programs_4_tf_pd_np

August 7, 2023

[1]:
```
import numpy as np
# import tensorflow as tf
import pandas as pd
```

[15]:
```
a = np.array([[1, 2], [3, 4]])
print(a)
print(a.shape)
# [3, 2] => pad axis=0 with 3 values before and 2 values after.
# [2, 3] => pad axis=1 with 2 values before and 3 values after.
print(np.pad(a, pad_width=[[3, 2], [2, 3]],))
print(np.pad(a, pad_width=[[3, 2], [2, 3]],).shape)
```

```
[[1 2]
 [3 4]]
(2, 2)
[[0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0]
 [0 0 1 2 0 0 0]
 [0 0 3 4 0 0 0]
 [0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0]]
(7, 7)
```

[4]:
```
a = tf.constant([[1,2,3],[4,5,6]], tf.int32)
print(a)
b = tf.constant([1,2], tf.int32)
# output tensor's i'th dimension has input.dims(i) * multiples[i] elements, -
# - and the values of input are kept multiples[i] times along the 'i'th␣
 ↪dimension.

# output tensor's 0'th dimension has 2 * 1 elements, -
# - and the values of input are kept 1 times along the '0'th dimension.

# output tensor's 1'th dimension has 3 * 2 elements, -
# - and the values of input are kept 2 times along the '1'th dimension.
```

```
print(tf.tile(a, b))
```

```
tf.Tensor(
[[1 2 3]
 [4 5 6]], shape=(2, 3), dtype=int32)
tf.Tensor(
[[1 2 3 1 2 3]
 [4 5 6 4 5 6]], shape=(2, 6), dtype=int32)
```

[5]:
```
c = tf.constant([2,1], tf.int32)
print(tf.tile(a, c))
```

```
tf.Tensor(
[[1 2 3]
 [4 5 6]
 [1 2 3]
 [4 5 6]], shape=(4, 3), dtype=int32)
```

[6]:
```
d = tf.constant([2,2], tf.int32)
print(tf.tile(a, d))
```

```
tf.Tensor(
[[1 2 3 1 2 3]
 [4 5 6 4 5 6]
 [1 2 3 1 2 3]
 [4 5 6 4 5 6]], shape=(4, 6), dtype=int32)
```

[9]:
```
```

[9]: 2

[4]:
```
############################################################

df = pd.DataFrame([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9],
                   [np.nan, np.nan, np.nan]],
                  columns=['A', 'B', 'C'])
df
```

[4]:

	A	B	C
0	1.0	2.0	3.0
1	4.0	5.0	6.0
2	7.0	8.0	9.0
3	NaN	NaN	NaN

[5]:
```
df.aggregate('max', axis=1)
```

[5]:
```
0    3.0
1    6.0
2    9.0
3    NaN
dtype: float64
```

[6]:
```
##################################################
mydict = [{'a': 1, 'b': 2, 'c': 3, 'd': 4},
          {'a': 100, 'b': 200, 'c': 300, 'd': 400},
          {'a': 1000, 'b': 2000, 'c': 3000, 'd': 4000 },
          {'a': 10, 'b': 9, 'c': 8, 'd': 7 },
          {'a': 11, 'b': 12, 'c': 13, 'd': 14 }]
df = pd.DataFrame(mydict)
df
```

[6]:

	a	b	c	d
0	1	2	3	4
1	100	200	300	400
2	1000	2000	3000	4000
3	10	9	8	7
4	11	12	13	14

[8]:
```
# iloc[start:stop:step]
df.iloc[0::2]
```

[8]:

	a	b	c	d
0	1	2	3	4
2	1000	2000	3000	4000
4	11	12	13	14

[ ]:
```
##################################################################################
```

[3]:
```
# The expression lst[::-1] is used to reverse the elements of a list in Python.
 ↪Let me explain how it works.
# The general syntax for indexing is list_name[start_index:stop_index:step],
# The step value is -1, which means it iterates through the list in reverse
 ↪order with steps of 1.
lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
reverse_list = lst[::-1]
print(reverse_list)
```

```
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

[1]:
```
# lst[-5:], you are specifying a range starting from the fifth element from the
 ↪end (inclusive) and continuing until the end of the list.
lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
last_five_elements = lst[-5:]
```

```
print(last_five_elements)
```

```
[6, 7, 8, 9, 10]
```

[4]:
```
# [:5], the stop_index is 5, indicating that the sublist should include␣
 ↪elements up to, but not including, the element at index 5.
lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
first_five_elements = lst[:5]
print(first_five_elements)
```

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

[5]:
```
# [:-5], the stop_index is -5,
lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
first_five_elements = lst[:-5]
print(first_five_elements)
```

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

[5]:
```
# OR ====================================

a = {"hello":1, 2:4 ,"hello":6}
# a => {'hello': 6, 2: 4}

b = {"hello":1, 2:4 ,"hello_1":6}
print(list(b))
```

```
['hello', 2, 'hello_1']
```

[ ]:
```
##############################################################################################
```

[6]:
```
x=np.random.randint(0, 50, size=(2,3,4,2))
tf_1 = tf.convert_to_tensor(x, dtype=tf.float32)
tf_1[:,1:,:,:].shape
```

[6]:
```
TensorShape([2, 2, 4, 2])
```

[ ]:
```
############################################################
```

[20]:
```
z => array(['', 'archivist', 'archivist', 'block'], dtype='<U9')
z = z.view(np.uint32)
z =>
array([  0,   0,   0,   0,   0,   0,   0,   0,   0,  97, 114,  99, 104,
       105, 118, 105, 115, 116,  97, 114,  99, 104, 105, 118, 105, 115,
       116,  98, 108, 111,  99, 107,   0,   0,   0,   0], dtype=uint32)


# array([ 97, 114,  99, 104, 105, 118, 105, 115, 116,  97, 114,  99, 104,
```

```
#        105, 118, 105, 115, 116], dtype=uint32)
```

```
array(['', 'archivist', 'archivist', 'block'], dtype='<U9')
```

[20]:
```
array([  0,   0,   0,   0,   0,   0,   0,   0,   0,  97, 114,  99, 104,
       105, 118, 105, 115, 116,  97, 114,  99, 104, 105, 118, 105, 115,
       116,  98, 108, 111,  99, 107,   0,   0,   0,   0], dtype=uint32)
```

[21]:
```
################################################################################

aa = np.arange(32) #np.random.rand(32,32,32)
print(aa)
aa.shape
```

```
[ 0  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]
```

[21]:
```
(32,)
```

[23]:
```
a31 = aa[-19:31]
print(a31)
print(a31.shape)
aa[-19:32].shape
```

```
[13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30]
(18,)
```

[23]:
```
(19,)
```

[26]:
```
print(aa[-19:13])
print(aa[-19:14])
```

```
[]
[13]
```

[30]:
```
a31 = aa[19:-7]
print(a31)
print(a31.shape)
```

```
[19 20 21 22 23 24]
(6,)
```

[31]:
```
import random
```

[32]:
```
ch_arr = list(range(3))
random.sample(ch_arr, len(ch_arr))
```

[32]:
```
[1, 0, 2]
```

[48]:
```
#################################################################################
ab = np.array([[range(5), range(5,10)], [range(10,15), range(15,20)]])
print(ab.shape)
ab
```

```
(2, 2, 5)
```

[48]:
```
array([[[ 0,  1,  2,  3,  4],
        [ 5,  6,  7,  8,  9]],

       [[10, 11, 12, 13, 14],
        [15, 16, 17, 18, 19]]])
```

[60]:
```
print(ab[:,[1,0]].shape)
ab[:,[1,0]] # interchange indices for dimension 1.
```

```
(2, 2, 5)
```

[60]:
```
array([[[ 5,  6,  7,  8,  9],
        [ 0,  1,  2,  3,  4]],

       [[15, 16, 17, 18, 19],
        [10, 11, 12, 13, 14]]])
```

[58]:
```
print(ab[:,[1,0],[4,0]].shape)
ab[:,[1,0],[4,0]]
# pick 4th index in dimension 2 along the index 1 of dimension 1.
# pick 0th index in dimension 2 along the index 0 of dimension 1.
```

```
(2, 2)
```

[58]:
```
array([[ 9,  0],
       [19, 10]])
```

[56]:
```
print(ab[:,[1,0],[4]].shape)
ab[:,[1,0],[4]]
# pick 4th index in dimension 2 along the index 1 of dimension 1.
# pick 4th index in dimension 2 along the index 0 of dimension 1.
```

```
(2, 2)
```

[56]:
```
array([[ 9,  4],
       [19, 14]])
```

[64]:
```
print(ab[:,[1,0,1]].shape)
ab[:,[1,0,1]]
```

```
(2, 3, 5)
```

[64]:
```
array([[[ 5,  6,  7,  8,  9],
        [ 0,  1,  2,  3,  4],
        [ 5,  6,  7,  8,  9]],

       [[15, 16, 17, 18, 19],
        [10, 11, 12, 13, 14],
        [15, 16, 17, 18, 19]]])
```

[63]:
```
print(ab[:,[1,0,1],[1,0,1]].shape)
ab[:,[1,0,1],[1,0,1]] # interchange indices for dimension 1.
```

```
(2, 3)
```

[63]:
```
array([[ 6,  0,  6],
       [16, 10, 16]])
```

[ ]:
```

```

[169]:
```
###############################################################################
ab = np.random.randint(0, 179, (3,2,3,2,3))
ab = torch.from_numpy(ab)
print(ab.shape)
ab
```

```
torch.Size([3, 2, 3, 2, 3])
```

[169]:
```
tensor([[[[[177,  63, 164],
           [ 22,  61,  71]],

          [[ 67, 112,  76],
           [139,  36, 125]],

          [[101, 126,  36],
           [160, 147, 154]]],


         [[[127, 124, 123],
           [144,  22, 117]],

          [[ 35, 133, 163],
           [178, 168,  92]],

          [[ 76, 138, 160],
           [ 57,  91,  69]]]],
```

```
[[[[ 38, 173,  95],
   [ 11,  83, 113]],

  [[ 50,  15,  13],
   [170,  94,  39]],

  [[ 19,  51, 126],
   [165, 104,  54]]],


 [[[ 64, 142,  13],
   [110,  33,  96]],

  [[134,  13,   0],
   [178,  10,  99]],

  [[ 49, 107,  64],
   [120, 155,  67]]]],



[[[[ 68,  49,  20],
   [115,  12,  32]],

  [[ 30,  20,  96],
   [143,  39, 117]],

  [[ 36, 127,   9],
   [  4,  63,  92]]],


 [[[165,  22,  88],
   [ 34, 169,  45]],

  [[158,  38,  62],
   [100, 167, 113]],

  [[150, 116,  78],
   [ 15,  35, 141]]]])
```

[170]:
```
# ab.shape => (3, 2, 3, 2, 3)
# ab[0].shape => (2, 3, 2, 3)
ab[0]
```

[170]:
```
tensor([[[[177,  63, 164],
          [ 22,  61,  71]],
```

```
         [[ 67, 112,  76],
          [139,  36, 125]],

         [[101, 126,  36],
          [160, 147, 154]]],


        [[[127, 124, 123],
          [144,  22, 117]],

         [[ 35, 133, 163],
          [178, 168,  92]],

         [[ 76, 138, 160],
          [ 57,  91,  69]]]])
```

[171]:
```
# ab[0,:].shape => (2, 3, 2, 3)
ab[0,:]
```

[171]:
```
tensor([[[[177,  63, 164],
          [ 22,  61,  71]],

         [[ 67, 112,  76],
          [139,  36, 125]],

         [[101, 126,  36],
          [160, 147, 154]]],


        [[[127, 124, 123],
          [144,  22, 117]],

         [[ 35, 133, 163],
          [178, 168,  92]],

         [[ 76, 138, 160],
          [ 57,  91,  69]]]])
```

[172]:
```
# ab.shape => (3, 2, 3, 2, 3)
# ab[0,:,0].shape => (2, 2, 3)
ab[0,:,0]
```

[172]:
```
tensor([[[177,  63, 164],
         [ 22,  61,  71]],

        [[127, 124, 123],
         [144,  22, 117]]])
```

[ ]:

[173]:
```
ch = [[1,0,2],[2,0,1]]
for i,c in enumerate(ch):
    # this doesnot work if ab is numpy array.
    ab[i] = ab[i,:,c] # interchange indices for dimension 2.
    break
print(ab.shape)
ab[0]
```

```
torch.Size([3, 2, 3, 2, 3])
```

[173]:
```
tensor([[[[ 67, 112,  76],
          [139,  36, 125]],

         [[177,  63, 164],
          [ 22,  61,  71]],

         [[101, 126,  36],
          [160, 147, 154]]],


        [[[ 35, 133, 163],
          [178, 168,  92]],

         [[127, 124, 123],
          [144,  22, 117]],

         [[ 76, 138, 160],
          [ 57,  91,  69]]]])
```

[181]:
```
##############################################
a = np.array([1,2,3,4,5,6,7,8,9,10,11,12])
z = np.
  ↪array([False,False,False,False,True,False,False,True,False,False,False,False])
print(z.shape)
print(np.where(z))
print(np.where(z)[0])
a[np.where(z)]
```

```
(12,)
(array([4, 7]),)
[4 7]
```

[181]:
```
array([5, 8])
```

[ ]:

[ ]: