

第5讲 TensorFlow 基础

- 计算模型-Graph
- 数据结构-Tensor
- 运行模型-Session
- Operation
- Placeholder
- Variable
- Name Scope
- TensorFlow建模流程

20190326

TensorFlow

- Tensorflow: 2015年底开源的Google深度学习框架
- 使用数据流图进行数值计算。图中的节点表示数学运算，而图形边表示在它们之间传递的多维数据阵列（张量）
2019.3.26
- 目前被广泛应用于Google的各项产品中
- 已经成为GitHub上Star数和fork数最多的机器学习框架

Tensorflow

- 官方定义:
- Tensorflow is an Open Source Software Library for Machine Intelligence
- Tensorflow™ is an open source software library for numerical computation using data flow graph
- Open Source: 自由下载、修改和使用其代码
- Library for Numerical Computation: 创建自定义模型 + 做复杂数学计算
- Data Flow Graph: 计算模型

TensorFlow安装

- Tensorflow的windows安装也分CPU版本和GPU版本
- CPU版本安装（保证path中有相应的环境变量，如C:\Users\Anaconda3\Scripts）
- 在cmd中执行pip install tensorflow



A screenshot of a Windows command prompt window titled 'C:\Windows\system32\cmd.exe - pip install tensorflow'. The window shows the following text:
Microsoft Windows [版本 6.1.7601]
版权所有 © 2009 Microsoft Corporation。保留所有权利。
C:\Users\zgl>pip install tensorflow
Collecting tensorflow
 Downloading tensorflow-1.1.0-cp35-cp35m-win_amd64.whl (19.4MB)
 1% :■

- 确保最新的 pip，可运行如下命令：
`python -m pip install --upgrade pip`
- 官方 <https://www.tensorflow.org/install/>

Tensorflow初体验

```
#tensorflow初体验
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

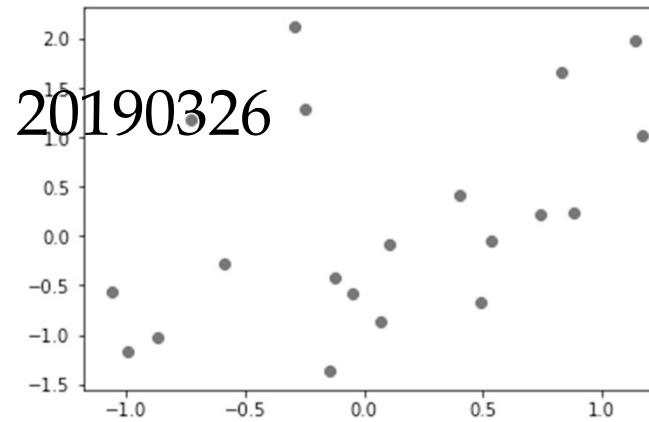
#在浏览器中显示matplotlib图表
%matplotlib inline

#定义2x20的随机数矩阵
a = tf.random_normal([2, 20])

#启动一个tensorflow会话
sess = tf.Session()

#在sess会话里执行a, 结果存放在out
out = sess.run(a)
x, y = out

#用pyplot创建一系列散列点, 坐标为x和y
plt.scatter(x, y)
#plt.show()
```



TensorFlow概念

TensorFlow = Tensor + Flow

Tensor 张量

数据结构：多维数组

20190326

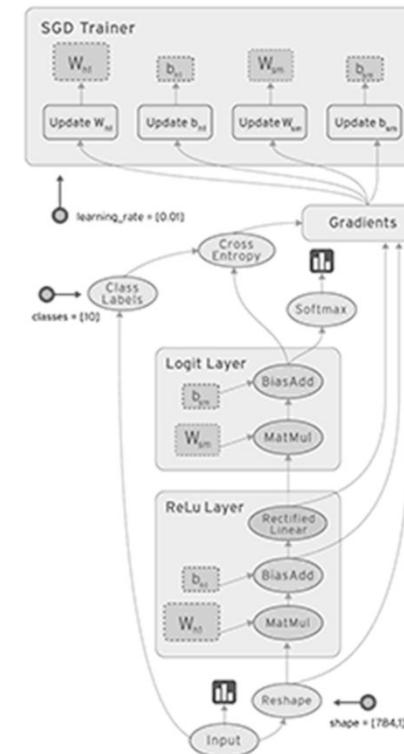
Flow 流

计算模型：张量之间通过计算而转换的过程

TensorFlow是一个通过计算图的形式表述计算的编程系统

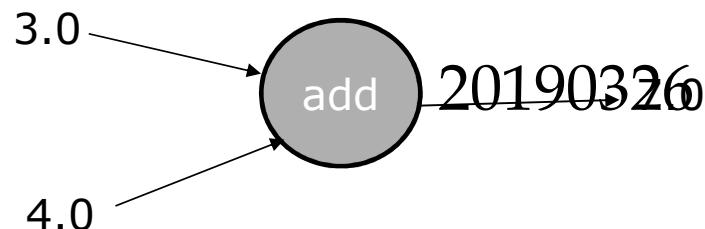
每一个计算都是计算图上的一个节点

节点之间的边描述了计算之间的关系



实例 一个简单计算图

□ 基本加法运算的数据流程



□ 数据流程用公式表示:

$$f(3.0, 4.0) = 3.0 + 4.0 = 7.0$$

构建计算图

□ TensorFlow的Graph对象：

```
import tensorflow as tf

# 创建一个新的数据流图          20190326
g = tf.Graph()

# 利用Graph.as_default()方法访问其上下文管理器为其添加op
with g.as_default():
    # 添加一些op, 它们将被添加到Graph对象g中
    a = add(2, 3)
    ...
```

构建计算图

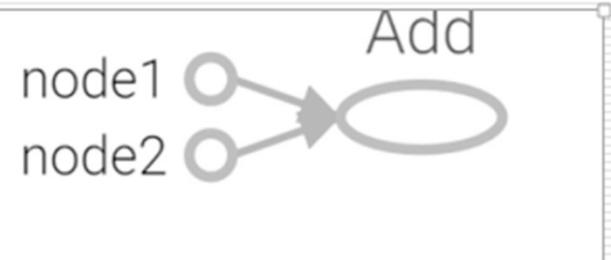
```
# 一个简单计算图
node1 = tf.constant(3.0,tf.float32,name="node1")
node2 = tf.constant(4.0,tf.float32,name="node2")
node3 = tf.add(node1, node2)
```

20190326

增加的Op、Tensor对象会自动放置在默认的数据流图中

```
print(node3)
```

```
Tensor("Add:0", shape=(), dtype=float32)
```



执行计算图

- 创建计算图只是建立静态计算模型
- 使用会话（session）执行图中的op来提供数据和获得结果

```
In [8]: # 建立会话
sess=tf.Session()
# 显示运行结果
print("运行sess.run(node1)的结果: ", sess.run(node1))
print("运行sess.run(node2)的结果: ", sess.run(node2))
print("运行sess.run(node3)的结果: ", sess.run(node3))

# 关闭session
sess.close()

20190326
运行sess.run(node1)的结果:  3.0
运行sess.run(node2)的结果:  4.0
运行sess.run(node3)的结果:  7.0
```

Session会话

- Session对象在使用完成后需要关闭以释放资源，除了显式调用close外，也可以使用with代码来自动完成关闭动作：

```
In [18]: node1 = tf.constant(3.0,tf.float32,name="node1")
node2 = tf.constant(4.0,tf.float32,20190326, name="node2")
result = tf.add(node1, node2)

#创建一个会话，并通过Python中的上下文管理器来管理这个会话
with tf.Session() as sess:
    #使用这创建好的会话来计算关心的结果
    print(sess.run(result))

# 不需要再调用 Session.close() 函数来关闭会话
# 当上下文退出时会话关闭和资源释放也自动完成了
```

7.0

默认会话

TensorFlow不会自动生成默认的会话，需要手动指定

当默认的会话被指定之后可以通过 `tf.Tensor.eval` 函数来计算一个张量的取值

```
In [19]: node1 = tf.constant(3.0,tf.float32,name="node1")
node2 = tf.constant(4.0,tf.float32,name="node2")
result = tf.add(node1, node2)
```

20190326

```
sess = tf.Session()
with sess.as_default():
    print(result.eval())|
```

7.0

```
In [20]: sess = tf.Session()
```

```
#下面两个命令有相同的功能
print(sess.run(result))
```

```
print(result.eval(session=sess))
```

7.0

7.0

19

右边代码也可以完成相同的功能

交互式会话

- 使用诸如IPython之类的python交互环境，可用InteractiveSession代替Session类，使用Tensor.eval()和Operation.run()方法代替Session.run()

tf.InteractiveSession 使用这个函数会自动将生成的会话注册为默认会话

20190326

```
In [23]: node1 = tf.constant(3.0,tf.float32,name="node1")
node2 = tf.constant(4.0,tf.float32,name="node2")
result = tf.add(node1, node2)

sess = tf.InteractiveSession()

print(result.eval())
sess.close()
```

7.0

Tensor 张量

□ Tensor: 数据结构

- 在TensorFlow中，所有的数据都通过张量的形式来表示
- 从功能的角度，张量可以简单理解为多维数组
零阶张量表示标量（scalar），也就是一个数；
一阶张量为向量（vector），也就是一维数组；
n阶张量可以理解为一个n维数组；
- 张量并没有真正保存数字，它保存的是计算过程

张量的属性

```
print(node3)  
Tensor("Add:0", shape=(), dtype=float32)
```

Tensor("Add:0", shape=(), dtype=float32)

20190326

名字 (name)

“node:src_output” : node 节点名称, src_output 来自节点的第几个输出

形状 (shape)

张量的维度信息, shape=() , 表示是标量

类型 (type)

每一个张量会有一个唯一的类型

TensorFlow会对参与运算的所有张量进行类型的检查,发现类型不匹配时会报错

张量的阶

张量的阶 (rank) 表征了张量的维度

阶	数学实体	代码示例
0	Scalar	Scalar = 20190926
1	Vector	Vector = [2,8,3]
2	Matrix	Matrix = [[4,2,1],[5,3,2],[5,5,6]]
3	3-tensor	Tensor = [[[4],[3],[2]],[[6],[100],[4]],[[5],[1],[4]]]
n	N-tensor	...

获取张量元素

阶为1的张量等价于向量；

阶为2的张量等价于矩阵，通过 $t[i, j]$ 获取元素；

阶为3的张量，通过 $t[i, j, k]$ 获取元素；

例：

```
In [13]: import tensorflow as tf  
tens1 = tf.constant([[1,2],[2,3]],[[3,4],[5,6]])  
  
sess = tf.Session()  
print(sess.run(tens1)[1,1,0])  
sess.close()
```

5

下标从 0 开始

张量的形状

三个术语描述张量的维度：阶（rank）、形状（shape）、维数（dimension number）

阶	形状	维数	例子
0	()	0-D	20190326 4
1	(D0)	1-D	[2,3,5]
2	(D0,D1)	2-D	[[2,3],[3,4]]
3	(D0,D1,D2)	3-D	[[[7],[3]],[[2],[4]]]
N	(D0,D1,...,Dn-1)	n-D	形为(D0,D1,...,Dn-1)的张量

张量的形状

```
In [12]: import tensorflow as tf

scalar = tf.constant(100)
vector = tf.constant([1, 2, 3, 4, 5])
matrix = tf.constant([[1, 2, 3], [4, 5, 6]])
cube_matrix = tf.constant([[[1], [2], [3], [4], [5], [6]], [[7], [8], [9]]])

print(scalar.get_shape())
print(vector.get_shape())
print(matrix.get_shape())
print(cube_matrix.get_shape())

()
(5,)
(2, 3)
(3, 3, 1)
```

张量的数据类型

数据类型 (dtype)	描 述
tf.float32	32 位浮点型
tf.float64	64 位浮点型
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.string	字符串 (作为非 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
tf.qint8	8 位有符号整数 (用于量化 Op)
tf.qint32	32 位有符号整数 (用于量化 Op)
tf.quint8	8 位无符号整数 (用于量化 Op)

默认类型：

不带小数点的数会被默认为 int32

带小数点的会被默认为 float32

张量的数据类型

```
In [14]: import tensorflow as tf  
a = tf.constant([1, 2], name="a")  
b = tf.constant([2.0, 3.0], name="b")  
result = a + b
```

```
-----  
ValueError                                20190326  
<ipython-input-14-ca86e03340c0> in <module>()
```

运行报错：

```
ValueError: Tensor conversion requested dtype int32 for Tensor with dtype  
float32: 'Tensor("b_3:0", shape=(2,), dtype=float32)'
```

TensorFlow会对参与运算的所有张量进行类型的检查，发现类型不匹配时会报错

Operation 操作

- 操作Op: 计算图中的节点，对Tensor对象执行运算的节点
- 创建Op，需要在Python中调用其构造方法。调用时需要传入计算所需的Tensor参数（称为输入）以及为正确创建Op的任何附加信息（属性）
- 构造方法将返回一个指向所创建Op的输出（0个或多个Tensor对象）的句柄

20190326

```
#一个简单计算图
node1 = tf.constant(3.0, tf.float32, name="node1")
node2 = tf.constant(4.0, tf.float32, name="node2")
#变量node3wei为指向Op输出的Tensor对象的句柄
node3 = tf.add(node1, node2, name="add_op")
```

- 无输入输出的运算：并非所有节点都需要与其它节点连接

Operation 操作

```
import tensorflow as tf

# 本例用到了TensorBoard, 具体使用后面讲解

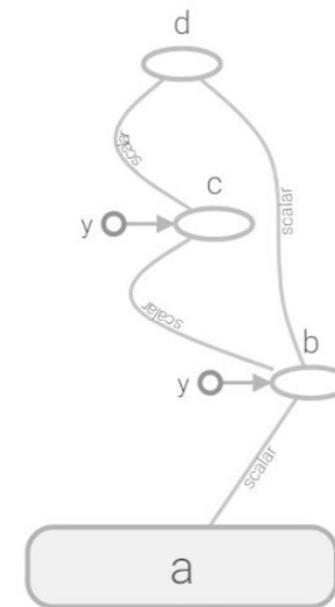
tf.reset_default_graph() #清除default graph和不断增加的节点

# 定义变量 a
a = tf.Variable(1, name="a")
# 定义操作b为a+1
b = tf.add(a, 1, name="b")
# 定义操作c为b*4
c = tf.multiply(b, 4, name="c")
# 定义 d为c-b
d = tf.subtract(c, b, name="d")

# Logdir改为自己机器上的合适路径
logdir='D:/log'

#生成一个写日志的writer, 并将当前的TensorFlow计算图写入日志。
writer = tf.summary.FileWriter(logdir,tf.get_default_graph())
writer.close()
```

20190326



23

张量的一些常见操作

Operation	Description
tf.shape	To find a shape of a <i>tensor</i>
tf.size	To find the size of a <i>tensor</i>
tf.rank	To find a rank of a * <i>tensor</i> *
tf.reshape	To change the shape of a <i>tensor</i> keeping the same elements contained
tf.squeeze	To delete in a <i>tensor</i> dimensions of size 1
tf.expand_dims	To insert a dimension to a * <i>tensor</i> *
tf.slice	To remove a portions of a <i>tensor</i>
tf.split	To divide a <i>tensor</i> into several tensors along one dimension
tf.tile	To create a new <i>tensor</i> replicating a <i>tensor</i> multiple times
tf.concat	To concatenate <i>tensors</i> in one dimension
tf.reverse	To reverse a specific dimension of a <i>tensor</i>
tf.transpose	To transpose dimensions in a <i>tensor</i>
tf.gather	To collect portions according to an index

20190326

Constant 常量

在运行过程中值不会改变的单元，在TensorFlow中无须进行初始化操作

创建语句：

```
constant_ name = tf.constant(value)
```

20190326

```
In [2]: a = tf.constant(1.0, name='a')
b = tf.constant(2.5, name='b')
c = tf.add(a, b, name='c')

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

3.5

Variable 变量

在运行过程中值会改变的单元，在TensorFlow中须进行初始化操作

创建语句：

```
name_variable = tf.Variable(value, name)
```

20190326

注意V是大写字母

个别变量初始化：

```
init_op = name_variable.initializer()
```

所有变量初始化：

```
init_op = tf.global_variables_initializer()
```

Variable 变量

```
In [3]: node1 = tf.Variable(3.0,tf.float32,name="node1")
node2 = tf.Variable(4.0,tf.float32,name="node2")
result = tf.add(node1, node2, name='add')

sess = tf.Session()
#变量初始化
init = tf.global_variables_initializer()
sess.run(init)

print(sess.run(result))
```

20190326

7.0

增加了一个init初始化变量，并调用会话的run命令对参数进行初始化

Variable 变量

```
In [4]: node1 = tf.Variable(3.0,tf.float32,name="node1")
node2 = tf.Variable(4.0,tf.float32,name="node2")
result = tf.add(node1, node2, name='add')

sess = tf.Session()

#变量初始化
init = tf.global_variables_initializer()
print(sess.run(result))
```

20190326

```
-----
FailedPreconditionError                         Traceback (most recent call last)
C:\Users\mingh\Anaconda3\lib\site-packages\tensorflow\python\client\session.py in _do_call(self, fn, *args)
    1138     try:
-> 1139         return fn(*args)
    1140     except errors.OpError as e:
```

使用了Variable变量类型，不进行初始化数值会出现运行错误

变量赋值

- 通常将一个统计模型中的参数表示为一组变量。
- 自动训练机器学习模型的Optimizer类，能够自动修改Variable对象的值，而无需显式做出请求。
20190326
- 如果要求Graph对象中的一些Variable对象只能手工修改，而不允许使用Optimizer类时，可在创建Variable对象时将其trainable参数设为False：
`not_trainable = tf.Variable(0, trainable=False)`
- 对于迭代计数器或其他任何不涉及机器学习模型参数计算的variable对象，通常都需要上述设置

变量赋值

```
In [7]: # 通过变量赋值输出1、2、3...10
```

```
import tensorflow as tf

value = tf.Variable(0, name="value")
one = tf.constant(1)          20190326
new_value = tf.add(value, one) 1
update_value = tf.assign(value, new_value) 2
init = tf.global_variables_initializer() 3
with tf.Session() as sess: 4
    sess.run(init) 5
    for _ in range(10): 6
        sess.run(update_value) 7
        print(sess.run(value)) 8
    9
10
```

RR

Placeholder 占位符

- TensorFlow中的Variable变量类型，在定义时需要初始化，但有些变量定义时并不知道其数值，只有当真正开始运行程序时，才由外部输入，比如训练数据，这时候需要用到占位符
- tf.placeholder 占位符，是TensorFlow中特有的一种数据结构，类似动态变量，函数的参数、或者C语言或者Python语言中格式化输出时的“%”占位符

Placeholder 占位符

- TensorFlow占位符Placeholder，先定义一种数据，其参数为数据的Type和Shape

占位符Placeholder的函数接口如下：

20190326
tf.placeholder(dtype, shape=None, name=None)

```
x = tf.placeholder(tf.float32, [2, 3], name='tx')  
# 此代码生成一个2x3的二维数组，矩阵中每个元素的类型都是tf.float32，内部对应的符号名称是tx
```

feed_dict参数传递数据

如果构建了一个包含placeholder操作的计算图，当在session中调用run方法时，placeholder占用的变量必须通过**feed_dict**参数传递进去，否则报

```
In [1]: import tensorflow as tf  
  
a = tf.placeholder(tf.float32, name='a') 20190326  
b = tf.placeholder(tf.float32, name='b')  
c = tf.multiply(a, b, name='c')  
  
init = tf.global_variables_initializer()  
  
with tf.Session() as sess:  
    sess.run(init)  
    # 通过feed_dict的参数传值，按字典格式  
    result = sess.run(c, feed_dict={a:8.0, b:3.5})  
  
    print(result)
```

28.0

feed_dict参数传递数据

多个操作可以通过一次Feed完成执行

```
In [2]: import tensorflow as tf

a = tf.placeholder(tf.float32, name='a')
b = tf.placeholder(tf.float32, name='b')
c = tf.multiply(a, b, name='c')
d = tf.subtract(a, b, name='d')

init = tf.global_variables_initializer()

with tf.Session() as sess:
    sess.run(init)

    result = sess.run([c,d], feed_dict={a:[8.0,2.0,3.5], b:[1.5,2.0,4.]})

    print(result)
    # 取结果中的第一个
    print(result[0])

[array([ 12.,   4.,  14.], dtype=float32), array([ 6.5,   0. , -0.5], dtype=float32)]
[ 12.   4.  14.]
```

20190326

fetches参数返回多个值

一次返回多个值分别赋给多个变量

```
In [6]: import tensorflow as tf

a = tf.placeholder(tf.float32, name='a')
b = tf.placeholder(tf.float32, name='b')
c = tf.multiply(a, b, name='c')          20190326
d = tf.subtract(a, b, name='d')

init = tf.global_variables_initializer()

with tf.Session() as sess:
    sess.run(init)

#返回的两个值分别赋给两个变量
rc,rd = sess.run([c,d], feed_dict={a:[8.0,2.0,3.5], b:[1.5,2.0,4.]})

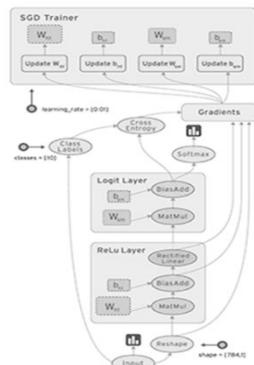
print("value of c=",rc,"value of d=",rd)

value of c= [ 12.   4.  14.] value of d= [ 6.5   0.  -0.5]
```

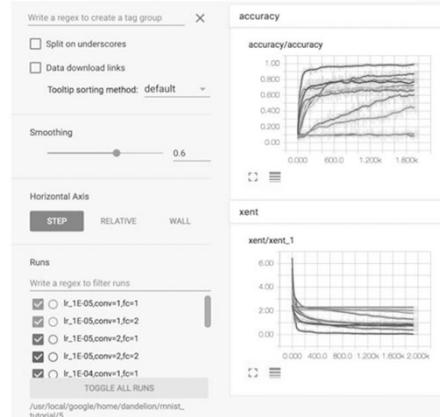
TensorBoard

- TensorBoard是TensorFlow的可视化工具
- 通过TensorFlow程序运行过程中输出的日志文件可视化TensorFlow程序的运行状态
- TensorBoard和TensorFlow程序跑在不同的进程中

20190326



计算图可视化



度量可视化

TensorBoard 应用

```
In [7]: import tensorflow as tf

#清除default graph和不断增加的节点
tf.reset_default_graph()

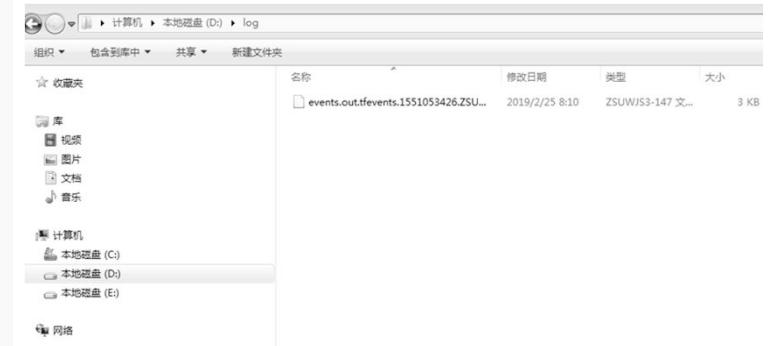
# Logdir改为自己机器上的合适路径
logdir='D:/log'

#定义一个简单的计算图，实现向量加法的操作
input1 = tf.constant([1.0, 2.0, 3.0], name="input1")
input2 = tf.Variable(tf.random_uniform([3]), name="input2")
output = tf.add_n([input1, input2], name="add")

#生成一个写日志的writer，并将当前的TensorFlow计算图写入日志。
writer = tf.summary.FileWriter(logdir,tf.get_default_graph())
writer.close()
```

20190326

- 运行后指定目录产生日志文件



启动TensorBoard

- 进入日志存放的目录（非常重要）
- 执行启动tensorboard命令，并指定日志输出目录

tensorboard --logdir= path\to\logs

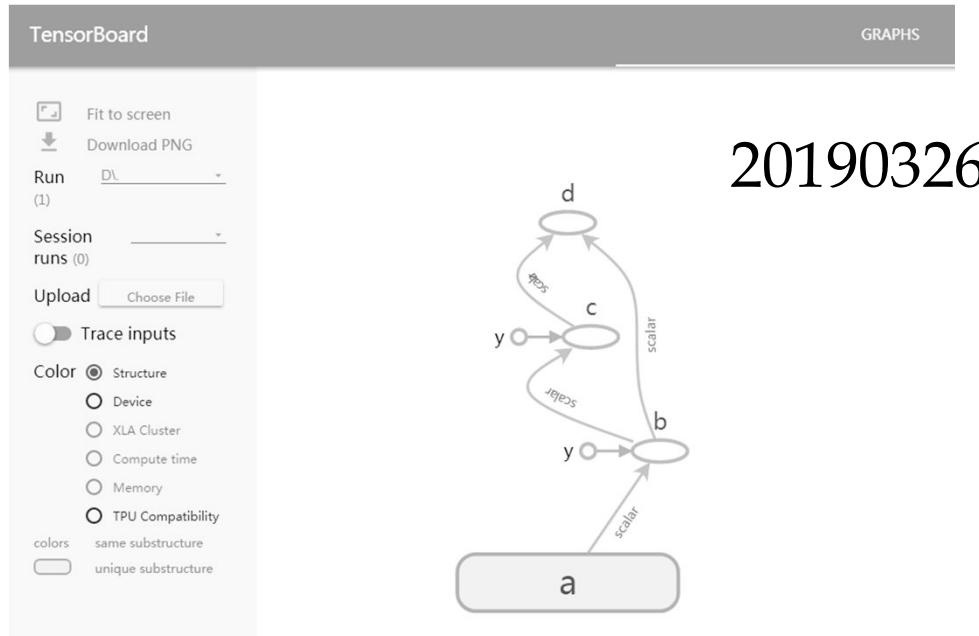
```
管理员: C:\Windows\System32\cmd.exe - tensorboard --logdir=D:\log
Microsoft Windows [版本 6.1.7601]
版权所有 <c> 2009 Microsoft Corporation。保留所有权利。

D:\log>tensorboard --logdir=D:\log
W0225 08:14:14.585541 Reloader tf_logging.py:86] Found more than one graph event
per run, or there was a metagraph containing a graph_def, as well as one or more
graph events. Overwriting the graph with the newest event.
W0225 08:14:14.586541 Reloader tf_logging.py:86] Found more than one metagraph event
per run. Overwriting the metagraph with the newest event.
TensorBoard 0.4.0rc3 at http://ZSUWJS3-147:6006 <Press CTRL+C to quit>
```

默认下，Tensorboard服务器启动后会自动监听端口6006，使用—port参数可改变启动服务端口

访问TensorBoard

□ 打开浏览器并在地址栏输入`http://localhost:6006`



TensorBoard 常用 API

API	描述
tf.summary.FileWriter()	创建FileWriter和事件文件，会在logdir中创建一个新的事件文件
tf.summary.FileWriter.add_summary()	将摘要添加到事件文件
tf.summary.FileWriter.add_event()	向事件文件添加一个事件
tf.summary.FileWriter.add_graph()	向事件文件添加一个图 20190326
tf.summary.FileWriter.get_logdir()	获取事件文件的路径
tf.summary.FileWriter.flush()	将所有事件都写入磁盘
tf.summary.FileWriter.close()	将事件写入磁盘，并关闭文件操作符
tf.summary.scalar()	输出包含单个标量值的摘要
tf.summary.histogram()	输出包含直方图的摘要
tf.summary.audio()	输出包含音频的摘要
tf.summary.image()	输出包含图片的摘要
tf.summary.merge()	合并摘要，包含所有输入摘要的值

Name Scope 名称作用域

```
import tensorflow as tf

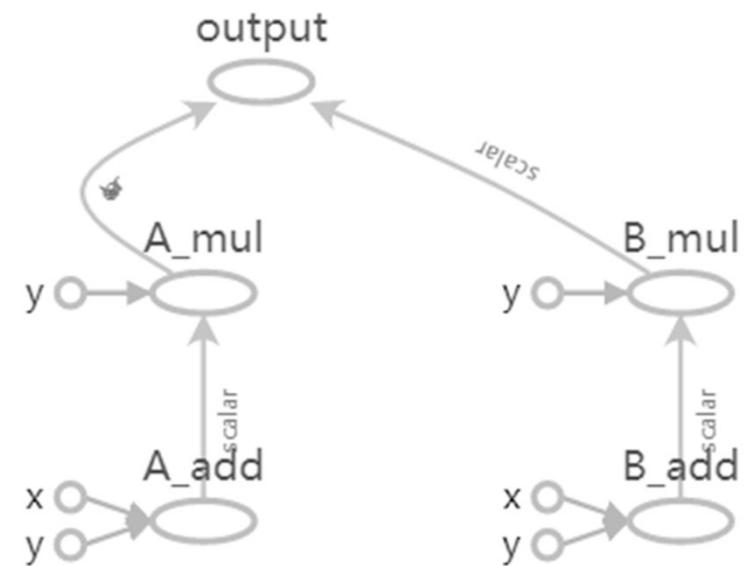
# 清除default_graph和不断增加的节点
tf.reset_default_graph()

a = tf.add(1, 2, name="A_add")
b = tf.multiply(a, 3, name="A_mul")
c = tf.add(4, 5, name="B_add")
d = tf.multiply(c, 6, name="B_mul")
e = tf.add(b, d, name="output")

# logdir改为自己机上的合法路径
logdir = 'D:improved_graph'

# 用到TensorBoard可视化
# 生成一个写日志的writer并将当前的TensorFlow计算图写入日志
writer = tf.summary.FileWriter(logdir, graph=tf.get_default_graph())
writer.close()
```

20190326



Name Scope名称作用域

- 通过名称作用域组织数据流图。名称作用域的基本用法是将Op添加到with tf.name_scope (<name>) 语句块中

```
# Name scope命名域的应用
import tensorflow as tf

# 清除defuse_graph和不断增加的节点
tf.reset_default_graph()

with tf.name_scope("scope_A"):
    a = tf.add(1, 2, name="A_add")
    b = tf.multiply(a, 3, name="A_mul")

with tf.name_scope("scope_A"):
    c = tf.add(4, 5, name="B_add")
    d = tf.multiply(c, 6, name="B_mul")
    e = tf.add(b, d, name="output")

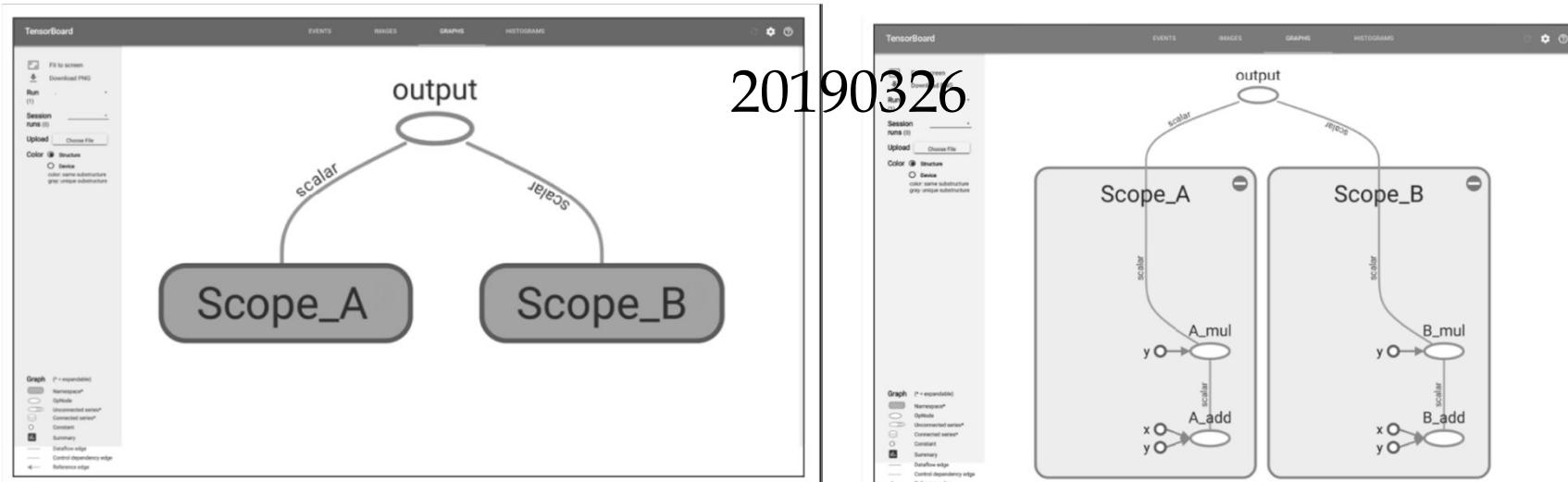
# logdir改为自己的合法路径
logdir = 'D:improved_graph'

# 用到TensorBoard可视化
# 生成一个写日志的writer并将当前的TensorFlow计算图写入日志
writer = tf.summary.FileWriter(logdir, graph=tf.get_default_graph())
writer.close()
```

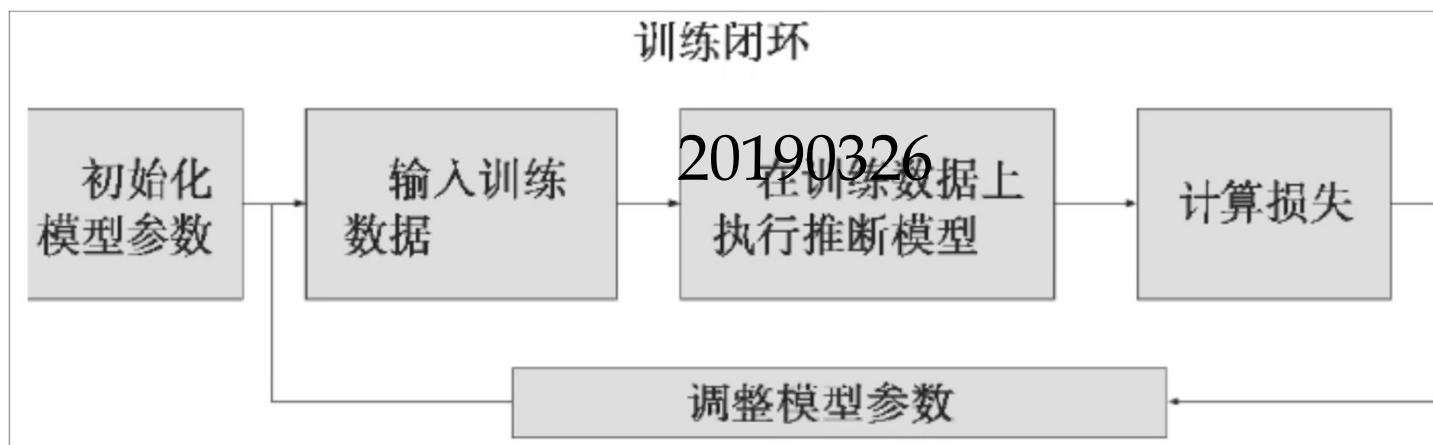
20190326

Name Scope

□ >tensorboard --logdir=D:\improved_graph



有监督机器学习过程



Tf 进行Iris分类

```
In [5]: # 导入相关库
import tensorflow as tf
import numpy as np

#直接从sklearn自带的datasets中导入iris
from sklearn.datasets import load_iris
iris = load_iris()

# 查看数据集
# print(iris.target)
# print(iris.feature_names)
# print(iris.data)
# print (iris.data.shape)

# 将数据集按照3:1分为训练集和测试集，每次分割的结果不同
from sklearn.model_selection import train_test_split
Xtrain, Xtest, Ytrain, Ytest = train_test_split(iris.data, iris.target, test_size = 0.25)

# 查看分割后的数据集
# print(Xtrain)
# print(Ytrain)
# print(Xtest)
# print(Ytest)
```

20190326

Tf 进行Iris分类

```
# 定义占位符: 用于传入输入数据, 形如[ 5.4  3.2  4.5  1.5]
x_train = tf.placeholder("float", [None, 4])
x_test = tf.placeholder("float", [4])

# 计算距离:
distance = tf.reduce_sum(tf.abs(tf.add(x_train, tf.negative(x_test)))), axis=1)
20190326

# 获得距离最小的index
pred = tf.argmin(distance, 0)

# 定义正确率标量
accuracy = 0

# 初始化变量
# init = tf.global_variables_initializer()
```

Tf进行Iris分类

```
with tf.Session() as sess:  
    #sess.run(init)  
    m = len(Xtest)  
    for i in range(m):  
        # index[0]为最小值所在的索引, index[1]为所有距离大小的列表,  
        index = sess.run([pred, distance], feed_dict={x_train: Xtrain, x_test: Xtest[i,:]})  
  
        # 预测值  
        pred_label = Ytrain[index[0]]  
  
        # 真值  
        true_label = Ytest[i]  
  
        # 计算预测正确的样本数  
        if pred_label == true_label:  
            accuracy += 1  
        print("test", i, "predict label:", pred_label, "true label:", true_label)  
  
    print("accuracy:", accuracy / m)
```

20190326

```
test 20 predict label: 0 true label: 0  
test 21 predict label: 1 true label: 1  
test 22 predict label: 0 true label: 0  
test 23 predict label: 2 true label: 2  
test 24 predict label: 0 true label: 0  
test 25 predict label: 2 true label: 2  
test 26 predict label: 2 true label: 2  
test 27 predict label: 0 true label: 0  
test 28 predict label: 1 true label: 1  
test 29 predict label: 2 true label: 2  
test 30 predict label: 2 true label: 2  
test 31 predict label: 0 true label: 0  
test 32 predict label: 2 true label: 2  
test 33 predict label: 1 true label: 1  
test 34 predict label: 1 true label: 1  
test 35 predict label: 0 true label: 0  
test 36 predict label: 2 true label: 2  
test 37 predict label: 0 true label: 0  
done  
accuracy: 0.9473684210526315
```

Tensorflow 原理

- 使用图(graph)来表示任务
- 图在会话(Session)中执行
- 使用tensor表示数据 20190326
- 通过变量(Variable)维护状态
- 使用feed和fetch可以为任意操作(arbitrary operation)赋值

或者从其中获取数据

重要连接

- TensorFlow官网: <https://tensorflow.google.cn/>
- TensorFlow Github: <https://tensorflow.google.cn/hub/>
- TensorFlow中文社区: <http://www.tensorfly.cn/>
- 机器学习速成课程: <https://developers.google.cn/machine-learning/crash-course/>