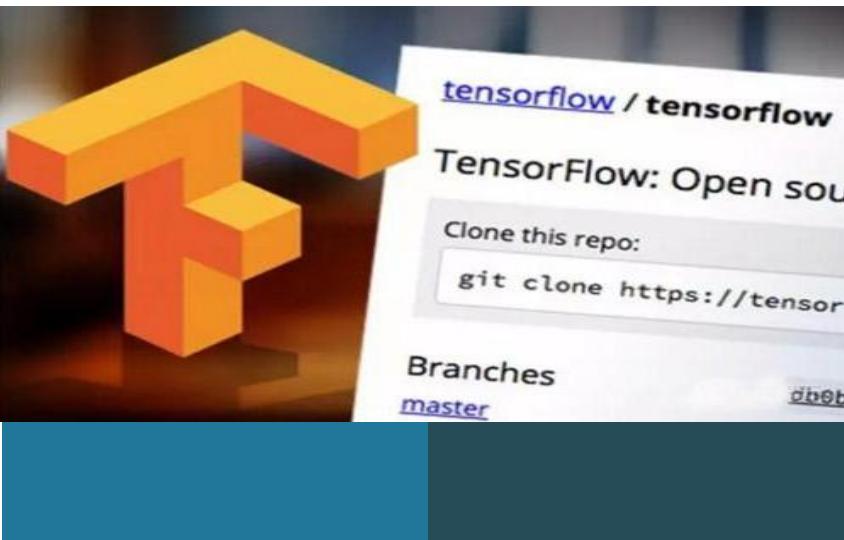




## 08 TensorFlow2.0基础

西安科技大学 牟琦  
[muqi@xust.edu.cn](mailto:muqi@xust.edu.cn)



## 8.1 TensorFlow2.0特性

## An end-to-end open source machine learning platform

□ **end-to-end**: 端到端

□ **open source**: 开放源代码，开放设计和实现框架

*TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*

*Wide & Deep Learning for Recommender Systems*

*The YouTube Video Recommendation System*

□ **machine learning**: 机器学习生态系统

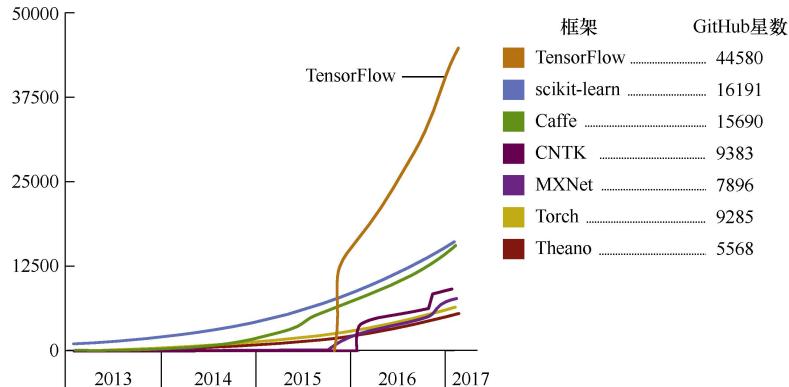
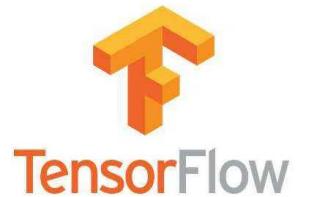


复旦科技大学

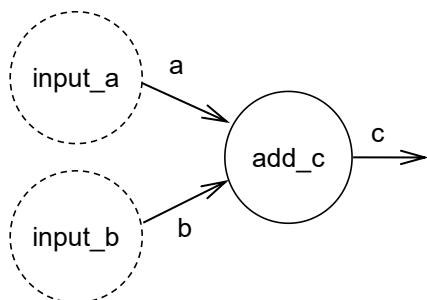
计算机科学与技术学院

## ■ TensorFlow的发展历程

- 2011: DistBelief
- 2015.11: TensorFlow 0.5.0
- 2017.02: TensorFlow 1.0
  - 高层API，将Keras库整合进其中
  - 动态图机制：Eager Execution
  - 面向移动智能终端：TensorFlow Lite
  - 面向网页前端：TensorFlow.js
  - 自动生成计算图：AutoGraph
- 2019: TensorFlow 2.0



## ■ TensorFlow1.x——延迟执行机制 (deferred execution) / 静态图机制



```
a=tf.constant(2,name="input_a")
b=tf.constant(3,name="input_b")
c=tf.add(a,b,name="add_c")
```

构建阶段

只是描述了计算图，  
并没有实质的计算发生

```
sess=tf.Session()
print(sess.run(c))
sess.close()
```

执行阶段

在会话中运行计算图

- 代码运行效率高，便于优化
- 程序不够简洁



## ■ TensorFlow2.0——动态图机制 (Eager Execution)

```
a=tf.constant(2,name="input_a")
b=tf.constant(3,name="input_b")
print(a+b)
```

- 无需首先创建静态图，可以立刻执行计算，并返回结果
- 能够快速的建立和调试模型
- 执行效率不高

兼顾**易用性**和**执行效率**——

在**程序调试**阶段使用**动态图**，快速建立模型、调试程序；

在**部署阶段**，采用**静态图**机制，从而提高模型的性能和部署能力



## ■ TensorFlow1.x——重复、冗余的API

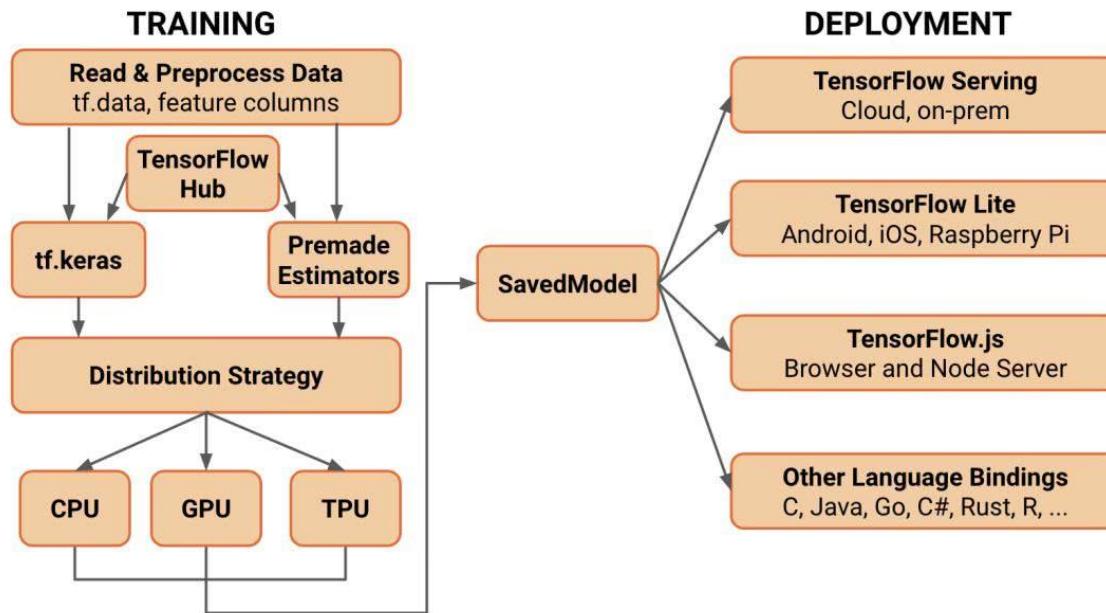
构建神经网络: tf.slim, tf.layers, tf.contrib.layers, tf.keras  
混乱, 不利于程序共享, 维护的成本高

## ■ TensorFlow2.0——清理 / 整合API

清理、整合了重复的API  
将tf.keras作为构建和训练模型的标准高级API



## TensorFlow2.0架构



## ■ TensorFlow框架特性

### □ 多种环境支持

可运行于移动设备、个人计算机、服务器、集群等  
云端、本地、浏览器、移动设备、嵌入式设备

### □ 支持分布式模式

TensorFlow会自动检测GPU和CPU，并充分利用它们并行、分布的执行

### □ 简洁高效

构建、训练、迭代模型：Eager Execution, Keras  
部署阶段：转化为静态图，提高执行效率。

### □ 社区支持



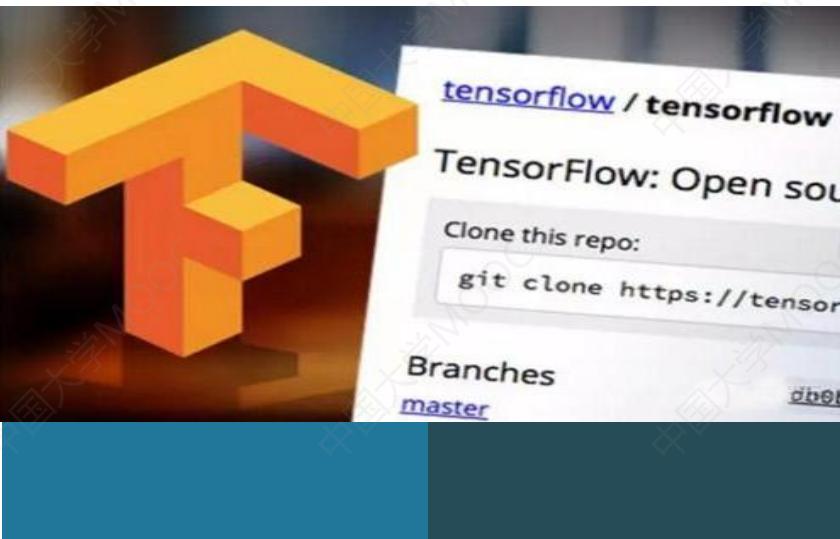
西安科技大学

计算机科学与技术学院

# 8.1 TensorFlow2.0特性

## □ 使用 TensorFlow 提供解决方案的部分企业





## 8.2 创建张量

## 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



## 8.2 创建张量

### ■ 更新到最新版本

```
In [ ]: !pip install --upgrade tensorflow  
!pip install --upgrade tensorflow-gpu
```

更新到最新版本

在Jupyter Notebook中运行cmd命令

### ■ 查看版本号

```
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



## 8.2 创建张量

在TensorFlow2.0的环境中，运行TensorFlow1.x代码，常常会**出现错误提示**

```
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")
      4
----> 5 sess=tf.Session()
      6 print(sess.run(c))
      7 sess.close()
```

**AttributeError: module 'tensorflow' has no attribute 'Session'**



## 8.2 创建张量

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



西安科技大学

计算机科学与技术学院

## ■ 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



## 8.2 创建张量

### ■ 参数为 Python 列表

```
In [4]: tf.constant([[1, 2], [3, 4]])
```

```
Out[4]: <tf.Tensor: id=3, shape=(2, 2), dtype=int32, numpy=
array([[1, 2],
       [3, 4]])>
```



## 8.2 创建张量

### ■ 张量的.numpy()方法

```
In [5]: a=tf.constant([[1, 2], [3, 4]])  
a.numpy()
```

```
Out[5]: array([[1, 2],  
[3, 4]])
```

TensorFlow2.0中的所有的张量，  
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
```

```
In [7]: print(a)
```

Eager Execution模式下的张量

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



## 8.2 创建张量

### ■ 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Unicode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位浮点型



## 8.2 创建张量

### ■ 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)
```

Traceback (most recent call last)

```
NameError: name 'float64' is not defined
```



## 8.2 创建张量

### ■ 参数为NumPy数组

```
In [11]: import numpy as np
```

numpy创建浮点数数组时，默认的浮点型是64位浮点数。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.])>
```

```
In [14]: tf.constant(np.array([1.0, 2.0]), dtype=tf.float32)
```

指明数据类型为32位浮点数

```
Out[14]: <tf.Tensor: id=10, shape=(2,), dtype=float32, numpy=array([1., 2.], dtype=float32)>
```



## 8.2 创建张量

### □ 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>

在进行数据类型转换时，自动将低精度的数据类型向高精度转换



## 8.2 创建张量

### ■ 参数为布尔型

```
In [17]: tf.constant(True)
```

```
Out[17]: <tf.Tensor: id=15, shape=(), dtype=bool, numpy=True>
```

```
In [18]: a = tf.constant([True, False])  
tf.cast(a, tf.int32)
```

布尔型转换为整型，0: False, 1 : True

```
Out[18]: <tf.Tensor: id=17, shape=(2,), dtype=int32, numpy=array([1, 0])>
```

```
In [19]: a = tf.constant([-1, 0, 1, 2])  
tf.cast(a, tf.bool)
```

整型变量转换为布尔型，将非 0 数字都视为 True

```
Out[19]: <tf.Tensor: id=19, shape=(4,), dtype=bool, numpy=array([ True, False,  True,  True])>
```



## 8.2 创建张量

### ■ 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



## 8.2 创建张量

### □ 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
```



## 8.2 创建张量

### □ is\_tensor()函数

```
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
```



## 8.2 创建张量

### □ 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 [28]: `tf.ones(shape=(2, 1))`

Out[28]: <tf.Tensor: id=24, shape=(2, 1), dtype=float32, numpy=array([1., 1.], dtype=float32)>

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., 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]])>



## 8.2 创建张量

### □ 创建元素值都相同的张量——tf.fill()函数

tf.fill( dims, value )

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), dtype=int32, numpy=  
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)>



## 8.2 创建张量

### □ 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 $2 \times 2$ 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>
```



## 8.2 创建张量

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



## 8.2 创建张量

### □ 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)>
```



## 8.2 创建张量

### □ 设置随机种子——tf.random.set\_seed()函数

```
In [39]: tf.random.set_seed(8)  
        tf.random.normal([2, 2])
```

设置随机种子，可以产生同样的随机数张量

```
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)>
```



## 8.2 创建张量

### □ 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, 不包括最大值

```
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]])>
```



## 8.2 创建张量

### □ 随机打乱——tf.random.shuffle()函数

```
In [49]: x = tf.constant([[1, 2], [3, 4], [5, 6]])  
tf.random.shuffle(x)
```

参数为张量，随机打乱第一维

```
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])>
```



## 8.2 创建张量

### □ 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])>



## 8.2 创建张量

### 口 小结：创建张量

类型	函数	功能
常量	tf.constant(value, dtype, shape)	创建张量
	tf.convert_to_tensor()	创建张量
	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
随机量	tf.random.normal()	创建元素取值符合正态分布的张量
	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



## 8.2 创建张量

### □ Tensor对象的属性——**ndim**、**shape**、**dtype**

```
In [4]: tf.constant([[1, 2], [3, 4]])  
  
Out[4]: <tf.Tensor: id=3, shape=(2, 2), dtype=int32, numpy=  
array([[1, 2],  
       [3, 4]])>
```

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

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



## 8.2 创建张量

### □ 获得Tensor对象的形状、元素总数和维度

In [46]: tf.shape(a)

返回张量的形状

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>

In [48]: tf.rank(a)

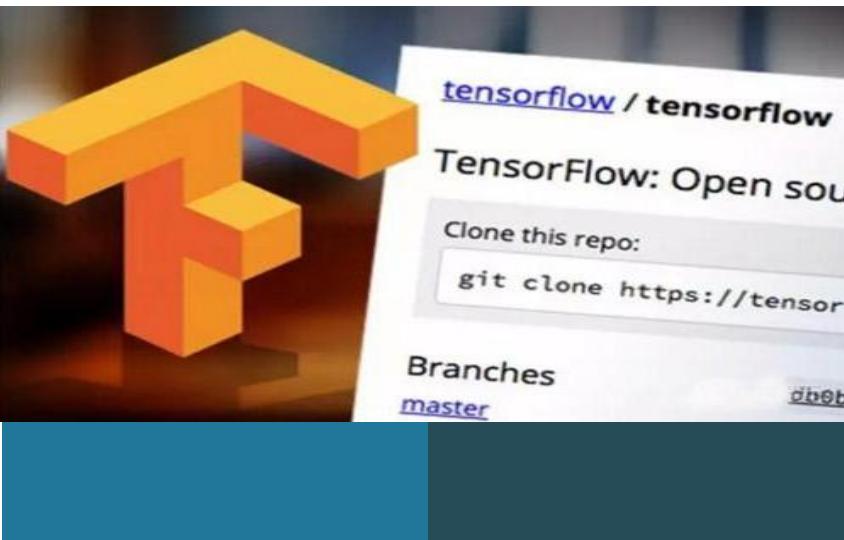
返回张量的维度

Out[48]: <tf.Tensor: id=88, shape=(), dtype=int32, numpy=2>



### 口 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

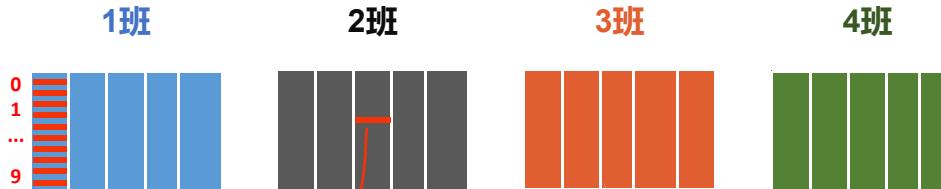


## 8.3 维度变换

## ■ 张量的存储和视图

多维张量在物理上以一维的方式连续存储  
通过定义维度和形状，在逻辑上把它理解为多维张量

逻辑组织：视图



物理组织：存储

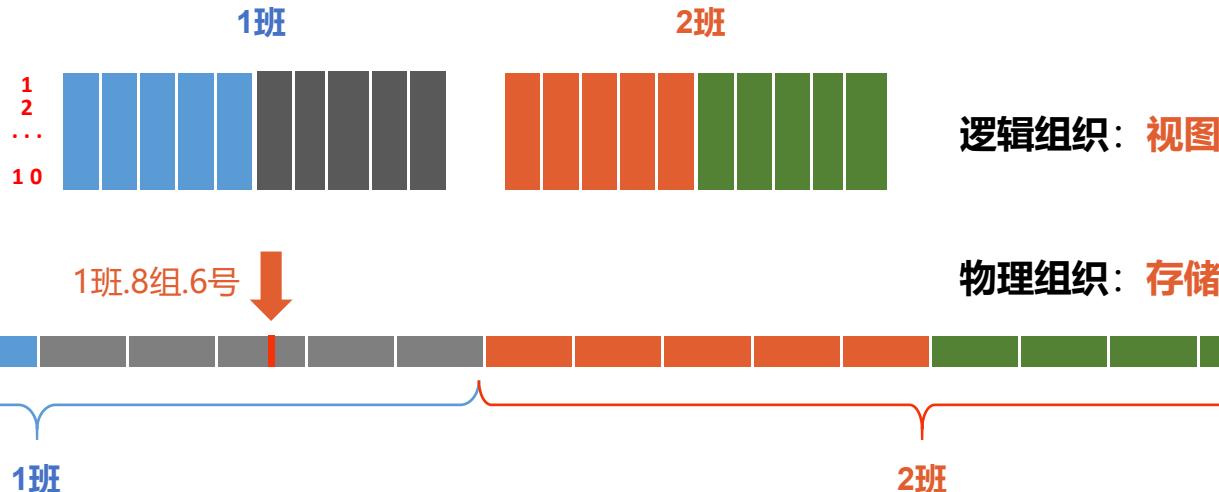


班级号.组号.组内序号



## ■ 张量的存储和视图

当对多维张量进行维度变换时，  
只是改变了逻辑上索引的方式，没有改变内存中的存储方式



1班

2班



兰州交通大学  
计算机科学与技术学院

## 改变张量的形状

tf.reshape (tensor, shape)

没有封装到Tensor对象中，  
前缀是tf，而不是张量对象

In [1]: `import tensorflow as tf`

In [2]: `a=tf.range(24)`  
`b=tf.reshape(a, [2, 3, 4])`  
`b`

Out[2]: <tf.Tensor: id=5, shape=(2, 3, 4), dtype=int32, numpy=  
array([[[ 0, 1, 2, 3],  
 [ 4, 5, 6, 7],  
 [ 8, 9, 10, 11]],  
  
 [[12, 13, 14, 15],  
 [16, 17, 18, 19],  
 [20, 21, 22, 23]]])>



## 8.3 维度变换

```
In [3]: import numpy as np
tf.constant(np.arange(24).reshape(2, 3, 4))
```

```
Out[3]: <tf.Tensor: id=6, shape=(2, 3, 4), dtype=int32, numpy=
array([[[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]],
      [[12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23]])>
```

```
In [4]: tf.reshape(b, [4, -1])
```

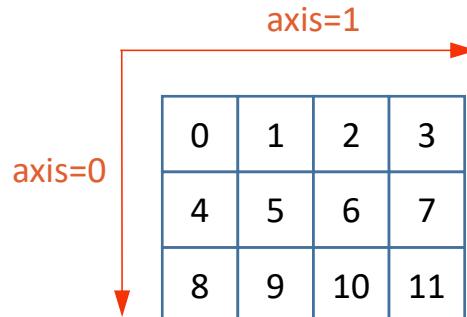
```
Out[4]: <tf.Tensor: id=8, shape=(4, 6), dtype=int32, numpy=
array([[ 0,  1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11],
       [12, 13, 14, 15, 16, 17],
       [18, 19, 20, 21, 22, 23]])>
```

shape参数=-1：  
自动推导出长度

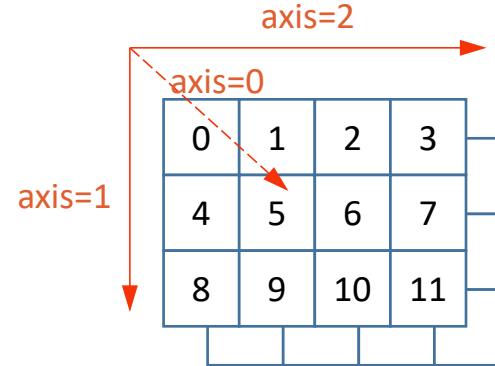
## ■ 多维张量的轴：张量的维度



$shape=(4,)$   
 $axis: 0$



$shape=(3, 4)$   
 $axis: 0, 1$



$shape=(2, 3, 4)$   
 $axis: 0, 1, 2$   
 $-3, -2, -1$

轴也可以是负数，  
 表示从后向前索引

张量中的轴的概念和用法，和NumPy数组是完全一样的



哈尔滨科技大学  
 计算机科学与技术学院

## ■ 增加和删除维度

### □ 增加维度

`tf.expand_dims(input, axis)`

增加的这个维度上，长度为1

```
In [5]: t = tf.constant([1, 2])  
print(t.shape)
```

(2, )

```
In [6]: t1 = tf.expand_dims(t, 1)  
print(t1.shape)
```

(2, 1)

在axis=1的轴上增加维度

```
In [7]: t1
```

```
Out[7]: <tf.Tensor: id=11, shape=(2, 1), dtype=int32, numpy=  
array([[1],  
       [2]])>
```



兰州交通大学

计算机科学与技术学院

## 8.3 维度变换

```
In [8]: t2 = tf.expand_dims(t, 0)    在axis=0的轴上增加维度  
print(t2.shape)
```

(1, 2)

```
In [9]: t2
```

```
Out[9]: <tf.Tensor: id=13, shape=(1, 2), dtype=int32, numpy=array([[1, 2]])>
```

```
In [10]: t3 = tf.expand_dims(t, -1)  
print(t3.shape)
```

(2, 1)

```
In [11]: t3
```

```
Out[11]: <tf.Tensor: id=15, shape=(2, 1), dtype=int32, numpy=  
array([[1],  
       [2]])>
```

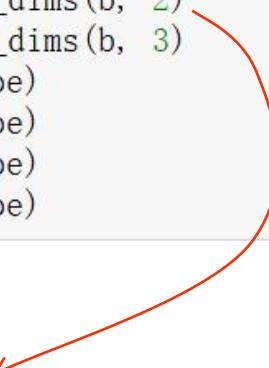


## 8.3 维度变换

```
In [12]: a=tf.range(24)
          b=tf.reshape(a, [2, 3, 4])
          print(b. shape)
```

(2, 3, 4)

```
In [13]: b1=tf.expand_dims(b, 0)
          b2=tf.expand_dims(b, 1)
          b3=tf.expand_dims(b, 2)
          b4=tf.expand_dims(b, 3)
          print(b1. shape)
          print(b2. shape)
          print(b3. shape)
          print(b4. shape)
```

(1, 2, 3, 4)  
(2, 1, 3, 4)  
(2, 3, 1, 4)   
(2, 3, 4, 1)



西安科技大学

计算机科学与技术学院

## 8.3 维度变换

### □ 删除维度

```
tf.squeeze( input, axis=None )
```

原始张量

要删除的维度

只能删除**长度为 1** 的维度，  
省略时删除所有长度为1的维度

**例：** # *t is a tensor of shape ( 1, 2, 1, 3, 1 )*

```
>>>tf.shape(tf.squeeze(t))  
(2, 3)
```

```
>>>tf.shape(tf.squeeze(t,[2,4]))  
(1,2, 3)
```

增加维度和删除维度，**只是改变了张量的视图**，**不会改变张量的存储**



兰州交通大学

计算机科学与技术学院

## ■ 交换维度

tf.transpose( a, perm )

对二维张量交换维度，就是矩阵的转置

```
In [14]: x = tf.constant([[1, 2, 3], [4, 5, 6]])  
x
```

```
Out[14]: <tf.Tensor: id=30, shape=(2, 3), dtype=int32, numpy=  
array([[1, 2, 3],  
       [4, 5, 6]])>
```

```
In [15]: tf.transpose(x)
```

```
Out[15]: <tf.Tensor: id=32, shape=(3, 2), dtype=int32, numpy=  
array([[1, 4],  
       [2, 5],  
       [3, 6]])>
```



## ■ 交换维度

`tf.transpose( a, perm )`

调整张量中各个轴的顺序

```
In [16]: x = tf.constant([[1, 2, 3], [4, 5, 6]])  
x
```

```
Out[16]: <tf.Tensor: id=33, shape=(2, 3), dtype=int32, numpy=  
array([[1, 2, 3],  
       [4, 5, 6]])>
```

```
In [17]: tf.transpose(x, perm=[1, 0])
```

改变轴的顺序，从而改变张量的形状

```
Out[17]: <tf.Tensor: id=35, shape=(3, 2), dtype=int32, numpy=  
array([[1, 4],  
       [2, 5],  
       [3, 6]])>
```



## 8.3 维度变换

```
In [18]: a=tf.range(24) 0 1 2  
b=tf.reshape(a, [2, 3, 4])  
b
```

```
Out[18]: <tf.Tensor: id=41, shape=(2, 3, 4), dtype=int32, numpy  
=  
array([[[ 0,  1,  2,  3],  
       [ 4,  5,  6,  7],  
       [ 8,  9, 10, 11]],  
      [[12, 13, 14, 15],  
       [16, 17, 18, 19],  
       [20, 21, 22, 23]]])>
```

交换维度，不仅改变了张量的视图，同时也**改变了张量的存储顺序**

In [19]: tf.transpose(b, (1, 0, 2)) 最后一维不变，前两维交换

```
Out[19]: <tf.Tensor: id=43, shape=(3, 2, 4), dtype=int32, numpy  
=  
array([[[ 0,  1,  2,  3],  
       [12, 13, 14, 15]],  
      [[ 4,  5,  6,  7],  
       [16, 17, 18, 19]],  
      [[ 8,  9, 10, 11],  
       [20, 21, 22, 23]]])>
```



## ■ 拼接和分割

### □ 拼接张量

- 将多个张量在某个维度上**合并**
- 拼接并不会产生新的维度

`tf.concat( tensors, axis )`

所有需要拼接的张量列表

指定在哪个轴上进行拼接

```
In [20]: t1 = [[1, 2, 3], [4, 5, 6]]  
t2 = [[7, 8, 9], [10, 11, 12]]
```

shape: (2,3)

```
In [21]: tf.concat([t1, t2], 0) (2, 3)  
(2, 3)
```

在axis=0的轴上拼接,  
2+2=4

```
Out[21]: <tf.Tensor: id=47, shape=(4, 3), dtype=int32, numpy=  
array([[ 1,  2,  3],  
       [ 4,  5,  6],  
       [ 7,  8,  9],  
       [10, 11, 12]])>
```

axis=0

```
In [22]: tf.concat([t1, t2], 1) (2, 3)  
(2, 3)
```

在axis=1的轴上拼接,  
3+3=6

```
Out[22]: <tf.Tensor: id=51, shape=(2, 6), dtype=int32, numpy=  
array([[ 1,  2,  3,  7,  8,  9],  
       [ 4,  5,  6, 10, 11, 12]])>
```

axis=1



### □ 分割张量：将一个张量拆分成多个张量，分割后维度不变

```
tf.split( value, num_or_size_splits, axis=0 )
```

待分割张量

分割方案

指明分割的轴

**分割方案**：是一个**数值**时，表示**等长分割**，数值是切割的**份数**；

是一个**列表**时，表示**不等长切割**，列表中是切割后每份的长度

**例：**

**2**：分割成**2个**张量

**[1:2:1]**：就表示分割成**3个**张量，长度分别是**1,2,1**



西安科技大学

计算机科学与技术学院

## □ 分割张量

### ■ 在axis=0轴上分割

```
In [23]: x=tf.range(24)
x=tf.reshape(x, [4, 6])
x
```

```
Out[23]: <tf.Tensor: id=57, shape=(4, 6), dtype=int32, numpy=
array([[ 0,  1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11],
       [12, 13, 14, 15, 16, 17],
       [18, 19, 20, 21, 22, 23]])>
```

In [24]: tf.split(x, 2, 0) 在axis=0的轴上分割，平均分成2份

```
Out[24]: [<tf.Tensor: id=60, shape=(2, 6), dtype=int32, numpy=
array([[ 0,  1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11]]),
<tf.Tensor: id=61, shape=(2, 6), dtype=int32, numpy=
array([[12, 13, 14, 15, 16, 17],
       [18, 19, 20, 21, 22, 23]])]
```



## 8.3 维度变换

### ■ 在axis=0轴上分割

```
In [23]: x=tf.range(24)  
x=tf.reshape(x, [4, 6])  
x
```

```
Out[23]: <tf.Tensor: id=57, shape=(4, 6), dtype=int32, numpy=  
array([[ 0,  1,  2,  3,  4,  5],  
       [ 6,  7,  8,  9, 10, 11],  
       [12, 13, 14, 15, 16, 17],  
       [18, 19, 20, 21, 22, 23]])>
```

axis=0

```
In [25]: tf.split(x, [1, 2, 1], 0)
```

在axis=0的轴上分割  
分割成3个张量，axis=0的轴上，长度依次为1,2,1

```
Out[25]: [<tf.Tensor: id=64, shape=(1, 6), dtype=int32, numpy=a  
rray([[0, 1, 2, 3, 4, 5]]),  
      <tf.Tensor: id=65, shape=(2, 6), dtype=int32, numpy=  
array([[ 6,  7,  8,  9, 10, 11],  
           [12, 13, 14, 15, 16, 17]]),  
      <tf.Tensor: id=66, shape=(1, 6), dtype=int32, numpy=a  
rray([[18, 19, 20, 21, 22, 23]])]
```

## 8.3 维度变换

### ■ 在axis=1轴上分割

```
In [23]: x=tf.range(24)
x=tf.reshape(x, [4, 6])
x
```

```
Out[23]: <tf.Tensor: id=57, shape=(4, 6), dtype=int32, numpy=
array([[ 0,  1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11],
       [12, 13, 14, 15, 16, 17],
       [18, 19, 20, 21, 22, 23]])>
```

axis=1

图像的分割与拼接，  
改变了张量的视图，  
张量的存储顺序并  
没有改变。



聊城大学

计算机科学与技术学院

```
In [26]: tf.split(x, [2, 4], 1)
```

在axis=1的轴上分割  
分割成2个张量，axis=1的轴上，长度依次为2,4

```
Out[26]: [<tf.Tensor: id=69, shape=(4, 2), dtype=int32, numpy=
array([[ 0,  1],
       [ 6,  7],
       [12, 13],
       [18, 19]])>, <tf.Tensor: id=70, shape=(4, 4),
dtype=int32, numpy=
array([[ 2,  3,  4,  5],
       [ 8,  9, 10, 11],
       [14, 15, 16, 17],
       [20, 21, 22, 23]])>]
```

## 堆叠和分解

### 堆叠张量

- 在合并张量时，  
创建一个新的维度
- 和NumPy中堆叠函数的功能完全一样

3个张量堆叠：

shape: (4,)      axis=0 → (3,4)  
                   axis=1 → (4,3)

`tf.stack( values, axis )`

要堆叠的多个张量

指定插入新维度的位置

In [27]: `x=tf.constant([1, 2, 3])`  
`y=tf.constant([4, 5, 6])` shape: (3,)

In [28]: `tf.stack((x, y), axis=0)` 在axis=0的轴上堆叠

Out[28]: <tf.Tensor: id=73, shape=(2, 3), dtype=int32, numpy=array([[1, 2, 3], [4, 5, 6]])>

In [29]: `tf.stack((x, y), axis=1)` 在axis=1的轴上堆叠

Out[29]: <tf.Tensor: id=74, shape=(3, 2), dtype=int32, numpy=array([[1, 4], [2, 5], [3, 6]])>



复旦科技大学

计算机科学与技术学院

## 8.3 维度变换

### □ 分解张量

`tf.unstack( values, axis )`

- 是张量堆叠的逆运算
- 张量分解为多个张量
- 分解后得到的每个张量，和原来的张量相比，维数都少了一维

```
In [30]: c = tf.constant([[1, 2, 3], [4, 5, 6]])  
c
```

```
Out[30]: <tf.Tensor: id=75, shape=(2, 3), dtype=int32, numpy=  
array([[1, 2, 3],  
       [4, 5, 6]])>
```

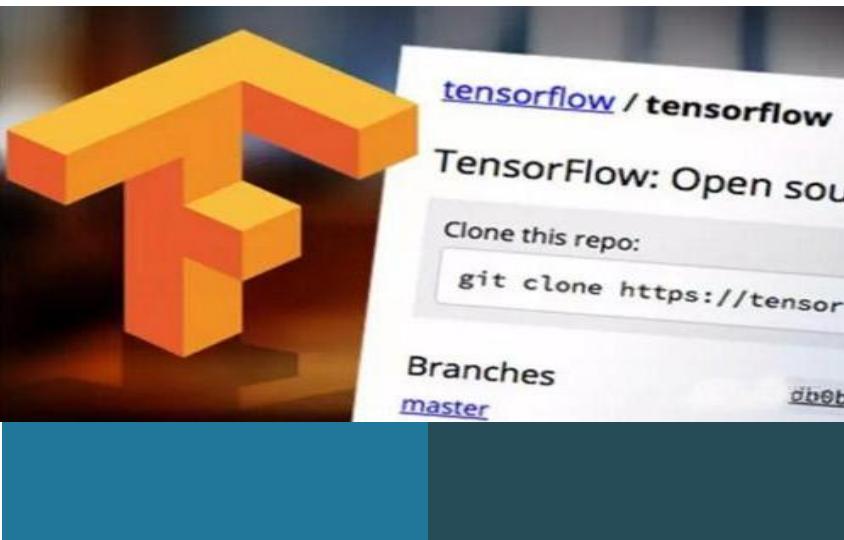
```
In [31]: tf.unstack(c, axis=0)
```

```
Out[31]: [<tf.Tensor: id=76, shape=(3,), dtype=int32, numpy=arr  
ay([1, 2, 3])>,  
          <tf.Tensor: id=77, shape=(3,), dtype=int32, numpy=arr  
ay([4, 5, 6])>]
```

```
In [32]: tf.unstack(c, axis=1)
```

```
Out[32]: [<tf.Tensor: id=78, shape=(2,), dtype=int32, numpy=arr  
ay([1, 4])>,  
          <tf.Tensor: id=79, shape=(2,), dtype=int32, numpy=arr  
ay([2, 5])>,  
          <tf.Tensor: id=80, shape=(2,), dtype=int32, numpy=arr  
ay([3, 6])>]
```





## 8.4 部分采样

## ■ 索引和切片

### □ 索引

#一维张量  
`a[1]`

#二维张量  
`b[1][1]`  
`b[1,1]`

#三维张量  
`c[1][1][1]`  
`c[1,1,1]`

**例：**手写数字图像数据集MNIST: **(6000, 28, 28)**

**mnist[0]**: 取第1张图片中的数据

**mnist[0][1]** : 取第1张图片中的第2行

**mnist[0][1][2]**: 取第1张图片中的第2行的第3列



 切片

起始位置:结束位置:步长

- 起始位置: 结束位置, 是前闭后开的, 切片中不包含结束位置
- 起始位置、结束位置、步长都可以省略
- 步长可以是负数, 这时起始位置的索引号, 应该大于结束位置

In [1]: `import tensorflow as tf`In [2]: `a=tf.range(10)`In [3]: `a[:, :, :]` 当3个参数全部省略时, 表示读取所有数据, 步长为1

Out[3]: &lt;tf.Tensor: id=7, shape=(10,), dtype=int32, numpy=array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])&gt;

In [4]: `a[:, :, 2::2]` 从第一个元素开始, 隔行采样, 直到最后一个元素为止

Out[4]: &lt;tf.Tensor: id=11, shape=(5,), dtype=int32, numpy=array([0, 2, 4, 6, 8])&gt;



## 8.4 部分采样

### □ 一维张量切片

In [5]: `a[1::2]` 读出所有的奇数

Out[5]: <tf.Tensor: id=15, shape=(5,), dtype=int32, numpy=array([1, 3, 5, 7, 9])>

In [6]: `a[::-1]` 从最后一个元素开始，逆序取出所有元素

Out[6]: <tf.Tensor: id=19, shape=(10,), dtype=int32, numpy=array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])>

In [7]: `a[::-2]` 从最后一个元素开始，逆序取出间隔的元素

Out[7]: <tf.Tensor: id=23, shape=(5,), dtype=int32, numpy=array([9, 7, 5, 3, 1])>



## 8.4 部分采样

### □ 下载鸢尾花数据集

下载鸢尾花数据集

```
In [8]: TRAIN_URL = "http://download.tensorflow.org/data/iris_training.csv"  
train_path = tf.keras.utils.get_file(TRAIN_URL.split('/')[-1], TRAIN_URL)
```

```
In [9]: import pandas as pd  
df_iris=pd.read_csv(train_path) 使用Pandas读取到二维表格对象df_iris中
```

```
In [10]: import numpy as np  
np_iris=np.array(df_iris) 转换为NumPy数组np_iris
```

```
In [11]: iris=tf.convert_to_tensor(np_iris) 转换为张量  
iris.shape
```

```
Out[11]: TensorShape([120, 5]) 二维张量, 共120个样本,  
前4列是属性, 最后1列是标记
```



兰州交通大学

计算机科学与技术学院

### □ 二维张量切片：维度之间用逗号隔开

```
In [12]: iris[0, :] 读取第一个样本的所有列（包括属性和标记）
```

```
Out[12]: <tf.Tensor: id=28, shape=(5,), dtype=float64, numpy=array([6.4, 2.8, 5.6, 2.2, 2.])>
```

```
In [13]: iris[0:5, 0:4] 读取前5个样本的所有属性
```

```
Out[13]: <tf.Tensor: id=32, shape=(5, 4), dtype=float64, numpy=
array([[6.4, 2.8, 5.6, 2.2],
       [5. , 2.3, 3.3, 1. ],
       [4.9, 2.5, 4.5, 1.7],
       [4.9, 3.1, 1.5, 0.1],
       [5.7, 3.8, 1.7, 0.3]])>
```



## 8.4 部分采样

In [14]: `iris[:, 0]`

读取所有样本的第1个属性

```
Out[14]: <tf.Tensor: id=36, shape=(120,), dtype=float64, numpy=
array([6.4, 5. , 4.9, 4.9, 5.7, 4.4, 5.4, 6.9, 6.7, 5.1, 5.2, 6.9, 5.8,
       5.4, 7.7, 6.3, 6.8, 7.6, 6.4, 5.7, 6.7, 6.4, 5.4, 6.1, 7.2, 5.2,
       5.8, 5.9, 5.4, 6.7, 6.3, 5.1, 6.4, 6.8, 6.2, 6.9, 6.5, 5.8, 5.1,
       4.8, 7.9, 5.8, 6.7, 5.1, 4.7, 6. , 4.8, 7.7, 4.6, 7.2, 5. , 6.6,
       6.1, 5. , 7. , 6. , 7.4, 5.8, 6.2, 5. , 5.6, 6.7, 6.3, 6.4, 6.2,
       7.3, 4.4, 7.2, 6.5, 5. , 4.7, 6.6, 5.5, 7.7, 6.1, 4.9, 5.5, 5.7,
       6. , 6.4, 5.4, 6.1, 6.5, 5.6, 6.3, 4.9, 6.8, 5.7, 6. , 5. , 6.5,
       6.1, 5.1, 4.6, 6.5, 4.6, 4.6, 7.7, 5.9, 5.1, 4.9, 4.9, 4.5, 5.8,
       5. , 5.2, 5.3, 5. , 5.6, 4.8, 6.3, 5.7, 5. , 6.3, 5. , 5.5, 5.7,
       4.4, 4.8, 5.5])>
```

In [15]: `iris[0:10, 0:4]`

读取前10个样本的所有属性

```
Out[15]: <tf.Tensor: id=40, shape=(10, 4), dtype=float64, numpy=
array([[6.4, 2.8, 5.6, 2.2],
       [5. , 2.3, 3.3, 1. ],
       [4.9, 2.5, 4.5, 1.7],
       [4.9, 3.1, 1.5, 0.1],
       [5.7, 3.8, 1.7, 0.3],
       [4.4, 3.2, 1.3, 0.2],
       [5.4, 3.4, 1.5, 0.4],
       [6.9, 3.1, 5.1, 2.3],
       [6.7, 3.1, 4.4, 1.4],
       [5.1, 3.7, 1.5, 0.4]])>
```



西安科技大学

计算机科学与技术学院

## □ 三维张量切片——手写数字数据集MNIST (60000,28,28)

**mnist[0, ::, ::]**: 第1张图片

**[0, :, :]** 为了更加简洁，两个冒号可以简写为一个冒号

**mnist[0:10, :, :]**: 前10张图片

**mnist[0:20, 0:28:2, :]**: 前20张图片的所有的行，隔行采样

**mnist[:, 0:28:2, 0:28:2]**: 对所有的图片，隔行采样，并且隔列采样



### □ 三维张量切片——彩色图片lena (**512, 512, 3**)

`lena[:, :, 0]`: R通道的图片

`lena[:, :, 2]`: B通道的图片

### □ 四维张量切片——多张彩色图片 (**4, 512, 512, 3**)

`image[0, :, :, 0]`: 第1张图片的R通道

`image[0:2, :, :, 2]`: 前2张图片的B通道

`image[0:2, 0:512:2, :, 2]`: 对前2张图片的B通道图片隔行采样

采用切片的方式，只能进行**连续的、或者有规律的**采样



## ■ 数据提取：根据索引，抽取出没有规律的、特定的数据

□ **gather()**函数：用一个**索引列表**，将**给定张量**中对应索引值的元素提取出来

gather( params, indices )

输入张量

索引值列表

```
In [16]: a=tf.range(5)
```

```
In [17]: tf.gather(a, indices=[0, 2, 3])
```

从张量a中提取索引值分别为 0,2,3 的元素

```
Out[17]: <tf.Tensor: id=47, shape=(3,), dtype=int32, numpy=array([0, 2, 3])>
```



## □ 对多维张量采样——gather()、gather\_nd()函数

gather( params, **axis**, indices )

gather()函数一次对一个维度进行索引

说明在哪个轴上采样

创建张量

```
In [18]: a=tf.range(20)  
b=tf.reshape(a, [4, 5])  
b
```

创建二维张量

```
Out[18]: <tf.Tensor: id=53, shape=(4, 5), dtype=int32, numpy=  
array([[ 0,  1,  2,  3,  4],  
       [ 5,  6,  7,  8,  9],  
       [10, 11, 12, 13, 14],  
       [15, 16, 17, 18, 19]])>
```



兰州交通大学

计算机科学与技术学院

## 8.4 部分采样

对axis=0的轴采样 (对行采样)

In [19]: `tf.gather(b, axis=0, indices=[0, 2, 3])`

	axis=1				
0	0, 1, 2, 3, 4				
1	5, 6, 7, 8, 9				
2	10, 11, 12, 13, 14				
3	15, 16, 17, 18, 19				

Out[19]: <tf.Tensor: id=56, shape=(3, 5), dtype=int32, numpy=  
array([[ 0, 1, 2, 3, 4],  
 [10, 11, 12, 13, 14],  
 [15, 16, 17, 18, 19]])>

对axis=1的轴采样 (对列采样)

In [20]: `tf.gather(b, axis=1, indices=[0, 2, 3])`

Out[20]: <tf.Tensor: id=59, shape=(4, 3), dtype=int32, numpy=  
array([[ 0, 2, 3],  
 [ 5, 7, 8],  
 [10, 12, 13],  
 [15, 17, 18]])>

	0	1	2	3	4
0	0, 1, 2, 3, 4				
1	5, 6, 7, 8, 9				
2	10, 11, 12, 13, 14				
3	15, 16, 17, 18, 19				

axis=0



## □ 同时采样多个点——gather\_nd()函数

```
[[ 0,  1,  2,  3,  4],
 [ 5,  6,  7,  8,  9],
 [10, 11, 12, 13, 14],
 [15, 16, 17, 18, 19]]
```

[21]: `tf.gather_nd(b, [[0, 0], [1, 1], [2, 3]])`

[21]: <tf.Tensor: id=61, shape=(3,), dtype=int32, numpy=array([0, 6, 13])>

和使用3次索引的结果是一样的

In [22]: `b[0, 0]`

Out[22]: <tf.Tensor: id=65, shape=(), dtype=int32, numpy=0>

In [23]: `b[1, 1]`

Out[23]: <tf.Tensor: id=69, shape=(), dtype=int32, numpy=6>

In [24]: `b[2, 3]`

Out[24]: <tf.Tensor: id=73, shape=(), dtype=int32, numpy=13>



### □ 选择采样维度

三维张量——彩色图片lena (**512, 512, 3**)

```
tf.gather_nd(lena, [[0, 0],[1, 1],[2,3]])
```

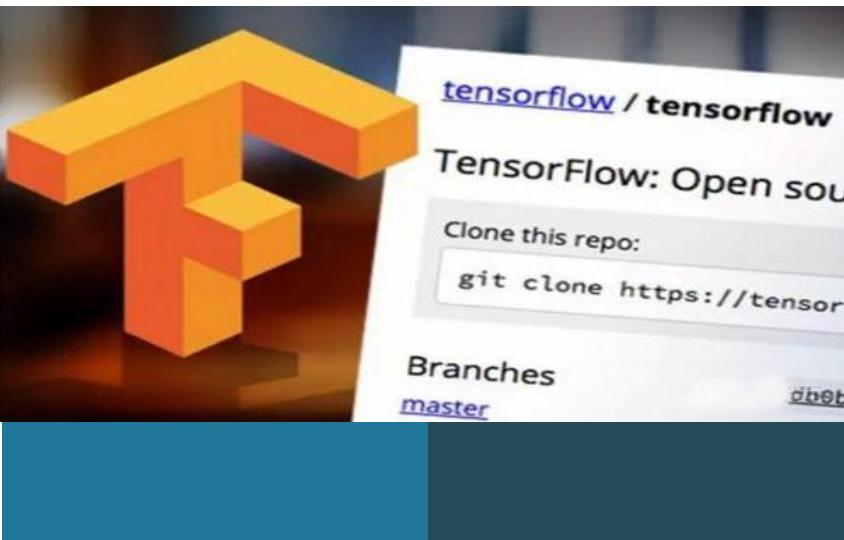
对前两维采样，表示取得这些点所有通道的值

三维张量——手写数字数据集MNIST (**60000,28,28**)

```
tf.gather_nd(mnist, [[0],[2],[3]])
```

只对第一维采样，表示取到索引值为0,2,3的3张图片





## 8.5 张量运算

## ■ 基本数学运算

### □ 加减乘除运算

参数分别是参与运算的两个张量

#### 算术操作      描述

tf.add(x, y)      将x和y逐元素相加

tf.subtract(x, y)      将x和y逐元素相减

tf.multiply(x, y)      将x和y逐元素相乘

tf.divide(x, y)      将x和y逐元素相除

tf.math.mod(x, y)      对x逐元素取模

In [1]: `import tensorflow as tf`

In [2]: `a = tf.constant([0, 1, 2])  
b = tf.constant([3, 4, 5])  
tf.add(a, b)`

Out[2]: <tf.Tensor: id=2, shape=(3,), dtype=int32, numpy=array([3, 5, 7])>



西安科技大学

计算机科学与技术学院

## □ 幂指对数运算

算术操作	描述
<code>tf.pow(x, y)</code>	对x求y的幂次方
<code>tf.square(x)</code>	对x逐元素求计算平方
<code>tf.sqrt(x)</code>	对x逐元素开平方根
<code>tf.exp(x)</code>	计算e的x次方
<code>tf.math.log(x)</code>	计算自然对数,底数为e



## ■ 一维张量幂运算

```
In [3]: x = tf.range(4)  
x
```

```
Out[3]: <tf.Tensor: id=6, shape=(4,), dtype=int32, numpy=array([0, 1, 2, 3])>
```

```
In [4]: tf.pow(x, 2)
```

```
Out[4]: <tf.Tensor: id=8, shape=(4,), dtype=int32, numpy=array([0, 1, 4, 9])>
```



对x张量中的每个元素计算了2次方



兰州交通大学

计算机科学与技术学院

## 8.5 张量运算

### ■ 二维张量幂运算

```
In [5]: x= tf.constant([[2, 2], [3, 3]])
y= tf.constant([[8, 16], [2, 3]])
tf.pow(x, y)
```

```
Out[5]: <tf.Tensor: id=11, shape=(2, 2), dtype=int32, numpy=
array([[ 256, 65536],
       [     9,     27]])>
```

```
In [6]: x= tf.constant([1., 4., 9., 16.])
pow(x, 0.5)
```

```
Out[6]: <tf.Tensor: id=14, shape=(4, ), dtype=float32, numpy=array([1., 2., 3., 4.], dtype=float32)>
```

张量的元素必须是浮点数类型

平方根



哈尔滨科技大学

计算机科学与技术学院

## 8.5 张量运算

### ■ 平方和平方根运算

```
In [7]: x=tf.constant([1, 2, 3, 4])  
tf.square(x)
```

```
Out[7]: <tf.Tensor: id=16, shape=(4,), dtype=int32, numpy=array([ 1,  4,  9, 16])>
```

```
In [8]: x=tf.constant([1., 4., 9., 16.])  
tf.sqrt(x)
```

```
Out[8]: <tf.Tensor: id=18, shape=(4,), dtype=float32, numpy=array([1., 2., 3., 4.], dtype=float32)>
```

```
In [9]: f = tf.constant([[1., 9.], [16., 100.]])  
tf.sqrt(f)
```

```
Out[9]: <tf.Tensor: id=20, shape=(2, 2), dtype=float32, numpy=  
array([[ 1.,  3.],  
       [ 4., 10.]], dtype=float32)>
```



## ■ 自然指数和自然对数运算

TensorFlow中只有以e为底的自然对数，没有提供以其他数值为底的对数运算函数

```
In [10]: tf.exp(1.)
```

张量的元素必须是浮点数类型

```
Out[10]: <tf.Tensor: id=22, shape=(), dtype=float32, numpy=2.7182817>
```

```
In [11]: x=tf.exp(3.)
```

```
tf.math.log(x)
```

自然对数在math模块中

```
Out[11]: <tf.Tensor: id=25, shape=(), dtype=float32, numpy=3.0>
```



## ■ 对数运算

$$\log_a b = \frac{\log_c b}{\log_c a}$$

要计算其他底数的对数，可以利用对数的换底公式，间接的通过 `tf.math.log(x)` 函数来实现。

```
In [12]: x=tf.constant(256.)
y=tf.constant(2.)
tf.math.log(x)/tf.math.log(y)
```

$$\log_2 256 = \frac{\log_e 256}{\log_e 2}$$

Out[12]: <tf.Tensor: id=30, shape=(), dtype=float32, numpy=8.0>

```
In [13]: x = tf.constant([[1., 9.], [16., 100.]])  
y = tf.constant([[2., 3.], [2., 10.]])  
tf.math.log(x) / tf.math.log(y)
```

真数

底数

Out[13]: <tf.Tensor: id=35, shape=(2, 2), dtype=float32, numpy=  
array([[0., 2.],  
 [4., 2.]], dtype=float32)>



## □ 其他运算

函数	描述
<code>tf.sign(x)</code>	返回x的符号
<code>tf.abs(x)</code>	对x逐元素求绝对值
<code>tf.negative(x)</code>	对x逐元素求相反数, $y = -x$
<code>tf.reciprocal(x)</code>	取x的倒数
<code>tf.logical_not(x)</code>	对x逐元素求的逻辑非
<code>tf.ceil(x)</code>	向上取整
<code>tf.floor(x)</code>	向下取整
<code>tf.rint(x)</code>	取最接近的整数
<code>tf.round(x)</code>	对x逐元素求舍入最接近的整数
<code>tf.maximum(x, y)</code>	返回两tensor中的最大值
<code>tf.minimum(x, y)</code>	返回两tensor中的最小值

## □ 三角函数和反三角函数运算

函 数	描 述
<code>tf.cos(x)</code>	三角函数cos
<code>tf.sin(x)</code>	三角函数sin
<code>tf.tan(x)</code>	三角函数tan
<code>tf.acos(x)</code>	反三角函数arccos
<code>tf.asin(x)</code>	反三角函数arcsin
<code>tf.atan(x)</code>	反三角函数arctan



## ■ 重载运算符

运算符	构造方法	运算符	构造方法
$x+y$	<code>tf.add( )</code>	$x\&y$	<code>tf.logical_and( )</code>
$x-y$	<code>tf.subtract( )</code>	$x y$	<code>tf.logical_or( )</code>
$x^*y$	<code>tf.multiply( )</code>	$x^y$	<code>tf.logical_xor( )</code>
$x/y(\text{python}2.0)$	<code>tf.divide( )</code>	$\sim x$	<code>tf.logical_not()</code>
$x/y(\text{python}3.0)$	<code>tf.truediv( )</code>	$x < y$	<code>tf.less( )</code>
$X//y(\text{python}3.0)$	<code>tf.floordiv( )</code>	$x \leq y$	<code>tf.less_equal( )</code>
$x \% y$	<code>tf.math.mod( )</code>	$x > y$	<code>tf.greater( )</code>
$x^{**}y$	<code>tf.pow( )</code>	$x \geq y$	<code>tf.greater_equal( )</code>
$-x$	<code>tf.neg( )</code>		
$\text{abs}(x)$	<code>tf.abs()</code>		

## 8.5 张量运算

```
In [14]: a = tf.constant([0, 1, 2, 3])
b = tf.constant([4, 5, 6, 7])
a+b
```

```
Out[14]: <tf.Tensor: id=38, shape=(4,), dtype=int32, numpy=array([ 4,  6,  8, 10])>
```

```
In [15]: a-b
```

```
Out[15]: <tf.Tensor: id=39, shape=(4,), dtype=int32, numpy=array([-4, -4, -4, -4])>
```

```
In [16]: a*b
```

```
Out[16]: <tf.Tensor: id=40, shape=(4,), dtype=int32, numpy=array([ 0,  5, 12, 21])>
```

```
In [17]: a/b
```

```
Out[17]: <tf.Tensor: id=43, shape=(4,), dtype=float64, numpy=array([0.          , 0.2
, 0.33333333, 0.42857143])>
```



## 8.5 张量运算

```
In [18]: a=tf.constant([0, 1, 2, 3])  
b = 2  
a%b
```

```
Out[18]: <tf.Tensor: id=46, shape=(4,), dtype=int32, numpy=array([0, 1, 0, 1])>
```

```
In [19]: a=tf.constant([[0, 1, 2, 3], [0, -1, -2, -3]])  
a//b
```

```
Out[19]: <tf.Tensor: id=49, shape=(2, 4), dtype=int32, numpy=  
array([[ 0,  0,  1,  1],  
       [ 0, -1, -1, -2]])>
```

```
In [20]: a=tf.constant([0, 1, 2, 3])  
b=2  
a**b
```

```
Out[20]: <tf.Tensor: id=52, shape=(4,), dtype=int32, numpy=array([0, 1, 4, 9])>
```



## ■ 广播机制 (broadcasting)

```
In [21]: a = tf.constant([1, 2, 3])  
a
```

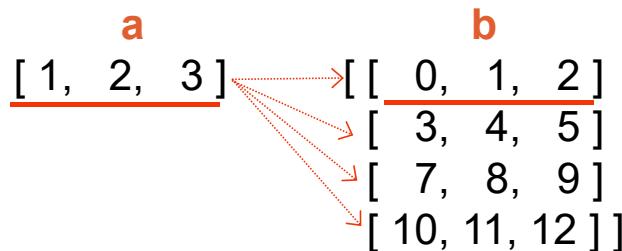
```
Out[21]: <tf.Tensor: id=53, shape=(3,), dtype=int32, numpy=array([1, 2, 3])>
```

```
In [22]: import numpy as np  
b=tf.constant(np.arange(12).reshape(4, 3))  
b
```

```
Out[22]: <tf.Tensor: id=54, shape=(4, 3), dtype=int32, numpy=  
array([[ 0,  1,  2],  
       [ 3,  4,  5],  
       [ 6,  7,  8],  
       [ 9, 10, 11]])>
```



## □ 一维张量+二维张量



两个张量最后一个维度的长度必须相等

```
In [23]: a+b
```

```
Out[23]: <tf.Tensor: id=55, shape=(4, 3), dtype=int32, numpy=
array([[ 1,  3,  5],
       [ 4,  6,  8],
       [ 7,  9, 11],
       [10, 12, 14]])>
```



兰州交通大学

计算机科学与技术学院

## 8.5 张量运算

### □ 一维张量 + 三维张量

```
In [24]: a = tf.constant([1, 2, 3])  
b = tf.constant(np.arange(12).reshape(2, 2, 3))  
b
```

最后一个维度的长度相等

```
Out[24]: <tf.Tensor: id=57, shape=(2, 2, 3), dtype=int32, numpy=  
array([[[ 0,  1,  2],  
       [ 3,  4,  5]],  
  
      [[ 6,  7,  8],  
       [ 9, 10, 11]]])>
```

```
In [25]: a+b
```

```
Out[25]: <tf.Tensor: id=58, shape=(2, 2, 3), dtype=int32, numpy=  
array([[[ 1,  3,  5],  
       [ 4,  6,  8]],  
  
      [[ 7,  9, 11],  
       [10, 12, 14]]])>
```



## 8.5 张量运算

```
In [26]: a*b
```

```
Out[26]: <tf.Tensor: id=59, shape=(2, 2, 3), dtype=int32, numpy=
array([[[ 0,  2,  6],
       [ 3,  8, 15]],
      [[ 6, 14, 24],
       [ 9, 20, 33]]])>
```

### □ 数字+N维张量

当张量和一个**数字**进行运算时，会将这个数字值广播到张量的各个元素



复旦科技大学

计算机科学与技术学院

### ■ 张量和NumPy数组之间的相互转换

**NumPy数组转化为张量**: `tf.constant(); tf.convert_to_tensor`

**张量转换为NumPy数组**: `Tensor.numpy()`



## 当张量和NumPy数组共同参与运算时：

- 执行**TensorFlow操作**,

TensorFlow将自动的  
把NumPy数组转换为  
张量

```
In [27]: nd = np.ones([2, 2])
t = tf.multiply(nd, 36)
t
```

```
Out[27]: <tf.Tensor: id=62, shape=(2, 2), dtype=float64, numpy=
array([[36., 36.],
       [36., 36.]])>
```

- 执行**NumPy操作**,

NumPy将自动的张量  
转换为NumPy数组

```
In [28]: np.add(nd, t)
```

```
Out[28]: array([[37., 37.],
       [37., 37.]])
```



西安科技大学

计算机科学与技术学院

## 使用运算符操作

只要操作数中有一个  
**Tensor对象**，就把所有  
的操作数都转化为张量，  
然后再进行运算。

In [29]: `a=tf.constant([1, 2, 3])` 张量  
`b=np.array((4, 5, 6))`  
`a+b` 运算符重载为张量加法

Out[29]: `<tf.Tensor: id=65, shape=(3,), dtype=int32, numpy=array([5, 7, 9])>`

In [30]: `a=np.array((4, 5, 6))` 2个NumPy数组相加  
`b=np.ones(3)`  
`a+b`

Out[30]: `array([5., 6., 7.])`

In [31]: `a=1` Python整数相加  
`b=2`  
`a+b`

Out[31]: 3



## ■ 张量乘法

□ 元素乘法: `tf.multiply()`, `*`运算符

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} \times \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} = \begin{bmatrix} 0 & 3 \\ 4 & 12 \end{bmatrix}$$

□ 向量乘法: `tf.matmul()`, `@`运算符

$$\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \end{bmatrix} = \begin{bmatrix} 10 & 13 \\ 28 & 40 \end{bmatrix}$$



## 8.5 张量运算

### 创建张量

$$\begin{bmatrix} 0 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \end{bmatrix} = \begin{bmatrix} 10 & 13 \\ 28 & 40 \end{bmatrix}$$

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

TensorFlow version: 2.0.0

```
In [2]: import numpy as np  
a = tf.constant(np.arange(6), shape=(2, 3))  
a
```

定义张量a

```
Out[2]: <tf.Tensor: id=2, shape=(2, 3), dtype=int32, numpy=  
array([[0, 1, 2],  
       [3, 4, 5]])>
```

```
In [3]: b = tf.constant(np.arange(6), shape=(3, 2))  
b
```

定义张量b

```
Out[3]: <tf.Tensor: id=5, shape=(3, 2), dtype=int32, numpy=  
array([[0, 1],  
       [2, 3],  
       [4, 5]])>
```



## ■ 向量乘法

tf.matmul(), @运算符

$$\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \end{bmatrix} = \begin{bmatrix} 10 & 13 \\ 28 & 40 \end{bmatrix}$$

In [4]: tf.matmul(a, b)Out[4]: <tf.Tensor: id=6, shape=(2, 2), dtype=int32, numpy=  
array([[10, 13],  
 [28, 40]])>In [5]: a@bOut[5]: <tf.Tensor: id=7, shape=(2, 2), dtype=int32, numpy=  
array([[10, 13],  
 [28, 40]])>

哈尔滨科技大学

计算机科学与技术学院

## ■ 多维向量乘法——三维张量×二维张量

- 最后两维做向量乘法
- 高维采用广播机制

```
In [6]: a = tf.random.normal([2, 3, 5])
b = tf.random.normal([5, 4])      (3,5) × (5,4) → (3,4) → (2,3,4)
tf.matmul(a, b)
```

```
Out[6]: <tf.Tensor: id=20, shape=(2, 3, 4), dtype=float32, numpy=
array([[[ 0.13672318, -0.3784006 , -0.2231093 , -1.2624099 ],
       [ 2.196111  ,  1.2696512 , -4.518535  ,  3.6673522 ],
       [ 0.62343776, -2.2611952 ,  1.4009535 ,  0.22248505]],

      [[-0.7526479 , -0.5529423 ,  2.294831  , -2.4406729 ],
       [-0.12710276,  1.5016136 ,  1.420251  ,  0.81047237],
       [ 1.014232  ,  1.3020521 , -4.1052294 ,  1.0766649 ]]],  
      dtype=float32)>
```



## ■ 多维向量乘法——三维张量×三维张量

创建张量

```
In [7]: a = tf.constant(np.arange(12), shape=(2, 2, 3))  
a
```

```
Out[7]: <tf.Tensor: id=23, shape=(2, 2, 3), dtype=int32, numpy=  
array([[[ 0,  1,  2],  
       [ 3,  4,  5]],  
  
       [[ 6,  7,  8],  
       [ 9, 10, 11]]])>
```

```
In [8]: b = tf.constant(np.arange(12), shape=(2, 3, 2))  
b
```

```
Out[8]: <tf.Tensor: id=26, shape=(2, 3, 2), dtype=int32, numpy=  
array([[[ 0,  1],  
       [ 2,  3],  
       [ 4,  5]],  
  
       [[ 6,  7],  
       [ 8,  9],  
       [10, 11]]])>
```



## 8.5 张量运算

### ■ 多维向量乘法——三维张量×三维张量

#### 张量乘法过程

- 最后两维做向量乘法
- 高维采用广播机制

$(2,3) \times (3,2) \rightarrow (2,2)$

广播  $\rightarrow (2,2,2)$

```
In [7]: a = tf.constant(np.arange(12), shape=(2, 2, 3))  
a
```

```
Out[7]: <tf.Tensor: id=23, shape=(2, 2, 3), dtype=int32, numpy=  
array([[ [ 0,  1,  2],  
       [ 3,  4,  5]],
```

$\begin{bmatrix} 6 & 7 & 8 \\ 9 & 10 & 11 \end{bmatrix}$

$$\begin{bmatrix} 10 & 13 \\ 28 & 40 \end{bmatrix}$$

```
In [8]: b = tf.constant(np.arange(12), shape=(2, 3, 2))  
b
```

```
Out[8]: <tf.Tensor: id=26, shape=(2, 3, 2), dtype=int32, numpy=  
array([[ [ 0,  1],  
       [ 2,  3],  
       [ 4,  5]],
```

$\begin{bmatrix} 6 & 7 \\ 8 & 9 \\ 10 & 11 \end{bmatrix}$

$$\begin{bmatrix} 172 & 193 \\ 244 & 274 \end{bmatrix}$$


## ■ 多维向量乘法——三维张量×三维张量

运行结果

In [9]: a @ b

```
Out[9]: <tf.Tensor: id=27, shape=(2, 2, 2), dtype=int32, numpy=
array([[[ 10,  13],
       [ 28,  40]],

      [[172, 193],
       [244, 274]]])>
```



西安科技大学

计算机科学与技术学院

## 8.5 张量运算

### ■ 多维向量乘法——四维张量×四维张量

```
In [10]: a = tf.constant(np.arange(24), shape=(2, 2, 2, 3))  
a
```

```
Out[10]: <tf.Tensor: id=30, shape=(2, 2, 2, 3), dtype=int32, numpy=  
array([[[[ 0,  1,  2],  
        [ 3,  4,  5]],  
  
       [[ 6,  7,  8],  
        [ 9, 10, 11]]],  
  
      [[[12, 13, 14],  
        [15, 16, 17]],  
  
       [[18, 19, 20],  
        [21, 22, 23]]])>
```

创建张量

```
In [11]: b = tf.constant(np.arange(24), shape=(2, 2, 3, 2))  
b
```

```
Out[11]: <tf.Tensor: id=33, shape=(2, 2, 3, 2), dtype=int32, numpy=  
array([[[[ 0,  1],  
        [ 2,  3],  
        [ 4,  5]],  
  
       [[ 6,  7],  
        [ 8,  9],  
        [10, 11]]],  
  
      [[[12, 13],  
        [14, 15],  
        [16, 17]],  
  
       [[18, 19],  
        [20, 21],  
        [22, 23]]]])>
```



哈尔滨科技大学

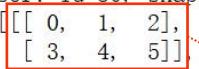
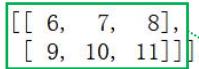
计算机科学与技术学院

## 8.5 张量运算

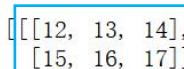
### ■ 多维向量乘法——四维张量×四维张量

```
In [10]: a = tf.constant(np.arange(24), shape=(2, 2, 2, 3))  
a
```

```
Out[10]: <tf.Tensor: id=30, shape=(2, 2, 2, 3), dtype=int32, numpy=  
array([[[[ 0,  1,  2],  
       [ 3,  4,  5]],
```

$$\begin{bmatrix} [6, 7, 8], \\ [9, 10, 11] \end{bmatrix}$$



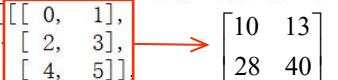
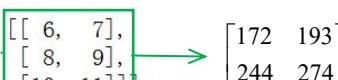
$$\begin{bmatrix} [12, 13, 14], \\ [15, 16, 17] \end{bmatrix}$$



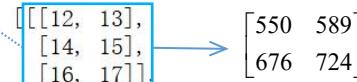
$$\begin{bmatrix} [18, 19, 20], \\ [21, 22, 23] \end{bmatrix})]$$

```
In [11]: b = tf.constant(np.arange(24), shape=(2, 2, 3, 2))  
b
```

```
Out[11]: <tf.Tensor: id=33, shape=(2, 2, 3, 2), dtype=int32, numpy=  
array([[[[ 0,  1],  
       [ 2,  3],  
       [ 4,  5]],
```

$$\begin{bmatrix} [6, 7], \\ [8, 9], \\ [10, 11] \end{bmatrix}, \begin{bmatrix} 10 & 13 \\ 28 & 40 \end{bmatrix}$$



$$\begin{bmatrix} [12, 13], \\ [14, 15], \\ [16, 17] \end{bmatrix}, \begin{bmatrix} 550 & 589 \\ 676 & 724 \end{bmatrix}$$



$$\begin{bmatrix} [18, 19], \\ [20, 21], \\ [22, 23] \end{bmatrix}), \begin{bmatrix} 1144 & 1201 \\ 1324 & 1390 \end{bmatrix}]$$

张量乘法过程

$(2,3) \times (3,2) \rightarrow (2,2)$   
广播  $\rightarrow (2,2,2,2)$



兰州交通大学

计算机科学与技术学院

## ■ 多维向量乘法——四维张量×四维张量

运行结果

In [12]: `a@b`

```
Out[12]: <tf.Tensor: id=34, shape=(2, 2, 2, 2), dtype=int32, numpy=
array([[[[ 10,   13],
          [ 28,   40]],

         [[ 172,   193],
          [ 244,   274]]],

        [[[ 550,   589],
          [ 676,   724]],

         [[1144,  1201],
          [1324,  1390]]]])>
```



西安科技大学

计算机科学与技术学院

口 **数据统计**: 求张量在**某个维度**上、或者**全局**的统计值

函 数	描 述
<code>tf.reduce_sum(input_tensor, axis)</code>	求和
<code>tf.reduce_mean(input_tensor, axis)</code>	求平均值
<code>tf.reduce_max(input_tensor, axis)</code>	求最大值
<code>tf.reduce_min(input_tensor, axis)</code>	求最小值

$$\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \end{bmatrix}$$

全局最大值: 5  
每行的最大值: [2,5]  
每列的最大值: [3,4,5]



## 8.5 张量运算

### ■ 求和函数——tf.reduce\_sum()

```
In [13]: a = tf.constant([[1, 5, 3], [4, 2, 6]])  
a
```

```
Out[13]: <tf.Tensor: id=35, shape=(2, 3), dtype=int32, numpy=  
array([[1, 5, 3],  
       0  
       [4, 2, 6]])>
```

```
In [14]: tf.reduce_sum(a, axis=0) axis=0
```

```
Out[14]: <tf.Tensor: id=37, shape=(3,), dtype=int32, numpy=array([5, 7, 9])>
```

```
In [15]: tf.reduce_sum(a, axis=1) axis=1
```

```
Out[15]: <tf.Tensor: id=39, shape=(2,), dtype=int32, numpy=array([ 9, 12])>
```

```
In [16]: tf.reduce_sum(a)
```

```
Out[16]: <tf.Tensor: id=41, shape=(), dtype=int32, numpy=21>
```

## 8.5 张量运算

### ■ 求均值函数——tf.reduce\_mean()

$$\begin{array}{c} 0 \\ \downarrow \\ \begin{bmatrix} 1 & 5 & 3 \\ 4 & 2 & 6 \end{bmatrix} \\ 2.5 \quad 3.5 \quad 4.5 \end{array}$$

张量元素的数据类型是int32,  
因此求得的均值也是int32

In [17]: `tf.reduce_mean(a, axis=0)`

Out[17]: <tf.Tensor: id=43, shape=(3,), dtype=int32, numpy=array([2, 3, 4])>

In [18]: `a = tf.constant([[1., 5., 3.], [4., 2., 6.]])`  
`tf.reduce_mean(a, axis=0)`

张量元素采用浮点数

得到浮点数的均值

Out[18]: <tf.Tensor: id=46, shape=(3,), dtype=float32, numpy=array([2.5, 3.5, 4.5],  
dtype=float32)>

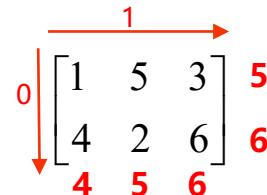
In [19]: `a = tf.constant([[1, 5, 3], [4, 2, 6]])`  
`tf.reduce_mean(tf.cast(a, tf.float32), axis=0)`

将张量的数据类型转换  
为浮点数，再求均值

Out[19]: <tf.Tensor: id=50, shape=(3,), dtype=float32, numpy=array([2.5, 3.5, 4.5],  
dtype=float32)>



## ■ 求最大值、最小值函数——tf.max(), tf.min()



```
In [20]: tf.reduce_max(a, axis=0)
```

```
Out[20]: <tf.Tensor: id=52, shape=(3,), dtype=int32, numpy=array([4, 5, 6])>
```

```
In [21]: tf.reduce_max(a, axis=1)
```

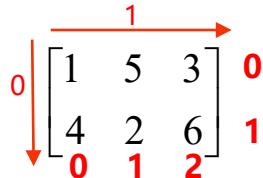
```
Out[21]: <tf.Tensor: id=54, shape=(2,), dtype=int32, numpy=array([5, 6])>
```

```
In [22]: tf.reduce_max(a)
```

```
Out[22]: <tf.Tensor: id=56, shape=(), dtype=int32, numpy=6>
```



## ■ 求最值的索引——tf.argmax(), tf.argmin()



```
In [23]: a = tf.constant([[1, 5, 3], [4, 2, 6]])
tf.argmax(a, axis=0)
```

```
Out[23]: <tf.Tensor: id=59, shape=(3,), dtype=int64, numpy=array([1, 0, 1], dtype=int64)>
```

```
In [24]: tf.argmax(a, axis=1)
```

```
Out[24]: <tf.Tensor: id=61, shape=(2,), dtype=int64, numpy=array([1, 2], dtype=int64)>
```

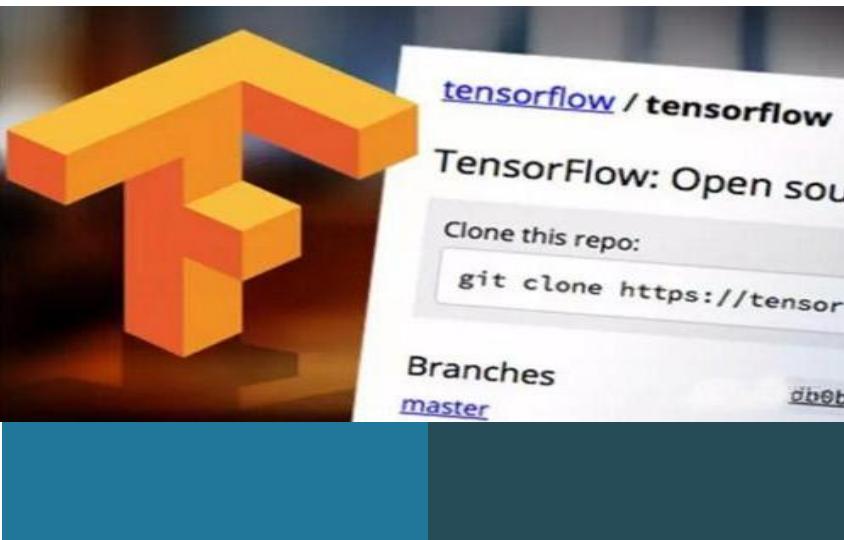
  

```
In [25]: tf.argmax(a)
```

```
Out[25]: <tf.Tensor: id=63, shape=(3,), dtype=int64, numpy=array([1, 0, 1], dtype=int64)>
```

没有指定axis参数时，默认axis=0





## 8.6 使用GPU

## ■ 导入TensorFlow，查看版本

```
In [1]: import tensorflow as tf  
print(tf.__version__)
```

```
2.0.0
```

## ■ 查看当前主机上的运算设备

```
In [2]: gpus = tf.config.experimental.list_physical_devices(device_type='GPU')  
cpus = tf.config.experimental.list_physical_devices(device_type='CPU')  
print(gpus)  
print(cpus)
```

```
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]  
[PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU')]
```



## 8.6 使用GPU

### ■ 指定在CPU上执行

```
In [3]: # 指定在CPU上执行
with tf.device('/cpu:0'):
    cpu_a = tf.random.normal([10000, 1000])
    cpu_b = tf.random.normal([1000, 2000])
    cpu_c = tf.matmul(cpu_a, cpu_b)
```

```
In [4]: print("cpu_a:", cpu_a.device)
print("cpu_b:", cpu_b.device)
print("cpu_c:", cpu_c.device)
```

```
cpu_a: /job:localhost/replica:0/task:0/device:CPU:0
cpu_b: /job:localhost/replica:0/task:0/device:CPU:0
cpu_c: /job:localhost/replica:0/task:0/device:CPU:0
```



## 8.6 使用GPU

### ■ 指定在GPU上执行

```
In [5]: #查看GPU是否可用  
tf.test.is_gpu_available()
```

```
Out[5]: True
```

```
In [6]: # 指定在GPU上执行随机数操作  
with tf.device('/gpu:0'):  
    gpu_a = tf.random.normal([10000, 1000])  
    gpu_b = tf.random.normal([1000, 2000])  
    gpu_c = tf.matmul(gpu_a, gpu_b)
```

```
In [7]: print("gpu_a:", gpu_a.device)  
print("gpu_b:", gpu_b.device)  
print("gpu_c:", gpu_c.device)
```

```
gpu_a: /job:localhost/replica:0/task:0/device:GPU:0  
gpu_b: /job:localhost/replica:0/task:0/device:GPU:0  
gpu_c: /job:localhost/replica:0/task:0/device:GPU:0
```



## 8.6 使用GPU

### ■ 创建函数cpu\_run()和gpu()\_run

```
In [8]: # 函数cpu_run()
def cpu_run():
    with tf.device('/cpu:0'):
        cpu_a = tf.random.normal([10000, 1000])
        cpu_b = tf.random.normal([1000, 2000])
        c = tf.matmul(cpu_a, cpu_b)
    return c
```

```
In [9]: # 函数gpu_run()
def gpu_run():
    with tf.device('/gpu:0'):
        gpu_a = tf.random.normal([10000, 1000])
        gpu_b = tf.random.normal([1000, 2000])
        c = tf.matmul(gpu_a, gpu_b)
    return c
```



## 8.6 使用GPU

### ■ 比较在CPU和GPU上执行乘法操作的时间

```
In [10]: import timeit # 导入timeit模块
```

```
In [11]: # 使用timeit工具来统计执行10次的时间
cpu_time = timeit.timeit(cpu_run, number=10)
gpu_time = timeit.timeit(gpu_run, number=10)
print("cpu:", cpu_time, "gpu:", gpu_time)
```

```
cpu: 1.6976891000000052 gpu: 0.002923800000004917
```

```
In [12]: # 使用timeit工具来统计执行100次的时间
cpu_time = timeit.timeit(cpu_run, number=100)
gpu_time = timeit.timeit(gpu_run, number=100)
print("cpu:", cpu_time, "gpu:", gpu_time)
```

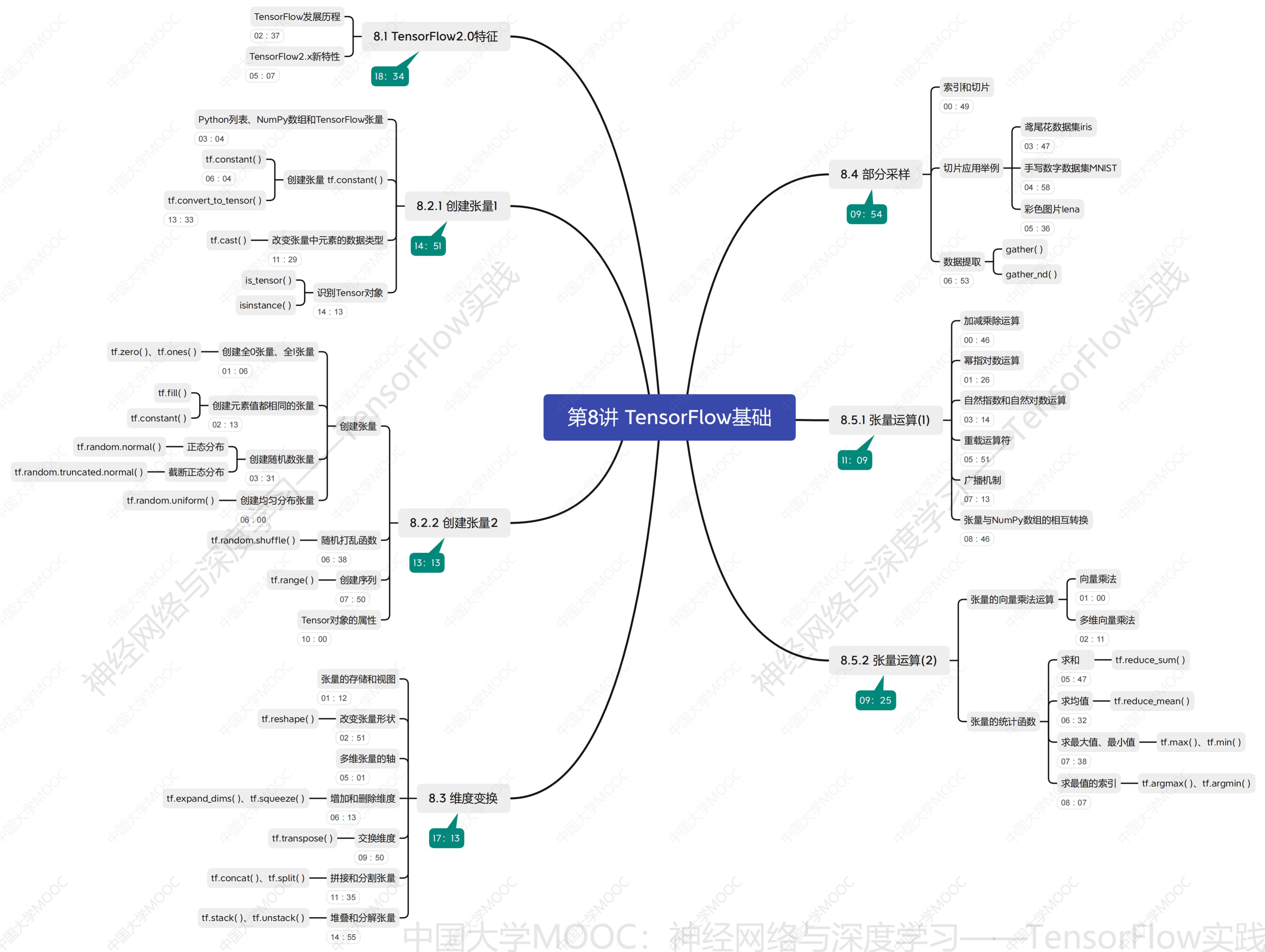
```
cpu: 16.43405639999999 gpu: 0.02554969999999912
```



- 在TensorFlow中，张量可以运行在CPU、GPU或TPU上
- 一般无需指定设备，TensorFlow会自动调用所有可用资源进行计算，决定执行操作的设备，并在需要时将操作复制到该设备



## 第8讲 TensorFlow基础



## 第 8 讲 Tensorflow 基础拓展题目

### 题目一：

在单元作业 2 的基础上，已知  $x$ 、 $y$ 、 $w$ 、 $b$ ，计算  $L$  的值并输出结果。

$$(1) L(w, b) = \frac{\sum_{i=1}^n (wx_i + b - y_i)}{n}$$

其中， $xi$  是  $x$  中索引值为  $i$  的元素； $yi$  是  $y$  中索引值为  $i$  的元素； $n$  是张量中元素的个数

(2) 输出  $L$  的值

- ① 代码
- ② 实验结果