

使用内置函数创建 ndarray

The screenshot shows a Jupyter Notebook interface with the following content:

```

In [2]: import numpy as np
        my_list = [1, 2, 3, 4, 5]
        np.array(my_list)

Out[2]: array([1, 2, 3, 4, 5])

In [ ]:

```

Below the code cell, there is a text overlay that reads: "In the last video,"

00:00 / 09:32

1x CC

NumPy 的一个非常节省时间的功能是使用内置函数创建 ndarray。借助这些函数，我们只需编写一行代码就能创建某些类型的 ndarray。以下是一些创建 ndarray 的最实用内置函数，你在进行 AI 编程时将遇到这些函数。

我们先创建一个具有指定形状的 ndarray，其中的元素全是 0。为此，我们可以使用 `np.zeros()` 函数。函数 `np.zeros(shape)` 会创建一个全是 0 并且为给定 形状 的 ndarray。因此，例如如果你想创建一个秩为 2 的数组，其中包含 3 行和 4 列，你将以（行，列）的形式将该形状传递给函数，如下示例所示：

```
# We print X
print()
print('X = \n', X)
print()

# We print information about X
print('X has dimensions:', X.shape)
print('X is an object of type:', type(X))
print('The elements in X are of type:', X.dtype)
```

```
X =
[[ 0. 0. 0. 0.]
 [ 0. 0. 0. 0.]
 [ 0. 0. 0. 0.]
```

```
X has dimensions: (3, 4)
X is an object of type: class 'numpy.ndarray'
The elements in X are of type: float64
```

可以看出，`np.zeros()` 函数默认地创建一个 dtype 为 float64 的数组。你可以使用关键字 `dtype` 更改数据类型。

同样，我们可以创建一个具有指定形状的 ndarray，其中的元素全是 1。为此，我们可以使用 `np.ones()` 函数。和 `np.zeros()` 函数一样，`np.ones()` 函数会用一个参数来指定你要创建的 ndarray 的形状。我们来看一个示例：

```
# We create a 3 x 2 ndarray full of ones.
X = np.ones((3,2))

# We print X
print()
print('X = \n', X)
print()

# We print information about X
print('X has dimensions:', X.shape)
print('X is an object of type:', type(X))
print('The elements in X are of type:', X.dtype)
```

```
[ 1. 1.]
[ 1. 1.]]
```

```
X has dimensions: (3, 2)
X is an object of type: class 'numpy.ndarray'
The elements in X are of type: float64
```

可以看出，`np.ones()` 函数也默认地创建一个 dtype 为 float64 的数组。你可以使用关键字 `dtype` 更改数据类型。

我们还可以创建一个具有指定形状的 ndarray，其中的元素全是我们想指定的任何数字。为此，我们可以使用 `np.full()` 函数。`np.full(shape, constant value)` 函数有两个参数。第一个参数是你要创建的 ndarray 的 **形状**，第二个参数是你要向数组中填充的 **常数值**。我们来看一个示例：

```
# We create a 2 x 3 ndarray full of fives.
X = np.full((2,3), 5)

# We print X
print()
print('X = \n', X)
print()

# We print information about X
print('X has dimensions:', X.shape)
print('X is an object of type:', type(X))
print('The elements in X are of type:', X.dtype)
```

```
X =
[[5 5 5]
 [5 5 5]]
```

```
X has dimensions: (2, 3)
X is an object of type: class 'numpy.ndarray'
The elements in X are of type: int64
```

`np.full()` 函数默认地创建一个数据类型和用于填充数组的常数值相同的数组。你可以使用关键字 `dtype` 更改数据类型。

ndarray。因为所有单位矩阵都是方形，因此，`np.eye()` 函数仅接受一个整数作为参数。我们来看一个示例：

```
# We create a 5 x 5 Identity matrix.
X = np.eye(5)

# We print X
print()
print('X = \n', X)
print()

# We print information about X
print('X has dimensions:', X.shape)
print('X is an object of type:', type(X))
print('The elements in X are of type:', X.dtype)
```

```
X =
[[ 1. 0. 0. 0. 0.]
 [ 0. 1. 0. 0. 0.]
 [ 0. 0. 1. 0. 0.]
 [ 0. 0. 0. 1. 0.]
 [ 0. 0. 0. 0. 1.]]
```

```
X has dimensions: (5, 5)
X is an object of type: class 'numpy.ndarray'
The elements in X are of type: float64
```

可以看出，`np.eye()` 函数也默认地创建一个 dtype 为 float64 的数组。你可以使用关键字 `dtype` 更改数据类型。你将在这门课程的线性代数部分深入学习单位矩阵及其用途。我们还可以使用 `np.diag()` 函数创建对角矩阵。对角矩阵是仅在主对角线上有值的方形矩阵。`np.diag()` 函数会创建一个对应于对角矩阵的 ndarray，如以下示例所示：

```
# Create a 4 x 4 diagonal matrix that contains the numbers 10,20,30, and 50
# on its main diagonal
X = np.diag([10,20,30,50])

# We print X
print()
print('X = \n', X)
print()
```

```
[ 0 20 0 0]
[ 0 0 30 0]
[ 0 0 0 50]]
```

NumPy 还允许你创建在给定区间内值均匀分布的 ndarray。NumPy 的 `np.arange()` 函数非常强大，可以传入一个参数、两个参数或三个参数。下面将介绍每种情况，以及如何创建不同种类的 ndarray。

先仅向 `np.arange()` 中传入一个参数。如果只传入一个参数，`np.arange(N)` 将创建一个秩为 1 的 ndarray，其中包含从 0 到 $N - 1$ 的连续整数。因此，注意，如果我希望数组具有介于 0 到 9 之间的整数，则需要将 N 设为 10，而不是将 N 设为 9，如以下示例所示：

```
# We create a rank 1 ndarray that has sequential integers from 0 to 9
x = np.arange(10)

# We print the ndarray
print()
print('x = ', x)
print()

# We print information about the ndarray
print('x has dimensions:', x.shape)
print('x is an object of type:', type(x))
print('The elements in x are of type:', x.dtype)
```

```
x = [0 1 2 3 4 5 6 7 8 9]
```

```
x has dimensions: (10,)
x is an object of type: class 'numpy.ndarray'
The elements in x are of type: int64
```

如果传入两个参数，`np.arange(start, stop)` 将创建一个秩为 1 的 ndarray，其中包含位于半开区间 `[start, stop)` 内并均匀分布的值。也就是说，均匀分布的数字将包括 `start` 数字，但是不包括 `stop` 数字。我们来看一个示例

```
# We print the ndarray
print()
print('x = ', x)
print()

# We print information about the ndarray
print('x has dimensions:', x.shape)
print('x is an object of type:', type(x))
print('The elements in x are of type:', x.dtype)
```

```
x = [4 5 6 7 8 9]
```

```
x has dimensions: (6,)
x is an object of type: class 'numpy.ndarray'
The elements in x are of type: int64
```

可以看出，函数 `np.arange(4,10)` 生成了一个包含 4 但是不含 10 的整数序列。

最后，如果传入三个参数，`np.arange(start,stop,step)` 将创建一个秩为 1 的 ndarray，其中包含位于半开区间 `[start, stop)` 内并均匀分布的值，`step` 表示两个相邻值之间的差。我们来看一个示例：

```
# We create a rank 1 ndarray that has evenly spaced integers from 1 to 13 in steps of 3.
x = np.arange(1,14,3)

# We print the ndarray
print()
print('x = ', x)
print()

# We print information about the ndarray
print('x has dimensions:', x.shape)
print('x is an object of type:', type(x))
print('The elements in x are of type:', x.dtype)
```

```
x = [ 1  4  7 10 13]
```

```
x has dimensions: (5,)
x is an object of type: class 'numpy.ndarray'
```

可以看出，`x` 具有在 1 和 13 之间的序列整数，但是所有相邻值之间的差为 3。

虽然 `np.arange()` 函数允许间隔为非整数，例如 0.3，但是由于浮点数精度有限，输出通常不一致。因此，如果需要非整数间隔，通常建议使用函数 `np.linspace()`。

`np.linspace(start, stop, N)` 函数返回 `N` 个在闭区间 `[start, stop]` 内均匀分布的数字。即 `start` 和 `stop` 值都包括在内。此外注意，在调用 `np.linspace()` 函数时，必须至少以 `np.linspace(start, stop)` 的形式传入两个参数。在此示例中，指定区间内的默认元素数量为 `N=50`。`np.linspace()` 比 `np.arange()` 效果更好，是因为

`np.linspace()` 使用我们希望在特定区间内的元素数量，而不是值之间的间隔。我们来看一些示例：

```
# We create a rank 1 ndarray that has 10 integers evenly spaced between 0 and 25.
x = np.linspace(0,25,10)

# We print the ndarray
print()
print('x = \n', x)
print()

# We print information about the ndarray
print('x has dimensions:', x.shape)
print('x is an object of type:', type(x))
print('The elements in x are of type:', x.dtype)
```

```
x = [ 0. 2.77777778 5.55555556 8.33333333 11.11111111 13.88888889
      16.66666667 19.44444444 22.22222222 25.]
```

```
x has dimensions: (10,)
x is an object of type: class 'numpy.ndarray'
The elements in x are of type: float64
```

从上述示例中可以看出，函数 `np.linspace(0,25,10)` 返回一个 ndarray，其中包含 10 个在闭区间 `[0, 25]` 内均匀分布的元素。还可以看出，在此示例中，起始和结束点 0 和 25 都包含在内。但是，可以不包含区间的结束点（就像 `np.arange()` 函数一样），方法是在 `np.linspace()` 函数中将关键字 `endpoint` 设为 `False`。我们创建和上面一样的 `x` ndarray，但是这次不包含结束点：

```
# with 25 excluded.
x = np.linspace(0,25,10, endpoint = False)

# We print the ndarray
print()
print('x = ', x)
print()

# We print information about the ndarray
print('x has dimensions:', x.shape)
print('x is an object of type:', type(x))
print('The elements in x are of type:', x.dtype)
```

```
x = [ 0. 2.5 5. 7.5 10. 12.5 15. 17.5 20. 22.5]
```

```
x has dimensions: (10,)
x is an object of type: class 'numpy.ndarray'
The elements in x are of type: float64
```

可以看出，因为排除了结束点，值之间的间隔需要更改，因为需要在给定区间内填充 10 个均匀分布的数字。

到目前为止，我们仅使用了内置函数 `np.arange()` 和 `np.linspace()` 来创建秩为 1 的 ndarray。但是，我们可以将这些函数与 `np.reshape()` 函数相结合，创建秩为 2 的任何形状 ndarray。`np.reshape(ndarray, new_shape)` 函数会将给定 ndarray 转换为指定的 `new_shape`。请务必注意：`new_shape` 应该与给定 ndarray 中的元素数量保持一致。例如，你可以将秩为 1 的 6 元素 ndarray 转换为秩为 2 的 3 x 2 ndarray，或秩为 2 的 2 x 3 ndarray，因为这两个秩为 2 的数组元素总数都是 6 个。但是，你无法将秩为 1 的 6 元素 ndarray 转换为秩为 2 的 3 x 3 ndarray，因为这个秩为 2 的数组将包含 9 个元素，比原始 ndarray 中的元素数量多。我们来看一些示例：


```
# We print x
print()
print('Original x = ', x)
print()

# We reshape x into a 4 x 5 ndarray
x = np.reshape(x, (4,5))

# We print the reshaped x
print()
print('Reshaped x = \n', x)
print()

# We print information about the reshaped x
print('x has dimensions:', x.shape)
print('x is an object of type:', type(x))
print('The elements in x are of type:', x.dtype)
```

Original x = [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19]

Reshaped x =

```
[[ 0 1 2 3 4]
 [ 5 6 7 8 9]
 [10 11 12 13 14]
 [15 16 17 18 19]]
```

x has dimensions: (4, 5)

x is an object of type: class 'numpy.ndarray'

The elements in x are of type: int64

NumPy 的一大特性是某些函数还可以当做方法使用。这样我们便能够在一行代码中按顺序应用不同的函数。ndarray 方法和 ndarray 属性相似，它们都使用点记法 (.)。我们来看看如何只用一行代码实现上述示例中的相同结果：

```
Y = np.arange(20).reshape(4, 5)

# We print Y
print()
print('Y = \n', Y)
print()

# We print information about Y
print('Y has dimensions:', Y.shape)
print('Y is an object of type:', type(Y))
print('The elements in Y are of type:', Y.dtype)
```

```
Y =
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]
```

Y has dimensions: (4, 5)

Y is an object of type: class 'numpy.ndarray' The elements in Y are of type: int64

可以看出，我们获得了和之前完全一样的结果。注意，当我们将 `reshape()` 当做方法使用时，它应用为 `ndarray.reshape(new_shape)`。这样会将 `ndarray` 转换为指定形状 `new_shape`。和之前一样，请注意，`new_shape` 应该与 `ndarray` 中的元素数量保持一致。在上述示例中，函数 `np.arange(20)` 创建了一个 `ndarray` 并当做将被 `reshape()` 方法调整形状的 `ndarray`。因此，如果将 `reshape()` 当做方法使用，我们不需要将 `ndarray` 当做参数传递给 `reshape()` 函数，只需传递 `new_shape` 参数。

同样，我们也可以使用 `reshape()` 与 `np.linspace()` 创建秩为 2 的数组，如以下示例所示。



```
X = np.linspace(0,50,10, endpoint=False).reshape(5,2)
```

```
# We print X
print()
print('X = \n', X)
print()

# We print information about X
print('X has dimensions:', X.shape)
print('X is an object of type:', type(X))
print('The elements in X are of type:', X.dtype)
```

```
X =
[[ 0.  5.]
 [10. 15.]
 [20. 25.]
 [30. 35.]
 [40. 45.]]
```

```
X has dimensions: (5, 2)
```

```
X is an object of type: class 'numpy.ndarray' The elements in X are of type:
float64
```

我们将创建的最后一种 ndarray 是随机 ndarray。随机 ndarray 是包含随机数字的数组。在机器学习中，通常需要创建随机指标，例如，在初始化神经网络的权重时。NumPy 提供了各种随机函数来帮助我们创建任何形状 of 随机 ndarray。

我们先使用 `np.random.random(shape)` 函数创建具有给定 形状 的 ndarray，其中包含位于半开区间 `[0.0, 1.0)` 内的随机浮点数。



```
X = np.random.random((3,3))
```

```
# We print X
print()
print('X = \n', X)
print()

# We print information about X
print('X has dimensions:', X.shape)
print('X is an object of type:', type(X))
print('The elements in x are of type:', X.dtype)
```

```
X =
[[ 0.12379926 0.52943854 0.3443525 ]
 [ 0.11169547 0.82123909 0.52864397]
 [ 0.58244133 0.21980803 0.69026858]]
```

```
X has dimensions: (3, 3)
```

```
X is an object of type: class 'numpy.ndarray' The elements in X are of type:
float64
```

NumPy 还允许我们创建由特定区间内的随机整数构成的 ndarray。函数 `np.random.randint(start, stop, size = shape)` 会创建一个具有给定 形状 的 ndarray , 其中包含在半开区间 `[start, stop)` 内的随机整数。我们来看一个示例 :

```
# We create a 3 x 2 ndarray with random integers in the half-open interval
[4, 15).
```

```
X = np.random.randint(4,15,size=(3,2))
```

```
# We print X
print()
print('X = \n', X)
print()
```

```
# We print information about X
print('X has dimensions:', X.shape)
print('X is an object of type:', type(X))
print('The elements in X are of type:', X.dtype)
```

```
X =
[[ 7 11]
```

X has dimensions: (3, 2)

X is an object of type: class 'numpy.ndarray' The elements in X are of type: int64

在某些情况下，你可能需要创建由满足特定统计学特性的随机数字组成的 ndarray。例如，你可能希望 ndarray 中的随机数字平均值为 0。NumPy 使你能够创建从各种概率分布中抽样的数字组成的随机 ndarray。例如，函数

`np.random.normal(mean, standard deviation, size=shape)` 会创建一个具有给定形状的 ndarray，其中包含从 正态 高斯分布（具有给定 均值 和 标准差）中抽样的随机数字。我们来创建一个 1,000 x 1,000 ndarray，其中包含从正态分布（均值为 0，标准差为 0.1）中随机抽样的浮点数。

```
# We create a 1000 x 1000 ndarray of random floats drawn from normal (Gaussian) distribution
# with a mean of zero and a standard deviation of 0.1.
X = np.random.normal(0, 0.1, size=(1000,1000))

# We print X
print()
print('X = \n', X)
print()

# We print information about X
print('X has dimensions:', X.shape)
print('X is an object of type:', type(X))
print('The elements in X are of type:', X.dtype)
print('The elements in X have a mean of:', X.mean())
print('The maximum value in X is:', X.max())
print('The minimum value in X is:', X.min())
print('X has', (X < 0).sum(), 'negative numbers')
print('X has', (X > 0).sum(), 'positive numbers')
```

X =

```
[[ 0.04218614 0.03247225 -0.02936003 ..., 0.01586796 -0.05599115 -0.03630946]
 [ 0.13879995 -0.01583122 -0.16599967 ..., 0.01859617 -0.08241612 0.09684025]
 [ 0.14422252 -0.11635985 -0.04550231 ..., -0.09748604 -0.09350044 0.02514799]
 ...,
 [-0.10472516 -0.04643974 0.08856722 ..., -0.02096011 -0.02946155 0.12930844]
 [-0.26596955 0.0829783 0.11032549 ..., -0.14492074 -0.00113646 -0.03566034]
 [-0.12044482 0.20355356 0.13637195 ..., 0.06047196 -0.04170031 -0.04957684]]
```



float64

The elements in X have a mean of: -0.000121576684405

The maximum value in X is: 0.476673923106

The minimum value in X is: -0.499114224706 X 具有 500562 个负数 X 具有 499438 个正数

可以看出，ndarray 中的随机数字的平均值接近 0，X 中的最大值和最小值与 0（平均值）保持对称，正数和负数的数量很接近。

下一项