

# 深度学习

Tensorflow

DAY04

# Tensorflow概述

## Tensorflow概述

### Tensorflow简介

什么是Tensorflow

Tensorflow的特点

Tensorflow的发展历史

Tensorflow安装

案例1：快速开始

案例2：张量相加

### Tensorflow体系结构

体系结构概述

单机模式与分布式

后端逻辑层次

### 基本概念

张量

数据流

操作

图和会话

变量和占位符

# Tensorflow简介

---

# 什么是Tensorflow

- TensorFlow由谷歌人工智能团队谷歌大脑（Google Brain）开发和维护的开源深度学习平台，是目前人工智能领域主流的开发平台，在全世界有着广泛的用户群体。



# Tensorflow的特点

- 优秀的构架设计，通过“张量流”进行数据传递和计算，用户可以很容易地、可视化地看到张量流动的每一个环节
- 可轻松地在CPU/GPU上部署，进行分布式计算，为大数据分析提供计算能力的支撑
- 跨平台性好，灵活性强。TensorFlow不仅可在Linux、Mac和Windows系统中运行，甚至还可在移动终端下工作



# Tensorflow发展历史

- 2011年，Google公司开发了它的第一代分布式机器学习系统DistBelief。  
著名计算机科学家杰夫·迪恩（Jeff Dean）和深度学习专家吴恩达（Andrew Y. Ng）都是这个项目的核心成员
- 2015年11月，Google将它的升级版实现正式开源，协议遵循Apache 2.0并更名为TensorFlow
- 目前，TensorFlow最新版为2.X，教学使用1.14.0



# Tensorflow安装

## ➤ 在线安装

- ✓ 安装纯净包 : `pip install tensorflow`
- ✓ 安装GPU版本 : `pip install tf-nightly-gpu`

## ➤ 离线安装

- ✓ 下载离线包 : <https://pypi.org/project/tensorflow/#files> 执行安装



# Tensorflow安装（续）

- 修改源进行安装。如果安装包time out错误，则可以修改pip源，重新进行安装，修改方式：

（1）编辑或新建pip配置文件（~/.pip/pip.conf），在配置文件下加入：

[global]

index-url = http://mirrors.aliyun.com/pypi/simple/

[install]

trusted-host = mirrors.aliyun.com

（2）安装时将timeout时间设置长一点

sudo pip3 --timeout 600

install tensorflow-1.14.0-cp35-cp35m-manylinux1\_x86\_64.whl





# Tensorflow安装（续1）

➤ 也可使用如下完整命令安装：

```
pip3 install --user tensorflow==1.14.0 --index-url  
https://pypi.tuna.tsinghua.edu.cn/simple/ --trusted-host  
https://pypi.tuna.tsinghua.edu.cn --timeout 600
```



# 案例1：快速开始

```
# tf的helloworld程序
import tensorflow as tf

hello = tf.constant('Hello, world!') # 定义一个常量
sess = tf.Session() # 创建一个session
print(sess.run(hello)) # 计算
sess.close()
```



## 案例2：张量相加

```
1  # 常量加法运算示例
2  import tensorflow as tf
3  import os
4
5  os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3' #调整警告级别
6
7  a = tf.constant(5.0) # 定义常量a
8  b = tf.constant(1.0) # 定义常量a
9  c = tf.add(a, b)
10
11 with tf.Session() as sess:
12     print(sess.run(c)) # 执行计算
```



# Tensorflow体系结构

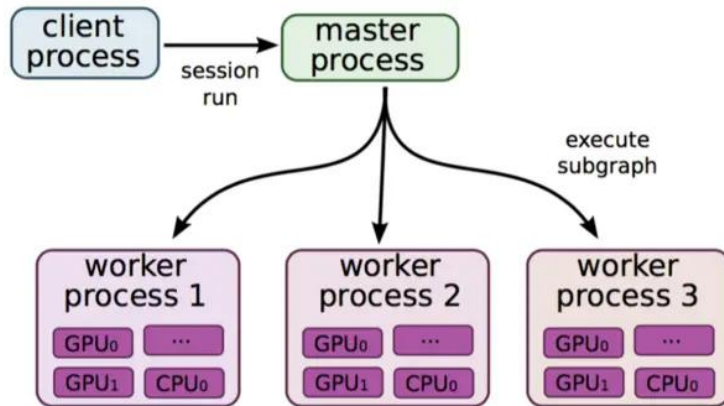
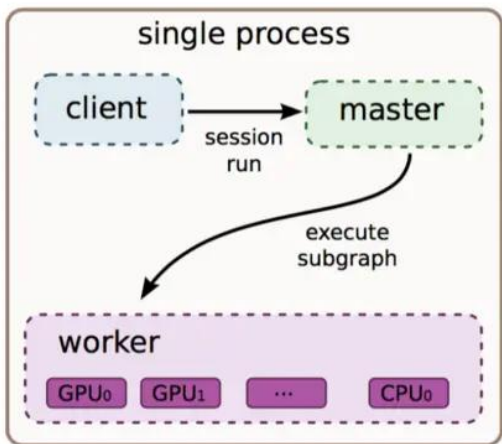
---

- TensorFlow属于“定义”与“运行”相分离的运行机制。从操作层面可以抽象成两种：模型构建和模型运行
  - ✓ 客户端：用户编程、执行使用
  - ✓ master：用来与客户端交互，并进行任务调度
  - ✓ worker process：工作节点，每个worker process可以访问一到多个device
  - ✓ device：TF的计算核心，执行计算

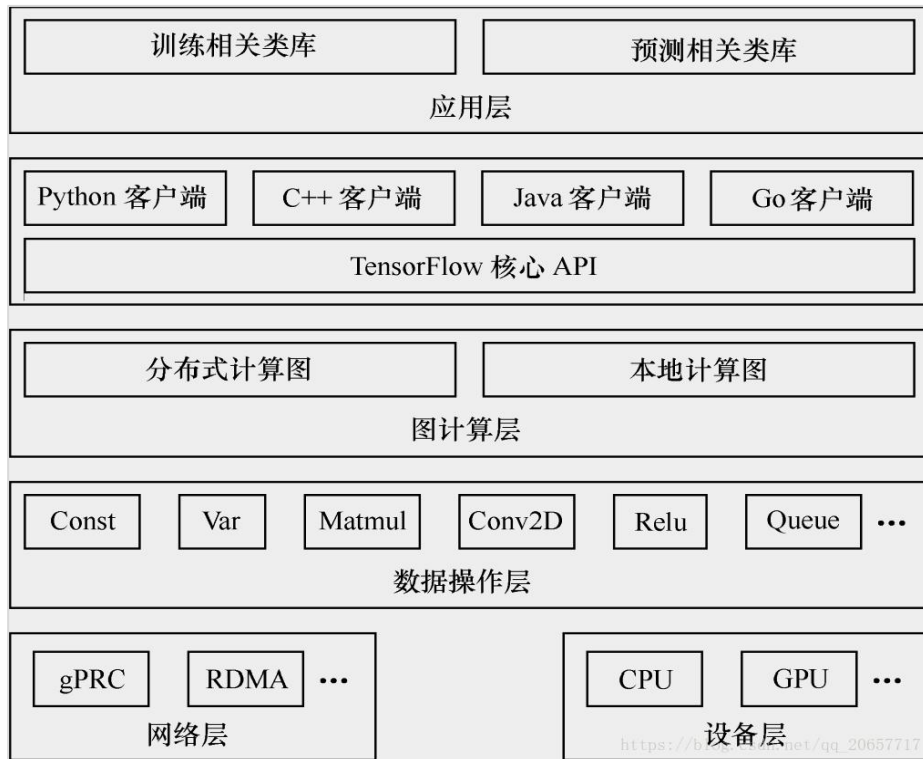


# 单机模式与分布式模式

- TF的实现分为“单机实现”和“分布式实现”



# 后端逻辑层次



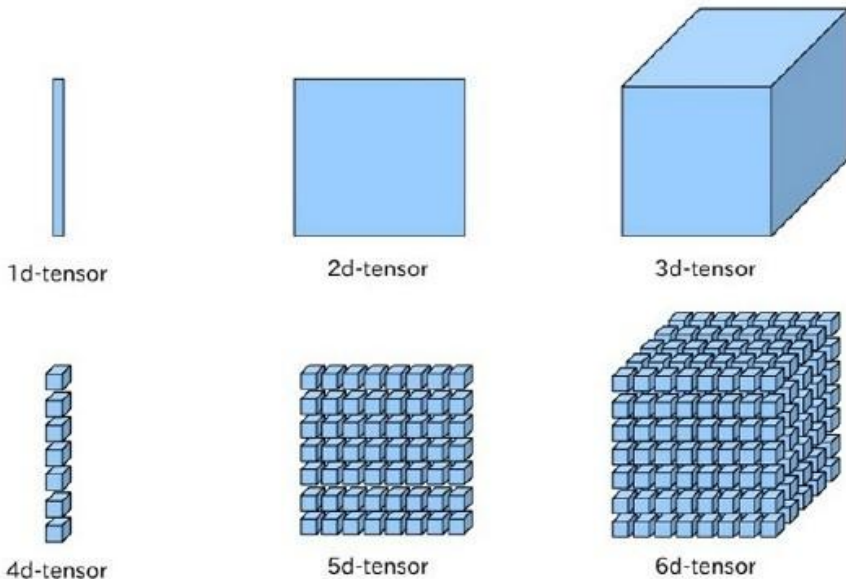
# 基本概念

---



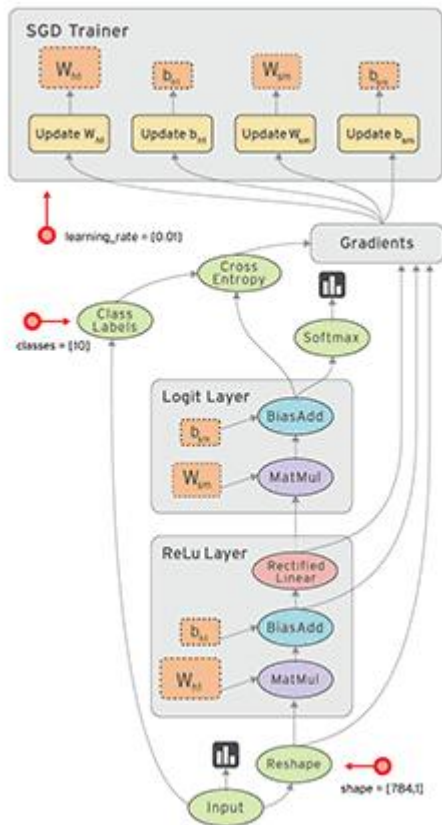
# 张量

- 张量 ( Tensor ) : 多维数组或向量，张量是数据的载体，包含名字、形状、数据类型等属性



# 数据流

- 数据流图 ( Data Flow Graph ) 用 “结点” ( nodes ) 和 “线” ( edges ) 的有向图来描述数学计算
- “节点” 一般用来表示数学操作，也可以表示数据输入 ( feed in ) 的起点/输出 ( push out ) 的终点，或者是读取/写入持久变量 ( persistent variable ) 的终点
- “线” 表示 “节点” 之间的输入/输出关系。这些数据 “线” 可以输运多维数据数组，即 “张量” ( tensor )
- 一旦输入端的所有张量准备好，节点将被分配到各种计算设备完成异步并行地执行运算



- 操作（Operation，简称op）指专门执行计算的节点，tensorflow函数或API定义的都是操作。常用操作包括：
  - 标量运算，向量运算，矩阵运算
  - 带状态的运算
  - 神经网络组建
  - 存储、恢复
  - 控制流
  - 队列及同步运算



# 图和会话

- 图 ( Graph ) 描述整个程序结构，Tensorflow中所有的计算都构建在图中
- 会话 ( Session ) 用来执行图的运算



# 变量和占位符

- 在Tensorflow中，变量（Variable）是一种操作，变量是一种特殊的张量，能够进行存储持久化（张量不能进行持久化），它的值是张量
- 占位符（placeholder）是变量占位符，当不能确定变量的值时，可以先声明一个占位符，真正执行时再传入变量



# Tensorflow基本使用

## Tensorflow基本使用

图和会话操作

张量及基本运算

什么是图

会话及相关操作

张量的阶与形状

张量的数据类型

张量常用属性

张量类型转换

占位符

张量形状改变

张量数学计算

变量

共享变量

# 图和会话操作

---

# 什么是图

- 图 ( Graph ) 描述了计算的过程。TensorFlow 程序通常被组织成一个构建阶段和一个执行阶段。在构建阶段, op 的执行步骤 被描述成一个图. 在执行阶段, 使用会话执行执行图中的 op.
- TensorFlow Python 库有一个默认图 (default graph), op 构造器可以为其增加节点. 这个默认图对 许多程序来说已经足够用了 , 也可以创建新的图来描述计算过程
- 在Tensorflow中 , op/session/tensor都有graph属性





## 案例3：查看图对象

```
2 import tensorflow as tf
3 import os
4
5 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2' # 调整警告级别
6
7 a = tf.constant(5.0) # 定义常量a
8 b = tf.constant(1.0) # 定义常量a
9 c = tf.add(a, b)
10 print("c:", c)
11
12 graph = tf.get_default_graph() # 获取缺省图
13 print(graph)
14
15 with tf.Session() as sess:
16     print(sess.run(c)) # 执行计算
17     print(a.graph) # 通过tensor获取graph对象
18     print(c.graph) # 通过op获取graph对象
19     print(sess.graph) # 通过session获取graph对象
```



# 会话及相关操作

- 会话（session）用来执行图中的计算，并且保存了计算张量对象的上下文信息。会话的作用主要有：
  - 运行图结构
  - 分配资源
  - 掌握资源（如变量、队列、线程）
- 一个session只能执行一个图的运算。可以在会话对象创建时，指定运行的图。如果在构造会话时未指定图形参数，则将在会话中使用默认图。如果在同一进程中使用多个图（使用`tf.graph()`创建），则必须为每个图使用不同的会话，但每个图可以在多个会话中使用。



# 会话及相关操作（续）

- 创建会话
  - `tf.Session()` # 使用默认图
- 运行
  - `session.run(fetches, feed_dict=None)`
  - 参数：`fetches` 图中的单个操作，或多个操作的列表  
`feed_dict` 运行传入的参数构成的字典，可以覆盖之前的值
- 关闭
  - `session.close()`



## 案例4：指定会话运行某个图

```
2 import tensorflow as tf
3 import os
4 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2' # 调整警告级别
5
6 a = tf.constant(5.0) # 定义常量a
7 b = tf.constant(1.0) # 定义常量a
8 c = tf.add(a, b)
9
10 graph = tf.get_default_graph() # 获取缺省图
11 print(graph)
12
13 graph2 = tf.Graph()
14 print(graph2)
15 with graph2.as_default(): # 设置为默认图
16     d = tf.constant(11.0)
17
18 with tf.Session(graph=graph2) as sess:
19     print(sess.run(d)) # 执行计算
20     # print(sess.run(c)) # 报错
```



# 会话常见的错误及原因

- 调用run()方法时，可能会出现的错误及原因
  - RuntimeError : Session处于无效（如关闭）
  - TypeError : fetches或feed\_dict的键是不合适的值
  - ValueError : fetches或feed\_dict的键无效或引用的值不存在



# 张量及基本运算

---

# 张量的阶与形状

- 阶：张量的维度（数方括号的层数）
- 形状表示方法
  - ✓ 0维： $()$
  - ✓ 1维： $(5)$ ，1行5个元素
  - ✓ 2维： $(2,3)$ ，2行3列
  - ✓ 3维： $(2,3,4)$ ，两个3行4列的矩阵



# 张量的数据类型

数据类型	Python 类型	描述
DT_FLOAT	tf.float32	32 位浮点数.
DT_DOUBLE	tf.float64	64 位浮点数.
DT_INT64	tf.int64	64 位有符号整型.
DT_INT32	tf.int32	32 位有符号整型.
DT_INT16	tf.int16	16 位有符号整型.
DT_INT8	tf.int8	8 位有符号整型.
DT_UINT8	tf.uint8	8 位无符号整型.
DT_STRING	tf.string	可变长度的字节数组. 每一个张量元素都是一个字节数组.
DT_BOOL	tf.bool	布尔型.
DT_COMPLEX64	tf.complex64	由两个32位浮点数组成的复数: 实数和虚数.
DT_QINT32	tf.qint32	用于量化Ops的32位有符号整型.
DT_QINT8	tf.qint8	用于量化Ops的8位有符号整型.
DT_QUINT8	tf.quint8	用于量化Ops的8位无符号整型.





# 张量常用属性

属性名称	说明
graph	所属的默认图
op	张量的操作名
name	名称
shape	形状
dtype	元素类型



## 案例5：查看张量属性

课堂练习

```
2 import tensorflow as tf
3 import os
4
5 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2' # 调整警告级别
6
7 a = tf.constant(5.0) # 定义常量a
8
9 with tf.Session() as sess:
10     print(sess.run(a)) # 执行计算
11     print("name:", a.name)
12     print("dtype:", a.dtype)
13     print("shape:", a.shape)
14     print("op:", a.op)
15     print("graph:", a.graph)
```



## 案例6：生成张量

```
2 import tensorflow as tf
3
4 # 生成值全为0的张量
5 tensor_zeros = tf.zeros(shape=[2, 3], dtype="float32")
6 # 生成值全为1的张量
7 tensor_ones = tf.ones(shape=[2, 3], dtype="float32")
8 # 创建正态分布张量
9 tensor_nd = tf.random_normal(shape=[10],
10                               mean=1.7,
11                               stddev=0.2,
12                               dtype="float32")
13 # 生成和输入张量形状一样的张量，值全为1
14 tensor_zeros_like = tf.zeros_like(tensor_ones)
15
16 with tf.Session() as sess:
17     print(tensor_zeros.eval()) # eval表示在session中计算该张量
18     print(tensor_ones.eval())
19     print(tensor_nd.eval())
20     print(tensor_zeros_like.eval())
```



# 张量类型转换

函数名称	说明
<code>tf.string_to_number(string_tensor)</code>	字符串转换为数字
<code>tf.to_double(x)</code>	转换为64位浮点型
<code>tf.to_float(x)</code>	转换为32位浮点型
<code>tf.to_int32(x)</code> <code>tf.to_int64(x)</code>	转换为32/64位整型
<code>tf.cast(x, dtype)</code>	将x转换为dtype所指定的类型



## 案例7：张量类型转换

```
# 张量类型转换
import tensorflow as tf

tensor_ones = tf.ones(shape=[2, 3], dtype="int32")
tensor_float = tf.constant([1.1, 2.2, 3.3])

with tf.Session() as sess:
    print(tf.cast(tensor_ones, tf.float32).eval())
    # print(tf.cast(tensor_float, tf.string).eval()) #不支持浮点数到字符串直接转换
```



# 占位符

- 不确定张量内容情况下，可以使用占位符先占个位置，然后执行计算时，通过参数传入具体数据执行计算（通过feed\_dict参数指定）。placeholder节点被声明的时候是未初始化的，也不包含数据，如果没有为它供给数据，则TensorFlow运算的时候会产生错误
- 占位符定义：

```
name = placeholder(dtype, shape=None, name=None)
```



## 案例8：占位符使用

```
2 import tensorflow as tf
3
4 # 不确定数据，先使用占位符占个位置
5 plhd = tf.placeholder(tf.float32, [2, 3]) # 2行3列的tensor
6 plhd2 = tf.placeholder(tf.float32, [None, 3]) # N行3列的tensor
7
8 with tf.Session() as sess:
9     d = [[1, 2, 3],
10         [4, 5, 6]]
11     print(sess.run(plhd, feed_dict={plhd: d}))
12     print("shape:", plhd.shape)
13     print("name:", plhd.name)
14     print("graph:", plhd.graph)
15     print("op:", plhd.op)
16     print(sess.run(plhd2, feed_dict={plhd2: d}))
```



# 张量形状改变

➤ 静态形状：在创建一个张量，初始状态的形状

✓ `tf.Tensor.get_shape()`：获取Tensor对象的静态形状

✓ `tf.Tensor.set_shape()`：更新Tensor对象的静态形状

注意：转换静态形状的时候，1-D到1-D，2-D到2-D，不能跨阶数改变形状；

对于已经固定或者设置静态形状的张量/变量，不能再次设置静态形状

➤ 动态形状：在运行图时，动态形状才是真正用到的，这种形状是一种描述原始张量在执行过程中的一种张量

✓ `tf.reshape(tf.Tensor, shape)`：创建一个具有不同动态形状的新张量

✓ 可以跨纬度转换，如1D-->2D, 1D-->3D





## 案例9：修改张量形状

课堂练习

```
1  # 改变张量形状示例(重点)
2  import tensorflow as tf
3
4  pld = tf.placeholder(tf.float32, [None, 3])
5  print(pld)
6
7  pld.set_shape([4, 3])
8  print(pld)
9  # pld.set_shape([3, 3]) #报错, 静态形状一旦固定就不能再设置静态形状
10
11 # 动态形状可以创建一个新的张量, 改变时候一定要注意元素的数量要匹配
12 new_pld = tf.reshape(pld, [3, 4])
13 print(new_pld)
14 # new_pld = tf.reshape(pld, [2, 4]) # 报错, 元素的数量不匹配
15
16 with tf.Session() as sess:
17     pass
```



# 张量数学计算

函数名称	说明
<code>tf.add(x, y)</code>	张量相加
<code>tf.matmul(x, y)</code>	张量相乘
<code>tf.log(x)</code>	求张量的自然对数
<code>tf.reduce_sum(x, axis)</code>	计算张量指定维度上的总和
<code>tf.segment_sum(data, segment_ids)</code>	计算张量片段总和



## ➤ 矩阵乘法说明

- ✓ 当矩阵A的列数（column）等于矩阵B的行数（row）时，A与B可以相乘
- ✓ 矩阵C的行数等于矩阵A的行数，C的列数等于B的列数
- ✓ 乘积C的第m行第n列的元素等于矩阵A的第m行的元素与矩阵B的第n列对应元素乘积之和

$$C = AB = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix} = \begin{pmatrix} 1 \times 1 + 2 \times 2 + 3 \times 3 & 1 \times 4 + 2 \times 5 + 3 \times 6 \\ 4 \times 1 + 5 \times 2 + 6 \times 3 & 4 \times 4 + 5 \times 5 + 6 \times 6 \end{pmatrix} = \begin{pmatrix} 14 & 32 \\ 32 & 77 \end{pmatrix}$$



# 案例10：张量数学计算

课堂练习

```
2 import tensorflow as tf
3
4 x = tf.constant([[1, 2], [3, 4]], dtype=tf.float32)
5 y = tf.constant([[4, 3], [3, 2]], dtype=tf.float32)
6
7 x_add_y = tf.add(x, y) # 张量相加
8 x_mul_y = tf.matmul(x, y) # 张量相乘
9 log_x = tf.log(x) # log(x)
10
11 # reduce_sum: 此函数计算一个张量的各个维度上元素的总和
12 # 按照axis中已经给定的维度来减少的; 除非 keep_dims 是true
13 # 否则张量的秩将在axis的每个条目中减少1
14 # 如果keep_dims为true,则减小的维度将保留为长度1
15 # 如果axis没有条目,则缩小所有维度,并返回具有单个元素的张量.
16 x_sum_1 = tf.reduce_sum(x, axis=[1], keepdims=False)
```



## 案例10：张量数学计算（续）

课堂练习

```
18 # segment_sum: 沿张量的片段计算总和
19 # 函数返回的是一个Tensor,它与data有相同的类型,与data具有相同的形状
20 # 但大小为 k(段的数目)的维度0除外
21 data = tf.constant([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], dtype=tf.float32)
22 segment_ids = tf.constant([0, 0, 0, 1, 1, 2, 2, 2, 2, 2], dtype=tf.int32)
23 x_seg_sum = tf.segment_sum(data, segment_ids) # [6, 9, 40]
24
25 with tf.Session() as sess:
26     print(x_add_y.eval())
27     print(x_mul_y.eval())
28     print(x_mul_y.eval())
29     print(log_x.eval())
30     print(x_seg_sum.eval())
```



# 变量

- 变量是一种op，它的值是张量
- 变量能够持久化保存，普通张量则不可
- 当定义一个变量时，需要在会话中进行初始化
- 变量创建

```
tf.Variable(initial_value=None, name=None)
```



# 案例11：变量使用

```
1  # 变量OP示例
2  import tensorflow as tf
3
4  # 创建普通张量
5  a = tf.constant([1, 2, 3, 4, 5])
6  # 创建变量
7  var = tf.Variable(tf.random_normal([2, 3], mean=0.0, stddev=1.0),
8                    name="variable")
9
10 # 变量必须显式初始化，这里定义的是初始化操作，并没有运行
11 init_op = tf.global_variables_initializer()
12
13 with tf.Session() as sess:
14     sess.run(init_op)
15     print(sess.run([a, var]))
```



# Tensorboard可视化

## Tensorboard可视化

Tensorboard工具

综合案例：实现线性回归

什么是可视化

启动Tensorboard

Tensorboard主页说明

摘要与事件文件操作

实现线性回归



# Tensorboard工具

---

# 什么是可视化

- 可视化是用来查看在Tensorflow平台下程序运行的过程，包括：张量/变量，操作，数据流，学习过程等，从而方便 TensorFlow 程序的理解、调试与优化
- Tensorflow提供了专门的可视化工具tensorboard，它将tensorflow执行的数据、模型、过程用图形方式进行显示。tensorflow在执行过程中，可以通过某些操作，将模型、数据、graph等信息，保存到磁盘中的Events文件中，从而提供给tensorboard进行可视化



# 启动tensorboard

- 使用以下命令启动tensorboard

```
tensorboard --logdir="PycharmProjects/tensorflow_study/summary/"
```

其中，logdir参数的值为事件文件存储目录，启动成功后可以看到如下信息，使用提示的URL地址和端口进行访问：

```
TensorBoard 1.14.0 at http://tedu:6006/ (Press CTRL+C to quit)
```



# tensorboard主页说明

The screenshot shows the TensorBoard web interface. At the top, there are three tabs: "TensorBoard", "SCALARS", and "GRAPHS". The "SCALARS" tab is highlighted with a red box, and a red arrow points from the Chinese character "标量" (Scalar) to it. The "GRAPHS" tab is also highlighted with a red box, and a red arrow points from the Chinese character "图" (Graph) to it. On the left side, there is a sidebar with various controls: a search bar, "Fit to Screen" and "Download PNG" buttons, "Run" and "Tag" dropdowns, an "Upload" button, radio buttons for "Graph", "Conceptual Graph", and "Profile", a "Trace inputs" toggle, and radio buttons for "Color" (Structure, Device, XLA Cluster, Compute time). Below these is a legend for the graph visualization. The main area displays a computational graph with nodes "variable" and "random\_nor...", connected by a dataflow edge. The "variable" node has an "init" operation. On the right side, there is a vertical list of "CUSTOM SCALARS" including IMAGES, AUDIO, DISTRIBUTIONS, HISTOGRAMS, PROJECTOR, TEXT, PR CURVES, PROFILE, BEHOLDER, WHAT-IF TOOL, HPARAMS, MESH, and DEBUGGER.

TensorBoard

SCALARS

GRAPHS

CUSTOM SCALARS

IMAGES

AUDIO

DISTRIBUTIONS

HISTOGRAMS

PROJECTOR

TEXT

PR CURVES

PROFILE

BEHOLDER

WHAT-IF TOOL

HPARAMS

MESH

DEBUGGER

Search nodes. Regexes supported.

Fit to Screen

Download PNG

Run (1)

Tag (1) Default

Upload

Choose File

Graph

Conceptual Graph

Profile

Trace inputs

Color

Structure

Device

XLA Cluster

Compute time

Close legend.

Graph (\* = expandable)

Namespace\* 2

OpNode 2

Unconnected series\* 2

Connected series\* 2

Constant 2

Summary 2

Dataflow edge 2

Control dependency edge 2

Reference edge 2

variable

init

random\_nor...

variable

a

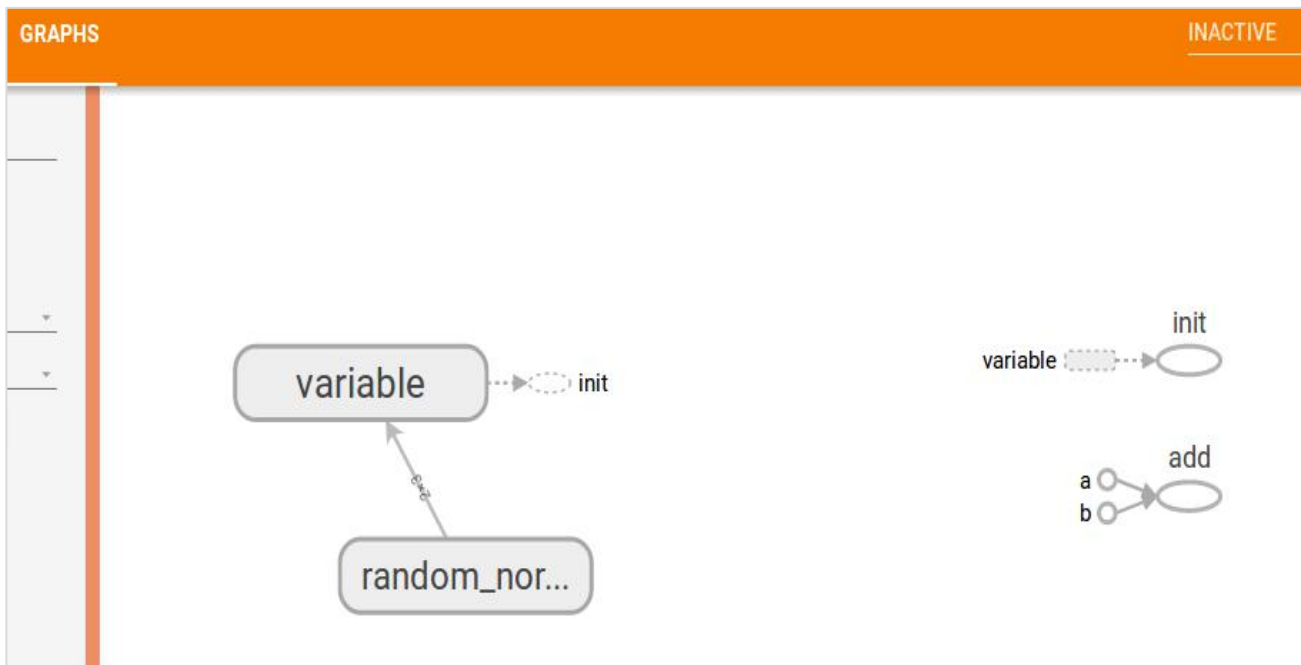
b

# 案例12：为操作添加可视化

```
2 import tensorflow as tf
3 import os
4 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2' # 调整警告级别
5
6 # 创建普通张量
7 a = tf.constant([1, 2, 3, 4, 5])
8 # 创建变量
9 var = tf.Variable(tf.random_normal([2, 3], mean=0.0, stddev=1.0),
10                  name="variable")
11
12 b = tf.constant(3.0, name="a")
13 c = tf.constant(4.0, name="b")
14 d = tf.add(b, c, name="add")
15
16 # 变量必须显式初始化，这里定义的是初始化操作，并没有运行
17 init_op = tf.global_variables_initializer()
18
19 with tf.Session() as sess:
20     sess.run(init_op)
21     # 将程序图结构写入事件文件
22     fw = tf.summary.FileWriter("../summary/", graph=sess.graph)
23     print(sess.run([a, var]))
```



# 案例12：为操作添加可视化（续）



注：张量如果未使用默认情况下不显示



# 摘要与事件文件操作

- 如果需要将变量/张量在tensorboard中显示，需要执行以下两步：

- ✓ 收集变量

`tf.summary.scalar(name, tensor)` # 收集标量，name为名字，tensor为值

`tf.summary.histogram(name, tensor)` # 收集高维度变量参数

`tf.summary.image(name, tensor)` # 收集图片张量

- ✓ 合并变量并写入事件文件

`merged = tf.summary.merge_all()` # 合并所有变量

`summary = sess.run(merged)` # 运行合并，每次迭代训练都需要运行

`FileWriter.add_summary(summary, i)` # 添加摘要，i表示第几次的值



# 综合案例：实现线性回归

---



# 案例13：实现线性回归

- 任务描述：
  - 给定一组输入、输出作为样本
  - 定义线性模型，并进行训练
  - 将训练过程可视化



# 案例13：实现线性回归（续1）

```
1  # 线性回归示例
2  import tensorflow as tf
3
4  # 第一步：创建数据
5  x = tf.random_normal([100, 1], mean=1.75, stddev=0.5, name="x_data")
6  y_true = tf.matmul(x, [[2.0]]) + 5.0 # 矩阵相乘必须是二维的
7
8  # 第二步：建立线性回归模型
9  # 建立模型时，随机建立权重、偏置  $y = wx + b$ 
10 # 权重需要不断更新，所以必须是变量类型。trainable指定该变量是否能随梯度下降一起变化
11 weight = tf.Variable(tf.random_normal([1, 1], name="w"),
12                       trainable=True) # 训练过程中值是否允许变化
13 bias = tf.Variable(0.0, name="b", trainable=True) # 偏置
14 y_predict = tf.matmul(x, weight) + bias # 计算  $wx + b$ 
15
16 ## 第三步：求损失函数，误差(均方差)
17 loss = tf.reduce_mean(tf.square(y_true - y_predict))
```



## 案例13：实现线性回归（续2）

```
19 # # 第四步：使用梯度下降法优化损失
20 # 学习率是比价敏感的参数，过小会导致收敛慢，过大可能导致梯度爆炸
21 train_op = tf.train.GradientDescentOptimizer(0.1).minimize(loss)
22
23 # 收集损失值
24 tf.summary.scalar("losses", loss)
25 merged = tf.summary.merge_all() # 将所有的摘要信息保存到磁盘
26
27 init_op = tf.global_variables_initializer()
28 with tf.Session() as sess: # 通过Session运行op
29     sess.run(init_op)
30     # 打印初始权重、偏移值
31     print("weight:", weight.eval(), " bias:", bias.eval())
32
33     # 指定事件文件
34     fw = tf.summary.FileWriter("../summary/", graph=sess.graph)
35
36     for i in range(500): # 循环执行训练
37         sess.run(train_op) # 执行训练
38         summary = sess.run(merged) # 运行合并后的tensor
39         fw.add_summary(summary, i)
40         print(i, ":", i, "weight:", weight.eval(), " bias:", bias.eval())
```



## 今日总结

- Tensorflow简介
- 图与会话
- 张量基本操作
- Tensorboard可视化
- 综合案例：线性回归