numpy 库常用函数

昆明理工大学津桥学院电气与信息工程学院

智能信息技术工作室

目录

numpy 数据类型

名称	描述
bool_	布尔型数据类型（True 或者 False）
int_	默认的整数类型（类似于 C 语言中的 long，int32 或 int64）
intc	与 C 的 int 类型一样，一般是 int32 或 int 64
intp	用于索引的整数类型（类似于 C 的 ssize_t，一般情况下仍然是 int32 或 int64）
int8	字节（-128 to 127）
int16	整数（-32768 to 32767）
int32	整数（-2147483648 to 2147483647）
int64	整数（-9223372036854775808 to 9223372036854775807）
uint8	无符号整数（0 to 255）
uint16	无符号整数（0 to 65535）
uint32	无符号整数（0 to 4294967295）
uint64	无符号整数（0 to 18446744073709551615）
float_	float64 类型的简写
float16	半精度浮点数，包括：1 个符号位，5 个指数位，10 个尾数位
float32	单精度浮点数，包括：1 个符号位，8 个指数位，23 个尾数位
float64	双精度浮点数，包括：1 个符号位，11 个指数位，52 个尾数位

名称	描述
complex_	complex128 类型的简写，即 128 位复数
complex64	复数，表示双 32 位浮点数（实数部分和虚数部分）
complex128	复数，表示双 64 位浮点数（实数部分和虚数部分）

数据生成函数

```
>>>np.arange(5)#生成 0 到 n-1 的 n 个数
array([0, 1, 2, 3, 4])
>>>np.arange(4,9)#指定起始数和终止数
array([4, 5, 6, 7, 8])
>>>np.arange(0,10,2)#指定步长为 2
array([0, 2, 4, 6, 8])
>>>np.arange(3,9,1.5)#步长可以是浮点数
array([3. , 4.5, 6. , 7.5])
>>>np.ones(5)#生成一维的全 1 数组
array([1., 1., 1., 1., 1.])
>>>np.ones((2,3))#二维
[[1. 1. 1.]
 [1. 1. 1.]]
>>>np.ones((2,3,4))#三维
[[[1. 1. 1. 1.]
  [1. 1. 1. 1.]
  [1. 1. 1. 1.]]
 [[1. 1. 1. 1.]
  [1. 1. 1. 1.]
  [1. 1. 1. 1.]]]
np.zeros(shape)用法与 np.ones(shape)类似
>>>np.full((2,4),2)#生成 2 行 4 列的全 2 数组
[[2 2 2 2]
 [2 2 2 2]]
>>>np.eye(3)#生成 n*n 的对角矩阵
[[1. 0. 0.]
 [0. 1. 0.]
```

```
 [0.  0.  1.]]
>>>a=np.eye(3)
>>>np.ones_like(a)
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
```

类似地函数还有 np.zeros_like(a)和 np.full_like(a,val)

```
>>>np.linspace(1,100,50)#在 1-100 之间生成 50 个均匀分布的数据，即间隔相
等
array([   1.        ,    3.02040816,    5.04081633,    7.06122449,
          9.08163265,   11.10204082,   13.12244898,   15.14285714,
         17.16326531,   19.18367347,   21.20408163,   23.2244898 ,
         25.24489796,   27.26530612,   29.28571429,   31.30612245,
         33.32653061,   35.34693878,   37.36734694,   39.3877551 ,
         41.40816327,   43.42857143,   45.44897959,   47.46938776,
         49.48979592,   51.51020408,   53.53061224,   55.55102041,
         57.57142857,   59.59183673,   61.6122449 ,   63.63265306,
         65.65306122,   67.67346939,   69.69387755,   71.71428571,
         73.73469388,   75.75510204,   77.7755102 ,   79.79591837,
         81.81632653,   83.83673469,   85.85714286,   87.87755102,
         89.89795918,   91.91836735,   93.93877551,   95.95918367,
         97.97959184,  100.        ])
np.linspace(1,100,50,endpoint=False)#不包括最后一个数 100
array([  1.  ,   2.98,   4.96,   6.94,   8.92,  10.9 ,  12.88,  14.8
6,
        16.84,  18.82,  20.8 ,  22.78,  24.76,  26.74,  28.72,  30.7
 ,
        32.68,  34.66,  36.64,  38.62,  40.6 ,  42.58,  44.56,  46.5
4,
        48.52,  50.5 ,  52.48,  54.46,  56.44,  58.42,  60.4 ,  62.3
8,
        64.36,  66.34,  68.32,  70.3 ,  72.28,  74.26,  76.24,  78.2
2,
        80.2 ,  82.18,  84.16,  86.14,  88.12,  90.1 ,  92.08,  94.0
6,
        96.04,  98.02])
```

数组维度处理函数

```
a=np.array([[1,2],[3,4]])
b=np.array([[5,6],[7,8]])
np.concatenate((a,b))#数组合并
array([[1, 2],
       [3, 4],
       [5, 6],
       [7, 8]])
np.concatenate((a,b),axis=1)#在第一个轴上合并
array([[1, 2, 5, 6],
       [3, 4, 7, 8]])
np.concatenate((a,b.T))#a 和 b 的转置合并
array([[1, 2],
       [3, 4],
       [5, 7],
       [6, 8]])
>>> b=np.arange(16).reshape(2,4,2)
>>> b.ndim #数组的维度
3
>>> b.shape #数组的形状
(2, 4, 2)
>>> b.size #数组元素个数
16
>>> b.dtype #数组元素的类型
dtype('int32')
>>> b.itemsize #每个数组元素占用的字节数
4
>>> np.arange(16).reshape(4,4)#返回 shape 形状的数组
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

```
>>> np.arange(16).reshape(-1)#扁平化操作，返回一行数组，功能与 flatten()一致
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
>>> np.arange(16).reshape(-1,2)#若不知道数组的形状，用-1 代替行数，让其自动计算行数，生成 2 列的数组
array([[ 0,  1],
       [ 2,  3],
       [ 4,  5],
       [ 6,  7],
       [ 8,  9],
       [10, 11],
       [12, 13],
       [14, 15]])
```

注意 reshape()函数不改变原数组的形状，

```
>>> a=np.arange(16)
>>> a
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
>>> a.reshape(2,8)
array([[ 0,  1,  2,  3,  4,  5,  6,  7],
       [ 8,  9, 10, 11, 12, 13, 14, 15]])
>>> a #数组不变
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
```

resize()函数会改变原数组的形状

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

```
       [ 2,  3],
       [ 4,  5],
       [ 6,  7],
       [ 8,  9],
       [10, 11]])
>>> a=np.arange(16).reshape((2,2,4))
>>> a
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7]],
       [[ 8,  9, 10, 11],
        [12, 13, 14, 15]]])
>>> a.swapaxes(0,2) #第一个维度与第三个维度交换，第二个维度不变，等价于 a.transpose(2,1,0)
array([[[ 0,  8],
        [ 4, 12]],
       [[ 1,  9],
        [ 5, 13]],
       [[ 2, 10],
        [ 6, 14]],
       [[ 3, 11],
        [ 7, 15]]])
```

交换前 7 的下标是[0,1,3]，交换后 7 的下标是[3,1,0]

矩阵的计算

```
>>> b=np.arange(12).reshape(6,2)
>>> b
array([[ 0,  1],
       [ 2,  3],
```

```
       [ 4,  5],
       [ 6,  7],
       [ 8,  9],
       [10, 11]])
>>> b.T
array([[ 0,  2,  4,  6,  8, 10],
       [ 1,  3,  5,  7,  9, 11]])
>>> b.transpose()
array([[ 0,  2,  4,  6,  8, 10],
       [ 1,  3,  5,  7,  9, 11]])
        [ 8,  9, 10, 11, 12, 13, 14, 15]])
>>> a=np.arange(4).reshape(2,2)
>>> a
array([[0, 1],
       [2, 3]])
>>> b=np.arange(5,9).reshape(2,2)
>>> b
array([[5, 6],
       [7, 8]])
>>> a+b #矩阵加法
array([[ 5,  7],
       [ 9, 11]])
>>> a-b #矩阵减法
array([[-5, -5],
       [-5, -5]])
>>> a.dot(b) #矩阵乘法，注意与 a*b 的区别
array([[ 7,  8],
       [31, 36]])
>>> a*b #相同位置上的元素相乘
```

```
array([[ 0,  6],
       [14, 24]])
```

数组元素类型转换

```
>>> a=np.arange(16) #整型
>>> a
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
>>> b=a.astype(float) #转换为浮点型
>>> b
array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10., 11., 12.,
       13., 14., 15.])
>>> a.tolist() #少了 arrary()形式
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
```

数组的索引与切片

```
>>> a=np.arange(16)
>>> a
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
>>> a[1] #下标从一开始
1
>>> a[3:9] #不包含最后一个数
array([3, 4, 5, 6, 7, 8])
>>> a[6:] #从第 6 个元素开始到最后一个
array([ 6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
>>> a[:6] #从第 1 个到第 6 个
array([0, 1, 2, 3, 4, 5])
```

高维数组的用法类似

```
>>> a.resize(2,4,2) #理解为 2 个 4*2 矩阵
>>> a
array([[[ 0,  1],
        [ 2,  3],
        [ 4,  5],
        [ 6,  7]],
       [[ 8,  9],
        [10, 11],
        [12, 13],
        [14, 15]]])
>>> a[:,:1] #选择两个矩阵，每个矩阵的第 1 行所有列，
array([[[0, 1]],
       [[8, 9]]])
>>> a[:,::2] #选择两个矩阵，每个矩阵从第 1 行开始每隔 2 行选择所有列
array([[[ 0,  1],
        [ 4,  5]],
       [[ 8,  9],
        [12, 13]]])
```

数值计算

函数	功能
np.abs(x)	返回 x 的绝对值
np.sqrt(x)	返回 x 的平方根，x 为非负
np.square(x)	返回 x 的平方
np.ceil(x)	返回不小于 x 的最小整数
np.floor(x)	返回不大于 x 的最大整数
np.rint(x)	返回 x 的四舍五入值

np.modf(x)	返回 x 的小数部分和整数部分
np.mod(x,y)	返回 x 对 y 模运算的结果
np.sin(x),np.cos(x),np.tan(x)	返回 x 的对应的三角函数值
np.exp(x)	返回 e 的 x 次方
np.power(x,y)	返回 x 的 y 次方
np.sign(x)	返回 x 的符号，正数返回 1，负数返回-1，零返回 0
np.max(x),np.min(x)	返回 x 的最大值（最小值），x 必须为列表
np.maximum(x,y),np.minimum(x,y)	返回 x,y 的最大值（最小值）

统计函数

```
>>> a=np.arange(10)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a.mean() #平均值
4.5
>>> a.min() #最小值
0
>>> a.max() #最大值
9
>>> a.sum() #求和
45
>>> np.median(a) #中位数
4.5
>>> a.ptp() #最大值与最小值的差
9
>>> np.std(a) #标准差
```

```
2.8722813232690143
>>> np.var(a) #方差
8.25
>>> np.argmax(a) #最大值的下标
9
>>> np.argmin(a) #最小值的下标
0
>>> np.unravel_index(1,(2,5)) #将一维数组转换成二维数组后下标为 1 的元素新的下标值
(0, 1)
>>> np.average(range(1,11), weights=range(10,0,-1)) #加权平均
4.0
```

等价于

```
>>> a=np.arange(1,11)
>>> a
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
>>> b=np.arange(10,0,-1)
>>> b
array([10,  9,  8,  7,  6,  5,  4,  3,  2,  1])
>>> np.sum(a*b)/np.sum(b)
4.0
>>> f = np.array([1, 2, 4, 7, 11, 16])
>>> f
array([ 1,  2,  4,  7, 11, 16])
>>> np.gradient(f) #梯度函数
array([1. , 1.5, 2.5, 3.5, 4.5, 5. ])
```

第一个元素的梯度值为其本身，最后一个元素的梯度值是它减去前一个元素的值，中间其余各元素的梯度值等于后一个元素的值减去前一个元素值再除以 2，比如 2 的梯度值为(4-1)/2

```
>>> c=np.array([[18,49,1,5,26],[40,38,39,46,47],[46,23,6,31,36]])
>>> c
array([[18,  49,   1,   5,  26],
       [40,  38,  39,  46,  47],
       [46,  23,   6,  31,  36]])
>>> np.gradient(c) #二维数组求梯度
[array([[ 22. , -11. ,  38. ,  41. ,  21. ],
       [ 14. , -13. ,   2.5,  13. ,   5. ],
       [  6. , -15. , -33. , -15. , -11. ]]),
 array([[ 31. ,  -8.5, -22. ,  12.5,  21. ],
       [ -2. ,  -0.5,   4. ,   4. ,   1. ],
       [-23. , -20. ,   4. ,  15. ,   5. ]])]
```

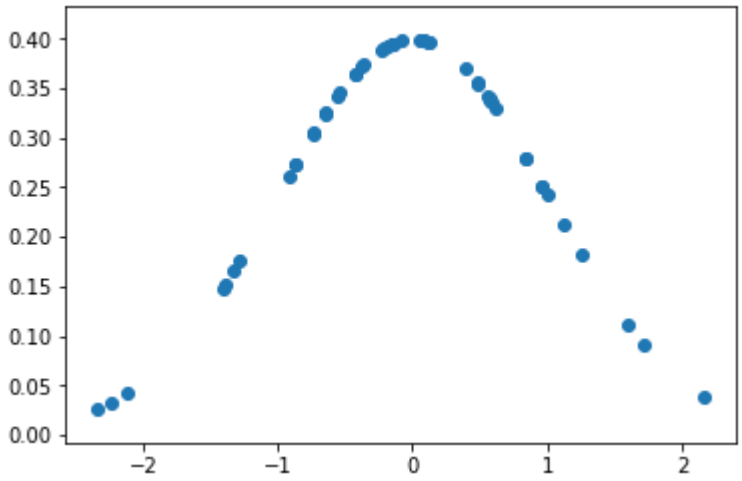
二维数组求梯度的结果是两个数组，其中第一个数组是在第一个维度上求梯度的结果，第二个数组是在第二个维度求梯度的结果，如第一个数组中 22=40-18，14=(46-18)/2，第二个数组中 31=49-18.

随机函数

```
>>> np.random.rand(4,2)#4*2 的浮点数
array([[0.59604968, 0.17794049],
       [0.11143599, 0.30248439],
       [0.4233824 , 0.12969034],
       [0.28665763, 0.64885854]])
>>> np.random.randn(2,5)#从标准正态分布中返回在行维度上的采样点，参数表示两次采样，每次采 5 个点，参数可以是一维或多维
array([[  0.56798466,  -2.60254611,   0.01499502,   0.11006608,
0.59028144],
       [  1.62147223,   0.75339603,   1.55385909,   0.3813979 ,
0.36035391]])
```

比如：

```
import numpy as np
import matplotlib.pyplot as plt
x=np.random.randn(50) #从 N（0，1）中采 50 个点
mu=0
sigma=1
y = 1./(sigma*np.sqrt(2*np.pi))*np.exp(-(x-mu)**2/(2*sigma**2))
plt.scatter(x,y)
plt.show()
```



还可以为非标准正态分布中采样，比如，在 N（3，6.25）中采样

```
2.5 * np.random.randn(2, 4) + 3
array([[ 3.12212888,  7.14469619,  2.12220358,  3.23972579],
       [-0.93781705,  2.04657674, -2.26521597,  5.34359608]])
```

返回的结果中每行数据都是从图像中随机采样 4 个点

```
>>> np.random.randint(10,50,8)#在[10,50)的范围内随机产生 8 个整数
array([43, 45, 15, 25, 33, 38, 48, 42])
```

产生的数据可以是多维的，比如：

```
>>> np.random.randint(1,100,(4,3))在[1,100)中产生 4 行 3 列的随机整数
array([[55, 26,  6],
       [45, 76, 18],
       [11, 41, 65],
```

```
       [85,  6, 62]])
>>> np.random.seed(0)#设置随机数种子，相同的随机数种子可以产生相同的随机数据，不管执行多少遍
>>> np.random.randn(10)
array([ 1.76405235,  0.40015721,  0.97873798,  2.2408932 ,
1.86755799,
       -0.97727788,  0.95008842, -0.15135721, -0.10321885,
0.4105985 ])
>>> a=np.arange(10)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> np.random.shuffle(a) #将数组原地随机改变顺序，改变数组本身的值
>>> a
array([5, 2, 3, 4, 1, 0, 9, 8, 7, 6])
```

若是多维数组，则只随机改变第一个维度的顺序，例如：

```
>>> b=np.arange(16).reshape(4,4)
>>> b
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
>>> np.random.shuffle(b) #改变每行的顺序，列顺序不变
>>> b
array([[ 8,  9, 10, 11],
       [12, 13, 14, 15],
       [ 4,  5,  6,  7],
       [ 0,  1,  2,  3]])
```

类似的函数还有

```
>>> np.random.permutation(10)#直接产生一个特定范围内的乱序数组
```

```
array([4, 1, 6, 7, 2, 8, 5, 9, 0, 3])
```

若是多维数组，也是在第一个维度上打乱顺序

```
>>> c=np.arange(9).reshape(3,3)
>>> c
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
>>> np.random.permutation(c)
array([[0, 1, 2],
       [6, 7, 8],
       [3, 4, 5]])
```

>>> np.random.choice(5,3)#在 0 至 4 的范围内随机抽取 3 个数，范围内每个数被抽到的概率相等

```
array([3, 0, 1])
```

还可以对范围内每个数据设置被抽取的概率

```
>>> np.random.choice(5, 3, p=[0.1, 0, 0.3, 0.6, 0])# 0 的概率是 0.1，1 的概率是 0，以此类推。
array([2, 3, 3], dtype=int64)
>>> np.random.choice(5, 3, replace=False)#结果集不重复
array([1, 4, 0])
>>> name=['Jack','Smith','Jenny','Decker','Rose']#原始数据集不一定是数值型
>>> np.random.choice(name)
'Jack'
```

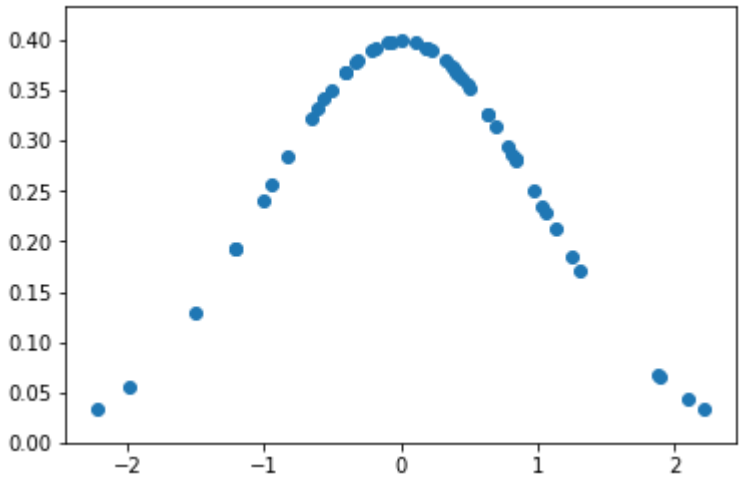
>>> np.random.uniform(0,10,3)#在 0 至 10 的半开区间随机产生一个 3 个在此范围内均匀分布的数据

```
array([6.16933997, 9.43748079, 6.81820299])
```

类似还有随机产生服从正态分布的的数据，比如

```
import numpy as np
```

```
import matplotlib.pyplot as plt
x=np.random.normal(0,1,50) #等价于 np.random.randn(50)，只不过此处设置正态分布的均值与标准差较为方便，第一个参数为均值，第二个参数是标准差，第三个参数为数据规模
mu=0
sigma=1
y = 1./(sigma*np.sqrt(2*np.pi))*np.exp(-(x-mu)**2/(2*sigma**2))
plt.scatter(x,y)
plt.show()
```




```
>>> np.random.poisson(2,10)#随机生成 lamda=2 的服从泊松分布的 10 个数据
array([4, 1, 2, 1, 1, 2, 1, 0, 0, 5])
```

参考文献

[1]URL: https://docs.scipy.org/doc/numpy/

[2]URL: https://blog.csdn.net/WWWQ2386466490/article/details/79938347

[3]URL: https://www.icourse163.org/course/BIT-1001870002

[4]嵩天,礼欣,黄天羽.Python 语言程序设计基础(第 2 版).北京：高等教育出版社，2017 年 2 月

致谢

本文得到机器学习爱好者 V 中众多群友的技术支持，特别感谢中国海洋大学黄海广博士组织的 QQ 群，使得国内大量机器学习爱好者得以集中学习讨论，还有中国科学院大学王楠博士的技术指导，在此一并表示谢意！

后记

Numpy 库非常庞大，本文只列出部分常用函数。

由于作者水平有限，难免有错误和不当之处，欢迎各位 python 爱好者批评指正，如有错误或意见请联系作者：sjgaokm@gmail.com