为是把第 0 维压扁了）。

axis 设置为 1，代表消除了列的维度，保留行的维度。

排序

一维数组的排序

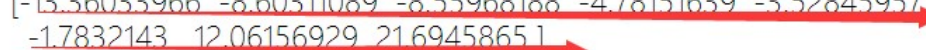
使用 sort 方法可以对元素进行排序。这和 python 下的排序很像。

```
arr=np.random.randn(8)*10  
print(arr)
```

```
[ -8.60311089 -8.55968188 -13.36033966 -4.78151639 -3.52845957  
 12.06156929 21.6945865 -1.7832143 ]
```

```
arr.sort()  
print(arr)
```

```
[-13.36033966 -8.60311089 -8.55968188 -4.78151639 -3.52845957  
 -1.7832143 12.06156929 21.6945865]
```



```
arr=np.random.randn(8)*10  
print(arr)  
arr.sort()  
print(arr)
```

多维数组排序

如果 ndarray 对象的 ndim 是 2(也就是一个矩阵)那么排序可以在不同的维度上进行。

```
arr=np.random.randn(5,3)*10
```

```
np.sort(arr,axis=1)#按维度 1，即列进行排序
```

```
np.sort(arr,axis=0)#按维度 0，即行进行排序
```

#sort 指定 axis=0，即是排序的时候，依据的维度是第一维（即行），于是我们可以观察到排序后矩阵，在行的方向上，是从小到大的。


```
arr=np.random.randn(5,3)*10
print(arr)
```

```
[[ 0.8641842 -1.46376309 -0.95208888]
 [ 2.45886205  5.02778185  3.88853153]
 [-0.72858119  3.19362638 -14.46825729]
 [-8.20412356 -11.45805838  1.67328358]
 [ 2.11967167 -9.06694682 -12.53959922]]
```

```
arr.sort(axis=0)
print(arr)
```

```
[[ -8.20412356 -11.45805838 -14.46825729]
 [ -0.72858119 -9.06694682 -12.53959922]
 [  0.8641842  -1.46376309 -0.95208888]
 [  2.11967167  3.19362638  1.67328358]
 [  2.45886205  5.02778185  3.88853153]]
```

排序小实验

找出排序后位置处于在 5%的数字

```
large_arr =np.random.randn(1000)
large_arr.sort()
print(Large_arr)
print(large_arr[int(0.05*len(large_arr))])
```

```
print(large_arr[int(0.05*len(large_arr))])
```

-1.71805070642

广播算法

当我们使用 ufunc 函数对两个数组进行计算时，ufunc 函数会对这两个数组的对应元素进行计算因此它要求这两个

数组有相同的大小(shape 相同)。如果两个数组的 shape 不同的话会进行如下的广播(broadcasting)处理。

```
a=np.random.randn(3,5)
a+10
```

```
a=np.random.randn(3,5)
```

```
a+10
```

```
array([[ 10.21238375,  8.62786757,  9.44075954, 10.13046306,
        12.81143018],
       [ 9.28626067, 10.05615553,  9.10308209,  8.41242968,
        10.90757693],
       [10.15879886,  9.29795614,  8.82812299, 10.92703946,
        10.66109087]])
```

这里的 a 的 shape 为 (3,5)，第二行和 a 进行运算的是一标量对象 10，这时就会用到广播算法。具体步骤是，将 10 的在第一个维度扩展 3 次，第二个维度扩展 5 次。这时两个对象的 shape 一致，就可以进行运算了。

什么时候会进行广播呢：

当操作两个 array 时，numpy 会比较它们的 shape，当出现下面两种情况时，两 array 会兼容和输出

broadcasting 结果：

- 相等
- 其中一个为 1

以下情况会进行广播：

比如求和的时候有：

```
Image (3d array): 256 x 256 x 3
Scale (1d array): 3
Result (3d array): 256 x 256 x 3
```

```
A (4d array): 8 x 1 x 6 x 1
B (3d array): 7 x 1 x 5
Result (4d array): 8 x 7 x 6 x 5
```

```
A (2d array): 5 x 4
B (1d array): 1
Result (2d array): 5 x 4
```

```
A (2d array): 15 x 3 x 5
B (1d array): 15 x 1 x 5
```

高级的 ndarray 处理

变形 ndarray 对象

reshape 可以轻松将数组转换为二维，三维，甚至更高维度的数组

如果在某个维度指定-1,则 numpy 会自动推导出正确的形状

```
import numpy as np

a1=np.arange(16)#含有 16 个元素的一维数组

np.reshape(a1,(4,4))#把一维数组 reshape 为一个矩阵，shape 为(4,4)

np.reshape(a1,(2,-1))#把一个矩阵 reshape 为另一个矩阵，0 维度为 2，1 维度为-1(numpy 会自动推导出为 8)
```

```
In [1]: import numpy as np
a1=np.arange(16)
np.reshape(a1,(4,4))
```

```
Out[1]: array([[ 0,  1,  2,  3],
               [ 4,  5,  6,  7],
               [ 8,  9, 10, 11],
               [12, 13, 14, 15]])
```

```
In [3]: np.reshape(a1,(2,-1))
```

```
Out[3]: array([[ 0,  1,  2,  3,  4,  5,  6,  7],
               [ 8,  9, 10, 11, 12, 13, 14, 15]])
```

拉平 ndarray 对象

使用 ravel()或 flatten()函数都可以把高维的 ndarray 对象拉平为一维。

```
a1.ravel()
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
```

```
a1.flatten()
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
```

拼接 ndarray 对象

先建立两个矩阵

```
arr1=np.array([[1,2,3],[4,5,6]])  
arr2=np.array([[7,8,9],[10,11,12]])
```

```
arr1=np.array([[1,2,3],[4,5,6]])  
arr2=np.array([[7,8,9],[10,11,12]])  
print(arr1,\n\n',arr2)
```

```
[[1 2 3]  
 [4 5 6]]
```

```
[[ 7  8  9]  
 [10 11 12]]
```

左右拼（横向拼接）

使用 np.hstack() 函数，传入 arr1, arr2 这两个矩阵即可。

```
np.hstack([arr1,arr2])
```

图示中的红框为 arr1, 黑框为 arr2，被左右拼接为一个新的矩阵，新矩阵的 shape 为 (2,6)

stack 是堆叠的意思, h 是水平 horizon 的缩写。

```
np.hstack([arr1,arr2])
```

```
array([[ 1,  2,  3,  7,  8,  9],  
       [ 4,  5,  6, 10, 11, 12]])
```

使用 np.c_[] 也可以实现。这里的 c，指 column 列。

```
np.c_[arr1,arr2]
```

```
array([[ 1,  2,  3,  7,  8,  9],  
       [ 4,  5,  6, 10, 11, 12]])
```

np.c_[] 还有生成矩阵的用法, c 即 column

```
np.c_[1:6, -5:0]
```

```
array([[ 1, -5],  
       [ 2, -4],  
       [ 3, -3],  
       [ 4, -2],  
       [ 5, -1]])
```

concatenate 这个方法也可以，但个人感觉 hstack 或 np.c_[] 更直观，

```
np.concatenate([arr1, arr2], axis=0)
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

所谓堆叠，参考叠盘子。。。连接的另一种表述 垂直stack与水平stack

垂直拼（纵向拼接）

使用 np.vstack() 函数，传入 arr1, arr2 这两个矩阵即可。

np.vstack([arr1, arr2])

```
np.vstack([arr1, arr2])
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

#参考 hstack 就很容易理解。

使用 np.r_[] 也可以实现，这里的 r，指 row 行。

```
np.r_[arr1, arr2]
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

```
np.vstack((arr1, arr2)) # vertical
```

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

```
np.hstack((arr1, arr2)) # horizontal
```

```
array([[ 1,  2,  3,  7,  8,  9],
       [ 4,  5,  6, 10, 11, 12]])
```

r_用于按行堆叠

c_用于按列堆叠

```
np.c_[arr1,arr2]
```

拆分 ndarray 对象

水平分割 ()

```
a,b,c=np.hsplit(arr1,3)
print(a,'\n\n',b,'\n\n',c)
```

#这里将 arr1 切为了 3 个 ndarray 对象，并赋值给了 a,b,c 三个变量。可以看到，这把刀是竖着切的。

```
a,b,c=np.hsplit(arr1,3)
print(a,'\n\n',b,'\n\n',c)
```

```
[[1]
 [4]]
```

```
[[2]
 [5]]
```

```
[[3]
 [6]]
```

垂直分割

```
np.vsplit(arr1,2)
```

#这里将原来 shape 为(2,3)的矩阵分割为 2 个 shape 为(1,3)的一维数组。可以看到，这把刀是横着切的。

```
np.vsplit(arr1,2)
```

```
[array([[1, 2, 3]]), array([[4, 5, 6]])]
```

重复 ndarray 对象

按元素重复

```
arr = np.arange(3)
print(arr)

[0 1 2]

print(arr.repeat(3))

[0 0 0 1 1 1 2 2 2]

print(arr.repeat([2,3,5]))

[0 0 1 1 1 2 2 2 2 2]
```

每个元素统一重复 3 次

与指定每个元素，重复的次数[2,3,4]次

按轴(维度)进行重复

```
arr = np.random.rand(2,3)
print(arr)

[[ 0.90063544  0.36862431  0.46734451]
 [ 0.61467785  0.63962631  0.61288228]]

print(arr.repeat(2, axis=0))

[[ 0.90063544  0.36862431  0.46734451]
 [ 0.90063544  0.36862431  0.46734451]
 [ 0.61467785  0.63962631  0.61288228]
 [ 0.61467785  0.63962631  0.61288228]]
```

按 ndarray 对象重复

可以理解为以传入的 ndarray 对象为模板，生产出一块瓷砖，就可以像**贴瓷砖**一样，横向，纵向进行复制。

对 arr 对象的复制两次，默认按 0 维进行。

```
print(arr)
print()
print(np.tile(arr, 2))
```

```
[[ 0.90063544  0.36862431  0.46734451]
 [ 0.61467785  0.63962631  0.61288228]]

[[ 0.90063544  0.36862431  0.46734451  0.90063544  0.36862431  0.46734451]
 [ 0.61467785  0.63962631  0.61288228  0.61467785  0.63962631  0.61288228]]
```

0 维复制 2 次，1 维复制 3 次

```
print(np.tile(arr, (2,3)))
```

```
[[ 0.90063544  0.36862431  0.46734451  0.90063544  0.36862431  0.46734451
  0.90063544  0.36862431  0.46734451]
 [ 0.61467785  0.63962631  0.61288228  0.61467785  0.63962631  0.61288228
  0.61467785  0.63962631  0.61288228]
 [ 0.90063544  0.36862431  0.46734451  0.90063544  0.36862431  0.46734451
  0.90063544  0.36862431  0.46734451]
 [ 0.61467785  0.63962631  0.61288228  0.61467785  0.63962631  0.61288228
  0.61467785  0.63962631  0.61288228]]
```

ndarray 对象的输入与输出

保存

```
a1=np.arange(50).reshape(2,5,5)
np.save('d:/a1',a1)
```

```
a1=np.arange(50).reshape(2,5,5)
np.save('d:/a1',a1)
```

保存好的数组,默认后缀为 npy

a1.npy

多个数组保存使用 savez 方法。

```
arr3 = np.arange(15).reshape(3,5)
np.savez("array_archive.npz", a=arr, b=arr2, c=arr3)
```

多个数组可以一起压缩存储

读取

load 方法载入 numpy 格式的数据

```
np.load('d:/a1.npy')
```

```
np.load('d:/a1.npy')
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, 24]],
       [[25, 26, 27, 28, 29],
        [30, 31, 32, 33, 34],
        [35, 36, 37, 38, 39],
        [40, 41, 42, 43, 44],
        [45, 46, 47, 48, 49]]])
```

savetxt, loadtxt 方法载入文本格式的数据同理

课后练题

简单加法：

在 Python 下对一个二维列表，l1=[[1,2,3],[4,5,6]]，然后进行每个元素+1 的操作。

要求使用

- Python 循环
- Map 函数
- Numpy 计算（广播算法）

矩阵转置：

对 arr= [[1, 2, 3], [4, 5, 6], [7,8, 9], [10, 11, 12]]进行行列互转

- Python 列表表达式

- Numpy 进行转置

矩阵内部运算

- 在 Python 下对一个二维列表，`l1=[[1,2,3],[4,5,6]]`，要求求出按行与按列求和。
- 在 numpy 下对一个二维列表，`l1=[[1,2,3],[4,5,6]]`，要求求出按行与按列求和。

NDArray 创建与属性

- 使用 Numpy 创建一个多维数组，请输出以下属性
- 该数组的形状
- 该数组的维度
- 该数组元素的个数
- 该数组的数据类型

NDArray 的访问

创建一个包含从数字 1 到 60，的数组，并将于其形状变为为（3，2，10）的三维数组，并按如下要求访问这个三维数组

- 访问第 0 维度中第二个元素的所有信息
- 访问第 0 维度中所有元素的全部第 0 个维度（行），和最后一个维度（列）6-7 列
- 将该数组降维至二维，形状自行定义
- 打乱矩阵内元素的顺序
- 按行维度进行排序
- 按列维度进行排序

- 将降维后的数组进行行列转置
- 将转至后的数组展平至一维数组。

拼接与切分 NDAarray 对象

- 先建立两个矩阵 `arr1=[[1,2,3],[4,5,6]]`和 `arr2=[[7,8,9],[10,11,12]]`
- 对两个矩阵进行横向拼接，并输出结果
- 对两个矩阵进行纵向拼接，赋值给 `arr3` 并输出结果
- 对 `arr3` 按列进行分割为 3 个新元素并输出
- 对 `arr3` 按行进行分割为 2 个新元素并输出
- 将 `arr3` 转换为 Python 列表，并指定数据类型为 `int`

数据导入导出

- 生成一个随机数矩阵，形状为(3,5)
- 将其第 2 行第 4 列的元素修改为 998
- 保存至本地文件 `random_matrix.txt`
- 并尝试从这个文件中载入并验证修改是否正确

使用 numpy 实现 softmax

思路：

- 计算指数，并安全处理避免数字过大
- 按行求和
- 每行均除以计算的和

股票相关统计量计算

- 读入的给定的 ibm 股价数据中的收盘价及成交量
- 计算成交量加权平均价 VWAP (收盘价)
- 计算时间加权平均价 TWAP (收盘价)
- 找出 IBM 股票收盘价的极差
- 计算其收盘价的中位数, 均值及方差
- 计算股票的收益率 (简单收益率与对数收益率)
- 计算对数收益收益率大于 1% 的天数有多少?

```
c,v=np.loadtxt('d:/ibm.txt',delimiter='\t',usecols=(5,6),unpack=True,skiprows=1)
np.average(c,weights=v)

t=np.arange(len(c))
np.average(c,weights=t)
```

成交量加权平均价是将多笔交易的价格按各自的成交量加权而算出的平均价, 若是计算某一证券在某交易日的 VWAP, 将当日成交总值除以总成交量即可。VWAP 可作为交易定价的一种方法, 亦可作为衡量机构投资者或交易商的交易表现的尺度。英文 Volume Weighted Average Price。

假设某种商品最高价和最低价 2 种, 其中最高价 300, 占 10%; 最低价 200, 占 90%。

那该商品的加权平均价为:

a1 对应的是价格, a2 对应的是成交量

```
a1=np.array([200,300])
a2=np.array([900,100])
np.average(a1,weights=a2)
```

210.0

时间加权平均也是同样原理, 按照越近日期, 所占的权重越高这样的方式去计算。

```
np.ptp(c)
```

```
np.mean(c)
```

```
np.median(c)
np.var(c)
```

#普通收益率，注意：不含最后一个交易日的计算

```
c=c[:-1]#对价格顺序进行反转，最新交易日放到最后
return=np.diff(c) /c[0:-1]
[np.where(dif>0)]
```

numpy 中的 diff 函数用于行一行之间的求差运算。

```
a1=np.arange(15)
np.diff(a1[0:-1])

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

对数收益率，取对数可以让数据更平稳，但是不会改变数据间的相关关系。有利于计算。

```
logreturn=np.diff(np.log(c))
```

计算对数收益大于 1 的交易日

```
logreturn[logreturn*100>1].size
```

日期处理

导入文本文件中的股份数据，并将日期转化为对应周几后，再分别去计算从周一到周五的平均价，并找出周几平均价最高，周几的平均价去低？

数据：ibm 股票数据

提示：自定义日期处理函数，在 loadtxt 的参数中对应处理函数与列。

按周对股票价格进行汇总：

在上一题的基础上选取有连续交易的三周时间（共计 15 天）后，按周进行汇总要求得出每周的开，高，低，收。

提示：numpy 中的 apply_along_axis 函数是高阶函数，能接收处理函数。

`numpy.apply_along_axis(func, axis, arr, *args, **kwargs)` :

必选参数：`func, axis, arr`。

其中 `func` 是我们自定义的一个函数，函数 `func(arr)` 中的 `arr` 是一个数组，函数的主要功能就是对数组里的每一个元素进行处理，其中 `axis` 表示函数 `func` 对数组 `arr` 作用的轴。可选参数：`*args, **kwargs`。都是 `func()` 函数额外的参数。返回的是一个根据 `func()` 函数以及维度 `axis` 运算后得到的数组。

附：Numpy 常用函数

numpy 函数参考

生成函数	作用
<code>np.array(x)</code>	将输入数据转化为一个 ndarray
<code>np.array(x, dtype)</code>	将输入数据转化为一个类型为 <code>type</code> 的 ndarray
<code>np.asarray(array)</code>	将输入数据转化为一个新的（copy）ndarray
<code>np.ones(N)</code>	生成一个 N 长度的一维全一 ndarray
<code>np.ones(N, dtype)</code>	生成一个 N 长度类型是 <code>dtype</code> 的一维全一 ndarray
<code>np.ones_like(ndarray)</code>	生成一个形状与参数相同的全一 ndarray
<code>np.zeros(N)</code>	生成一个 N 长度的一维全零 ndarray
<code>np.zeros(N, dtype)</code>	生成一个 N 长度类型位 <code>dtype</code> 的一维全零 ndarray
<code>np.zeros_like(ndarray)</code>	类似 <code>np.ones_like(ndarray)</code>
<code>np.empty(N)</code>	生成一个 N 长度的未初始化一维 ndarray
<code>np.empty(N, dtype)</code>	q 生成一个 N 长度类型是 <code>dtype</code> 的未初始化一维 ndarray
<code>np.empty(ndarray)</code>	类似 <code>np.ones_like(ndarray)</code>
<code>np.eye(N)</code>	创建一个 N * N 的单位矩阵（对角线为 1，其余为 0）
<code>np.identity(N)</code>	
<code>np.arange(num)</code>	生成一个从 0 到 num-1 步数为 1 的一维 ndarray

np.arange(begin, end)	生成一个从 begin 到 end-1 步数为 1 的一维 ndarray
np.arange(begin, end, step)	生成一个从 begin 到 end-step 的步数为 step 的一维 ndarray
np.mersgrid(ndarray, ndarray,...)	生成一个 ndarray * ndarray * ...的多维 ndarray
np.where(cond, ndarray1, ndarray2)	根据条件 cond，选取 ndarray1 或者 ndarray2，返回一个新的 ndarray
np.in1d(ndarray, [x,y,...])	检查 ndarray 中的元素是否等于[x,y,...]中的一个，返回 bool 数组
矩阵函数	说明
np.diag(ndarray) np.diag([x,y,...])	以一维数组的形式返回方阵的对角线（或非对角线）元素 将一维数组转化为方阵（非对角线元素为 0）
np.dot(ndarray, ndarray)	矩阵乘法
np.trace(ndarray)	计算对角线元素的和

排序函数	说明
np.sort(ndarray)	排序，返回副本
np.unique(ndarray)	返回 ndarray 中的元素，排除重复元素之后，并进行排序
np.intersect1d(ndarray1, ndarray2) np.union1d(ndarray1, ndarray2) np.setdiff1d(ndarray1, ndarray2) np.setxor1d(ndarray1, ndarray2)	返回二者的交集并排序。 返回二者的并集并排序。 返回二者的差。 返回二者的对称差
一元计算函数	说明
np.abs(ndarray) np.fabs(ndarray)	计算绝对值 计算绝对值（非复数）
np.mean(ndarray)	求平均值
np.sqrt(ndarray)	计算 $x^{0.5}$
np.square(ndarray)	计算 x^2
np.exp(ndarray)	计算 e^x
log、log10、log2、log1p	计算自然对数、底为 10 的 log、底为 2 的 log、底为 (1+x)的 log

np.sign(ndarray)	计算正负号：1（正）、0（0）、-1（负）
np.ceil(ndarray) np.floor(ndarray) np rint(ndarray)	计算大于等于改值的最小整数 计算小于等于该值的最大整数 四舍五入到最近的整数，保留 dtype
np.modf(ndarray)	将数组的小数和整数部分以两个独立的数组方式返回
np.isnan(ndarray)	返回一个判断是否是 NaN 的 bool 型数组
np.isfinite(ndarray) np.isinf(ndarray)	返回一个判断是否有穷（非 inf，非 NaN）的 bool 型数组 返回一个判断是否是无穷的 bool 型数组
cos、cosh、sin、sinh、tan、tanh	普通型和双曲型三角函数
arccos、arccosh、arcsin、arcsinh、arctan、arctanh	反三角函数和双曲型反三角函数
np.logical_not(ndarray)	计算各元素 not x 的真值，相当于-ndarray
多元计算函数	说明
np.add(ndarray, ndarray) np.subtract(ndarray, ndarray) np.multiply(ndarray, ndarray) np.divide(ndarray, ndarray) np.floor_divide(ndarray, ndarray) np.power(ndarray, ndarray) np.mod(ndarray, ndarray)	相加 相减 乘法 除法 圆整除法（丢弃余数） 次方 求模
np.maximum(ndarray, ndarray) np.fmax(ndarray, ndarray) np.minimun(ndarray, ndarray) np.fmin(ndarray, ndarray)	求最大值 求最大值（忽略 NaN） 求最小值 求最小值（忽略 NaN）
np.copysign(ndarray, ndarray)	将参数 2 中的符号赋予参数 1
np.greater(ndarray, ndarray) np.greater_equal(ndarray, ndarray)	> >=

np.less(ndarray, ndarray)	<
np.less_equal(ndarray, ndarray)	<=
np.equal(ndarray, ndarray)	==
np.not_equal(ndarray, ndarray)	!=
logical_and(ndarray, ndarray)	&
logical_or(ndarray, ndarray)	
logical_xor(ndarray, ndarray)	^
np.dot(ndarray, ndarray)	计算两个 ndarray 的矩阵内积
np.ix_([x,y,m,n],...)	生成一个索引器，用于 Fancy indexing(花式索引)
文件读写	说明
np.save(string, ndarray)	将 ndarray 保存到文件名为 [string].npy 的文件中（无压缩）
np.savez(string, ndarray1, ndarray2, ...)	将所有的 ndarray 压缩保存到文件名为[string].npy 的文件中
np.savetxt(sring, ndarray, fmt, newline='\n')	将 ndarray 写入文件，格式为 fmt
np.load(string)	读取文件名 string 的文件内容并转化为 ndarray 对象（或字典对象）
np.loadtxt(string, delimiter)	读取文件名 string 的文件内容，以 delimiter 为分隔符转化为 ndarray

ndarray 函数参考

函数	说明
ndarray.astype(dtype)	转换类型，若转换失败则会出现 TypeError
ndarray.copy()	复制一份 ndarray(新的内存空间)
ndarray.reshape((N,M,...))	将 ndarray 转化为 N*M*... 的多维 ndarray（非 copy）
ndarray.transpose((xIndex,yIndex,...))	根据维索引 xIndex,yIndex... 进行矩阵转置，依赖于 shape，不能用于一维矩阵（非 copy）
ndarray.swapaxes(xIndex,yIndex)	交换维度（非 copy）
计算函数	说明
ndarray.mean(axis=0)	求平均值
ndarray.sum(axis= 0)	求和
ndarray.cumsum(axis=0)	累加
ndarray.cumprod(axis=0)	累乘
ndarray.std()	方差

<code>ndarray.var()</code>	标准差
<code>ndarray.max()</code>	最大值
<code>ndarray.min()</code>	最小值
<code>ndarray.argmax()</code>	最大值索引
<code>ndarray.argmin()</code>	最小值索引
<code>ndarray.any()</code>	是否至少有一个 True
<code>ndarray.all()</code>	是否全部为 True
<code>ndarray.dot(ndarray)</code>	计算矩阵内积
排序函数	说明
<code>ndarray.sort(axis=0)</code>	排序，返回源数据

矩阵转置:

对

```
arr= [[1, 2, 3], [4, 5, 6], [7,8, 9], [10, 11, 12]]
```

进行行列互转

```
print ([[r[col] for r in arr] for col in range(len(arr[0]))])#1
```

```
[list(item) for item in list(zip(*arr))>#2
```

```
print ([[r[col] for r in arr] for col in range(len(arr[0]))])#1
```

```
[list(item) for item in list(zip(*arr))>#2
```

```
np.array(arr).T
```

