这三个函数有些相似性，都是堆叠数组，里面最难理解的应该就是stack()函数了，我查阅了numpy的官方文档，在网上又看了几个大牛的博客，发现他们也只是把numpy文档的内容照搬，看完后还是不能理解，最后经过本人代码分析，算是理解了stack()函数增加维度的含义。以下内容我会用通俗易懂的语言解释，内容可能有点多，耐心看，如果哪里说的不对，欢迎纠正！

**1. stack()函数**   
函数原型为：stack(arrays, axis=0)，arrays可以传数组和列表。axis的含义我下面会讲解，我们先来看个例子，然后我会分析输出结果。

import numpy as np

a=[[1,2,3],

[4,5,6]]print("列表a如下：")print(a)

print("增加一维，新维度的下标为0")

c=np.stack(a,axis=0)print(c)

print("增加一维，新维度的下标为1")

c=np.stack(a,axis=1)print(c)

输出：

列表a如下：[[1, 2, 3], [4, 5, 6]]

增加一维，新维度下标为0[[1 2 3]

[4 5 6]]

增加一维，新维度下标为1[[1 4]

[2 5]

[3 6]]

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24

首先这里arrays我传的是一个列表，现在我开始讲解这个stack()函数的意思，它就是对arrays里面的**每个元素**(可能是个列表，元组，或者是个numpy的数组)变成numpy的数组后，再对**每个元素**增加一维(至于维度加在哪里，是靠axis控制的)，然后再把这些元素串起来(至于怎么串，我下面会说)。

**arrays里面的每个元素必须形状是一样的**，例如本例中列表a中的两个元素[1,2,3]和[4,5,6]的形状是一样的，如果把[4,5,6]换成[4,5] ，那么程序会报错！而axis代表的是在哪个维度上加一维，例如axis=0(**它是默认的**)代表的就是增加的这一维的下标为0，axis等于多少不是随便乱写的，如果参数arrays里面的每个元素是个1维的，那么调用stack()函数增加一维后会变成2维的，所以axis只能等于0和1（维度的下标是从0开始的），而参数axis=0和axis=1得到的结果是不一样的。

例如上面的代码中a列表中的第一个元素为[1,2,3]，那么当axis=0的时候，就是在它的中括号外面再加一个中括号，变成[ [1,2,3] ]**（其实1，2，3之间是没有逗号的，因为stack()函数会先把参数arrays中的每个元素变成numpy的数组，数组之间是没有逗号的，看看上面的代码输出就知道了，这里大家明白就行，我为了方便讲解，下面还会加上逗号）**，这样最外面那层中括号才代表维度下标为0的那维；当axis=1的时候，就是在里面加个中括号，变成了[ [1]，[2]，[3] ],这样里面加的那层中括号才代表维度下标为1的那维。同理当axis=0的时候[4,5,6]变成[ [ 4，5，6] ]，当axis=1的时候，变成[ [4]，[5]，[6] ]。下面我们讲如何把增加一维度后的每个元素串起来。

怎么把上面那两个元素增加维度后的结果串起来呢，其实很简单。现在我们已经知道了增加维度无非是增加中括号的意思，至于在哪里加中括号，取决于axis等于几。我们把增加的中括号想像成一个个的箱子。还拿上面的代码来说，**当axis=0**的时候，我们把套在[1,2,3]外面的中括号**(就是[ [1,2,3] ]最外层的那个中括号)**看做是箱子A，这个箱子A也会套在[4,5,6]的外面，所以我们就先把[1,2,3]和[4,5,6]放在一起，变成[1,2,3],[4,5,6]，然后再一起套上箱子A，变成[ [1,2,3],[4,5,6] ]这就是当axis=0的时候程序的输出结果。

现在再来看**当axis=1**的时候，对于[1，2，3]，我们把套在1外面的箱子**(就是上面讲的[ [1]，[2]，[3] ]中1外面的那层中括号)**看做A，套在2外面的看做B，套在3外面的看做C，同理，箱子A也会套在4的外面，箱子B也会套在5的外面，箱子C也会套在6的外面。那么我们就把1和4放一起，2和5放一起，3和6放一起，变成[ 1,4 ，2,5 ，3,6 ]然后把箱子A,B,C分别套在1,4 ， 2,5 ， 3,6的外面，变成[ [1,4] , [2,5] , [3,6] ]这就是程序中axis=1的时候程序的输出结果。

大家发现了没有，串起来的时候其实就是把arrays中每个元素在相同的位置套箱子的一些小块(这里叫小块这个名词可能不洽当，但是大家明白就行)放在一起后，再套箱子，就是外面套个中括号，这就是堆叠。

再看下面的代码的输出，测试下你理解的没有。

import numpy as np

a=[[1,2,3,4],

[5,6,7,8],

[9,10,11,12]]print("列表a如下：")print(a)

print("增加一维，新维度的下标为0")

c=np.stack(a,axis=0)print(c)

print("增加一维，新维度的下标为1")

c=np.stack(a,axis=1)print(c)

输出：

列表a如下：[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]

增加一维，新维度的下标为0[[ 1 2 3 4]

[ 5 6 7 8]

[ 9 10 11 12]]

增加一维，新维度的下标为1[[ 1 5 9]

[ 2 6 10]

[ 3 7 11]

[ 4 8 12]]

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27

不知道和你想象的输出一样不一样，还有另一种情况，先看下面的代码。

import numpy as np

a=[1,2,3,4]

b=[5,6,7,8]

c=[9,10,11,12]print("a=",a)print("b=",b)print("c=",c)

print("增加一维，新维度的下标为0")

d=np.stack((a,b,c),axis=0)print(d)

print("增加一维，新维度的下标为1")

d=np.stack((a,b,c),axis=1)print(d)

输出：

('a=', [1, 2, 3, 4])

('b=', [5, 6, 7, 8])

('c=', [9, 10, 11, 12])

增加一维，新维度的下标为0[[ 1 2 3 4]

[ 5 6 7 8]

[ 9 10 11 12]]

增加一维，新维度的下标为1[[ 1 5 9]

[ 2 6 10]

[ 3 7 11]

[ 4 8 12]]

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29

你会发现输出结果和上面的代码一样，其实它俩就是一样的。只不过当你对arrays传参的时候，如果你传的参数是类似于(a,b,c)这种，它会把(a,b,c)当做一个元组来看，a,b,c都是元组的每个元素。然后分别对每个元素处理，上面我已经说了，arrays传的参数可以是列表，元组，或者numpy数组。所以传(a,b,c)和传[a,b,c]或者当x=[a,b,c]的时候传x，效果都是一样的。

上面的代码处理的arrays元素都是一维变二维的情况，下面我们看看二维变三维是什么样的。

import numpy as np

a=[[1,2,3],

[4,5,6]]

b=[[1,2,3],

[4,5,6]]

c=[[1,2,3],

[4,5,6]]print("a=",a)print("b=",b)print("c=",c)

print("增加一维，新维度的下标为0")

d=np.stack((a,b,c),axis=0)print(d)

print("增加一维，新维度的下标为1")

d=np.stack((a,b,c),axis=1)print(d)print("增加一维，新维度的下标为2")

d=np.stack((a,b,c),axis=2)print(d)

输出：

('a=', [[1, 2, 3], [4, 5, 6]])

('b=', [[1, 2, 3], [4, 5, 6]])

('c=', [[1, 2, 3], [4, 5, 6]])

增加一维，新维度的下标为0[[[1 2 3]

[4 5 6]]

[[1 2 3]

[4 5 6]]

[[1 2 3]

[4 5 6]]]

增加一维，新维度的下标为1[[[1 2 3]

[1 2 3]

[1 2 3]]

[[4 5 6]

[4 5 6]

[4 5 6]]]

增加一维，新维度的下标为2[[[1 1 1]

[2 2 2]

[3 3 3]]

[[4 4 4]

[5 5 5]

[6 6 6]]]

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51

**当axis=0**的时候，列表a，b，c最外面都需要套箱子(就是加中括号)，那么我把你们先放一起，变成下面这样

[[1,2,3],[4,5,6]],[[1,2,3],[4,5,6]],[[1,2,3],[4,5,6]]

* 1
* 2
* 3

然后在最外面套箱子，变成

[

[[1,2,3],[4,5,6]],

[[1,2,3],[4,5,6]],

[[1,2,3],[4,5,6]]

]

* 1
* 2
* 3
* 4
* 5

当axis=1的时候，列表a，b，c中的[1,2,3]需要套同样的箱子，列表a，b，c中的[4,5,6]需要套同样的箱子，好，我先把你们放一块变成下面这样

[

[1,2,3],[1,2,3],[1,2,3]

,

[4,5,6],[4,5,6],[4,5,6]

]

* 1
* 2
* 3
* 4
* 5
* 6

然后开始分别在 [1,2,3],[1,2,3],[1,2,3]的外面和[4,5,6],[4,5,6],[4,5,6]的外面套箱子，变成下面这样

[

[[1,2,3],[1,2,3],[1,2,3]]

,

[[4,5,6],[4,5,6],[4,5,6]]

]

* 1
* 2
* 3
* 4
* 5
* 6

当axis=2的时候，列表a，b，c中的1，2，3，4，5，6都需要套箱子，我把你们先放一起变成：

[

[1,1,1 , 2,2,2 , 3,3,3],

[4,4,4 , 5,5,5 , 6,6,6]

]

* 1
* 2
* 3
* 4

然后在1,1,1 ………6,6,6的外面分别套箱子变成：

[

[[1,1,1] , [2,2,2] , [3,3,3]],

[[4,4,4] , [5,5,5] , [6,6,6]]

]

* 1
* 2
* 3
* 4

关于stack()函数就讲这么多，这也是我全部理解的部分。

**2. hstack()函数**   
函数原型：hstack(tup) ，参数tup可以是元组，列表，或者numpy数组，返回结果为numpy的数组。看下面的代码体会它的含义

import numpy as np

a=[1,2,3]

b=[4,5,6]

print(np.hstack((a,b)))

输出：[1 2 3 4 5 6 ]

* 1
* 2
* 3
* 4
* 5
* 6

import numpy as np

a=[[1],[2],[3]]

b=[[1],[2],[3]]

c=[[1],[2],[3]]

d=[[1],[2],[3]]print(np.hstack((a,b,c,d)))

输出：[[1 1 1 1]

[2 2 2 2]

[3 3 3 3]]

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11

它其实就是水平(按列顺序)把数组给堆叠起来，vstack()函数正好和它相反。

**3. vstack()函数**   
函数原型：vstack(tup) ，参数tup可以是元组，列表，或者numpy数组，返回结果为numpy的数组。看下面的代码体会它的含义

import numpy as np

a=[1,2,3]

b=[4,5,6]print(np.vstack((a,b)))

输出：[[1 2 3]

[4 5 6]]

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8

import numpy as np

a=[[1],[2],[3]]

b=[[1],[2],[3]]

c=[[1],[2],[3]]

d=[[1],[2],[3]]print(np.vstack((a,b,c,d)))

输出：[[1]

[2]

[3]

[1]

[2]

[3]

[1]

[2]

[3]

[1]

[2]

[3]]

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20

它是垂直（按照行顺序）的把数组给堆叠起来。

--------------------- 本文来自 neu\_张康 的CSDN 博客 ，全文地址请点击：https://blog.csdn.net/csdn15698845876/article/details/73380803?utm\_source=copy