



1 TensorFlow



教学内容

- 1. TensorFlow简介
- 2. 数据类型
- 3. 常用函数
- 4. 线性回归示例
- 5. 深度神经网络示例

1 TensorFlow简介

TensorFlow 是一款用于数值计算的强大的开源软件库，特别适用于大规模机器学习的微调。

■ TensorFlow 的亮点

- 多平台
- 提供丰富API
- TF.Learn2, TF-slim, Keras 或 Pretty Tensor
- 高效 C ++ 实现
- TensorBoard 可视化工具

1.1 Tensorflow版本

■ TensorFlow 1.x

静态图执行模式(Graph Execution)

符号式编程, 先创建计算图后运行。

■ TensorFlow 2.x

动态图优先模式(Eager Execution), 命令式编程, 是2.x版的默认模式。

二者的使用体验完全不同, 代码互不兼容, 同时在编程风格、函数接口设计等上也大相径庭。

`tf.executing_eagerly()`



Graph与Session (v1)

- **Graph**: 一张有边与点的图, 表示需要进行计算的任务(v1);
- **Operation**: 执行计算的单元, 图的节点;
- **Session**: 称之为会话的上下文, 用于执行图(v1)。

除了Variables指向的 tensor 外所有的 tensor 在流入下一个节点后都不再保存。

TensorFlow 1.x 工作流程

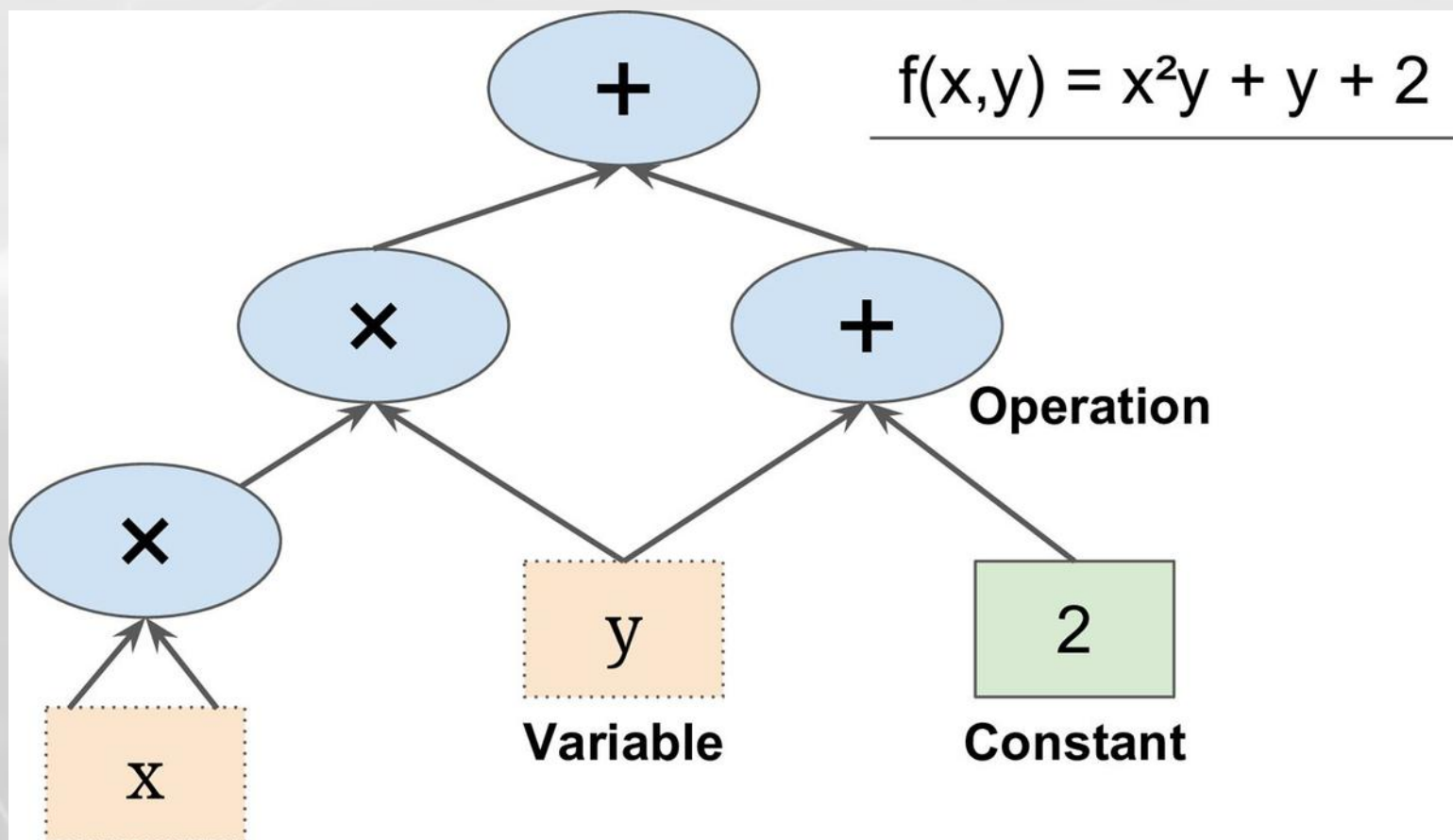
■ 创建计算图

```
import tensorflow as tf
tf.compat.v1.disable_v2_behavior()
a = tf.constant([1.0, 2.0])
b = tf.constant([3.0, 4.0])
c = a * b
```

■ 运行计算图

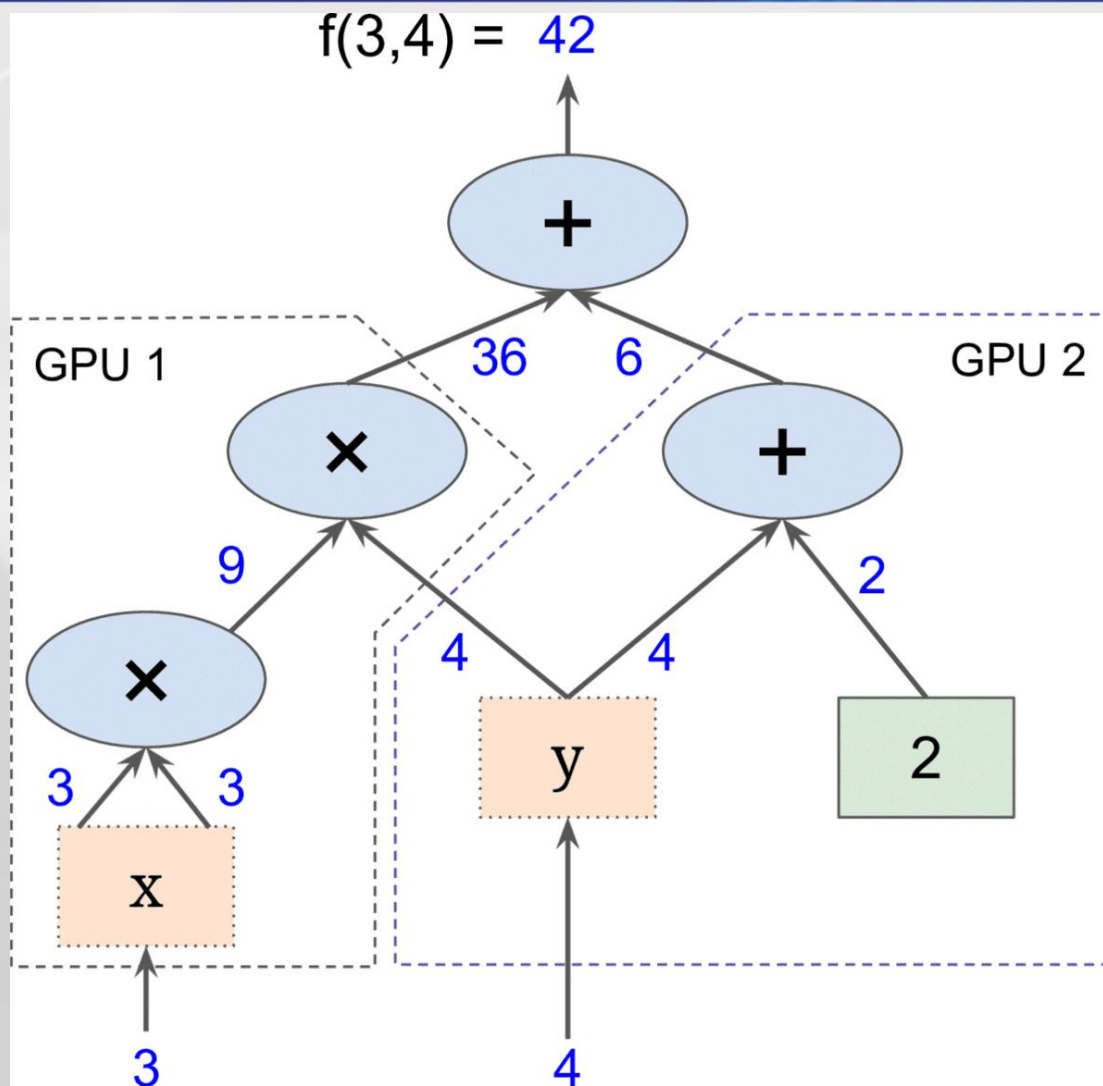
```
sess = tf.compat.v1.Session()
print(sess.run(c))    # 在该会话中执行计算
sess.close()          # 关闭会话
```

一个简单的计算图





- **Tensorflow** 可以将图分解为多个部分，并在多个 CPU 或 GPU 上并行运行。
TensorFlow 还支持分布式计算。





为什么要配置 Session ?

- 为了充分利用处理机的资源，提高会话运行的效率，可以配置并行线程数、GPU分配策略、运算超时时间等内容。
- 一般来说，动态图模型开发效率高，但是运行效率可能不如静态图模式，TensorFlow 2.x 也支持通过 `tf.function` 将动态图优先模式的代码转化为静态图模式。



TensorFlow 2.x

```
import tensorflow as tf  
a = tf.constant(2.)  
b = tf.constant(4.)  
print('a+b=',a+b)
```



运行模式转换

- **`tf.compat.v1.disable_v2_behavior()`**
- **`tf.compat.v1.disable_eager_execution()`**
- **`tf.compat.v1.enable_v2_behavior()`**
- **`tf.compat.v1.enable_eager_execution()`**



- 在 Eager Execution 期间，计算会自动分流到 GPU。如果想控制计算运行的位置，可通过`tf.device()`将其放在 GPU或 CPU 中执行。
- `tf.device("/cpu:0")`
- `tf.device("/gpu:0")`



```
def mul(x, steps):  
    for i in range(steps):  
        x = tf.matmul(x, x)  
# 指定在CPU中运行:  
with tf.device("/cpu:0"):  
    mul(tf.random.normal((1000,1000)), 100)  
# 指定在GPU中运行:  
if tf.config.list_physical_devices("GPU"):  
    with tf.device("/gpu:0"):  
        mul(tf.random.normal((1000,1000)), 100)  
else:  
    print("GPU: not found")
```



1.2 安装

```
$ pip3 install tensorflow
```

```
$ pip3 install tensorflow-cpu #cpu版
```

```
$ python
```

```
>>>import tensorflow as tf
```

```
>>>tf.__version__
```

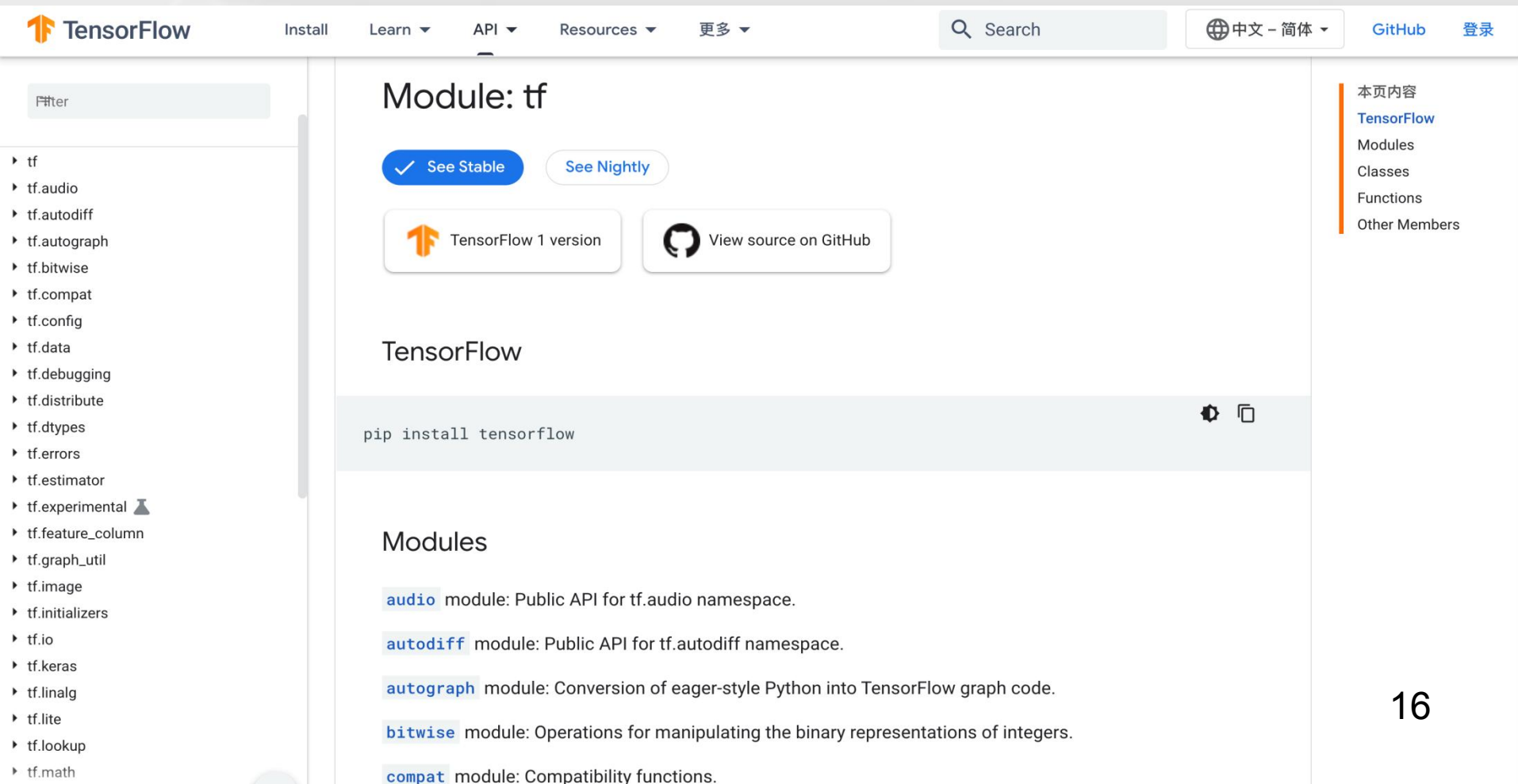
```
>>>import tensorflow as tf
```

```
>>>tf.config.list_physical_devices()
```

```
>>>tf.config.list_physical_devices('GPU')
```

1.3 TensorFlow文档

- https://www.tensorflow.org/api_docs/python/tf
- https://tensorflow.google.cn/api_docs/python/tf



The screenshot shows the TensorFlow API documentation page for the 'tf' module. The page is in Chinese (Simplified) and features a sidebar with a list of modules, a main content area with installation instructions and a list of modules, and a right sidebar with navigation links.

TensorFlow Install Learn ▾ API ▾ Resources ▾ 更多 ▾ Search 中文 - 简体 GitHub 登录

Module: tf

✓ See Stable See Nightly

TensorFlow 1 version View source on GitHub

TensorFlow

```
pip install tensorflow
```

Modules

- audio** module: Public API for tf.audio namespace.
- autodiff** module: Public API for tf.autodiff namespace.
- autograph** module: Conversion of eager-style Python into TensorFlow graph code.
- bitwise** module: Operations for manipulating the binary representations of integers.
- compat** module: Compatibility functions.

本页内容
TensorFlow
Modules
Classes
Functions
Other Members



- tf
- tf.audio
- tf.autodiff
- tf.autograph
- tf.bitwise
- tf.compat
- tf.config
- tf.data
- tf.debugging
- tf.distribute
- tf.dtypes
- tf.errors

- tf.estimator
- tf.experimental 
- tf.feature_column
- tf.graph_util
- tf.image
- tf.initializers
- tf.io
- tf.keras
- tf.linalg
- tf.lite
- tf.lookup
- tf.math

- tf.metrics
- tf.mixed_precision
- tf.mlir
- tf.nest
- tf.nn
- tf.optimizers
- tf.profiler
- tf.quantization
- tf.queue
- tf.ragged
- tf.random
- tf.raw_ops

- tf.saved_model
- tf.sets
- tf.signal
- tf.sparse
- tf.strings
- tf.summary
- tf.sysconfig
- tf.test
- tf.tpu
- tf.train
- tf.types
- tf.version



```
tf.math.ceil(  
    x, name=None  
)
```

For example:

```
>>> tf.math.ceil([-1.7, -1.5, -0.2, 0.2, 1.5, 1.7, 2.0])  
<tf.Tensor: shape=(7,), dtype=float32,  
numpy=array([-1., -1., -0.,  1.,  2.,  2.,  2.], dtype=float32)>
```



Args

x	A tf.Tensor . Must be one of the following types: bfloat16 , half , float32 , float64 . int32
name	A name for the operation (optional).

Returns

A **tf.Tensor**. Has the same type as **x**.



1.4 MNIST示例

MNIST为手写数字数据集

- `import tensorflow as tf`
- `#载入数据`
- `mnist = tf.keras.datasets.mnist`
- `(x_train, y_train), (x_test, y_test) = mnist.load_data()`
- `x_train, x_test = x_train / 255.0, x_test / 255.0`



■ #建模

```
■ model = tf.keras.models.Sequential([  
    tf.keras.layers.Flatten(input_shape=(28, 28)),  
    tf.keras.layers.Dense(128, activation='relu'),  
    tf.keras.layers.Dropout(0.2),  
    tf.keras.layers.Dense(10,  
activation='softmax')  
])
```



- #编译
- `model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy',
metrics=['accuracy'])`



■ # 训练

■ model.fit(x_train, y_train, epochs=5)

```
Epoch 1/5
 1/1875 [.....] - ETA: 46:15 - loss: 2.4089 - accuracy:
22/1875 [.....] - ETA: 4s - loss: 1.8428 - accuracy:
69/1875 [>.....] - ETA: 2s - loss: 1.1652 - accuracy:
109/1875 [>.....] - ETA: 3s - loss: 0.9519 - accuracy:
148/1875 [= >.....] - ETA: 2s - loss: 0.8290 - accuracy:
210/1875 [== >.....] - ETA: 2s - loss: 0.7002 - accuracy:
270/1875 [=== >.....] - ETA: 2s - loss: 0.6304 - accuracy:
331/1875 [==== >.....] - ETA: 1s - loss: 0.5769 - accuracy:
392/1875 [===== >.....] - ETA: 1s - loss: 0.5414 - accuracy:
456/1875 [===== >.....] - ETA: 1s - loss: 0.5080 - accuracy:
518/1875 [===== >.....] - ETA: 1s - loss: 0.4844 - accuracy:
580/1875 [===== >.....] - ETA: 1s - loss: 0.4604 - accuracy:
642/1875 [===== >.....] - ETA: 1s - loss: 0.4443 - accuracy:
```



- # 验证
- `model.evaluate(x_test, y_test, verbose=2)`

```
>>> model.evaluate(x_test, y_test, verbose=2)
313/313 - 0s - loss: 0.0753 - accuracy: 0.9770
[0.075251504778862, 0.9769999980926514]
```




1.5 优化器

tf.keras.optimizers

- **SGD**
- **Adam**
- **Adamax**
- **Adadelta**
- **Adagrad**
- **Nadam**
- **RMSprop**

SGD 随机梯度下降优化器

```
tf.keras.optimizers.SGD(  
    learning_rate=0.01, momentum=0.0,  
    nesterov=False, name='SGD', **kwargs  
)
```

- **learning_rate**: 学习率。
- **momentum**: 动量，用于加速 SGD 在相关方向上前进，并抑制震荡。
- **nesterov**: 是否使用 Nesterov 动量。



■ 1) 动量为0

$$w = w - \text{learning_rate} * g$$

■ 2) 动量不为0

$$\begin{aligned} \text{velocity} &= \text{momentum} * \text{velocity} - \text{learning_rate} * g \\ w &= w + \text{velocity} \end{aligned}$$

■ 3) nesterov=True

$$\begin{aligned} \text{velocity} &= \text{momentum} * \text{velocity} - \text{learning_rate} * g \\ w &= w + \text{momentum} * \text{velocity} - \text{learning_rate} * g \end{aligned}$$



- `opt = tf.keras.optimizers.SGD(learning_rate=0.1)`
- `var = tf.Variable(1.0)`
- `loss = lambda: (var ** 2)/2.0`
- `opt.minimize(loss, [var])`
- `var.numpy()`



Adam优化器

- 结合AdaGrad和RMSProp两种优化算法的优点。对梯度的一阶矩估计和二阶矩估计进行综合考虑，计算出更新步长。
- `tf.keras.optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-07, amsgrad=False, name='Adam', *kwargs)`



- `opt = tf.keras.optimizers.Adam(learning_rate=0.1)`
- `var1 = tf.Variable(10.0)`
- `loss = lambda: (var1 ** 2)/2.0`
- `opt.minimize(loss, [var1])`
- `var1.numpy()`

1.6 损失函数

tf.keras.losses

目标函数、优化评分函数

回归问题

均方根误差 MSE

平均绝对误差 MAE

Huber loss

分类问题

simmoid_cross_entropy_with_logits

交叉熵log_loss

softmax_cross_entropy_with_logits

sparse_softmax_cross_entropy_with_logits

weights_cross_entropy_with_logits

铰链损失hinge_loss



均方误差MSE

■ 均方误差MSE

```
tf.keras.losses.MeanSquaredError(  
    reduction=losses_utils.ReductionV2.AUTO,  
    name='mean_squared_error'  
)
```




- `y_true = [[0., 1.], [0., 0.]`
- `y_pred = [[1., 1.], [1., 0.]`
- `mse =`
`tf.keras.losses.MeanSquaredError()`
- `mse(y_true, y_pred).numpy()`
- `0.5`



BinaryCrossentropy

- 二值交叉熵
- **`tf.keras.losses.BinaryCrossentropy(`
 `from_logits=False,`
 `label_smoothing=0.0, axis=-1,`
 `reduction=losses_utils.ReductionV2.AUTO,`
 `name='binary_crossentropy'`
 `)`**



- **y_true = [0, 1, 0, 0]**
- **y_pred = [-18.6, 0.51, 2.94, -12.8]**
- **bce = tf.keras.losses.BinaryCrossentropy(
from_logits=True)**
- **bce(y_true, y_pred).numpy()**
- **0.865**



1.7 评价函数

tf.keras.metrics

metrics用于评估当前训练模型的性能

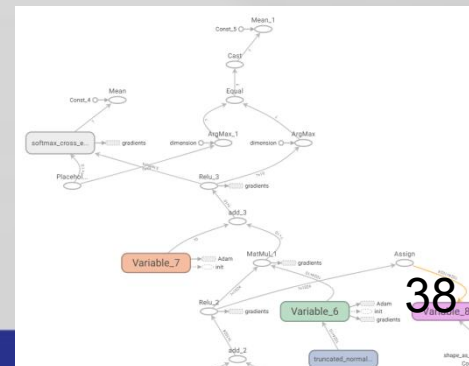
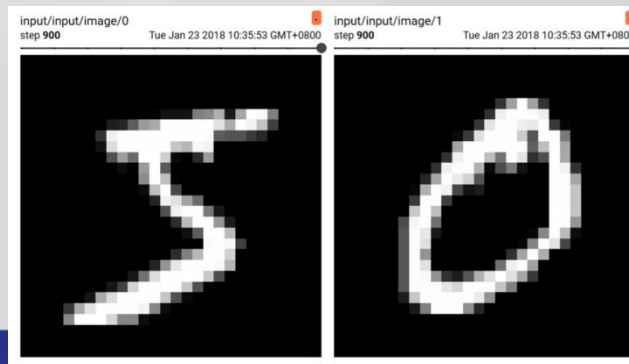
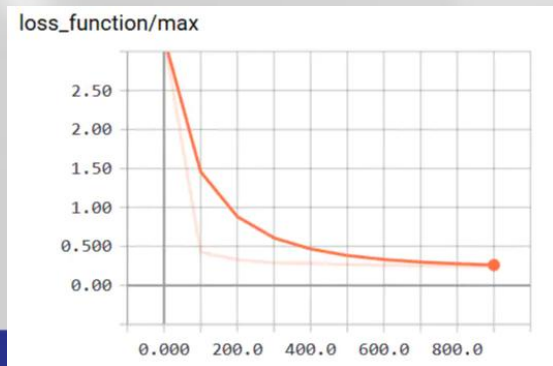
- **"accuracy"**:真实值(y_{true})和预测值(y_{pred}) 都是数值;
- **"sparse_accuracy"**: y_{true} 和 y_{pred} 都是以独热码和概率分布表示;
- **"sparse_categorical_accuracy"**: y_{true} 是以数值形式给出, y_{pred} 是以独热码给出



1.8 TensorBoard

- Tensorboard是tensorflow内置的一个可视化工具，它通过将tensorflow程序输出的**日志文件**的信息可视化，使得tensorflow程序的理解、调试和优化更加简单高效。
- Tensorboard的可视化依赖于tensorflow程序运行输出的日志文件，因而tensorboard和tensorflow程序在不同的进程中运行。
- <https://tensorflow.google.cn/tensorboard>

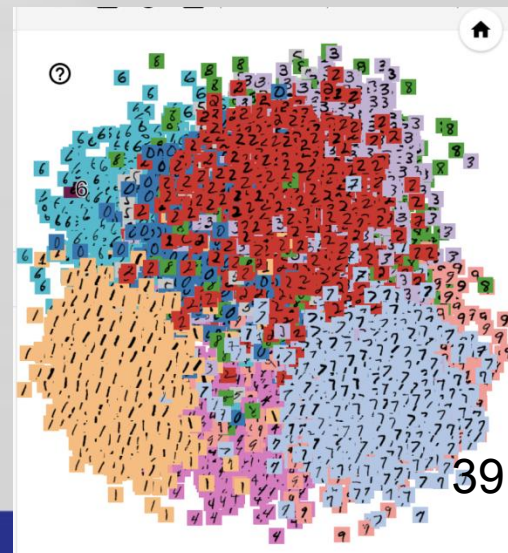
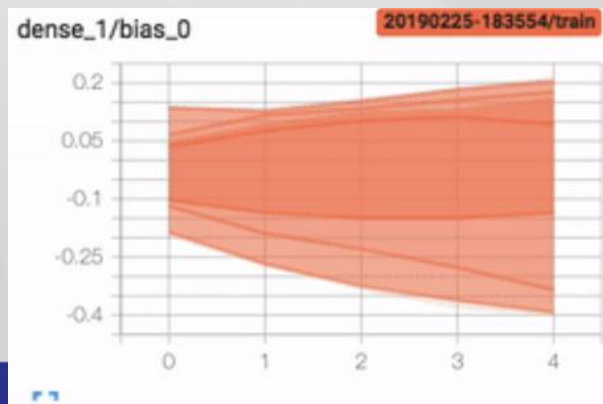
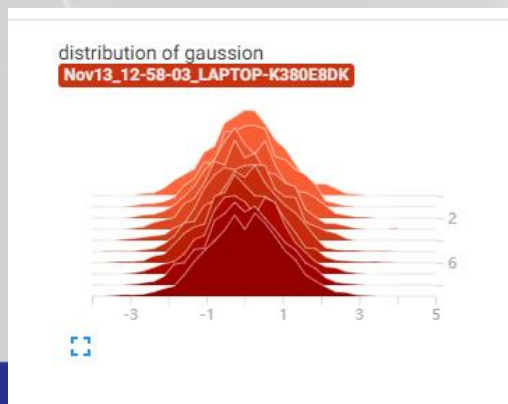
- (1) Scalars:展示训练过程中的准确率、损失值、权重/偏置的变化情况。
- (2) Images:展示训练过程中记录的图像。
- (3) Audio:展示训练过程中记录的音频。
- (4) Graphs:展示模型的数据流图，以及训练在各个设备上消耗的内存和时间。





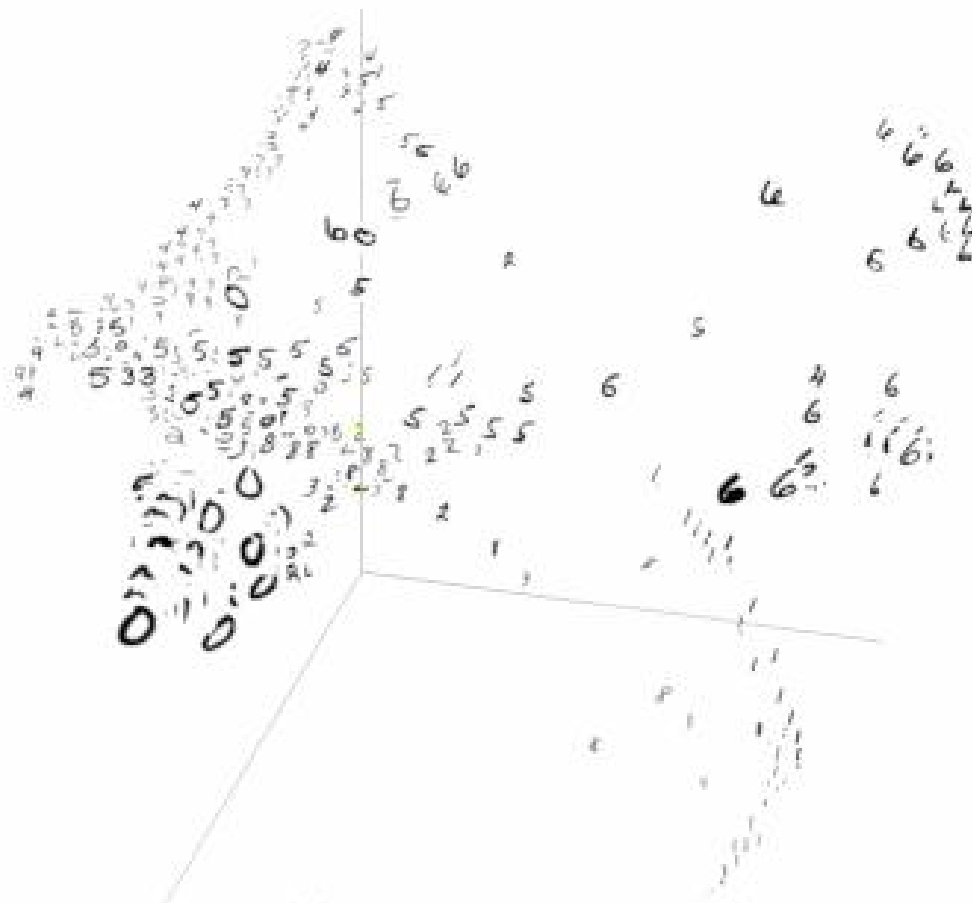
- (5) Distributions:展示训练过程中的数据分布。
- (6) Histograms:展示训练过程中数据的柱状图。
- (7) Embeddings:展示词向量的投影分布

○





②





TensorBoard的使用步骤

- 1) 设置保存路径;
- 2) 指定文件用来保存TensorFlow数据;
- 3) 添加需要展示的数据;
- 4) 展示图。

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, Dropout

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(10, activation='softmax') ])
```



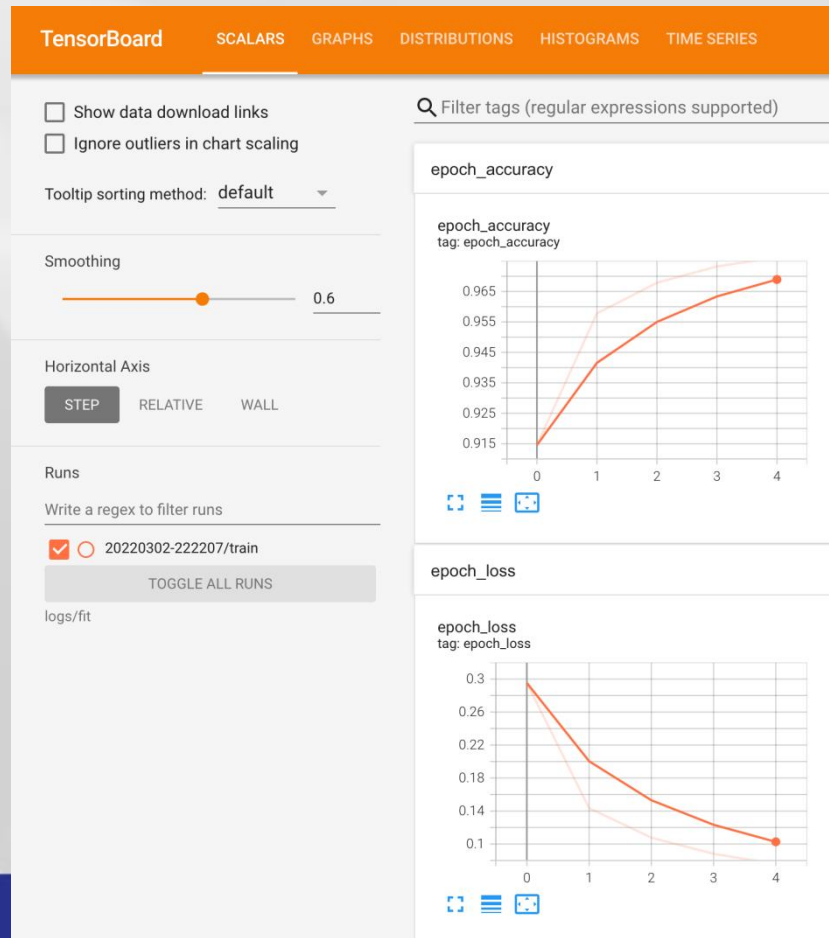
```
model.compile(optimizer='adam',  
              loss='sparse_categorical_crossentropy',  
              metrics=['accuracy' ])
```

```
log_dir = "logs/fit/mnist"  
callback = tf.keras.callbacks.TensorBoard(  
    log_dir=log_dir, histogram_freq=1)
```

```
model.fit(x_train, y_train,  
          epochs=500, callbacks=[callback])  
model.evaluate(x_test, y_test, verbose=2)
```



- `%python mnistv2_tb.py`
- `%tensorboard --logdir logs/fit`





2 数据类型

2.1 Tensor（张量）

Tensor是具有统一类型（称为 dtype）的多维数组

- 1、`tf.constant()`
- 2、`tf.Variable()`
- 3、`tf.zeros()`
- 4、`tf.ones()`
- 5、`tf.fill([2, 3], 9)`
- 6、`tf.convert_to_tensor(a, dtype=tf.int32)`
- 7、随机初始化进行创建（`tf.random`）



■ **tf.zeros(
 shape,
 dtype=tf.dtypes.float32)**

tf.zeros([3, 4], tf.int32)

[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]



```
■ tf.ones(  
    shape,  
    dtype=tf.dtypes.float32)  
tf.ones([2, 3], tf.int32)  
# [[1, 1, 1], [1, 1, 1]]
```



- **tf.fill(
 dims,
 value)**
- **# Output tensor has shape [2, 3].
fill([2, 3], 9) ==> [[9, 9, 9]
 [9, 9, 9]]**



■ **tf.eye(
 num_rows,
 num_columns=None,
 batch_shape=None,
 dtype=tf.dtypes.float32)**



2.2 数据结构

- Tensorflow的数据结构有rank, shape, data types等概念。



1) Rank

■ 一般是指数据的维度

阶	数学实例	python例子
0	Scalar纯量 (只有大小)	<code>s=483</code>
1	Vector向量 (大小和方向)	<code>v=[1.1, 2.2, 3.3]</code>
2	Matrix矩阵 (数据表)	<code>m=[[1, 2, 3], [4, 5, 6], [7, 8, 9]]</code>
3	3-Tensor 3阶张量	<code>t=[[[2], [4], [6]], [[8], [9], [10]], [[11], [12], [13]]]</code>
n	n-Tensor n阶张量	<code>...</code>

2) Shape

- 指tensor每个维度数据的个数，可以用python的list/tuple表示。

阶 rank	形状 shape	维数	实例
0	[]	0-D	一个0维张量，一个纯量
1	[D0]	1-D	一个1维张量的形式[5]
2	[D0, D1]	2-D	一个2维张量的形式[3, 4]
3	[D0, D1, D2]	3-D	一个3维张量的形式[1, 4, 3]
n	[D0, D1, ... Dn]	n-D	一个n维张量的形式[D0, D1, ..., Dn]



3) Data type

- | | |
|--------------------|-------------------------------------|
| 1、布尔型 (bool) | tf.bool |
| 2、整数型 (int) | |
| 1) 无符号整型 | tf.uint8 |
| 2) 有符号整型 | tf.int8 |
| 3、浮点型 (float) | |
| ■ tf.float16 (半精度) | tf.float16(1,5,10) |
| ■ tf.float32 (单精度) | tf.float32(1,8,23) |
| ■ tf.float64 (双精度) | tf.float64 (tf.double)
(1,11,52) |



4、字符串型 (string)

5、复数型 (complex)

- tf.complex64

- tf.complex128

6、裁剪类型 (Truncated float)

- tf.bfloat16 (TPU专用)

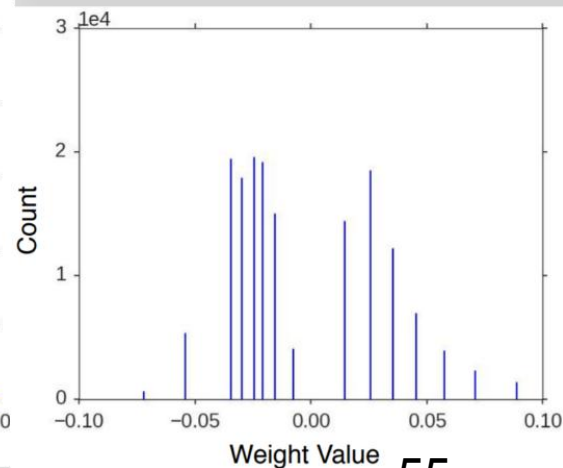
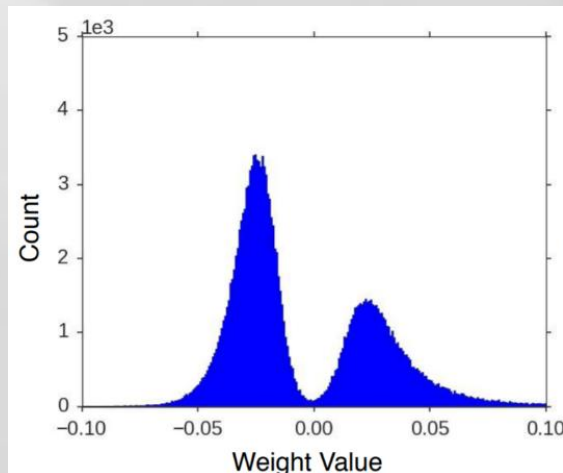
7、量化类型 (Quantized int)

1) quantized uint

- `tf.quint8`
- `tf.quint16`

2) quantized int

- `tf.qint8`
- `tf.qint16`
- `tf.qint32`





4) Variables与Constant

- TensorFlow 变量是用于表示程序处理的共享持久状态的方法。

```
biases = tf.Variable(tf.zeros([200]),  
name="var")
```

```
const = tf.constant(1.0,name="constant")
```




3 常用函数

- 随机数
- 数学函数
- 类型转换

3.1 随机数

tf.random

生成随机张量

- **normal()**

- **truncated_normal()**产生截断正态分布随机数，取值范围为 $[\text{mean} - 2 * \text{stddev}, \text{mean} + 2 * \text{stddev}]$

- **uniform()**

- **shuffle()**随机地将张量沿其第一维度打乱

tf.random.normal([2,4])



random.normal()参数

- **shape** 张量的形状
- **mean** 正态分布的均值
- **stddev** 指正态分布的标准差
- **dtype** 生成tensor的数据类型
- **seed** 设置生成随机数的种子
- **name** 生成的随机张量的名字

3.2 常用函数

操作组	操作
Maths	add,sub,mul,div,exp,log,round,sqrt,pow,exp
Array	Concat, Slice, Split, Constant, Rank, Shape, Shuffle
Matrix	MatMul, MatrixInverse, MatrixDeterminant
Neuronal Network	SoftMax, Sigmoid, ReLU, Convolution2D, MaxPool
Checkpointing	Save, Restore
Queues and synchronizations	Enqueue, Dequeue, MutexAcquire, MutexRelease
Flow control	Merge, Switch, Enter, Leave, NextIteration

3.3 数据类型转换

操作	描述
<code>tf.strings.to_number</code> (string, out_type=tf.dtypes.float32)	字符串转为数字
<code>tf.cast(x, dtype)</code>	将x或者x.values转换为dtype # tensor a is [1.8, 2.2], dtype=tf.float tf.cast(a, tf.int32) ==> [1, 2] # dtype=tf.int32



3.4 形状操作

- **tf.shape(input)**
- **返回数据的shape**

```
>>>t=[[[1, 1, 1], [2, 2, 2]], [[3, 3, 3], [4, 4, 4]]]
```

```
>>>tf.shape(t)
```

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

- **tf.size(input)**
- **返回数据的元素数量**

```
>>>t = [[[1, 1, 1], [2, 2, 2]], [[3,  
3, 3], [4, 4, 4]]]
```

```
>>>tf.size(t)
```

```
<tf.Tensor: shape=(), dtype=int32,  
numpy=12>
```



■ `tf.rank(input)`

■ 返回tensor的rank

tensor的rank表示一个tensor需要的索引数目,也就是通常所说的"order","degree"或"ndims"

```
>>>t = [[[1, 1, 1], [2, 2, 2]], [[3, 3, 3], [4, 4, 4]]]
```

```
>>>tf.rank(t)
```

```
<tf.Tensor: shape=(), dtype=int32, numpy=3>
```




- **tf.reshape(tensor, shape)**

- **改变tensor的形状**

```
>>>t=[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
>>>tf.reshape(t, [3, 3])
```

```
<tf.Tensor: shape=(3, 3),
```

```
dtype=int32, numpy=
```

```
array([[1, 2, 3],
```

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

```
       [7, 8, 9]])>
```



■ #如果shape有元素[-1],表示自动推导该维度值

-1 将自动推导得为 3:

`reshape(t, [3, -1]) ==>`

`[[1, 2, 3],`

`[4, 5, 6],`

`[7, 8, 9]]`

3.5 Tensor操作

■ `tf.expand_dims(input, dim)`

插入维度1进入一个tensor中

■ `t=[2,3]`, `t`的shape为 `[2]`

`shape(expand_dims(t, 0)) ==> [1, 2]`

`shape(expand_dims(t, 1)) ==> [2, 1]`

`shape(expand_dims(t, -1)) ==> [2, 1]`

■ `t2`的shape为 `[2, 3, 5]`

`shape(expand_dims(t2, 0)) ==> [1, 2, 3, 5]`

`shape(expand_dims(t2, 2)) ==> [2, 3, 1, 5]`

`shape(expand_dims(t2, 3)) ==> [2, 3, 5, 1]`



■ `tf.slice(input, begin, size)`对tensor进行切片操作

`begin`为每一个维度的开始索引值

```
input = [[[1, 1, 1], [2, 2, 2]], [[3, 3, 3], [4, 4, 4]], [[5, 5, 5], [6, 6, 6]]]
```

```
tf.slice(input, [1, 0, 0], [1, 1, 3]) ==> [[[3, 3, 3]]]
```

```
tf.slice(input, [1, 0, 0], [1, 2, 3]) ==>
[[[3, 3, 3],
[4, 4, 4]]]
```

```
tf.slice(input, [1, 0, 0], [2, 1, 3]) ==>
[[[3, 3, 3]],
[[5, 5, 5]]]
```

■ **tf.split(value, num_split,axis=0)**

沿着axis维度将value分割为num_split个tensor

x的shape为 [5, 30]

将x沿维度1分割为3个tensor

s0, s1, s2 = tf.split(1, 3, x)

tf.shape(s0) ==> [5, 10]

s0, s1, s2 = tf.split(x, [4, 15, 11], 1)

tf.shape(s0) ==> [5,4]

tf.shape(s1) ==> [5,15]

tf.shape(s2) ==> [5,11]



■ **tf.concat(values, concat_dim)** 沿着某一维度连结tensor

```
>>>t1 = [[1, 2, 3], [4, 5, 6]]
```

```
>>>t2 = [[7, 8, 9], [10, 11, 12]]
```

```
>>>tf.concat( [t1, t2],0)
```

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

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

```
 [7, 8, 9],
```

```
 [10, 11, 12]]
```

```
>>>tf.concat( [t1, t2],1)
```

```
[[1, 2, 3, 7, 8, 9],
```

```
 [4, 5, 6, 10, 11, 12]]
```

■ tf.reverse(tensor, dims)

沿着某维度进行序列反转

其中dim为列表，元素为bool型，size等于rank(tensor)

```
# tensor 't' is  
[[[ [ 0, 1, 2, 3],  
      [ 4, 5, 6, 7],  
      [ 8, 9, 10, 11]],  
  [[12, 13, 14, 15],  
   [16, 17, 18, 19],  
   [20, 21, 22, 23]]]
```

```
tensor 't' shape is [1, 2, 3, 4]
```

```
dims = [False, False, False, True]
```

```
reverse(t, dims) ==>  
[[[ [ 3, 2, 1, 0],  
      [ 7, 6, 5, 4],  
      [11,10, 9, 8]],  
  [[15, 14, 13, 12],  
   [19, 18, 17, 16],  
   [23, 22, 21, 20]]]
```



■ `tf.transpose(a, perm=None)`

调换tensor的维度顺序

按照列表perm的维度排列调换tensor顺序，
如果perm未定义，则perm为(n-1...0)

```
x = [[1, 2, 3], [4, 5, 6]]
```

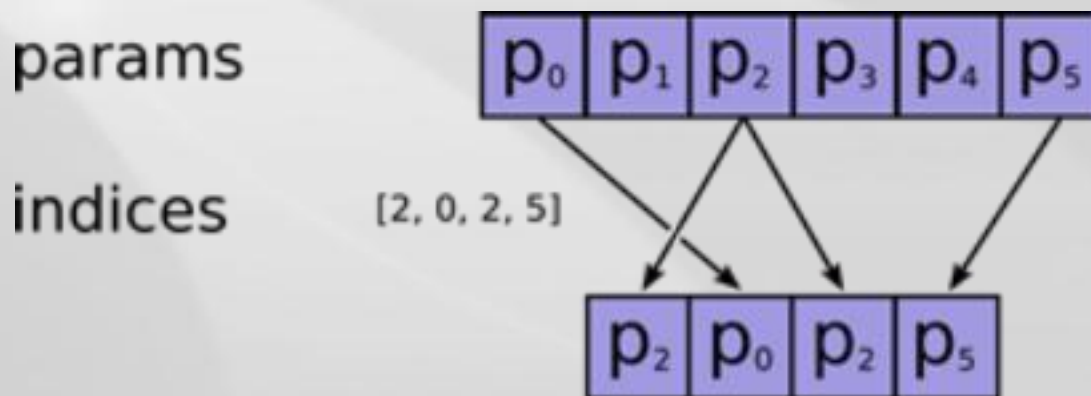
```
tf.transpose(x) ==> [[1 4], [2 5], [3 6]]
```

```
tf.transpose(x, [1, 0]) ==> [[1 4], [2 5], [3 6]]
```




■ **tf.gather(params, indices,
validate_indices=None)**

合并索引indices所指示params中的切片





```
■ tf.one_hot(  
    indices,  
    depth,  
    on_value=None,  
    off_value=None,  
    axis=None,  
    dtype=None  
)
```



- **indices**是一个列表,指定张量中独热向量的独热位置。
- 当**indices**的某个分量取-1时, 即对应的向量没有独热值。
- **depth**是每个独热向量的维度
- **on_value**是独热值
- **off_value**是非独热值
- **axis**指定第几阶为**depth**维独热向量, 默认为-1,即,指定张量的最后一维为独热向量



```
■ indices = [0, 1, 2]
  depth = 3
  tf.one_hot(indices, depth) # output: [3 x 3]
  # [[1., 0., 0.],
  #   [0., 1., 0.],
  #   [0., 0., 1.]]
```

```
indices = [0, 2, -1, 1]
depth = 3
tf.one_hot(indices, depth,
  on_value=5.0, off_value=0.0,
  axis=-1)  # output: [4 x 3]
# [[5.0, 0.0, 0.0], # one_hot(0)
#  [0.0, 0.0, 5.0], # one_hot(2)
#  [0.0, 0.0, 0.0], # one_hot(-1)
#  [0.0, 5.0, 0.0]] # one_hot(1)
```

```
indices = [[0, 2], [1, -1]]
depth = 3
tf.one_hot(indices, depth,
            on_value=1.0, off_value=0.0,
            axis=-1) # output: [2 x 2 x 3]
# [[[1.0, 0.0, 0.0], # one_hot(0)
#   [0.0, 0.0, 1.0]], # one_hot(2)
#   [[0.0, 1.0, 0.0], # one_hot(1)
#   [0.0, 0.0, 0.0]]] # one_hot(-1)
```



- **tf.unique(
 x,
 out_idx=tf.dtypes.int32)**
- 返回x中所有唯一元素，及其索引值。
- **# tensor 'x' is [1, 1, 2, 4, 4, 4, 7, 8, 8]**
y, idx = unique(x)
y ==> [1, 2, 4, 7, 8]
idx ==> [0, 0, 1, 2, 2, 2, 3, 4, 4]

3.6 矩阵运算

tf.linalg

■ **tf.linalg.diag(diagonal)**

diagonal = [1, 2, 3, 4]

tf.linalg.diag(diagonal) ==>

[[1, 0, 0, 0]

[0, 2, 0, 0]

[0, 0, 3, 0]

[0, 0, 0, 4]]



- **tf.linalg.trace(x)**

求二维tensor对角值之和

- **tf.linalg.det(x)**

求方阵行列式

- **tf.linalg.inv(x)**

求逆矩阵



```
■ tf.matmul(  
    a,  
    b,  
    transpose_a=False, #转置  
    transpose_b=False,  
    a_is_sparse=False, #稀疏矩阵  
    b_is_sparse=False)
```

矩阵相乘



■ **# 2-D tensor `a`**

[[1, 2, 3],

[4, 5, 6]]

a = tf.constant([1, 2, 3, 4, 5, 6], shape=[2, 3])

2-D tensor `b`

[[7, 8],

[9, 10],

[11, 12]]

b = tf.constant([7, 8, 9, 10, 11, 12], shape=[3, 2])



```
c = tf.matmul(a, b) # `a` * `b`  
# [[ 58, 64],  
# [139, 154]]
```



```
■ # 3-D tensor `a`  
# [[[ 1, 2, 3],  
#   [ 4, 5, 6]],  
#   [[ 7, 8, 9],  
#   [10, 11, 12]]]  
a = tf.constant(np.arange(1, 13,  
dtype=np.int32),  
■ shape=[2, 2, 3])
```



■ # 3-D tensor `b`

[[[13, 14],

[15, 16],

[17, 18]],

[[19, 20],

[21, 22],

[23, 24]]]

**b = tf.constant(np.arange(13, 25,
dtype=np.int32), shape=[2, 3, 2])**



```
■ # `a` * `b`  
# [[ 94, 100],  
# [229, 244],  
# [[508, 532],  
# [697, 730]]]  
c = tf.matmul(a, b)
```



3.7 复数操作

■ **`tf.complex(real, imag, name=None)`**

将两实数转换为复数形式

tensor 'real' is [2.25, 3.25]

tensor imag is [4.75, 5.75]

**`tf.complex(real, imag) ==> [[2.25 + 4.75j],
[3.25 + 5.75j]]`**



■ **tf.abs(x)**

计算复数的绝对值，即长度。

tensor 'x' is $[-2.25 + 4.75j], [-3.25 + 5.75j]$

tf.abs(x) ==> [5.25594902, 6.60492229]



- **`tf.math.conj(input, name=None)`**

计算共轭复数

- **`tf.math.imag(input, name=None)`**

- **`tf.math.real(input, name=None)`**

4 线性回归示例

- 用加利福尼亚房屋数据集进行线性回归

```
import numpy as np
from sklearn.datasets import
    fetch_california_housing
housing = fetch_california_housing()
m, n = housing.data.shape
#np.c_按column来组合array
housing_data_plus_bias =
    np.c_[np.ones((m, 1)), housing.data]
```



```
X = tf.constant(housing_data_plus_bias,  
dtype=tf.float32, name="X")  
y = tf.constant(housing.target.reshape(-1, 1),  
dtype=tf.float32, name="y")  
XT = tf.transpose(X)  
theta =  
tf.matmul(tf.matmul(tf.linalg.inv(tf.matmul(XT,  
X)), XT), y)  
print(theta)
```

5 深度神经网络示例

MNIST深度学习

■ 1) 加载模块

```
import tensorflow as tf  
import numpy as np
```



■ 2) 加载数据

```
def load_data(path):
```

```
    with np.load(path) as f:
```

```
        x_train, y_train = f['x_train'], f['y_train']
```

```
        x_test, y_test = f['x_test'], f['y_test']
```

```
    return (x_train, y_train), (x_test, y_test)
```



```
((trainX, trainY), (testX, testY)) =  
load_data("mnist.npz")  
trainX = trainX.astype("float32") / 255.0  
testX = testX.astype("float32") / 255.0
```



■ 建立模型

```
model = tf.keras.models.Sequential([  
    tf.keras.layers.Flatten(input_shape=(28, 28)),  
    tf.keras.layers.Dense(128, activation='relu'),  
    tf.keras.layers.Dropout(0.2),  
    tf.keras.layers.Dense(10, activation='softmax')  
])
```




■ 编译

```
model.compile(  
optimizer='adam',  
loss='sparse_categorical_crossentropy',  
metrics=['accuracy']  
)
```



■ 训练

model.fit(trainX, trainY, epochs=5)

■ 测试

model.evaluate(testX, testY, verbose=2)



作业

- 1、从MNIST的训练数据的0-9数字中各选择前100个，组成1000个数字的新训练集trainX1k。
- 2、使用 and 示例1中相同的训练模型，用trainX1k和trainX（6000个数字的训练集）各训练50epochs，比较二者的差别，并分析原因。
- 3、在TensorBoard中绘制二者的训练和测试过程数据曲线。