

8.2 创建张量

中国大学MOOC

TensorFlow: 开源机器学习框架

TensorFlow 2.0

- ■易用性
- 将Eager Execution作为默认执行模式
- 删除了大量过时和重复的API
- 支持更多的平台和语言

■ 更新到最新版本

```
更新到最新版本
In []: !pip install --upgrade tensorflow !pip install --upgrade tensorflow-gpu
```

■ 查看版本号

在Jupyter Notebook中运行cmd命令

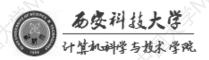
```
In [1]: import tensorflow as tf print("TensorFlow version:", tf. __version__) 查看版本号

TensorFlow version: 2.0.0

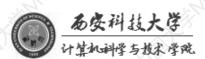
In [2]: print("Eager execution is:", tf. executing_eagerly())

Eager execution is: True

TensorFlow2.0默认为EagerExecution模式
```



```
In [3]: a=tf. constant(2, name="input a")
         b=tf. constant (3, name="input b")
         c=tf. add (a, b, name="add c")
         sess=tf. Session()
         print(sess.run(c))
         sess. close()
         AttributeError
                                                      Traceback (most recent call last)
         <ipython-input-3-495b1e0d9ff8> in <module>
               3 c=tf. add(a, b, name="add_c")
               5 sess=tf. Session()
               6 print(sess.run(c))
               7 sess. close()
        AttributeError: module 'tensorflow' has no attribute 'Session'
```

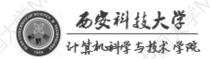




在TensorFlow2.0的环境中,运行TensorFlow1.x

import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

初学者建议直接从2.0开始学习!



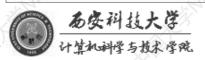


■ Python列表(list):

- □ 元素可以使用不同的数据类型,可以嵌套
- □ 在内存中不是连续存放的, 是一个动态的指针数组
- □ 读写效率低,占用内存空间大
- □ 不适合做数值计算

■ NumPy数组(ndarray):

- □ 元素数据类型相同
- □ 每个元素在内存中占用的空间相同,存储在一个连续的内存区域中
- □ 存储空间小,读取和写入速度快
- □ 不能够主动检测利用GPU进行运算
- TensorFlow张量(Tensor):
 - □ 可以高速运行于GPU和TPU之上,实现神经网络和深度学习中的复杂算法



■ 创建Tensor对象

张量由Tensor类实现,每个张量都是一个Tensor对象

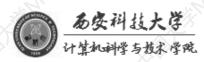
□ tf.constant()函数: 创建张量

tf.constant(value, dtype, shape)

■ value: 数字/Python列表/NumPy数组

■ dtype: 元素的数据类型

■ shape: 张量的形状



■ 参数为Python列表

■ 张量的.numpy()方法

```
In [5]: a=tf. constant([[1,2], [3,4]])
        a. numpy()
                        TensorFlow2.0中的所有的张量,
Out[5]:
        array([[1, 2],
                        都可以通过.numpy()方法,得到它对应的数组
In [6]:
        type(a)
Out[6]: tensorflow.python.framework.ops.EagerTensor
                                 Eager Execution模式下的张量
In [7]:
        print(a)
        tf. Tensor (
        [[1 \ 2]]
         [3 4]], shape=(2, 2), (dtype=int32)
```

■ 参数为数字

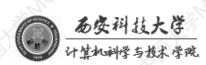
In [8]: tf. constant (1.0) 0维张量/数字/标量

Out[8]: <tf. Tensor: id=7, shape=(), dtype=float32, numpy=1.0>

In [9]: tf. constant (1.)

TensorFlow创建**浮点数**张量时,默认是**32位**浮点数

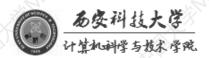
Out[9]: <tf.Tensor: id=8, shape=(), dtype=float32, numpy=1.0>





张量元素的数据类型

/					
数据类型		描	述		
tf.int8	8 位有符号整数				
tf.int16	16 位有符号整数				
tf.int32	32 位有符号整数				
tf.int64	64 位有符号整数				
tf.uint8	8 位无符号整数				
tf.float32	32 位浮点数				
tf.float64	64 位浮点数				
tf.string	字符串 (非Unicod	de编	码的字节	数组)	
tf.bool	布尔型				
tf.complex64	复数, 实部和虚部	分别]为32位汽	浮点型	_(,





■ 参数为数字

在创建张量时,指定元素的数据类型

In [10]: tf. constant (1.0, dtype=tf. float64)

Out[10]: <tf. Tensor: id=9, shape=(), dtype=float64, numpy=1.0>

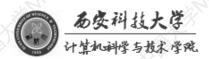
In [10]: tf.constant(1.0, dtype=float64)

NameError

(ipython-input-10-0ccd7816df6d) in module

---> 1 tf.constant(1.0, dtype=float64)

NameError: name 'float64' is not defined





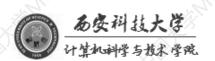
■ 参数为NumPy数组

```
numpy创建浮点数数组时,默认的浮点型是64位浮点数。
In [11]: import numpy as np
                                       当使用NumPy数组创建张量时, TensorFlow会接受数组
                                       元素的数据类型,使用64位浮点数保存数据。
In [12]: tf. constant (np. array ([1, 2]))
Out[12]: <tf. Tensor: id=10, shape=(2,), dtype=int32/numpy=array([1, 2])>
In [13]: tf. constant (np. array ([1.0, 2.0]))
Out[13]: <tf. Tensor: id=9, shape=(2,), dtype=float64, numpy=array([1., 2.])>
                                                           指明数据类型为32位浮点数
In [14]: tf. constant (np. array ([1.0, 2.0]), dtype=tf. float32)
Out[14]: <tf. Tensor: id=10, shape=(2,), dtype=float32, numpy=array([1., 2.], dtype=float32)>
```

□ tf.cast(x,dtype)函数: 改变张量中元素的数据类型

```
In [15]: a=tf. constant (np. array ([1, 2]))
          b=tf. cast(a, dtype=tf. float32)
          b. dtype
Out[15]:
          tf. float32
In [16]: a = tf. constant (123456789, dtype=tf. int32)
          tf. cast (a, tf. int16)
Out[16]: <tf. Tensor: id=14, shape=(), dtype=int16, numpy=-13035>
```

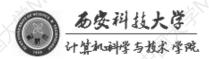
<u>在进行数据类型</u>转换时,自动将低精度的数据类型向高精度转换





■ 参数为布尔型

```
In [17]: tf. constant (True)
Out[17]: <tf. Tensor: id=15, shape=(), dtype=bool, numpy=True>
                                             布尔型转换为整型, 0: False, 1: True
In [18]: a = tf.constant([True, False])
         tf. cast (a, tf. int32)
Out[18]: \langle tf. Tensor: id=17, shape=(2,), dtype=int32, numpy=array([1, 0]) \rangle
In [19]: a = tf. constant([-1, 0, 1, 2])
                                          整型变量转换为布尔型,将非 0 数字都视为 True
         tf. cast(a, tf. bool)
Out[19]: <tf. Tensor: id=19, shape=(4,), dtype=bool, numpy=array([ True, False, True, True])>
```





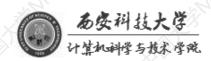
■ 参数为字符串

In [20]: tf. constant("hello")

Out[20]: <tf.Tensor: id=20, shape=(), dtype=string, numpy=b'hello'>

b: 表示这是一个字节串。

在Python3中,字符串默认是Unicode编码, 因此要转成字节串,就在原来的字符串前面加上一个b





□ tf.convert_to_tensor()函数

tf.convert_to_tensor(数组/列表/数字/布尔型/字符串)

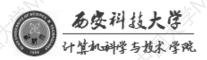
```
In [21]: na=np.arange(12).reshape(3, 4)
ta=tf.convert_to_tensor(na)
```

In [22]: type(na)

Out[22]: numpy. ndarray

In [23]: type(ta)

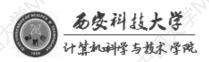
Out[23]: tensorflow.python.framework.ops.EagerTensor



```
In [24]: tf. is_tensor(ta)
Out[24]: True
In [25]: tf. is_tensor(na)
Out[25]: False
```

□ isinstance()函数

```
In [26]: isinstance(ta, tf. Tensor)
Out[26]: True
In [27]: isinstance(na, np. ndarray)
Out[27]: True
```



□ 创建全0张量和全1张量

tf.zeros(shape, dtype = tf.float32 默认为32位浮点数

tf.ones(shape, dtype=tf.float32)

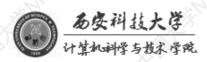
zeros函数的用法 和ones()相同

```
In [29]: tf. ones([6]) 形状也可以用方括号表示
```

```
Out[29]: <tf.Tensor: id=27, shape=(6,), dtype=float32, numpy=array ([1., 1., 1., 1., 1.], dtype=float32)>
```

```
In [30]: tf. ones([2, 3], tf. int32)
```

```
Out[30]: <tf.Tensor: id=30, shape=(2, 3), dtype=int32, numpy= array([[1, 1, 1], [1, 1, 1]])>
```



□ 创建元素值都相同的张量—— tf.fill()函数

tf.fill(<u>dims</u>, <u>value</u>)

形状

填充的数字

fill()函数没有dtype参数, 根据value参数自动判断数据类型

```
In [31]: tf.fill([2,3],9)
```

Out[31]: <tf. Tensor: id=33, shape=(2, 3), <pre>dtype=int32, numpy=
array([[9, 9, 9],

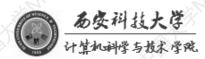
array([[9, 9, 9], [9, 9, 9]])>

In [32]: tf. fill([2, 3], 9.0)

Out[32]: <tf. Tensor: id=36, shape=(2, 3), dtype=float32, numpy=

array([[9., 9., 9.],

[9., 9., 9.]], dtype=float32)>



□ 创建元素值都相同的张量—— tf.constant()函数

```
填充的数字
                           形状
In [33]: tf. constant (9, shape=[2, 3])
Out[33]: <tf. Tensor: id=39, shape=(2, 3), dtype=int32, numpy=
         array([[9, 9, 9],
                [9, 9, 9]])
                                       为了避免混淆,建议加上参数名称
In [34]: tf. fill(dims=[2, 3], value=9)
Out[34]: <tf. Tensor: id=42, shape=(2, 3), dtype=int32, numpy=
         array([[9, 9, 9],
                [9, 9, 9]])>
In [35]: tf. constant (value=9, shape=[2, 3])
Out[35]: <tf. Tensor: (id=45, shape=(2, 3), dtype=int32, numpy=
         array([[9, 9, 9],
                [9, 9, 9]])>
```

□ 创建随机数张量——正态分布

tf.random.normal(shape, mean, stddev, dtype)

形状

均值

标准差

数据类型,默认是32位浮点数

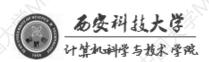
例: 创建一个2×2的张量, 其元素服从标准正态分布

In [36]: tf. random. normal([2, 2])

Out[36]: <tf. Tensor: id=51, shape=(2, 2), dtype=float32, numpy=

array([[-0.5383798, 0.63055885],

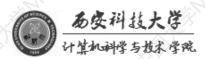
[0.8289045 , -0.58441126]], dtype=float32)>





例: 创建一个三维张量, 其元素服从正态分布

```
In [37]: tf. random. normal([3, 3, 3], mean=0.0, stddev=2.0)
Out[37]: <tf. Tensor: id=57, shape=(3, 3, 3), dtype=float32, numpy=
         array([[ 1.20538 , -0.8689349 , 0.38439623],
                 [-0.57567406, -1.5325377, -2.4028811],
                 [ 0.3453306 , -0.09237377, 0.8518045 ]],
                [[-2.8431058, 2.1145504, 0.00791129],
                 [ 1.849113 , -5.9299245 , 0.15747915],
                 [ 0. 29880348, -0. 22928385, 1. 5362241 ]],
                [[-0.64745957, 0.5811304, 2.894211],
                 [0.60582715, -3.3830724, -0.0153962],
                 [ 3.0395458 , 1.4132211 , 0.5158828 ]]], dtype=float32)>
```



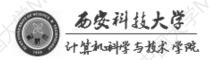


□ 创建随机数张量——截断正态分布

tf.random.truncated_normal(shape, mean, stddev, dtype)

- 返回一个截断的正态分布
- 截断的标准是2倍的标准差

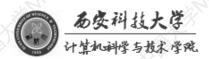
例如,当均值为0,标准差为1时, 使用tf.truncated_normal(),不可能出现区间[-2,2]以外的点 使用tf.random_normal(),可能出现[-2,2]以外的点





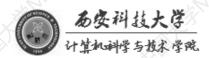
□ 创建随机数张量——截断正态分布

```
In [38]: tf. random. truncated normal([3, 3, 3], mean=0.0, stddev=2.0)
Out[38]: <tf. Tensor: id=63, shape=(3, 3, 3), dtype=float32, numpy=
         array([[[ 0.14917319, -1.1177629 , 1.882871 ],
                  [ 0.80246097, -1.269289 , 1.7788795 ],
                  [ 2. 2685118 , -0. 44797435, 1. 4552163 ]],
                [[ 0.44119573, 2.2127624, 1.6838118],
                 [ 2.4580297 , -1.8518591 , -3.7665844 ],
                 [ 2.6783562 , 0.34654975, -1.3416406 ]],
                [[-0.02359234, -2.911061, -3.7080057],
                 [-0.43191305, -0.7003556, -0.28586903],
                 [-1.0897896 , -3.2860935 , 0.21830656]]], dtype=float32)>
```



□ 设置随机种子——tf.random.set_seed()函数

```
设置随机种子,可以产生同样的随机数张量
In [39]: tf. random. set_seed(8)
         tf. random. normal ([2, 2])
Out[39]: <tf. Tensor: id=70, shape=(2, 2), dtype=float32, numpy=
         array([[ 1.2074401 , -0.7452463 ],
                 [ 0.6908678 , -0.76359874]], dtype=float32)>
In [40]: tf. random. set_seed(8)
         tf. random. normal ([2, 2])
Out[40]: <tf.Tensor: id=76, shape=(2, 2), dtype=float32, numpy=
         array([[ 1.2074401 , -0.7452463 ],
                 [ 0.6908678 , -0.76359874]], (dtype=float32)
```



□ 创建均匀分布张量——tf.random.uniform()函数

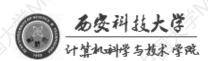
tf.random.uniform(shape,minval, maxval, dtype)

最小值,最大值 前闭后开,不包括<u>最大值</u>

```
In [41]: tf.random.uniform(shape=(3, 3), minval=0, maxval=10, dtype='int32')
```

```
Out[41]: <tf.Tensor: id=80, shape=(3, 3), dtype=int32, numpy= array([[6, 8, 3],
```

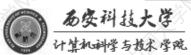
[9, 5, 1], [8, 6, 1]])>





□ 随机**打乱**——tf.random.shuffle()函数

```
In [49]: x = tf. constant([[1, 2], [3, 4], [5, 6]])
         tf. random. shuffle(x)
                                                 随机打乱第一维
                                      参数为张量,
Out[49]: <tf. Tensor: id=3, shape=(3, 2), dtype=int32, numpy=
         array([[1, 2],
                [5, 6],
                [3, 4])
In [50]: y=[1, 2, 3, 4, 5, 6]
         tf. random. shuffle(y)
                                   参数为Python列表
Out[50]: <tf. Tensor: id=5, shape=(6,), dtype=int32, numpy=array([2, 6, 5, 4, 1, 3])>
In [51]:
         z=np. arange (5)
         tf. random. shuffle(z)
                                  参数为NumPy数组
Out[51]: <tf. Tensor: id=7, shape=(5,), dtype=int32, numpy=array([1, 0, 2, 4, 3])>
```





□ 创建**序列**——tf.range()函数

tf.range(start, limit, delta=1,dtype)

起始数字,结束数字 前闭后开,不包括结束数字

In [42]: tf. range (10) 起始数字, 步长省略

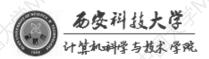
Out[42]: <tf.Tensor: id=84, shape=(10,), dtype=int32, numpy=array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])>

In [43]: tf. range(10, delta=2) 起始数字省略

Out[43]: <tf. Tensor: id=88, shape=(5,), dtype=int32, numpy=array([0, 2, 4, 6, 8])>

In [44]: tf. range(1, 10, delta=2)

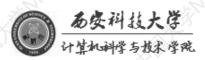
Out[44]: <tf.Tensor: id=92, shape=(5,), dtype=int32, numpy=array([1, 3, 5, 7, 9])>





□ 小结: 创建张量

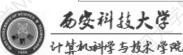
类 型	函 数	功 能		
	tf.constant(value, dtype, shape)	创建张量		
	tf.convert_to_tensor()	创建张量		
常量	tf.zeros(shape, dtype = tf.float32)	创建全0张量		
	tf.ones(shape, dtype = tf.float32)	创建全1张量		
	tf.fill(shape, value)	创建元素值全部相同的张量		
	tf.random.normal()	创建元素取值符合正态分布的张量		
随机量	tf.random.truncated_normal(shape, mean, stddev, dtype)	创建元素取值符合截断正态分布的张量		
	tf.random.uniform(shape,minval,maxval, dtype)	创建元素取值符合均匀分布的张量		
序列	tf.range(起始数字,结束数字,步长)	创建元素取值为整数序列的张量		



□ Tensor对象的属性——ndim、shape、dtype

```
In [45]: a=tf.constant([[1,2], [3,4]])
    print('ndim:', a.ndim)
    print('dtype:', a.dtype)
    print('shape:', a.shape)

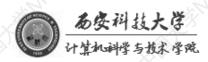
ndim: 2
    dtype: <dtype: 'int32'>
    shape: (2, 2)
```





□ 获得Tensor对象的形状、元素总数和维度

返回张量的形状 In [46]: tf. shape (a) Out [46]: <tf. Tensor: id=94, shape=(2,), dtype=int32, numpy=array([2, 2])> In [47]: tf. size(a) 返回张量中的元素总数 Out[47]: <tf. Tensor: id=95, shape=(), dtype=int32, numpy=4> 返回张量的维度 In [48]: tf. rank(a) Out[48]: <tf. Tensor: id=88, shape=(), dtype=int32, numpy=2>





□ 张量和NumPy数组

- 在TensorFlow中,所有的运算都是在张量之间进行的 NumPy数组仅仅是作为输入和输出来使用
- 张量可以运行于CPU,也可以运行于GPU和TPU 而NumPy数组只能够在CPU中运行

