3.9 大型数组运算¶

问题¶

你需要在大数据集(比如数组或网格)上面执行计算。

解决方案¶

涉及到数组的重量级运算操作，可以使用 `NumPy` 库。 `NumPy` 的一个主要特征是它会给Python提供一个数组对象，相比标准的Python列表而已更适合用来做数学运算。 下面是一个简单的小例子，向你展示标准列表对象和 `NumPy` 数组对象之间的差别：

```
>>> # Python lists
>>> x = [1, 2, 3, 4]
>>> y = [5, 6, 7, 8]
>>> x * 2
[1, 2, 3, 4, 1, 2, 3, 4]
>>> x + 10
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate list (not "int") to list
>>> x + y
[1, 2, 3, 4, 5, 6, 7, 8]

>>> # Numpy arrays
>>> import numpy as np
>>> ax = np.array([1, 2, 3, 4])
>>> ay = np.array([5, 6, 7, 8])
>>> ax * 2
array([2, 4, 6, 8])
>>> ax + 10
array([11, 12, 13, 14])
>>> ax + ay
array([ 6, 8, 10, 12])
>>> ax * ay
array([ 5, 12, 21, 32])
>>>
```

正如所见，两种方案中数组的基本数学运算结果并不相同。 特别的， `NumPy` 中的标量运算(比如 `ax*2` 或 `ax+10`)会作用在每一个元素上。 另外，当两个操作数都是数组的时候执行元素对等位置计算，并最终生成一个新的数组。

对整个数组中所有元素同时执行数学运算可以使得作用在整个数组上的函数运算简单而又快速。 比如，如果你想计算多项式的值，可以这样做：

```
>>> def f(x):
...  return 3*x**2 - 2*x + 7
...
>>> f(ax)
array([ 8, 15, 28, 47])
>>>
```

`NumPy` 还为数组操作提供了大量的通用函数，这些函数可以作为 `math` 模块中类似函数的替代。比如：

```
>>> np.sqrt(ax)
array([ 1. , 1.41421356, 1.73205081, 2. ])
>>> np.cos(ax)
array([ 0.54030231, -0.41614684, -0.9899925 , -0.65364362])
>>>
```

使用这些通用函数要比循环数组并使用 `math` 模块中的函数执行计算要快的多。 因此，只要有可能的话尽量选择 `NumPy` 的数组方案。

底层实现中， `NumPy` 数组使用了C或者Fortran语言的机制分配内存。 也就是说，它们是一个非常大的连续的并由同类型数据组成的内存区域。 所以，你可以构造一个比普通Python列表大的多的数组。 比如，如果你想构造一个10,000*10,000的浮点数二维网格，很轻松：

```
>>> grid = np.zeros(shape=(10000,10000), dtype=float)
>>> grid
   array([[ 0., 0., 0., ..., 0., 0., 0.],
   [ 0., 0., 0., ..., 0., 0., 0.],
   [ 0., 0., 0., ..., 0., 0., 0.],
   ...,
   [ 0., 0., 0., ..., 0., 0., 0.],
   [ 0., 0., 0., ..., 0., 0., 0.],
   [ 0., 0., 0., ..., 0., 0., 0.]])
>>>
```

所有的普通操作还是会同时作用在所有元素上：

```
>>> grid += 10
>>> grid
array([[ 10., 10., 10., ..., 10., 10., 10.],
   [ 10., 10., 10., ..., 10., 10., 10.],
   [ 10., 10., 10., ..., 10., 10., 10.],
   ...,
   [ 10., 10., 10., ..., 10., 10., 10.],
   [ 10., 10., 10., ..., 10., 10., 10.],
   [ 10., 10., 10., ..., 10., 10., 10.]])
>>> np.sin(grid)
array([[-0.54402111, -0.54402111, -0.54402111, ..., -0.54402111,
      -0.54402111, -0.54402111],
   [-0.54402111, -0.54402111, -0.54402111, ..., -0.54402111,
      -0.54402111, -0.54402111],
   [-0.54402111, -0.54402111, -0.54402111, ..., -0.54402111,
      -0.54402111, -0.54402111],
   ...,
   [-0.54402111, -0.54402111, -0.54402111, ..., -0.54402111,
      -0.54402111, -0.54402111],
   [-0.54402111, -0.54402111, -0.54402111, ..., -0.54402111,
      -0.54402111, -0.54402111],
   [-0.54402111, -0.54402111, -0.54402111, ..., -0.54402111,
      -0.54402111, -0.54402111]])
>>>
```

关于 `NumPy` 有一点需要特别的主意，那就是它扩展Python列表的索引功能 - 特别是对于多维数组。 为了说明清楚，先构造一个简单的二维数组并试着做些试验：

```
>>> 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]])

>>> # Select row1
>>> a[1]
array([5, 6, 7, 8])

>>> # Select column 1
>>> a[:,1]
array([ 2, 6, 10])

>>> # Select a subregion and change it
>>> a[1:3, 1:3]
array([[ 6, 7],
       [10, 11]])
>>> a[1:3, 1:3] += 10
>>> a
array([[ 1, 2, 3, 4],
       [ 5, 16, 17, 8],
       [ 9, 20, 21, 12]])

>>> # Broadcast a rowvector across an operation on all rows
>>> a + [100, 101, 102, 103]
array([[101, 103, 105, 107],
       [105, 117, 119, 111],
       [109, 121, 123, 115]])
>>> a
array([[ 1, 2, 3, 4],
       [ 5, 16, 17, 8],
       [ 9, 20, 21, 12]])

>>> # Conditional assignment on an array
>>> np.where(a < 10, a, 10)
array([[ 1, 2, 3, 4],
       [ 5, 10, 10, 8],
       [ 9, 10, 10, 10]])
>>>
```

讨论¶

NumPy 是Python领域中很多科学与工程库的基础，同时也是被广泛使用的最大最复杂的模块。 即便如此，在刚开始的时候通过一些简单的例子和玩具程序也能帮我们完成一些有趣的事情。

通常我们导入 NumPy 模块的时候会使用语句 import numpy as np 。 这样的话你就不用再你的程序里面一遍遍的敲入 numpy ，只需要输入 np 就行了，节省了不少时间。

如果想获取更多的信息，你当然得去 NumPy 官网逛逛了，网址是： http://www.numpy.org