用于视觉识别的CS231n卷积神经网络

Python Numpy教程

本教程由Justin Johnson提供。

我们将在本课程中将Python编程语言用于所有作业。Python本身就是一种很棒的通用编程语言，但是在一些流行的库（numpy，scipy，matplotlib）的帮助下，它成为科学计算的强大环境。

我们希望你们中的许多人都有一些Python和numpy的经验; 对于其他人来说，本节将作为Python编程语言和Python用于科学计算的快速速成课程。

你们中的一些人可能已经掌握了Matlab的知识，在这种情况下我们也推荐使用Matlab用户页面的numpy。

您还可以在这里找到由Volodymyr Kuleshov和Isaac Caswell为CS 228创建的本教程的IPython笔记本版本。

目录：

蟒蛇

Python是一种高级动态类型的多范式编程语言。Python代码通常被称为伪代码，因为它允许您在非常少的代码行中表达非常强大的想法，同时具有非常可读性。作为示例，这里是Python中经典快速排序算法的实现：

```python
def quicksort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quicksort(left) + middle + quicksort(right)

print(quicksort([3,6,8,10,1,2,1]))
# Prints "[1, 1, 2, 3, 6, 8, 10]"
```

Python版本

目前有两种不同的受支持版本的Python，2.7和3.5。有点令人困惑的是，Python 3.0引入了许多向后兼容的语言更改，因此为2.7编写的代码可能无法在3.5下运行，反之亦然。对于这个类，所有代码都将使用Python 3.5。

您可以通过运行在命令行中检查Python版本 `python --version`。

基本数据类型

与大多数语言一样，Python有许多基本类型，包括整数，浮点数，布尔值和字符串。这些数据类型的行为方式与其他编程语言相似。

数字：整数和浮点数的工作方式与其他语言相同：

```python
x = 3
print(type(x)) # Prints "<class 'int'>"
print(x)       # Prints "3"
print(x + 1)   # Addition; prints "4"
print(x - 1)   # Subtraction; prints "2"
print(x * 2)   # Multiplication; prints "6"
print(x ** 2)  # Exponentiation; prints "9"
x += 1
print(x)  # Prints "4"
x *= 2
print(x)  # Prints "8"
```

```
y = 2.5
print(type(y)) # Prints "<class 'float'>"
print(y, y + 1, y * 2, y ** 2) # Prints "2.5 3.5 5.0 6.25"
```

请注意，与许多语言不同，Python没有一元increment（`x++`）或减量（`x--`）运算符。

Python还有复杂数字的内置类型; 您可以在文档中找到所有详细信息。

布尔： Python中实现所有通常的运营商的布尔逻辑的，但使用英语词语而非符号（`&&`，`||`等）：

```
t = True
f = False
print(type(t)) # Prints "<class 'bool'>"
print(t and f) # Logical AND; prints "False"
print(t or f)  # Logical OR; prints "True"
print(not t)   # Logical NOT; prints "False"
print(t != f)  # Logical XOR; prints "True"
```

字符串： Python对字符串有很好的支持：

```
hello = 'hello'    # String literals can use single quotes
world = "world"    # or double quotes; it does not matter.
print(hello)       # Prints "hello"
print(len(hello))  # String length; prints "5"
hw = hello + ' ' + world  # String concatenation
print(hw)  # prints "hello world"
hw12 = '%s %s %d' % (hello, world, 12)  # sprintf style string formatting
print(hw12)  # prints "hello world 12"
```

String对象有很多有用的方法; 例如：

```
s = "hello"
print(s.capitalize())  # Capitalize a string; prints "Hello"
print(s.upper())       # Convert a string to uppercase; prints "HELLO"
print(s.rjust(7))      # Right-justify a string, padding with spaces; prints "  hello"
print(s.center(7))     # Center a string, padding with spaces; prints " hello "
print(s.replace('l', '(ell)'))  # Replace all instances of one substring with another;
                                # prints "he(ell)(ell)o"
print('  world '.strip())  # Strip leading and trailing whitespace; prints "world"
```

您可以在文档中找到所有字符串方法的列表。

集装箱

Python包含几种内置容器类型：列表，字典，集合和元组。

清单

列表是Python的等效数组，但是可以调整大小并且可以包含不同类型的元素：

```
xs = [3, 1, 2]    # Create a list
print(xs, xs[2])  # Prints "[3, 1, 2] 2"
print(xs[-1])     # Negative indices count from the end of the list; prints "2"
xs[2] = 'foo'     # Lists can contain elements of different types
print(xs)         # Prints "[3, 1, 'foo']"
xs.append('bar')  # Add a new element to the end of the list
print(xs)         # Prints "[3, 1, 'foo', 'bar']"
x = xs.pop()      # Remove and return the last element of the list
print(x, xs)      # Prints "bar [3, 1, 'foo']"
```

像往常一样，您可以在文档中找到有关列表的所有详细信息 。

切片： 除了一次访问一个列表元素外，Python还提供了访问子列表的简明语法; 这被称为*切片*：

```
nums = list(range(5))     # range is a built-in function that creates a list of integers
print(nums)               # Prints "[0, 1, 2, 3, 4]"
print(nums[2:4])          # Get a slice from index 2 to 4 (exclusive); prints "[2, 3]"
print(nums[2:])           # Get a slice from index 2 to the end; prints "[2, 3, 4]"
print(nums[:2])           # Get a slice from the start to index 2 (exclusive); prints "
print(nums[:])            # Get a slice of the whole list; prints "[0, 1, 2, 3, 4]"
print(nums[:-1])          # Slice indices can be negative; prints "[0, 1, 2, 3]"
nums[2:4] = [8, 9]        # Assign a new sublist to a slice
print(nums)               # Prints "[0, 1, 8, 9, 4]"
```

我们将在numpy数组的上下文中再次看到切片。

循环：您可以循环遍历列表的元素，如下所示：

```
animals = ['cat', 'dog', 'monkey']
for animal in animals:
    print(animal)
# Prints "cat", "dog", "monkey", each on its own line.
```

如果要访问循环体内每个元素的索引，请使用内置 `enumerate` 函数：

```
animals = ['cat', 'dog', 'monkey']
for idx, animal in enumerate(animals):
    print('#%d: %s' % (idx + 1, animal))
# Prints "#1: cat", "#2: dog", "#3: monkey", each on its own line
```

列表推导： 编程时，我们经常想要将一种数据转换为另一种数据。举个简单的例子，考虑以下计算平方数的代码：

```
nums = [0, 1, 2, 3, 4]
squares = []
for x in nums:
    squares.append(x ** 2)
print(squares)   # Prints [0, 1, 4, 9, 16]
```

您可以使用**列表推导**使此代码更简单：

```
nums = [0, 1, 2, 3, 4]
squares = [x ** 2 for x in nums]
print(squares)   # Prints [0, 1, 4, 9, 16]
```

列表推导还可以包含条件：

```
nums = [0, 1, 2, 3, 4]
even_squares = [x ** 2 for x in nums if x % 2 == 0]
print(even_squares)  # Prints "[0, 4, 16]"
```

字典

字典存储（键，值）对，类似于`Map`Java或Javascript中的对象。你可以像这样使用它：

```
d = {'cat': 'cute', 'dog': 'furry'}  # Create a new dictionary with some data
print(d['cat'])       # Get an entry from a dictionary; prints "cute"
print('cat' in d)     # Check if a dictionary has a given key; prints "True"
d['fish'] = 'wet'     # Set an entry in a dictionary
print(d['fish'])      # Prints "wet"
# print(d['monkey'])  # KeyError: 'monkey' not a key of d
print(d.get('monkey', 'N/A'))  # Get an element with a default; prints "N/A"
print(d.get('fish', 'N/A'))    # Get an element with a default; prints "wet"
del d['fish']         # Remove an element from a dictionary
print(d.get('fish', 'N/A')) # "fish" is no longer a key; prints "N/A"
```

您可以在文档中找到有关字典 的所有信息。

循环：很容易迭代字典中的键：

```
d = {'person': 2, 'cat': 4, 'spider': 8}
for animal in d:
    legs = d[animal]
    print('A %s has %d legs' % (animal, legs))
# Prints "A person has 2 legs", "A cat has 4 legs", "A spider has 8 legs"
```

如果要访问密钥及其对应的值，请使用以下`items`方法：

```
d = {'person': 2, 'cat': 4, 'spider': 8}
for animal, legs in d.items():
    print('A %s has %d legs' % (animal, legs))
# Prints "A person has 2 legs", "A cat has 4 legs", "A spider has 8 legs"
```

字典理解： 这些与列表理解类似，但允许您轻松构建字典。例如：

```
nums = [0, 1, 2, 3, 4]
even_num_to_square = {x: x ** 2 for x in nums if x % 2 == 0}
print(even_num_to_square)  # Prints "{0: 0, 2: 4, 4: 16}"
```

集

集合是不同元素的无序集合。举个简单的例子，请考虑以下事项：

```
animals = {'cat', 'dog'}
print('cat' in animals)   # Check if an element is in a set; prints "True"
print('fish' in animals)  # prints "False"
animals.add('fish')       # Add an element to a set
print('fish' in animals)  # Prints "True"
print(len(animals))       # Number of elements in a set; prints "3"
animals.add('cat')        # Adding an element that is already in the set does nothing
print(len(animals))       # Prints "3"
animals.remove('cat')     # Remove an element from a set
print(len(animals))       # Prints "2"
```

像往常一样，您可以在文档中找到有关集合 的所有信息。

循环： 对集合进行迭代与迭代列表具有相同的语法; 但是由于集合是无序的，因此您无法对访问集合元素的顺序进行假设：

```
animals = {'cat', 'dog', 'fish'}
for idx, animal in enumerate(animals):
    print('#%d: %s' % (idx + 1, animal))
# Prints "#1: fish", "#2: dog", "#3: cat"
```

设置理解： 像列表和词典一样，我们可以使用集合理解轻松构建集合：

```
from math import sqrt
nums = {int(sqrt(x)) for x in range(30)}
print(nums)  # Prints "{0, 1, 2, 3, 4, 5}"
```

元组

元组是（不可变的）有序值列表。元组在很多方面类似于列表; 其中一个最重要的区别是元组可以用作字典中的键和集合的元素，而列表则不能。这是一个简单的例子：

```
d = {(x, x + 1): x for x in range(10)}  # Create a dictionary with tuple keys
t = (5, 6)        # Create a tuple
print(type(t))    # Prints "<class 'tuple'>"
print(d[t])       # Prints "5"
print(d[(1, 2)])  # Prints "1"
```

该文档包含有关元组的更多信息。

功能

Python函数是使用`def`关键字定义的。例如：

```
def sign(x):
    if x > 0:
        return 'positive'
    elif x < 0:
        return 'negative'
    else:
        return 'zero'

for x in [-1, 0, 1]:
    print(sign(x))
# Prints "negative", "zero", "positive"
```

我们经常定义函数来获取可选的关键字参数，如下所示：

```
def hello(name, loud=False):
    if loud:
        print('HELLO, %s!' % name.upper())
    else:
        print('Hello, %s' % name)

hello('Bob') # Prints "Hello, Bob"
hello('Fred', loud=True)  # Prints "HELLO, FRED!"
```

有关Python函数的更多信息 ，请参阅文档。

类

在Python中定义类的语法很简单：

```
class Greeter(object):

    # Constructor
    def __init__(self, name):
        self.name = name  # Create an instance variable

    # Instance method
    def greet(self, loud=False):
        if loud:
            print('HELLO, %s!' % self.name.upper())
        else:
            print('Hello, %s' % self.name)

g = Greeter('Fred')  # Construct an instance of the Greeter class
g.greet()            # Call an instance method; prints "Hello, Fred"
g.greet(loud=True)   # Call an instance method; prints "HELLO, FRED!"
```

您可以在文档中阅读有关Python类的更多信息 。

NumPy的

Numpy是Python中科学计算的核心库。它提供了一个高性能的多维数组对象，以及用于处理这些数组的工具。如果您已经熟悉MATLAB，那么您可能会发现 本教程对Numpy入门非常有用。

数组

numpy数组是一个值网格，所有类型都相同，并由非负整数元组索引。维数是数组的*排名*;数组的*形状*是一个整数元组，给出了每个维度的数组大小。

我们可以从嵌套的Python列表初始化numpy数组，并使用方括号访问元素：

```
import numpy as np

a = np.array([1, 2, 3])   # Create a rank 1 array
print(type(a))            # Prints "<class 'numpy.ndarray'>"
print(a.shape)            # Prints "(3,)"
print(a[0], a[1], a[2])   # Prints "1 2 3"
a[0] = 5                  # Change an element of the array
print(a)                  # Prints "[5, 2, 3]"

b = np.array([[1,2,3],[4,5,6]])    # Create a rank 2 array
print(b.shape)                     # Prints "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0])   # Prints "1 2 4"
```

Numpy还提供了许多创建数组的函数：

```
import numpy as np

a = np.zeros((2,2))   # Create an array of all zeros
print(a)              # Prints "[[ 0.  0.]
                      #          [ 0.  0.]]"

b = np.ones((1,2))    # Create an array of all ones
print(b)              # Prints "[[ 1.  1.]]"

c = np.full((2,2), 7)  # Create a constant array
print(c)               # Prints "[[ 7.  7.]
                       #          [ 7.  7.]]"

d = np.eye(2)         # Create a 2x2 identity matrix
print(d)              # Prints "[[ 1.  0.]
                      #          [ 0.  1.]]"

e = np.random.random((2,2))  # Create an array filled with random values
print(e)                     # Might print "[[ 0.91940167  0.08143941]
                             #               [ 0.68744134  0.87236687]]"
```

您可以在文档中阅读有关其他数组创建方法 的信息。

数组索引

Numpy提供了几种索引数组的方法。

切片：与Python列表类似，可以切割numpy数组。由于数组可能是多维的，因此必须为数组的每个维指定一个切片：

```
import numpy as np

# Create the following rank 2 array with shape (3, 4)
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Use slicing to pull out the subarray consisting of the first 2 rows
# and columns 1 and 2; b is the following array of shape (2, 2):
# [[2 3]
#  [6 7]]
b = a[:2, 1:3]

# A slice of an array is a view into the same data, so modifying it
# will modify the original array.
print(a[0, 1])   # Prints "2"
b[0, 0] = 77     # b[0, 0] is the same piece of data as a[0, 1]
print(a[0, 1])   # Prints "77"
```

您还可以将整数索引与切片索引混合使用。但是，这样做会产生比原始数组更低级别的数组。请注意，这与MATLAB处理数组切片的方式完全不同：

```
import numpy as np

# Create the following rank 2 array with shape (3, 4)
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Two ways of accessing the data in the middle row of the array.
# Mixing integer indexing with slices yields an array of lower rank,
# while using only slices yields an array of the same rank as the
# original array:
row_r1 = a[1, :]    # Rank 1 view of the second row of a
row_r2 = a[1:2, :]  # Rank 2 view of the second row of a
print(row_r1, row_r1.shape)  # Prints "[5 6 7 8] (4,)"
print(row_r2, row_r2.shape)  # Prints "[[5 6 7 8]] (1, 4)"

# We can make the same distinction when accessing columns of an array:
col_r1 = a[:, 1]
```

```
col_r2 = a[:, 1:2]
print(col_r1, col_r1.shape)  # Prints "[ 2  6 10] (3,)"
print(col_r2, col_r2.shape)  # Prints "[[ 2]
                             #          [ 6]
                             #          [10]] (3, 1)"
```

整数数组索引： 使用切片索引到numpy数组时，生成的数组视图将始终是原始数组的子数组。相反，整数数组索引允许您使用另一个数组中的数据构造任意数组。这是一个例子：

```
import numpy as np

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

# An example of integer array indexing.
# The returned array will have shape (3,) and
print(a[[0, 1, 2], [0, 1, 0]])  # Prints "[1 4 5]"

# The above example of integer array indexing is equivalent to this:
print(np.array([a[0, 0], a[1, 1], a[2, 0]]))  # Prints "[1 4 5]"

# When using integer array indexing, you can reuse the same
# element from the source array:
print(a[[0, 0], [1, 1]])  # Prints "[2 2]"

# Equivalent to the previous integer array indexing example
print(np.array([a[0, 1], a[0, 1]]))  # Prints "[2 2]"
```

整数数组索引的一个有用技巧是从矩阵的每一行中选择或改变一个元素：

```
import numpy as np

# Create a new array from which we will select elements
a = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])

print(a)  # prints "array([[ 1,  2,  3],
          #                [ 4,  5,  6],
          #                [ 7,  8,  9],
          #                [10, 11, 12]])"

# Create an array of indices
b = np.array([0, 2, 0, 1])

# Select one element from each row of a using the indices in b
print(a[np.arange(4), b])  # Prints "[ 1  6  7 11]"

# Mutate one element from each row of a using the indices in b
```

```
a[np.arange(4), b] += 10

print(a)  # prints "array([[11,  2,  3],
          #                [ 4,  5, 16],
          #                [17,  8,  9],
          #                [10, 21, 12]])
```

布尔数组索引： 布尔数组索引允许您选择数组的任意元素。通常，这种类型的索引用于选择满足某些条件的数组元素。这是一个例子：

```
import numpy as np

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

bool_idx = (a > 2)   # Find the elements of a that are bigger than 2;
                     # this returns a numpy array of Booleans of the same
                     # shape as a, where each slot of bool_idx tells
                     # whether that element of a is > 2.

print(bool_idx)      # Prints "[[False False]
                     #          [ True  True]
                     #          [ True  True]]"

# We use boolean array indexing to construct a rank 1 array
# consisting of the elements of a corresponding to the True values
# of bool_idx
print(a[bool_idx])  # Prints "[3 4 5 6]"

# We can do all of the above in a single concise statement:
print(a[a > 2])     # Prints "[3 4 5 6]"
```

为简洁起见，我们遗漏了很多关于numpy数组索引的细节; 如果你想了解更多，你应该 阅读文档。

数据类型

每个numpy数组都是相同类型元素的网格。Numpy提供了一组可用于构造数组的大量数值数据类型。Numpy在创建数组时尝试猜测数据类型，但构造数组的函数通常还包含一个可选参数来显式指定数据类型。这是一个例子：

```
import numpy as np

x = np.array([1, 2])   # Let numpy choose the datatype
print(x.dtype)         # Prints "int64"

x = np.array([1.0, 2.0])   # Let numpy choose the datatype
```

```
print(x.dtype)             # Prints "float64"

x = np.array([1, 2], dtype=np.int64)    # Force a particular datatype
print(x.dtype)                          # Prints "int64"
```

您可以在文档中阅读有关numpy数据类型的所有内容。

数组数学

基本数学函数在数组上以元素方式运行，既可以作为运算符重载，也可以作为numpy模块中的函数：

```
import numpy as np

x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

# Elementwise sum; both produce the array
# [[ 6.0  8.0]
#  [10.0 12.0]]
print(x + y)
print(np.add(x, y))

# Elementwise difference; both produce the array
# [[-4.0 -4.0]
#  [-4.0 -4.0]]
print(x - y)
print(np.subtract(x, y))

# Elementwise product; both produce the array
# [[ 5.0 12.0]
#  [21.0 32.0]]
print(x * y)
print(np.multiply(x, y))

# Elementwise division; both produce the array
# [[ 0.2         0.33333333]
#  [ 0.42857143  0.5       ]]
print(x / y)
print(np.divide(x, y))

# Elementwise square root; produces the array
# [[ 1.          1.41421356]
#  [ 1.73205081  2.        ]]
print(np.sqrt(x))
```

请注意，与MATLAB不同，`*`是元素乘法，而不是矩阵乘法。我们使用该`dot`函数来计算向量的内积，将向量乘以矩阵，并乘以矩阵。`dot`既可以作为numpy模块中的函数，也可以作为数组对象的实例方法：

```
import numpy as np

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

v = np.array([9,10])
w = np.array([11, 12])

# Inner product of vectors; both produce 219
print(v.dot(w))
print(np.dot(v, w))

# Matrix / vector product; both produce the rank 1 array [29 67]
print(x.dot(v))
print(np.dot(x, v))

# Matrix / matrix product; both produce the rank 2 array
# [[19 22]
#  [43 50]]
print(x.dot(y))
print(np.dot(x, y))
```

Numpy提供了许多用于在数组上执行计算的有用函数; 其中一个最有用的是`sum`：

```
import numpy as np

x = np.array([[1,2],[3,4]])

print(np.sum(x))  # Compute sum of all elements; prints "10"
print(np.sum(x, axis=0))  # Compute sum of each column; prints "[4 6]"
print(np.sum(x, axis=1))  # Compute sum of each row; prints "[3 7]"
```

您可以在文档中找到numpy提供的完整数学函数列表。

除了使用数组计算数学函数之外，我们经常需要重新整形或以其他方式操纵数组中的数据。这种操作的最简单的例子是转置矩阵; 要转置矩阵，只需使用`T`数组对象的属性：

```
import numpy as np

x = np.array([[1,2], [3,4]])
print(x)    # Prints "[[1 2]
```

```
           #          [3 4]]"
print(x.T)  # Prints "[[1 3]
            #          [2 4]]"

# Note that taking the transpose of a rank 1 array does nothing:
v = np.array([1,2,3])
print(v)    # Prints "[1 2 3]"
print(v.T)  # Prints "[1 2 3]"
```

Numpy提供了更多用于操作数组的函数; 您可以在文档中看到完整列表 。

广播

广播是一种强大的机制，允许numpy在执行算术运算时使用不同形状的数组。我们经常有一个较小的数组和一个较大的数组，我们希望多次使用较小的数组来对较大的数组执行某些操作。

例如，假设我们想要向矩阵的每一行添加一个常量向量。我们可以这样做：

```
import numpy as np

# We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = np.empty_like(x)   # Create an empty matrix with the same shape as x

# Add the vector v to each row of the matrix x with an explicit loop
for i in range(4):
    y[i, :] = x[i, :] + v

# Now y is the following
# [[ 2  2  4]
#  [ 5  5  7]
#  [ 8  8 10]
#  [11 11 13]]
print(y)
```

这有效; 但是当矩阵 `x` 非常大时，在Python中计算显式循环可能会很慢。请注意，将向量添加 `v` 到矩阵的每一行 `x` 等同于 `vv` 通过堆叠多个 `v` 垂直副本来形成矩阵，然后执行和的元素 `x` 和求和 `vv` 。我们可以像这样实现这种方法：

```
import numpy as np

# We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
```

```
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
vv = np.tile(v, (4, 1))    # Stack 4 copies of v on top of each other
print(vv)                  # Prints "[[1 0 1]
                           #          [1 0 1]
                           #          [1 0 1]
                           #          [1 0 1]]"
y = x + vv  # Add x and vv elementwise
print(y)  # Prints "[[ 2  2  4
          #          [ 5  5  7]
          #          [ 8  8 10]
          #          [11 11 13]]"
```

Numpy广播允许我们执行此计算而无需实际创建多个副本`v`。考虑这个版本，使用广播：

```
import numpy as np

# We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = x + v  # Add v to each row of x using broadcasting
print(y)  # Prints "[[ 2  2  4]
          #          [ 5  5  7]
          #          [ 8  8 10]
          #          [11 11 13]]"
```

该生产线`y = x + v`的工作，即使`x`有形状`(4, 3)`和`v`具有形状`(3,)`由于广播; 这条线就像`v`实际上有形状一样`(4, 3)`，每行都是一个副本`v`，并且元素是按元素执行的。

将两个数组一起广播遵循以下规则：

1. 如果数组不具有相同的等级，则将较低等级数组的形状添加为1，直到两个形状具有相同的长度。
2. 如果两个数组在维度中具有相同的大小，或者如果其中一个数组在该维度中具有大小1，则称这两个数组在维度上是*兼容的*。
3. 如果阵列在所有维度上兼容，则可以一起广播。
4. 在广播之后，每个阵列的行为就好像它的形状等于两个输入数组的形状的元素最大值。
5. 在一个数组的大小为1且另一个数组的大小大于1的任何维度中，第一个数组的行为就像沿着该维度复制一样

如果此解释没有意义，请尝试阅读 文档 或此解释中的说明。

支持广播的功能称为*通用功能*。您可以在文档中找到所有通用功能的列表 。

以下是广播的一些应用：

```
import numpy as np

# Compute outer product of vectors
v = np.array([1,2,3])  # v has shape (3,)
w = np.array([4,5])    # w has shape (2,)
# To compute an outer product, we first reshape v to be a column
# vector of shape (3, 1); we can then broadcast it against w to yield
# an output of shape (3, 2), which is the outer product of v and w:
# [[ 4  5]
#  [ 8 10]
#  [12 15]]
print(np.reshape(v, (3, 1)) * w)

# Add a vector to each row of a matrix
x = np.array([[1,2,3], [4,5,6]])
# x has shape (2, 3) and v has shape (3,) so they broadcast to (2, 3),
# giving the following matrix:
# [[2 4 6]
#  [5 7 9]]
print(x + v)

# Add a vector to each column of a matrix
# x has shape (2, 3) and w has shape (2,).
# If we transpose x then it has shape (3, 2) and can be broadcast
# against w to yield a result of shape (3, 2); transposing this result
# yields the final result of shape (2, 3) which is the matrix x with
# the vector w added to each column. Gives the following matrix:
# [[ 5  6  7]
#  [ 9 10 11]]
print((x.T + w).T)
# Another solution is to reshape w to be a column vector of shape (2, 1);
# we can then broadcast it directly against x to produce the same
# output.
print(x + np.reshape(w, (2, 1)))

# Multiply a matrix by a constant:
# x has shape (2, 3). Numpy treats scalars as arrays of shape ();
# these can be broadcast together to shape (2, 3), producing the
# following array:
# [[ 2  4  6]
#  [ 8 10 12]]
print(x * 2)
```

广播通常会使您的代码更简洁，更快速，因此您应该尽可能地使用它。

Numpy文档

这个简短的概述涉及了许多关于numpy需要了解的重要事项，但还远未完成。查看 numpy参考资料 ，了解有关numpy的更多信息。

SciPy的

Numpy提供了一个高性能的多维数组和基本工具来计算和操作这些数组。 SciPy 以此为基础，提供了大量在numpy数组上运行的函数，可用于不同类型的科学和工程应用程序。

熟悉SciPy的最佳方法是 浏览文档。我们将重点介绍您可能会发现对此类有用的SciPy的一些部分。

图像操作

SciPy提供了一些处理图像的基本功能。例如，它具有将图像从磁盘读取到numpy数组，将numpy数组作为图像写入磁盘以及调整图像大小的功能。这是一个展示这些功能的简单示例：

```
from scipy.misc import imread, imsave, imresize

# Read an JPEG image into a numpy array
img = imread('assets/cat.jpg')
print(img.dtype, img.shape)  # Prints "uint8 (400, 248, 3)"

# We can tint the image by scaling each of the color channels
# by a different scalar constant. The image has shape (400, 248, 3);
# we multiply it by the array [1, 0.95, 0.9] of shape (3,);
# numpy broadcasting means that this leaves the red channel unchanged,
# and multiplies the green and blue channels by 0.95 and 0.9
# respectively.
img_tinted = img * [1, 0.95, 0.9]

# Resize the tinted image to be 300 by 300 pixels.
img_tinted = imresize(img_tinted, (300, 300))

# Write the tinted image back to disk
imsave('assets/cat_tinted.jpg', img_tinted)
```

左：原始图像。右图：着色和调整大小的图像。

MATLAB文件

功能`scipy.io.loadmat`和`scipy.io.savemat`允许您读取和写入MATLAB文件。您可以在文档中阅读它们。

点之间的距离

SciPy定义了一些用于计算点集之间距离的有用函数。

该函数`scipy.spatial.distance.pdist`计算给定集合中所有点对之间的距离：

```
import numpy as np
from scipy.spatial.distance import pdist, squareform

# Create the following array where each row is a point in 2D space:
# [[0 1]
#  [1 0]
#  [2 0]]
x = np.array([[0, 1], [1, 0], [2, 0]])
print(x)

# Compute the Euclidean distance between all rows of x.
# d[i, j] is the Euclidean distance between x[i, :] and x[j, :],
# and d is the following array:
```

```
# [[ 0.          1.41421356  2.23606798]
#  [ 1.41421356  0.          1.        ]
#  [ 2.23606798  1.          0.        ]]
d = squareform(pdist(x, 'euclidean'))
print(d)
```

您可以在文档中阅读有关此功能的所有详细信息 。

类似的函数（ `scipy.spatial.distance.cdist` ）计算两组点之间所有对之间的距离; 你可以在文档中阅读它 。

Matplotlib

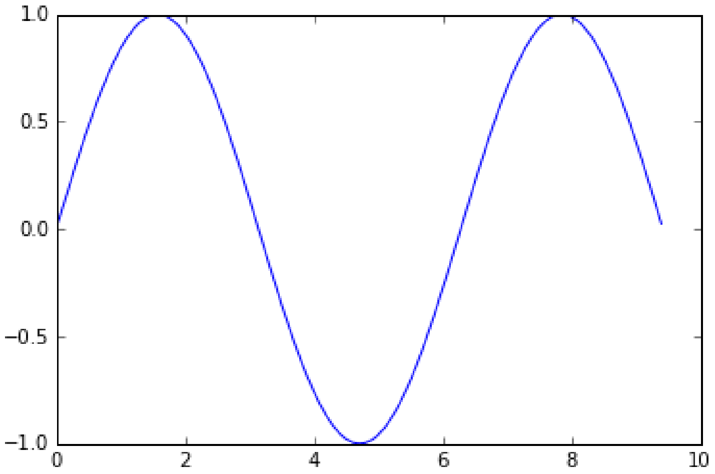
Matplotlib是一个绘图库。本节简要介绍该`matplotlib.pyplot`模块，该模块提供了类似于MATLAB的绘图系统。

绘制

matplotlib中最重要的功能是`plot`，它允许您绘制2D数据。这是一个简单的例子：

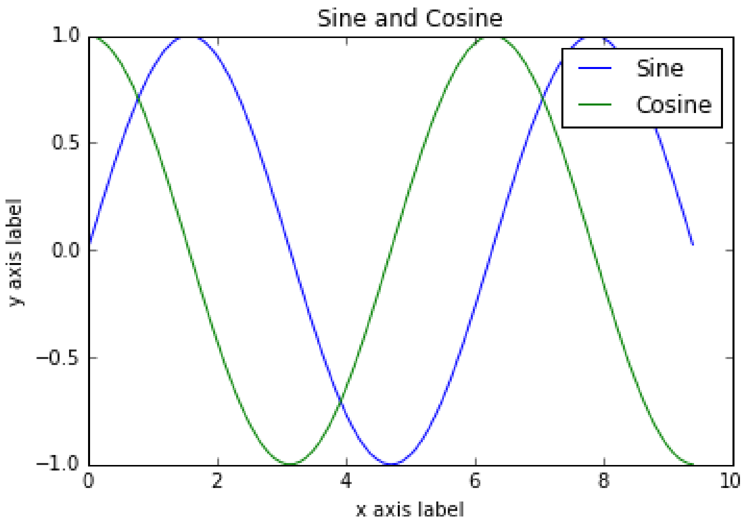
```
import numpy as np
import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on a sine curve
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)

# Plot the points using matplotlib
plt.plot(x, y)
plt.show()  # You must call plt.show() to make graphics appear.
```

运行此代码会生成以下图表：



通过一些额外的工作，我们可以轻松地一次绘制多条线，并添加标题，图例和轴标签：

```
import numpy as np
import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Plot the points using matplotlib
plt.plot(x, y_sin)
plt.plot(x, y_cos)
plt.xlabel('x axis label')
plt.ylabel('y axis label')
plt.title('Sine and Cosine')
plt.legend(['Sine', 'Cosine'])
plt.show()
```

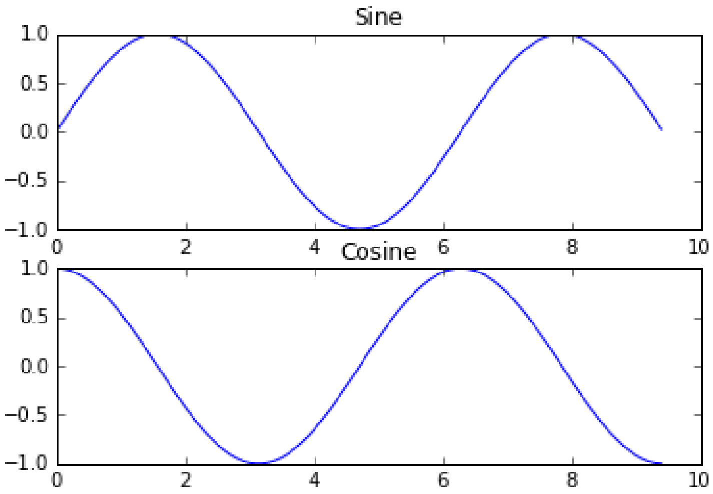


您可以在文档中阅读有关该`plot`功能的 更多信息。

次要情节

您可以使用该`subplot`函数在同一图中绘制不同的东西。这是一个例子：

```
import numpy as np
import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Set up a subplot grid that has height 2 and width 1,
# and set the first such subplot as active.
plt.subplot(2, 1, 1)

# Make the first plot
plt.plot(x, y_sin)
plt.title('Sine')

# Set the second subplot as active, and make the second plot.
plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine')

# Show the figure.
plt.show()
```

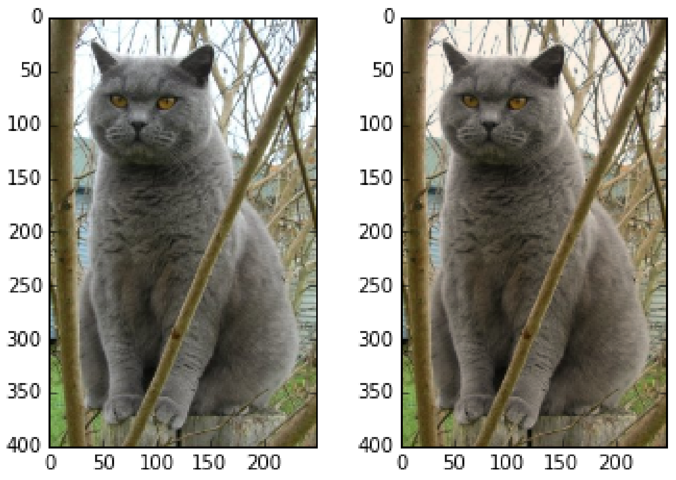


您可以在文档中阅读有关该`subplot`功能的 更多信息。

图片

您可以使用该`imshow`功能显示图像。这是一个例子：

```
import numpy as np
from scipy.misc import imread, imresize
import matplotlib.pyplot as plt

img = imread('assets/cat.jpg')
img_tinted = img * [1, 0.95, 0.9]

# Show the original image
plt.subplot(1, 2, 1)
plt.imshow(img)

# Show the tinted image
plt.subplot(1, 2, 2)

# A slight gotcha with imshow is that it might give strange results
# if presented with data that is not uint8. To work around this, we
# explicitly cast the image to uint8 before displaying it.
plt.imshow(np.uint8(img_tinted))
plt.show()
```



cs231n
cs231n
karpathy@cs.stanford.edu