理解python中的坐标轴

0. 引言

对于一个列表来说，求和显然是将所有元素相加，但是如果是一个矩阵，甚至更高维的存在，相加方式就有很多，所以可以把他们化归为列表的形式，将他们以某种方式看成一个列表，然后所有元素相加，即可得到答案。

1. 例子

以下为一个三维列表的例子：

```
[
    [
         [1., 1., 1., 1.],
         [2., 2., 2., 2.],
         [3., 3., 3., 3.]
    ],

    [
         [4., 4., 4., 4.],
         [5., 5., 5., 5.],
         [6., 6., 6., 6.]
    ]
]
```

2. 先讨论axis=0

对一个tensor张量来说，axis=0代表去掉最外层的[]，或者说进入最外层的[]，此时发现有两个元素。

```
[
    [1., 1., 1., 1.],
    [2., 2., 2., 2.],
    [3., 3., 3., 3.]
] # 记为A

[
    [4., 4., 4., 4.],
    [5., 5., 5., 5.],
    [6., 6., 6., 6.]
] # 记为B
```

如果此时对于他们进行操作，就是将其看成两个元素。

如列表[1, 2]，那么求和就是直接对这两个元素求和。

对应到原列表上就是两个矩阵的求和，见以下代码

```
import numpy as np
X = np.array([[[1., 1., 1., 1.],
               [2., 2., 2., 2.],
               [3., 3., 3., 3.]],
              [[4., 4., 4., 4.],
               [5., 5., 5., 5.],
               [6., 6., 6., 6.]]])
print(X.sum(axis=0))
```

输出为

```
[[5. 5. 5. 5.]
 [7. 7. 7. 7.]
 [9. 9. 9. 9.]]
```

3. 继续讨论axis=1、2

如果axis为1，则继续进入第二层[]，由于有两个元素，他会分别进入元素A、元素B，以元素A为例来看，可以发现他有三个元素[1., 1., 1., 1.], [2., 2., 2., 2.], [3., 3., 3., 3.]，由于并没有再进一层方括号，所以这三个就是基本元素，会对其相加，组成[6,6,6,6]，同理元素B也会如此。

```
>>> print(X.sum(axis=1))
[[ 6.  6.  6.  6.]
 [15. 15. 15. 15.]]
```

axis为2，则更进一层，进入第三层[]，如[1., 1., 1., 1.]，那会得到4，其余同理。

PS: 记得我并不是将方括号拆掉，而是进入，所以原本第一、第二、第三层的方括号同样都存在，所以会阻隔某一层次中不同元素内部的元素互相混合。

```
>>> print(X.sum(axis=2))
[[ 4.  8. 12.]
 [16. 20. 24.]]
```

4. 补充

事实上，通过这个理解可以发现，本质对某一axis的求和，就是把这一层的元素求和，这会导致这一层的元素个数变为1，所以这一层就没有必要存在，因此可以看到以上$2 \times 3 \times 4$的变量：

- axis=0时，输出是3×4
- axis=1时，输出是2×4
- axis=2时，输出是2×3