Python Numpy 教程

Lion · 10 个月前

这是一篇翻译。
原文：Python Numpy Tutorial

翻译：我

如果有什么问题欢迎留言私信指正探讨。

正文：

这篇教程是由Justin Johnson.贡献的。

在本门课程中我将会使用Python完成所有作业。Python是一个伟大的通用性编程语言，但是在一些流行的库(numpy, scipy, matplotlib)的帮助下Python成为了强大的科学计算的环境。

我们希望你对Python和numpy有些经验。在本章只是一个关于Pyhton和使用Python进行科学计算的速成课程。

Python

Python 是一种高级的，动态的，多泛型的编程语言。Python代码很多时候看起来就像是伪代码一样，因此你可以使用很少的几行可读性很高的代码来实现一个非常强大的想法。举个例子，下面是使用Python来实现非常经典的快速排序算法的代码。

```
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]"
```

当前有两种不同的Python版本被支持，分别是2.7和3.4。令人不解的是，Python3.0引入了一些不可向下兼容的变换，所以使用2.7写的代码不一定能在3.4版本上运行，反之亦然。在本课程中代码都是运行在2.7版本上的。

你可以在命令行中使用一下命令来检查一下自己的Python版本

```
python --version
```

基础数据类型

和很多语言一样，Python也有几种数据类型包括：整型，浮点型，波尔型和字符型。这些数据类型的表现方式和大家熟悉的其他语言一样。

Numbers：整型和浮点型的工作方式和其他语言一样。

```
x = 3
print type(x) # Prints "<type '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 "<type 'float'>"
print y, y + 1, y * 2, y ** 2 # Prints "2.5 3.5 5.0 6.25"
```

注意：Python没有类似于++和--的一元运算的操作。

Python也内置了长整型和复数。你可以在这篇文档里查看更多细节。

Booleans：Python的布尔型逻辑和其他语言都一样，不过使用了英文单词替换了符号(||,&&)

```
t = True
f = False
print type(t) # Prints "<type 'bool'>"
```

```
print not t   # Logical NOT; prints "False"
print t != f  # Logical XOR; prints "True"
```

Strings: Python对字符型的支持非常强大。

```
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 对象还有很多常用的方法，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"
```

你可以在这篇文章中找到包含string对象所有方法的列表。

Containers

Python包含了几个内置的容器类型：lists, dictionaries, sets, and tuples

Lists

list在Python中几乎等价于数组，但是是可调整大小的同时也可以包容不同类型的元素。

```
xs = [3, 1, 2]   # Create a list
print xs, xs[2]  # Prints "[3, 1, 2] 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']"
```

同样，你可以在这篇文档中找到关于list的所有细节。

Slicing：除了每次访问一个列表元素，Python还提供了一种简洁的语法去访问子列表，这被称为slicing:

```
nums = 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 "[0, 1]"
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]"
```

我们将会在numpy arrays章节中再看到slicing.

Loops：你可以用如下方法遍历列表元素：

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

如果你想使用循环结构得到元素的索引以及内容，使用内置的“enumerate”方法

```
animals = ['cat', 'dog', 'monkey']
for idx, animal in enumerate(animals):
```

列表推导：在编程的时候我们经常会想要将一种数据类型转换成另一种。举个简单的例子，思考下面这段代码，它计算了数据的平方：

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

你也可以使用列表推导写出更简单的代码：

```
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判断逻辑：

```
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

字典由键值对组成。就像JAVA里面的map，以及JavaScript中的Object。使用方法如下：

```
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"
```

Loops：根据字典的键很容易进行迭代。

```
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 spider has 8 legs", "A cat has 4 legs"
```

如果你想获得键以及对应的值，使用**iteritems**方法

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

Dictionary comprehensions:和推导列表差不多，不过可以更简单的构建字典。

```
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}"
```

Sets

sets是离散元的无序集合。下面举个简单的例子

```
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"
```

和之前一样，这篇文章的里面有sets所有的内容。

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

Set comprehensions:和列表以及字典一样。我们可以很简单的构建一个集合：

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

Tuples

元组是一个有序列表。在很多方面和列表一样，有一点非常重要的不同就是元组可以在字典中被用作键而列表不行。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 "<type 'tuple'>"
print d[t]       # Prints "5"
print d[(1, 2)]  # Prints "1"
```

这篇文档有很多关于

Functions

python 中方法的定义使用**def**关键字

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

for x in [-1, 0, 1]:
```

我们将会用到带有可选参数的方法，例如

```
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!"
```

更多关于方法的信息都在这篇文档中

Classes

定义类的语法在Python中很简单。

```
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!"
```

在这篇文档中你可以获得更多关于Python的类

Numpy

Arrays

numpy数组是一个栅格值（译者注：可以理解为矩阵），所有类型都是一样的，是一个被索引的非负实数的元祖。数组的维度大小是数组的rank，数组的*shape*是一个整型的元组，包含元组的大小和有几个这样的元组。

我们可以使用嵌套Python列表来初始化一个numpy数组，使用方括号来访问元素。

```
import numpy as np

a = np.array([1, 2, 3])  # Create a rank 1 array
print type(a)            # Prints "<type '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还提供很多方法来创建array

```
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.]]"
```

其他穿件array的方法你可以看这篇文档

Array indexing

Numpy提供了多种方法来索引数组。

Slicing：和Python的列表一样Numpy数组也是可以被切片的。因为数组可能是多维的，所以对数组的每一个元素都要指定slice

```
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"
```

你也可以混合整型索引和slice索引。然而这样做将会得到一个相比原始数组rank更低的数组。

注意这和MATLAB处理数组slicing的方法有很大的不同。

```
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]]
```

```
# 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)"
```

整型矩阵索引：当你使用slicing索引numpy矩阵时，结果的矩阵视图总是源矩阵的子矩阵。相比之下,整数数组索引允许您使用其他矩阵的数据构建任意阵列。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]"
```

整型数组索引的一个实用技巧是用来选择或变换矩阵的每一行的一个元素。

```
import numpy as np
```

```
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])


print a[np.arange(4), b]  # Prints "[ 1  6  7 11]"
a[np.arange(4), b] += 10
print a  # prints "array([[11,  2,  3],
         #                [ 4,  5, 16],
         #                [17,  8,  9],
         #                [10, 21, 12]])
```

Boolean array indexing: Boolean array indexing可以取得一个矩阵中的任意元素。通常这种索引方式用来选择符合一定条件的元素。例如：

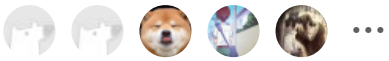
```
import numpy as np

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]"
```

Datatypes

每一个numpy矩阵都是由具有相同数据类型的元素组成的网格。Numpy提供了一个很大的可以用来构建矩阵的数据类型的集合。

（未完待续）

Python



7 条评论

写下你的评论...

太平洋的暖风

目前一点numpy的东西都没有。。。

4 个月前 1 赞

兄才

numpy呢?

4 个月前 1 赞

可可

你这好像是翻译的《深度学习Numpy 基础》一文啊

3 个月前

多谢，参考了，不然有些地方有些瑕疵，翻译也不完整

3 个月前

EWWE

Slicing：除了每次访问一个列表元素，Python还提供了一种简洁的语法去访问子列表，这被称为slicing:
nums = range(5) #应该改成 nums = arange(5)

2 个月前

沉迷代码无法自拔

快排写的有问题，自己都不跑了试一试吗，列表长度是基数用“/”结果是浮点数，要改为“//”取整除值

25 天前 1 赞

FromNowToNow

刚刚看到Stanford的CV教程，就看到了这个...

21 天前