# Tensorflow函数功能笔记

以下用tensorflow的实例tf

# tf.get\_variable

tensorflow中tf.get\_variable的API为

def get\_variable(name,

shape=None,

dtype=None,

initializer=None,

regularizer=None,

trainable=True,

collections=None,

caching\_device=None,

partitioner=None,

validate\_shape=True,

use\_resource=None,

custom\_getter=None):

常用的参数有：

name：变量名称

shape：变量维度

initializer：变量初始化方式

regularizer：正规化

caching\_device：可选的设备字符串或函数描述

其中，变量的初始化方式有

1. tf.constant\_initializer–常量初始化

#!/usr/bin/python

# coding:utf-8

import tensorflow as tf

# 默认值为0

v1 = tf.get\_variable('v1', shape=[5], initializer=tf.constant\_initializer())

# 也可以指定初始化值

v2 = tf.get\_variable('v2', shape=[5], initializer=tf.constant\_initializer(9))

with tf.Session() as sess:

sess.run(tf.global\_variables\_initializer())

print v1.eval()

print v2.eval()

---------------------

输出：

[ 0. 0. 0. 0. 0.]

[ 9. 9. 9. 9. 9.]

1. tf.random\_normal\_initializer–正太分布初始化

#!/usr/bin/python

# coding:utf-8

import tensorflow as tf

# 随机正太分布初始化

v1=tf.get\_variable('v1',shape=[5],initializer=tf.random\_normal\_initializer())

# 指定正太分布的均值方差种子值

v2=tf.get\_variable('v2',shape=[5],initializer=tf.random\_normal\_initializer(mean=0, stddev=1., seed=1))

with tf.Session() as sess:

sess.run(tf.global\_variables\_initializer())

print v1.eval()

print v2.eval()

输出：

[-0.96925646 -1.34411633 1.27495158 0.10656819 -0.01502592]

[-0.81131822 1.48459876 0.06532937 -2.4427042 0.0992484 ]

疑问：tf.random\_normal\_initializer的seed参数取不同的值有什么区别

1. tf.truncated\_normal\_initializer–截取正太分布

#!/usr/bin/python

# coding:utf-8

import tensorflow as tf

# 随机截取正太分布

v1=tf.get\_variable('v1',shape=[5],initializer=tf.truncated\_normal\_initializer())

# 指定截取正太分布的均值方差种子值

v2=tf.get\_variable('v2',shape=[5],initializer=tf.truncated\_normal\_initializer(mean=0, stddev=1., seed=1))

with tf.Session() as sess:

sess.run(tf.global\_variables\_initializer())

print v1.eval()

print v2.eval()

输出：

[ 0.33786374 -0.9326936 -0.27475813 0.07207271 1.80768931]

[-0.81131822 1.48459876 0.06532937 0.0992484 0.63969707]

1. tf.random\_uniform\_initializer–均匀分布

#!/usr/bin/python

# coding:utf-8

import tensorflow as tf

# 均匀分布

v1=tf.get\_variable('v1',shape=[5],initializer=tf.random\_uniform\_initializer())

# 指定均匀分布的上下限和种子值

v2=tf.get\_variable('v2',shape=[5],initializer=tf.random\_uniform\_initializer(maxval=-1., minval=1., seed=0))

with tf.Session() as sess:

sess.run(tf.global\_variables\_initializer())

print v1.eval()

print v2.eval()

输出：

[ 0.62208343 0.81940019 0.67432535 0.07955837 0.59436822]

[ 0.79827476 -0.9403336 -0.69752836 0.90343738 0.90295386]

1. tf.zeros\_initializer–全0初始化

tf.ones\_initializer–全1初始化

#!/usr/bin/python

# coding:utf-8

import tensorflow as tf

# 全0

v1 = tf.get\_variable('v1', shape=[5], initializer=tf.zeros\_initializer())

# 全1

v2 = tf.get\_variable('v2', shape=[5], initializer=tf.ones\_initializer())

with tf.Session() as sess:

sess.run(tf.global\_variables\_initializer())

print v1.eval()

print v2.eval()

输出：

[ 0. 0. 0. 0. 0.]

[ 1. 1. 1. 1. 1.]

---------------------

作者：阿卡蒂奥

来源：CSDN

原文：https://blog.csdn.net/akadiao/article/details/78517154

版权声明：本文为博主原创文章，转载请附上博文链接！

# sess.run(tf.global\_variables\_initializer())

sess.run(tf.global\_variables\_initializer()) 就是 run了 所有global Variable 的 assign op，这就是初始化参数的本来面目。

# tensorflow里面的op概念

TF中的op代表了对“操作”的抽象，说它抽象是因为，op仅仅说明了操作是做什么用的，但没有说明具体怎么做。举个例子，MatMul是一个操作，它表示了矩阵乘法，但并不包含矩阵乘法的具体实现，因为我们知道，在CPU和GPU上，矩阵乘法的高效率实现是完全不同的。为了把不同设备上的实现细节隐藏起来，为相同的计算提供统一的对外表示，TF提出了op的概念。

# tensorflow里面的张量

TensorFlow用张量这种数据结构来表示所有的数据.你可以把一个张量想象成一个n维的数组或列表.一个张量有一个静态类型和动态类型的维数.张量可以在图中的节点之间流通.

阶

在TensorFlow系统中，张量的维数来被描述为阶.但是张量的阶和矩阵的阶并不是同一个概念.张量的阶（有时是关于如顺序或度数或者是n维）是张量维数的一个数量描述.比如，下面的张量（使用Python中list定义的）就是2阶.

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

你可以认为一个二阶张量就是我们平常所说的矩阵，一阶张量可以认为是一个向量.对于一个二阶张量你可以用语句t[i, j]来访问其中的任何元素.而对于三阶张量你可以用't[i, j, k]'来访问其中的任何元素.

阶 数学实例 Python 例子

0 纯量 (只有大小) s = 483

1 向量(大小和方向) v = [1.1, 2.2, 3.3]

2 矩阵(数据表) m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

3 3阶张量 (数据立体) t = [[[2], [4], [6]], [[8], [10], [12]], [[14], [16], [18]]]

n n阶 (自己想想看) ....

张量是所有深度学习框架中最核心的组件，因为后续的所有运算和优化算法都是基于张量进行的。几何代数中定义的张量是基于向量和矩阵的推广，通俗一点理解的话，我们可以将标量视为零阶张量，矢量视为一阶张量，那么矩阵就是二阶张量。

---------------------

作者：pandamax

来源：CSDN

原文：https://blog.csdn.net/pandamax/article/details/63684633

版权声明：本文为博主原创文章，转载请附上博文链接！

# 5 tf.Graph()

tf.Graph() 函数非常重要，注意提现在两个方面

1. 它可以通过tensorboard用图形化界面展示出来流程结构

2. 它可以整合一段代码为一个整体存在于一个图中

声明情况大体有三种

1. tensor:通过张量本身直接出graph

# -\*- coding: utf-8 -\*-

import tensorflow as tf

c = tf.constant(4.0)

sess = tf.Session()

sess.run(tf.global\_variables\_initializer())

c\_out = sess.run(c)

print(c\_out)

print(c.graph == tf.get\_default\_graph())

print(c.graph)

print(tf.get\_default\_graph())

输出

4.0

True

<tensorflow.python.framework.ops.Graph object at 0x7f382f9ef110>

<tensorflow.python.framework.ops.Graph object at 0x7f382f9ef110>

2.通过声明一个默认的，然后定义张量内容，在后面可以调用或保存

# -\*- coding: utf-8 -\*-

import tensorflow as tf

g = tf.Graph()

with g.as\_default():

c = tf.constant(4.0)

sess = tf.Session(graph=g)

c\_out = sess.run(c)

print(c\_out)

print(g)

print(tf.get\_default\_graph())

输出

4.0

<tensorflow.python.framework.ops.Graph object at 0x7f65f1cb2fd0>

<tensorflow.python.framework.ops.Graph object at 0x7f65de447c90>

3.通过多个声明，在后面通过变量名来分别调用

# -\*- coding: utf-8 -\*-

import tensorflow as tf

g1 = tf.Graph()

with g1.as\_default():

c1 = tf.constant(4.0)

g2 = tf.Graph()

with g2.as\_default():

c2 = tf.constant(20.0)

with tf.Session(graph=g1) as sess1:

print(sess1.run(c1))

with tf.Session(graph=g2) as sess2:

print(sess2.run(c2))

输出

4.0

20.0

对graph的操作大体有三种

1.保存

# -\*- coding: utf-8 -\*-

import tensorflow as tf

g1 = tf.Graph()

with g1.as\_default():

# 需要加上名称，在读取pb文件的时候，是通过name和下标来取得对应的tensor的

c1 = tf.constant(4.0, name='c1')

g2 = tf.Graph()

with g2.as\_default():

c2 = tf.constant(20.0)

with tf.Session(graph=g1) as sess1:

print(sess1.run(c1))

with tf.Session(graph=g2) as sess2:

print(sess2.run(c2))

# g1的图定义，包含pb的path, pb文件名，是否是文本默认False

tf.train.write\_graph(g1.as\_graph\_def(),'.','graph.pb',False)

输出

4.0

20.0

并且在当前文件夹下面生成graph.pb文件

2.从pb文件中调用

# -\*- coding: utf-8 -\*-

import tensorflow as tf

from tensorflow.python.platform import gfile

#load graph

with gfile.FastGFile("./graph.pb",'rb') as f:

graph\_def = tf.GraphDef()

graph\_def.ParseFromString(f.read())

tf.import\_graph\_def(graph\_def, name='')

sess = tf.Session()

c1\_tensor = sess.graph.get\_tensor\_by\_name("c1:0")

c1 = sess.run(c1\_tensor)

print(c1)

输出

4.0

3.穿插调用

# -\*- coding: utf-8 -\*-

import tensorflow as tf

g1 = tf.Graph()

with g1.as\_default():

# 声明的变量有名称是一个好的习惯，方便以后使用

c1 = tf.constant(4.0, name="c1")

g2 = tf.Graph()

with g2.as\_default():

c2 = tf.constant(20.0, name="c2")

with tf.Session(graph=g2) as sess1:

# 通过名称和下标来得到相应的值

c1\_list = tf.import\_graph\_def(g1.as\_graph\_def(), return\_elements = ["c1:0"], name = '')

print(sess1.run(c1\_list[0]+c2))

输出

24.0

---------------------

作者：zj360202

来源：CSDN

原文：https://blog.csdn.net/zj360202/article/details/78539464

版权声明：本文为博主原创文章，转载请附上博文链接！

# eval()函数

eval() 其实就是tf.Tensor的Session.run() 的另外一种写法，但两者有差别

eval(): 将字符串string对象转化为有效的表达式参与求值运算返回计算结果

eval()也是启动计算的一种方式。基于Tensorflow的基本原理，首先需要定义图，然后计算图，其中计算图的函数常见的有run()函数，如sess.run()。同样eval()也是此类函数，

要注意的是，eval()只能用于tf.Tensor类对象，也就是有输出的Operation。对于没有输出的Operation, 可以用.run()或者Session.run()；Session.run()没有这个限制。

Tensor.run和Tensor.eval的区别

在会话中需要运行节点，会碰到两种方式：Session.run()和Tensor.eval()

解释一



如果t是一个tf.Tensor对象，则tf.Tensor.eval是tf.Session.run的缩写（其中sess是当前的tf.get\_default\_session。下面的两个代码片段是等价的：

在第二个示例中，会话充当上下文管理器，其作用是将其安装为with块的生命周期的默认会话。 上下文管理器方法可以为简单用例（比如单元测试）提供更简洁的代码; 如果您的代码处理多个图形和会话，则可以更直接地对Session.run（）进行显式调用。

解释二

如果你有一个Tensor t，在使用t.eval()时，等价于：tf.get\_default\_session().run(t).

举例：

t = tf.constant(42.0)

sess = tf.Session()

with sess.as\_default(): # or `with sess:` to close on exit

assert sess is tf.get\_default\_session()

assert t.eval() == sess.run(t)

这其中最主要的区别就在于你可以使用sess.run()在同一步获取多个tensor中的值，

例如：

t = tf.constant(42.0)

u = tf.constant(37.0)

tu = tf.mul(t, u)

ut = tf.mul(u, t)

with sess.as\_default():

tu.eval() # runs one step

ut.eval() # runs one step

sess.run([tu, ut]) # evaluates both tensors in a single step

注意到：每次使用 eval 和 run时，都会执行整个计算图，为了获取计算的结果，将它分配给tf.Variable，然后获取。

---------------------

作者：呆呆的猫

来源：CSDN

原文：https://blog.csdn.net/jiaoyangwm/article/details/79248535

版权声明：本文为博主原创文章，转载请附上博文链接！

# tensorflow函数：tf.Session()和 tf.Session().as\_default()的区别

 tf.Session()：创建一个会话

 tf.Session().as\_default()：创建一个默认会话

那么问题来了，会话和默认会话有什么区别呢？TensorFlow会自动生成一个默认的计算图，如果没有特殊指定，运算会自动加入这个计算图中。TensorFlow中的会话也有类似的机制，但是TensorFlow不会自动生成默认的会话，而是需要手动指定。

 tf.Session()创建一个会话，当上下文管理器退出时会话关闭和资源释放自动完成。

tf.Session().as\_default()创建一个默认会话，当上下文管理器退出时会话没有关闭，还可以通过调用会话进行run()和eval()操作，代码示例如下：

 tf.Session()代码示例：

import tensorflow as tf

a = tf.constant(1.0)

b = tf.constant(2.0)

with tf.Session() as sess:

print(a.eval())

print(b.eval(session=sess))

运行结果如下：

1.0

RuntimeError: Attempted to use a closed Session.

 在打印张量b的值时报错，报错为尝试使用一个已经关闭的会话。

 tf.Session().as\_default()代码示例：

import tensorflow as tf

a = tf.constant(1.0)

b = tf.constant(2.0)

with tf.Session().as\_default() as sess:

print(a.eval())

print(b.eval(session=sess))

运行结果如下：

1.0

2.0

 对于run()方法也是一样，如果想让默认会话在退出上下文管理器时关闭会话，可以调用sess.close()方法。

代码示例如下：

import tensorflow as tf

a = tf.constant(1.0)

b = tf.constant(2.0)

with tf.Session().as\_default() as sess:

print(a.eval())

sess.close()

print(b.eval(session=sess))

 运行结果如下：

1.0

RuntimeError: Attempted to use a closed Session.

---------------------

作者：Enchanted\_ZhouH

来源：CSDN

原文：https://blog.csdn.net/Enchanted\_ZhouH/article/details/77571939

版权声明：本文为博主原创文章，转载请附上博文链接！

# tf.name\_scope, tf.variable\_scope

1.tf.name\_scope('scope\_name')或tf.name\_scope(named\_scope)

主要与tf.Variable搭配使用；

当传入字符串时，用以给变量名添加前缀，类似于目录，如case1所示；

当传入已存在的name\_scope对象时，则其范围内变量的前缀只与当前传入的对象有关，与更上层的name\_scope无关，如case2所示。

import tensorflow as tf

# case 1:

with tf.name\_scope('l1'):

with tf.name\_scope('l2'):

wgt1 = tf.Variable([1,2,3], name='wgts')

bias1 = tf.Variable([0.1], name='biases')

print wgt1.name, bias1.name

(输出结果)# >>> l1/l2/wgts:0 l1/l2/biases:0

# case 2:

with tf.name\_scope('l1') as l1\_scp:

with tf.name\_scope('l2'):

wgt0 = tf.Variable([1,2,3], name='wgts')

bias0 = tf.Variable([0.1], name='biases')

with tf.name\_scope(l1\_scp):

wgt1 = tf.Variable([1,2,3], name='wgts')

bias1 = tf.Variable([0.1], name='biases')

print wgt0.name, bias0.name, wgt1.name, bias1.name

（输出结果）# >>> l1\_1/l2/wgts:0 l1\_1/l2/biases:0 l1\_1/wgts:0 l1\_1/biases:0

2. tf.variable\_scope('scope\_name', reuse=None)或

    tf.variable\_scope(named\_scope)

与name\_scope一样：当传入字符串时，用以给变量名添加前缀，类似于目录；

当传入已存在的variable\_scope对象时，则其范围内变量的前缀只与当前传入的对象有关，与更上层的variable\_scope无关。

常于get\_variable搭配使用，多用于变量共享；其中 reuse 参数可设为 None、tf.AUTO\_REUSE、True、False；

当 reuse=None（默认情况）时，与上层variable\_scope的reuse参数一样。

# case 1

with tf.variable\_scope('lv1'):

with tf.variable\_scope('lv2'):

init = tf.constant\_initializer(0.1)

wgt1 = tf.get\_variable('wgts', [2,2])

bias1 = tf.get\_variable('biases', [2,2])

print wgt1.name, bias1.name

（输出结果）# >>> lv1/lv2/wgts:0 lv1/lv2/biases:0

上面的代码是一个文件

当 reuse=tf.AUTO\_REUSE 时，自动复用，如果变量存在则复用，不存在则创建。这是最安全的用法。

with tf.variable\_scope('lv1'):

with tf.variable\_scope('lv2'):

init = tf.constant\_initializer(0.1)

wgt1 = tf.get\_variable('wgts', [2,2])

bias1 = tf.get\_variable('biases', [2,2])

print wgt1.name, bias1.name

（输出结果）# >>> lv1/lv2/wgts:0 lv1/lv2/biases:0

with tf.variable\_scope('lv1', reuse=tf.AUTO\_REUSE):

with tf.variable\_scope('lv2'):

init = tf.constant\_initializer(0.1)

wgt2 = tf.get\_variable('wgts', [2,2])

bias2 = tf.get\_variable('biases', [2,2])

print wgt2.name, bias2.name

（输出结果）# >>> lv1/lv2/wgts:0 lv1/lv2/biases:0

上面两段代码在一个文件里

with tf.variable\_scope('lv1', reuse=tf.AUTO\_REUSE):

with tf.variable\_scope('lv2'):

init = tf.constant\_initializer(0.1)

wgt2 = tf.get\_variable('wgts', [2,2])

bias2 = tf.get\_variable('biases', [2,2])

print wgt2.name, bias2.name

（输出结果）# >>> lv1/lv2/wgts:0 lv1/lv2/biases:0

上面一段程序是另一个文件

当 reuse=True 时，tf.get\_variable会查找该命名变量，如果没有找到，则会报错；所以设置reuse=True之前，要保证该命名变量已存在。

with tf.variable\_scope('lv1', reuse=True):

with tf.variable\_scope('lv2'):

init = tf.constant\_initializer(0.1)

wgt1 = tf.get\_variable('wgts', [2,2])

bias1 = tf.get\_variable('biases', [2,2])

print wgt1.name, bias1.name

(输出结果)# >>> ValueError: Variable lv1/lv2/wgts does not exist,

# or was not created with tf.get\_variable(). Did you mean

# to set reuse=tf.AUTO\_REUSE in VarScope?

上面一段代码是一个文件

命名变量已存在：

with tf.variable\_scope('lv1'):

with tf.variable\_scope('lv2'):

init = tf.constant\_initializer(0.1)

wgt1 = tf.get\_variable('wgts', [2,2])

bias1 = tf.get\_variable('biases', [2,2])

print wgt1.name, bias1.name

（输出结果）# >>> lv1/lv2/wgts:0 lv1/lv2/biases:0

# case 2

with tf.variable\_scope('lv1', reuse=True):

with tf.variable\_scope('lv2'):

init = tf.constant\_initializer(0.1)

wgt1 = tf.get\_variable('wgts', [2,2])

bias1 = tf.get\_variable('biases', [2,2])

print wgt1.name, bias1.name

（输出结果） # >>> lv1/lv2/wgts:0 lv1/lv2/biases:0

上面两段代码是一个文件

当 reuse=False 时，tf.get\_variable会调用tf.Variable来创建变量，并检查创建的变量是否以存在，如果已存在，则报错；

with tf.variable\_scope('lv1'):

with tf.variable\_scope('lv2'):

init = tf.constant\_initializer(0.1)

wgt1 = tf.get\_variable('wgts', [2,2])

bias1 = tf.get\_variable('biases', [2,2])

print wgt1.name, bias1.name

# >>> lv1/lv2/wgts:0 lv1/lv2/biases:0

# case 2

with tf.variable\_scope('lv1', reuse=False):

with tf.variable\_scope('lv2'):

init = tf.constant\_initializer(0.1)

wgt1 = tf.get\_variable('wgts', [2,2])

bias1 = tf.get\_variable('biases', [2,2])

print wgt1.name, bias1.name

# ValueError: Variable lv1/lv2/wgts already exists, disallowed.

# Did you mean to set reuse=True or reuse=tf.AUTO\_REUSE in VarScope?

上面两段代码是一个文件

---------------------

作者：yeqiustu

来源：CSDN

原文：https://blog.csdn.net/yeqiustu/article/details/79801982

版权声明：本文为博主原创文章，转载请附上博文链接！

# tf.constant()

tf.constant

tf.constant(value,dtype=None,shape=None,name=’Const’)   
创建一个常量tensor，按照给出value来赋值，可以用shape来指定其形状(维度)。value可以是一个数，也可以是一个list。   
如果是一个数，那么这个常亮中所有值的按该数来赋值。   
如果是list,那么len(value)一定要小于等于shape展开后的长度。赋值时，先将value中的值逐个存入。不够的部分，则全部存入value的最后一个值。dtype是value的类型，默认的就是value的类型，如果需要重新设置，则可以设置新的类型，name是张量的名称

# tf.Variable()

tf.Variable(initializer,name),参数initializer是初始化参数，name是可自定义的变量名称，用法如下：

import tensorflow as tf

v1=tf.Variable(tf.random\_normal(shape=[4,3],mean=0,stddev=1),name='v1')

v2=tf.Variable(tf.constant(2),name='v2')

v3=tf.Variable(tf.ones([4,3]),name='v3')

with tf.Session() as sess:

sess.run(tf.global\_variables\_initializer())

print(sess.run(v1))

print(sess.run(v2))

print(sess.run(v3))

结果如下：

[[-1.2115501   1.0484737   0.55210656]

 [-1.5301195   0.9060654  -2.6766613 ]

 [ 0.27101386 -0.32336152  0.44544214]

 [-0.0120788  -0.3409422  -0.48505628]]

2

[[1. 1. 1.]

 [1. 1. 1.]

 [1. 1. 1.]

 [1. 1. 1.]]

---------------------

作者：小天狼星666

来源：CSDN

原文：https://blog.csdn.net/qq\_37913997/article/details/80352191

版权声明：本文为博主原创文章，转载请附上博文链接！

# tensorflow游乐场地址

[http://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=circle&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=4,2&seed=0.98869&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false](http://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=circle&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=4,2&seed=0.98869&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&co)

# tf.matmul()

tf.matmul(a, b, transpose\_a=False, transpose\_b=False, adjoint\_a=False, adjoint\_b=False, a\_is\_sparse=False, b\_is\_sparse=False, name=None)###

将矩阵a 乘于矩阵b。

注意，在使用tf.matmul时，必须要把a和b转化成同一维度的矩阵，比如一个1\*2的矩阵和一个2\*3的矩阵相乘，在用tf.constant创建矩阵的时候必须要把shape设置同一维度，第一个矩阵要设置成shape=[1,2]，第二个矩阵的shape=[2,3]，第一个矩阵的shape不能用默认值，否则在相乘的时候会报出如下错误：

ValueError: Shape must be rank 2 but is rank 1 for 'MatMul' (op: 'MatMul') with input shapes: [2], [2,3].

输入必须是矩阵（或者是张量秩 >２的张量，表示成批的矩阵），并且其在转置之后有相匹配的矩阵尺寸。  
两个矩阵必须都是同样的类型，支持的类型如下：float16, float32, float64, int32, complex64, complex128。

任意一个矩阵都可以通过设置被转置 或 adjointed（共轭和转置）为真时进行对应的操作。两个参数都默认为假

如果一个或两个矩阵都包含大量的零，当\*\*a\_is\_sparse 或b\_is\_sparse \*\*设置为真时，会使用一个更有效率的乘法公式。  
他们都默认设置为假。这个优化规则只能适用于类型为bfloat16 或 bfloat32 的平面矩阵（等级2的张量）。

参数：  
a: 一个类型为 float16, float32, float64, int32, complex64, complex128 且张量秩 > 1 的张量。  
b: 一个类型跟张量a相同的张量。  
transpose\_a: 如果为真, a则在进行乘法计算前进行转置。  
transpose\_b: 如果为真, b则在进行乘法计算前进行转置。  
adjoint\_a: 如果为真, a则在进行乘法计算前进行共轭和转置。  
adjoint\_b: 如果为真, b则在进行乘法计算前进行共轭和转置。  
a\_is\_sparse: 如果为真, a会被处理为稀疏矩阵。  
b\_is\_sparse: 如果为真, b会被处理为稀疏矩阵。  
name: 操作的名字（可选参数）

返回值：  
一个跟张量a和张量b类型一样的张量且最内部矩阵是a和b中的相应矩阵的乘积。如果所有的转置或adjoint参数都为假，则公式为output[..., :, :] = a[..., :, :] \* b[..., :, :] ,

引发错误:  
ValueError: 如果transpose\_a 和 adjoint\_a, 或 transpose\_b 和 adjoint\_b 都被设置为真

作者：HabileBadger  
链接：https://www.jianshu.com/p/19ea2d15eb14  
來源：简书  
简书著作权归作者所有，任何形式的转载都请联系作者获得授权并注明出处。

# tensorflow随机数生成函数

|  |  |  |
| --- | --- | --- |
| 函数名称 | 随机数分布 | 主要参数 |
| tf.random\_normal | 正态分布 | 平均值，标准差，取值类型 |
| tf.truncated\_normal | 正态分布，但如果随即出来的值 偏离平均值两个标准差，那么这个数将会被重新随机 | 平均数，标准差，取值类型 |
| tf.random\_uniform | 均匀分布 | 最小、最大取值，取值类型 |
| tf.random\_gamma | Gamma分布 | 形状参数alpha、尺度参数beta，取值类型 |

# tensorflow常数生成函数

|  |  |  |
| --- | --- | --- |
| 函数名称 | 功能 | 样例 |
| tf.zeros | 产生全0的数组 | tf.zeros([2,3],int32)->[[0,0,0],[0.0.0] |
| tf.ones | 产生全1的数组 | tf.ones([2,3],int32)->[[1,1,1],[1,1,1]] |
| tf.fill | 产生一个全部为给定数字的数组 | tf.fill([2,3],9)->[[9,9,9],[9,9,9]] |
| tf.constant | 产生一个给定值的常量 | tf.constant([1,2,3])->[1,2,3] |

# 神经网络中的偏置值

就是b值....  
wx+b的b  
打个比方 有点(1,1) 属于1类 点(2,2)属于2类，请问是否能从原点画一条线把他们分开  
不可以，所以需要偏置值b，这样线段就不从(0,0)点出发了

# 13. tensorflow中维护的集合列表

|  |  |  |
| --- | --- | --- |
| 集合名称 | 集合内容 | 使用场景 |
| tf.GraphKeys.VARIABLES | 所有变量 | 持久化tensorflow模型 |
| tf.GraphKeys.TRAINABLE\_VARIABLES | 可学习的变量（一般指神经网络中的参数） | 模型训练，生成模型可视化内容 |
| tf.GraphKeys.SUMMARIES | 日志生成相关的张量 | tensorflow计算可视化 |
| tf.GraphKeys.QUEUE\_RUNNERS | 处理输入的QueueRunner | 输入处理 |
| tf.GraphKeys.MOVING\_AVERAGE\_VARIABLES | 所有计算了滑动平均值的变量 | 计算变量的滑动平均值 |

# tf.placeholder

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

placeholder，占位符，在tensorflow中类似于函数参数，运行时必须传入值。

dtype：数据类型。常用的是tf.float32,tf.float64等数值类型。

shape：数据形状。默认是None，就是一维值，也可以是多维，比如[2,3], [None, 3]表示列是3，行不定。

name：名称。

代码片段-1（计算3\*4=12）

#!/usr/bin/env python

# \_\*\_ coding: utf-8 \_\*\_

import tensorflow as tf

import numpy as np

input1 = tf.placeholder(tf.float32)

input2 = tf.placeholder(tf.float32)

output = tf.multiply(input1, input2)

with tf.Session() as sess:

print sess.run(output, feed\_dict = {input1:[3.], input2: [4.]})

代码片段-2（计算矩阵相乘，x\*x）

#!/usr/bin/env python

# \_\*\_ coding: utf-8 \_\*\_

import tensorflow as tf

import numpy as np

x = tf.placeholder(tf.float32, shape=(1024, 1024))

y = tf.matmul(x, x)

with tf.Session() as sess:

# print(sess.run(y)) # ERROR: x is none now

rand\_array = np.random.rand(1024, 1024)

print(sess.run(y, feed\_dict={x: rand\_array})) # Will succeed.

---------------------

作者：cc19

来源：CSDN

原文：https://blog.csdn.net/cc1949/article/details/78364615

版权声明：本文为博主原创文章，转载请附上博文链接！

# tf.reduce\_mean()

tf.reduce\_mean 函数用于计算张量tensor沿着指定的数轴（tensor的某一维度）上的的平均值，主要用作降维或者计算tensor（图像）的平均值。

reduce\_mean(input\_tensor,

axis=None,

keep\_dims=False,

name=None,

reduction\_indices=None)

第一个参数input\_tensor： 输入的待降维的tensor;

第二个参数axis： 指定的轴，如果不指定，则计算所有元素的均值;

第三个参数keep\_dims：是否降维度，设置为True，输出的结果保持输入tensor的形状，设置为False，输出结果会降低维度;

第四个参数name： 操作的名称;

第五个参数 reduction\_indices：在以前版本中用来指定轴，已弃用;

以一个维度是2，形状是[3,3]的tensor举例：

import tensorflow as tf

x = [[1,2,3],

[1,2,3]]

xx = tf.cast(x,tf.float32)

mean\_all = tf.reduce\_mean(xx, keep\_dims=False)

mean\_0 = tf.reduce\_mean(xx, axis=0, keep\_dims=False)

mean\_1 = tf.reduce\_mean(xx, axis=1, keep\_dims=False)

with tf.Session() as sess:

m\_a,m\_0,m\_1 = sess.run([mean\_all, mean\_0, mean\_1])

rint m\_a # output: 2.0

print m\_0 # output: [ 1. 2. 3.]

print m\_1 #output: [ 2. 2.]

如果设置保持原来的张量的维度，keep\_dims=True ，结果：

print m\_a # output: [[ 2.]]

print m\_0 # output: [[ 1. 2. 3.]]

print m\_1 #output: [[ 2.], [ 2.]]

类似函数还有:

tf.reduce\_sum ：计算tensor指定轴方向上的所有元素的累加和;

tf.reduce\_max  :  计算tensor指定轴方向上的各个元素的最大值;

tf.reduce\_all :  计算tensor指定轴方向上的各个元素的逻辑和（and运算）;

tf.reduce\_any:  计算tensor指定轴方向上的各个元素的逻辑或（or运算）;

---------------------

作者：-牧野-

来源：CSDN

原文：https://blog.csdn.net/dcrmg/article/details/79797826

版权声明：本文为博主原创文章，转载请附上博文链接！

# tf.clip\_by\_value()

tf.clip\_by\_value(A, min, max)：输入一个张量A，把A中的每一个元素的值都压缩在min和max之间。  
小于min的让它等于min，大于max的元素的值等于max。

例如：

import tensorflow as tf;

import numpy as np;

A = np.array([[1,1,2,4], [3,4,8,5]])

with tf.Session() as sess:

print sess.run(tf.clip\_by\_value(A, 2, 5))

输出：

[[2 2 2 4]

[3 4 5 5]]

作者：yalesaleng  
链接：https://www.jianshu.com/p/e79f6842cadb  
來源：简书  
简书著作权归作者所有，任何形式的转载都请联系作者获得授权并注明出处。

# tf.sigmoid()

tf.sigmoid(a,name)

把a转换为0~1之间的数值

# tf.square()

### tf.square(x, name=None)

对**x**内的所有元素进行平方操作

I.e., (y = x \* x = x^2).

即 (y = x \* x = x^2)。

参数:  
**x**: 一个张量或者是稀疏张量。必须是下列类型之一：half, float32, float64, int32, int64, complex64, complex128.  
**name**: 操作的名字 (可选参数).

Returns:  
一个张量或者是稀疏张量。有着跟**x**相同的类型。

作者：HabileBadger  
链接：https://www.jianshu.com/p/7a201b0814e3  
來源：简书  
简书著作权归作者所有，任何形式的转载都请联系作者获得授权并注明出处。

# tf.greater()

tf.greater(a,b)

功能：通过比较a、b两个值的大小来输出对错。

例如：当a=4，b=3时，输出结果为：true；当a=2，b=3时，输出结果为：false。

import tensorflow as tf

sess=tf.Session()

with sess.as\_default():

print(tf.greater(4,3).eval())

5

结果：

True

---------------------

作者：放下扳手&拿起键盘

来源：CSDN

原文：https://blog.csdn.net/william\_hehe/article/details/78636564

版权声明：本文为博主原创文章，转载请附上博文链接！

# tf.where()

有两种用法：

1、tf.where(tensor)

tensor 为一个bool 型张量，where函数将返回其中为true的元素的索引。如上图官方注释

2、tf.where(tensor,a,b)

a,b为和ｔｅｎｓｏｒ相同维度的ｔｅｎｓｏｒ，将ｔｅｎｓｏｒ中的ｔｒｕｅ位置元素替换为ａ中对应位置元素，ｆａｌｓｅ的替换为ｂ中对应位置元素。

例：

[复制代码](javascript:void(0);)

import tensorflow as tf

import numpy as np

sess=tf.Session()

a=np.array([[1,0,0],[0,1,1]])

a1=np.array([[3,2,3],[4,5,6]])

print(sess.run(tf.equal(a,1)))

print(sess.run(tf.where(tf.equal(a,1),a1,1-a1)))

[复制代码](javascript:void(0);)

＞＞［［ｔｒｕｅ，ｆａｌｓｅ，ｆａｌｓｅ］，［ｆａｌｓｅ，ｔｒｕｅ，ｔｒｕｅ］］

＞＞［［３，-１，-２］，［-３，５，６］］

# tf.train.exponential\_decay()

指数衰减发来优化学习率

tf.train.exponential\_decay(learning\_rate,global\_step,decay\_steps,decay\_rate,staircase=False, name=None)

参数:

learning\_rate:最初的学习速率。一个float32或float64张量或一个Python类型数字。

global\_step:用于衰变计算的全局步数，不能是负数，int32,或者int64，或者一个Python类型数字。

decay\_steps:int32或者int64的Tensor或者一个Python类型数字，参见上面的衰变计算。

decay\_rate: nt32或者int64的Tensor或者一个Python类型数字，衰减速率

staircase: Boolean型.它是true时，在离散时间间隔内衰减学习速率。

name: string型.可选项，默认值为 'ExponentialDecay'

返回值：

返回值类型与learn\_rate返回值类型相同，返回衰减学习速率

---------------------

import tensorflow as tf;

import numpy as np;

import matplotlib.pyplot as plt;

learning\_rate = 0.1

decay\_rate = 0.96

global\_steps = 1000

decay\_steps = 100

global\_ = tf.Variable(tf.constant(0))

c = tf.train.exponential\_decay(learning\_rate, global\_, decay\_steps, decay\_rate, staircase=True)

d = tf.train.exponential\_decay(learning\_rate, global\_, decay\_steps, decay\_rate, staircase=False)

T\_C = []

F\_D = []

with tf.Session() as sess:

for i in range(global\_steps):

T\_c = sess.run(c,feed\_dict={global\_: i})

T\_C.append(T\_c)

F\_d = sess.run(d,feed\_dict={global\_: i})

F\_D.append(F\_d)

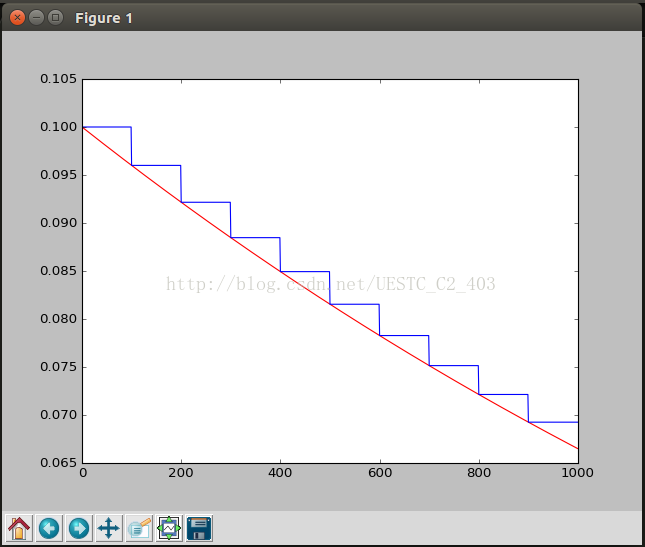
plt.figure(1)

plt.plot(range(global\_steps), F\_D, 'r-')

plt.plot(range(global\_steps), T\_C, 'b-')

plt.show()

---------------------



作者：UESTC\_C2\_403

来源：CSDN

原文：https://blog.csdn.net/uestc\_c2\_403/article/details/72213286

版权声明：本文为博主原创文章，转载请附上博文链接！

初始的学习速率是0.1，总的迭代次数是1000次，如果staircase=True，那就表明每decay\_steps次计算学习速率变化，更新原始学习速率，如果是False，那就是每一步都更新学习速率。

# tf.contrib.layers.12\_regularizer(lambda)(w) tf.contrib.layers.11\_regularizer(lambda)(w)

tensorflow提供了tf.contrib.layers.12\_regularizer(lambda)(w)函数，他可以返回一个函数，这个函数可以计算给定参数的L2正则化项的值

lambda参数表示正则化项的权重，也就是公式J(Θ)+λR(w)中的λ

w为需要计算正则化损失的参数

tf.contrib.layers.11\_regularizer(lambda)(w),计算给定参数的L1正则化项的值，参数意义与L2的一样

# tf.add\_n()

tf.add\_n([p1, p2, p3....])函数是实现一个列表的元素的相加。就是输入的对象是一个列表，列表里的元素可以是向量，矩阵，等

例如：

import tensorflow as tf;

import numpy as np;

input1 = tf.constant([1.0, 2.0, 3.0])

input2 = tf.Variable(tf.random\_uniform([3]))

output = tf.add\_n([input1, input2])

with tf.Session() as sess:

sess.run(tf.initialize\_all\_variables())

print sess.run(input1 + input2)

print sess.run(output)

输出：

[ 1.68921876  2.73008633  3.04061747]

[ 1.68921876  2.73008633  3.04061747]

---------------------

作者：UESTC\_C2\_403

来源：CSDN

原文：https://blog.csdn.net/uestc\_c2\_403/article/details/72808839

版权声明：本文为博主原创文章，转载请附上博文链接！

# tf.assign()

tf.assign(A, new\_number): 这个函数的功能主要是把A的值变为new\_number

例如：

import tensorflow as tf;

A = tf.Variable(tf.constant(0.0), dtype=tf.float32)

with tf.Session() as sess:

sess.run(tf.initialize\_all\_variables())

print sess.run(A)

sess.run(tf.assign(A, 10))

print sess.run(A)

输出：

0.0

10.0

开始给A赋值为0，经过tf.assign函数后，把A的值变为10

---------------------

作者：UESTC\_C2\_403

来源：CSDN

原文：https://blog.csdn.net/uestc\_c2\_403/article/details/72235310

版权声明：本文为博主原创文章，转载请附上博文链接！

# 24. tf.truncated\_normal()

tf.truncated\_normal的用法

tf.truncated\_normal(shape, mean, stddev) :shape表示生成张量的维度，mean是均值，stddev是标准差。这个函数产生正太分布，均值和标准差自己设定。这是一个截断的产生正太分布的函数，就是说产生正太分布的值如果与均值的差值大于两倍的标准差，那就重新生成。和一般的正太分布的产生随机数据比起来，这个函数产生的随机数与均值的差距不会超过两倍的标准差，但是一般的别的函数是可能的。

例如：

import tensorflow as tf;

import numpy as np;

import matplotlib.pyplot as plt;

c = tf.truncated\_normal(shape=[10,10], mean=0, stddev=1)

with tf.Session() as sess:

print sess.run(c)

输出：

[[ 1.95758033 -0.68666345 -1.83860338  0.78213859 -1.08119416 -1.44530308  
   0.38035342  0.57904619 -0.57145643 -1.22899497]  
 [-0.75853795  0.48202974  1.03464043  1.19210851 -0.15739718  0.8506189  
   1.18259966 -0.99061841 -0.51968449  1.38996458]  
 [ 1.05636907 -0.02668529  0.64182931  0.4110294  -0.4978295  -0.64912242  
   1.27779591 -0.01533993  0.47417602 -1.28639436]  
 [-1.65927458 -0.364887   -0.45535028  0.078814   -0.30295736  1.91779387  
  -0.66928798 -0.14847915  0.91875714  0.61889237]  
 [-0.01308221 -0.38468206  1.34700036  0.64531708  1.15899456  1.09932268  
   1.22457981 -1.1610316   0.59036094 -1.97302651]  
 [-0.24886213  0.82857937  0.09046989  0.39251322  0.21155456 -0.27749416  
   0.18883201  0.08812679 -0.32917103  0.20547724]  
 [ 0.05388507  0.45474565  0.23398806  1.32670367 -0.01957406  0.52013856  
  -1.13907862 -1.71957874  0.75772947 -1.01719368]  
 [ 0.27155915  0.05900437  0.81448066 -0.37997526 -0.62020499 -0.88820189  
   1.53407145 -0.01600445 -0.4236775  -1.68852305]  
 [ 0.78942037 -1.32458341 -0.91667277 -0.00963761  0.76824385 -0.5405798  
  -0.73307443 -1.19854116 -0.66179073  0.26329204]  
 [ 0.59473759 -0.37507254 -1.21623695 -1.30528259  1.18013096 -1.32077384  
  -0.59241474 -0.28063133  0.12341146  0.48480138]]

# tf.group()

tf.group()用于创造一个操作，可以将传入参数的所有操作进行分组。API手册如:

tf.group(

\*inputs,

\*\*kwargs

)

ops = tf.group(tensor1, tensor2,...)

其中\*inputs是0个或者多个用于组合tensor，一旦ops完成了，那么传入的tensor1,tensor2,...等等都会完成了，经常用于组合一些训练节点，如在Cycle GAN中的多个训练节点，例子如：

generator\_train\_op = tf.train.AdamOptimizer(g\_loss, ...)

discriminator\_train\_op = tf.train.AdamOptimizer(d\_loss,...)

train\_ops = tf.groups(generator\_train\_op ,discriminator\_train\_op)

with tf.Session() as sess:

sess.run(train\_ops)

# 一旦运行了train\_ops,那么里面的generator\_train\_op和discriminator\_train\_op都将被调用

注意的是，tf.group()返回的是个操作，而不是值，如果你想下面一样用，返回的将不是值

a = tf.Variable([5])

b = tf.Variable([6])

c = a+b

d = a\*b

e = a/b

ops = tf.group(c,d,e)

with tf.Session() as sess:

sess.run(tf.global\_variables\_initializer())

ee = sess.run(ops)

返回的将不是c,d,e的运算结果，而是一个None，就是因为这个是一个操作，而不是一个张量。如果需要返回结果，请参考tf.tuple()

---------------------

作者：FesianXu

来源：CSDN

原文：https://blog.csdn.net/LoseInVain/article/details/81703786

版权声明：本文为博主原创文章，转载请附上博文链接！

# tf.control\_dependencies(control\_input)

作用同tf.group相同

此函数指定某些操作执行的依赖关系

返回一个控制依赖的上下文管理器，使用 with 关键字可以让在这个上下文环境中的操作都在 control\_inputs 执行

with tf.control\_dependencies([a, b]):

c = ....

d = ...

在执行完 a，b 操作之后，才能执行 c，d 操作。意思就是 c，d 操作依赖 a，b 操作

with tf.control\_dependencies([train\_step, variable\_averages\_op]):

train\_op = tf.no\_op(name='train')

tf.no\_op()表示执行完 train\_step, variable\_averages\_op 操作之后什么都不做

# tf.tuple()

tf.tuple()用于组合多个张量输入组成的列表[tensor1,tensor2,...]，然后返回一个计算过后的张量列表[cal\_tensor1,cal\_tensor2,...]，这点和tf.group()是不同的，API手册如：

tf.tuple(

tensors,

name=None,

control\_inputs=None

)

ops = tf.tuple([tensor1,tensor2,...],control\_inputs=c\_ops)

其中tensors是由多个tensor组成的列表，其中的control\_inputs是添加额外的控制输入，添加的输入c\_ops必须在整个ops完成之前得到执行，但是c\_ops的输出是不会返回的。

API上描述，这个可以作为提供一种并行处理的机制，所有的输入的tensor可以并行计算，但是所有tensor的计算出来的值将会以tuple的形式返回，并且这个只能在并行计算完成之后得到。

使用例子：

a = tf.Variable([5])

b = tf.Variable([6])

c = a+b

d = a\*b

e = a/b

ops = tf.tuple([c,d,e])

with tf.Session() as sess:

sess.run(tf.global\_variables\_initializer())

ee = sess.run(ops)

print(ee)

输出

[array([11], dtype=int32), array([30], dtype=int32), array([0.83333333])]

1

可以和tf.group()用于组合多个操作的例子进行对比。

---------------------

作者：FesianXu

来源：CSDN

原文：https://blog.csdn.net/LoseInVain/article/details/81704072

版权声明：本文为博主原创文章，转载请附上博文链接！

# tf.cast()

函数 tf.cast()

cast(

x,

dtype,

name=None

)

将x的数据格式转化成dtype.例如，原来x的数据格式是bool，   
那么将其转化成float以后，就能够将其转化成0和1的序列。反之也可以

例子

a = tf.Variable([1,0,0,1,1])

b = tf.cast(a,dtype=tf.bool)

sess = tf.Session()

sess.run(tf.initialize\_all\_variables())

print(sess.run(b))

#[ True False False True True]

# tensorflow中的变量初始化

|  |  |  |
| --- | --- | --- |
| 初始化函数 | 功能 | 主要参数 |
| tf.constant\_initializer | 将变量初始化为给定常量 | 常量的取值 |
| tf.random\_normal\_initializer | 将变量初始化为满足正态分布的随机值 | 正态分布的均值和标准差 |
| tf.truncated\_uniform\_initializer | 将变量初始化为满足正态分布的随机值，但如果随机出来的值偏离平均值超过两个标准差，那么这个数将会被重新随机 | 正态分布的均值和标准差 |
| tf.random\_uniform\_initializer | 将变量初始化为满足平均分布的随机值 | 最大、最小值 |
| tf.uniform\_unit\_scaling\_initializer | 将变量初始化为满足平均分布但不影响输出数量级的随机值 | factor(产生随机值时) |
| tf.zeros\_initializer | 将变量设置为全0 | 变量维度 |
| tf.ones\_initializer | 将变量设置为全1 | 变量维度 |

# tf.train.Saver()

详细用法见tensorflow实战google深度学习框架第二版的5.4.1

# variables\_to\_restore()

variables\_to\_restore函数，是TensorFlow为滑动平均值提供。之前，也介绍过通过使用滑动平均值可以让神经网络模型更加的健壮。我们也知道，其实在TensorFlow中，变量的滑动平均值都是由影子变量所维护的，如果你想要获取变量的滑动平均值需要获取的是**影子变量**而不是**变量**本身。

**1、滑动平均值模型文件的保存**

**[python]** [view plain](http://blog.csdn.net/sinat_29957455/article/details/78508793) [copy](http://blog.csdn.net/sinat_29957455/article/details/78508793)

1. **import** tensorflow as tf
3. **if** \_\_name\_\_ == "\_\_main\_\_":
4. v = tf.Variable(0.,name="v")
5. #设置滑动平均模型的系数
6. ema = tf.train.ExponentialMovingAverage(0.99)
7. #设置变量v使用滑动平均模型，tf.all\_variables()设置所有变量
8. op = ema.apply([v])
9. #获取变量v的名字
10. **print**(v.name)
11. #v:0
12. #创建一个保存模型的对象
13. save = tf.train.Saver()
14. sess = tf.Session()
15. #初始化所有变量
16. init = tf.initialize\_all\_variables()
17. sess.run(init)
18. #给变量v重新赋值
19. sess.run(tf.assign(v,10))
20. #应用平均滑动设置
21. sess.run(op)
22. #保存模型文件
23. save.save(sess,"./model.ckpt")
24. #输出变量v之前的值和使用滑动平均模型之后的值
25. **print**(sess.run([v,ema.average(v)]))
26. #[10.0, 0.099999905]

上面的代码，是如何来保存一个滑动平均值的模型文件，之前有介绍过滑动平均值和模型文件的保存，所以这里就不再重复了。

**2、滑动平均值模型文件的读取**

**[python]** [view plain](http://blog.csdn.net/sinat_29957455/article/details/78508793) [copy](http://blog.csdn.net/sinat_29957455/article/details/78508793)

1. v = tf.Variable(1.,name="v")
2. #定义模型对象
3. saver = tf.train.Saver({"v/ExponentialMovingAverage":v})
4. sess = tf.Session()
5. saver.restore(sess,"./model.ckpt")
6. **print**(sess.run(v))
7. #0.0999999

对于模型文件的读取，在上一篇博客中有介绍过，这里特别需要注意的一个地方就是，在使用tf.train.Saver函数中，所传递的模型参数是**{"v/ExponentialMovingAverage":v}**而不是**{"v":v}**，如果你使用的是后面的参数，那么你得到的结果将是10而不是0.09，那是因为后者获取的是**变量**本身而不是**影子变量**。是不是感觉使用这种方式来读取模型文件的时候，还需要输入一大串的变量名称。

**3、variables\_to\_restore函数的使用**

**[python]** [view plain](http://blog.csdn.net/sinat_29957455/article/details/78508793) [copy](http://blog.csdn.net/sinat_29957455/article/details/78508793)

1. v = tf.Variable(1.,name="v")
2. #滑动模型的参数的大小并不会影响v的值
3. ema = tf.train.ExponentialMovingAverage(0.99)
4. **print**(ema.variables\_to\_restore())
5. #{'v/ExponentialMovingAverage': <tf.Variable 'v:0' shape=() dtype=float32\_ref>}
6. sess = tf.Session()
7. saver = tf.train.Saver(ema.variables\_to\_restore())
8. saver.restore(sess,"./model.ckpt")
9. **print**(sess.run(v))
10. #0.0999999

通过使用**variables\_to\_restore**函数，可以使在加载模型的时候将影子变量直接映射到变量的本身，所以我们在获取变量的滑动平均值的时候只需要获取到变量的本身值而不需要去获取影子变量。

# tf.gfile

tf.gfile模块的主要角色是：  
1.提供一个接近Python文件对象的API，以及  
2.提供基于TensorFlow C ++ FileSystem API的实现。

gfile API介绍

下面将分别介绍每一个gfile API！

2-1）tf.gfile.Copy(oldpath, newpath, overwrite=False)

拷贝源文件并创建目标文件，无返回，其形参说明如下：

oldpath：带路径名字的拷贝源文件；

newpath：带路径名字的拷贝目标文件；

overwrite：目标文件已经存在时是否要覆盖，默认为false，如果目标文件已经存在则会报错

2-2）tf.gfile.MkDir(dirname)

创建一个目录，dirname为目录名字，无返回。

2-3）tf.gfile.Remove(filename)

删除文件，filename即文件名，无返回。

2-4）tf.gfile.DeleteRecursively(dirname)

递归删除所有目录及其文件，dirname即目录名，无返回。

2-5）tf.gfile.Exists(filename)

判断目录或文件是否存在，filename可为目录路径或带文件名的路径，有该目录则返回True，否则False。

2-6）tf.gfile.Glob(filename)

查找匹配pattern的文件并以列表的形式返回，filename可以是一个具体的文件名，也可以是包含通配符的正则表达式。

2-7）tf.gfile.IsDirectory(dirname)

判断所给目录是否存在，如果存在则返回True，否则返回False，dirname是目录名。

2-8）tf.gfile.ListDirectory(dirname)

罗列dirname目录下的所有文件并以列表形式返回，dirname必须是目录名。

2-9）tf.gfile.MakeDirs(dirname)

以递归方式建立父目录及其子目录，如果目录已存在且是可覆盖则会创建成功，否则报错，无返回。

2-10）tf.gfile.Rename(oldname, newname, overwrite=False)

重命名或移动一个文件或目录，无返回，其形参说明如下：

oldname：旧目录或旧文件；

newname：新目录或新文件；

overwrite：默认为false，如果新目录或新文件已经存在则会报错，否则重命名或移动成功。

2-11）tf.gfile.Stat(filename)

返回目录的统计数据，该函数会返回FileStatistics数据结构，以dir(tf.gfile.Stat(filename))获取返回数据的属性如下：

2-12）tf.gfile.Walk(top, in\_order=True)

递归获取目录信息生成器，top是目录名，in\_order默认为True指示顺序遍历目录，否则将无序遍历，每次生成返回如下格式信息(dirname, [subdirname, subdirname, ...], [filename, filename, ...])。

2-13）tf.gfile.GFile(filename, mode)

获取文本操作句柄，类似于python提供的文本操作open()函数，filename是要打开的文件名，mode是以何种方式去读写，将会返回一个文本操作句柄。

tf.gfile.Open()是该接口的同名，可任意使用其中一个！

2-14）tf.gfile.FastGFile(filename, mode)

该函数与tf.gfile.GFile的差别仅仅在于“无阻塞”，即该函数会无阻赛以较快的方式获取文本操作句柄。

---------------------

作者：cs\_程序猿

来源：CSDN

原文：https://blog.csdn.net/a373595475/article/details/79693430

版权声明：本文为博主原创文章，转载请附上博文链接！

# tf.graph\_util模板

在 python 中操作张量图的帮助器。

**使用函数**

convert\_variables\_to\_constants(...)：用相同值的常量替换图表中的所有变量。

extract\_sub\_graph(...)：提取可以到达“dest\_nodes”中的任何节点的子图。

must\_run\_on\_cpu(...)：如果给定的 node\_def 必须在 CPU 上运行，则返回 True，否则为 False。

remove\_training\_nodes(...)：调整不需要推理的节点。

tensor\_shape\_from\_node\_def\_name(...)：从 NodeDef 的输入字符串获取形状的便利函数。

# tf.get\_default\_graph

功能：获取当前默认计算图。

例子：

import tensorflow as tf

a=tf.constant([1.0,2.0],name="a")

b=tf.constant([3.0,4.0],name="b")

c=a+b

print(c.graph)

print(tf.get\_default\_graph())

结果：

tensorflow.python.framework.ops.Graph object at 0x000001E4F528CB70

tensorflow.python.framework.ops.Graph object at 0x000001E4F528CB70

---------------------

作者：放下扳手&拿起键盘

来源：CSDN

原文：https://blog.csdn.net/william\_hehe/article/details/78616998

版权声明：本文为博主原创文章，转载请附上博文链接！

# GraphDef

只关注计算图的连接结构信息, 根据连接结构中”索引“在MetaInfoDef中寻找节点运算的具体

message GraphDef {

repeated NodeDef node = 1;

// 版本号

VersionDef versions = 4;

// 实验性的版本号

int32 version = 3 [deprecated = true];

FunctionDefLibrary library = 2;

};

## tf.get\_default\_graph().as\_graph\_def()

获取当前计算图中的所有结点的信息，如

import tensorflow as tf

a = tf.constant(1)

print(tf.get\_default\_graph().as\_graph\_def())

返回

node {

name: "Const"

op: "Const"

attr {

key: "dtype"

value {

type: DT\_INT32

}

}

attr {

key: "value"

value {

tensor {

dtype: DT\_INT32

tensor\_shape {

}

int\_val: 1

}

}

}

}

versions {

producer: 22

}

---------------------

作者：silent56\_th

来源：CSDN

原文：https://blog.csdn.net/silent56\_th/article/details/75557951

版权声明：本文为博主原创文章，转载请附上博文链接！

# 32. tf.import\_graph\_def()

# f.import\_graph\_def

tf.import\_graph\_def(  
    graph\_def,  
    input\_map=None,  
    return\_elements=None,  
    name=None,  
    op\_dict=None,  
    producer\_op\_list=None  
)

定义： [tensorflow/python/framework/importer.py](https://www.github.com/tensorflow/tensorflow/blob/r1.10/tensorflow/python/framework/importer.py).

参考:    [Building Graphs > Utility functions](https://tensorflow.google.cn/api_guides/python/framework#Utility_functions)

将图从graph\_def导入到当前默认图中. (即将舍弃的参数)

SOME ARGUMENTS ARE DEPRECATED. 它们将在未来的版本中被删除。更新说明:如果你需要此特性请在 https://github.com/tensorflow/tensorflow/issues 提出。

这个函数提供了一种方法来导入序列化的TensorFlow GraphDef协议缓冲区，并将GraphDef中的各个对象提取为tf。张量和tf。操作对象。一旦提取出来，这些对象就会被放到当前的默认图形中。。参考[tf.Graph.as\_graph\_def](https://tensorflow.google.cn/api_docs/python/tf/Graph#as_graph_def) 来创建 a GraphDef proto.

参数:

* **graph\_def**: 包含要导入到默认图中的操作的GraphDef proto。
* **input\_map**: 将graph\_def中的输入名称(作为字符串)映射到张量对象的字典。输入图中指定的输入张量的值将被重新映射到相应的张量值。
* **return\_elements**: 在graph\_def中包含操作名的字符串列表，将作为operationobject返回;和/或graph\_def中的张量名称，它们将作为张量对象返回。
* **name**: (可选.) 将前缀放在graph\_def中名称前面的前缀。注意，这并不适用于导入的函数名。默认为"import".
* **op\_dict**: (可选.) 已弃用，请勿使用
* **producer\_op\_list**: (可选.) 一个OpList原型，带有(可能是剥离的)图表生产者使用的OpDefs列表。如果提供了，那么根据producer\_op\_list的默认值，在graph\_def中无法识别的ops attrs将被删除。这将允许稍后的二进制文件生成更多的graphdef被早期的二进制文件所接受。

返回:

   从导入的图中得到的与return\_element中的名称相对应的操作和/或张量对象的列表。

错误类型:

* **TypeError**: 如果graph\_def不是GraphDef proto, input\_map不是字典映射字符串到Tensorobjects，或者return\_elements不是字符串列表。
* **ValueError**: 如果input\_map，或者return\_elements包含在graph\_def中没有出现的名称，或者graph\_def格式不好(例如，它指的是一个未知张量)。

# 持久化原理及数据格式

定义元图（MetaGraph）的类型MetaGraphDef（详细参见《tensorflow实战google深度学习框架》第二版p117）

# 持续化代码自动生成的四个文件

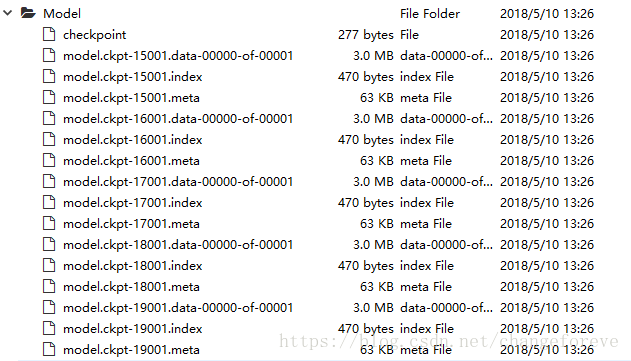
# tf.train.get\_checkpoint\_state

tf.train.get\_checkpoint\_state函数通过checkpoint文件找到模型文件名。

tf.train.get\_checkpoint\_state(checkpoint\_dir,latest\_filename=None)

该函数返回的是checkpoint文件CheckpointState proto类型的内容，其中有model\_checkpoint\_path和all\_model\_checkpoint\_paths两个属性。其中model\_checkpoint\_path保存了最新的tensorflow模型文件的文件名，all\_model\_checkpoint\_paths则有未被删除的所有tensorflow模型文件的文件名。

下图是在训练过程中生成的几个模型文件列表：



以下是测试程序里的部分代码：

with tf.Session() as sess:

ckpt=tf.train.get\_checkpoint\_state('Model/')

print(ckpt)

if ckpt and ckpt.all\_model\_checkpoint\_paths:

#加载模型

#这一部分是有多个模型文件时，对所有模型进行测试验证

for path in ckpt.all\_model\_checkpoint\_paths:

saver.restore(sess,path)

global\_step=path.split('/')[-1].split('-')[-1]

accuracy\_score=sess.run(accuracy,feed\_dict=validate\_feed)

print("After %s training step(s),valisation accuracy = %g"%(global\_step,accuracy\_score))

'''

#对最新的模型进行测试验证

saver.restore(sess,ckpt.model\_checkpoint\_paths)

global\_step=ckpt.model\_checkpoint\_path.split('/')[-1].split('-')[-1]

accuracy\_score=sess.run(accuracy,feed\_dict=validate\_feed)

print("After %s training step(s),valisation accuracy = %g"%(global\_step,accuracy\_score))

'''

else:

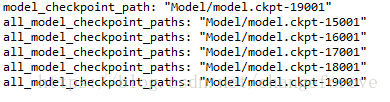
print('No checkpoint file found')

return

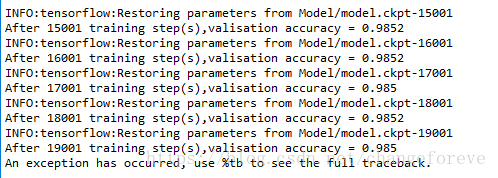
#time.sleep(eval\_interval\_secs)

return

在上面代码中，通过tf.train.get\_checkpoint\_state函数得到的相关模型文件名如下：



对所有模型进行测试，得到：



---------------------

作者：知了声日迟

来源：CSDN

原文：https://blog.csdn.net/changeforeve/article/details/80268522

版权声明：本文为博主原创文章，转载请附上博文链接！

# tf.shape()

f.shape(input,name=None,out\_type=tf.int32)

参数

（1）input：输入张量或稀疏张量；

（2）name：命名；

（3）out\_type：默认tf.int32类型；

输出

返回out\_type类型张量。

例子：将矩阵的维度输出成一个维度的矩阵

import tensorflow as tf

import numpy as np

A = np.array([[[1, 1, 1],

[2, 2, 2]],

[[3, 3, 3],

[4, 4, 4]],

[[5, 5, 5],

[6, 6, 6]]])

t = tf.shape(A)

with tf.Session() as sess:

print(sess.run(t))

https://img-blog.csdn.net/20181008111617451?watermark/2/text/aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3lxbF82MTc1NDAyOTg=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70

import tensorflow as tf

import numpy as np

A = np.array([[[1, 1],

[2, 2]],

[[3, 3],

[4, 4]]])

t = tf.shape(A)

with tf.Session() as sess:

print(sess.run(t))

https://img-blog.csdn.net/20181008111903181?watermark/2/text/aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3lxbF82MTc1NDAyOTg=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70

---------------------

作者：蹦跶的小羊羔

来源：CSDN

原文：https://blog.csdn.net/yql\_617540298/article/details/82965503

版权声明：本文为博主原创文章，转载请附上博文链接！

# tf.reshape()

tf.reshape(tensor,shape,name=None)

参数

（1）tensor：输入张量；

（2）shape：列表形式，可以存在-1；

        -1代表的含义是不用我们自己指定这一维的大小，函数会自动计算，但列表中只能存在一个-1；

（3）name：命名；

输出

将tensor变换为参数shape的形式。

例子

import numpy as np

a= np.array([0,1,2,3,4,5,6,7])

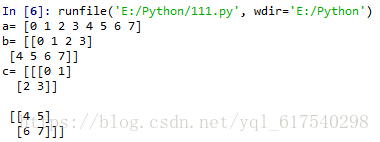
print('a=',a)

b = a.reshape((2,4))

print('b=',b)

c = a.reshape((2,2,2))

print('c=',c)

---------------------

作者：蹦跶的小羊羔

来源：CSDN

原文：https://blog.csdn.net/yql\_617540298/article/details/82965503

版权声明：本文为博主原创文章，转载请附上博文链接！

# tf.python\_io.TFRecordWriter

A class to write records to a TFRecords file.

# tf.train.Int64List,tf.train.BytesList, tf.train.FloatList

tf.train.FloatList：列表每个元素为float。  
  tf.train.Int64List：列表每个元素为int64。   
 tf.train.BytesList：列表每个元素为string。。

# tf.train.slice\_input\_producer和tf.train.string\_input\_producer两种队列批量读取方法

一.tf.train.slice\_input\_producer()

    tf.train.slice\_input\_producer([image,label],num\_epochs=10),随机产生一个图片和标签,num\_epochs=10,则表示把所有的数据过10遍，使用完所有的图片数据为一个epoch,这是重复使用10次。上面的用法表示你的数据集和标签已经全部加载到内存中了，如果数据集非常庞大，我们通过这个函数也可以只加载图片的路径，放入图片的path，注意path必须是一个list或者tensorlist.见下面代码实例

# -\*- coding: utf-8 -\*-

"""

Created on Mon Mar 26 22:02:22 2018

@author: Administrator

"""

import tensorflow as tf

import glob

import matplotlib.pyplot as plt

import time

datapath=r'/media/wsw/pythonfile\_withpycharm/SVMLearning/faceLibrary/人脸库/ORL/'

imgpath = glob.glob(datapath+'\*.bmp')

# 将路径转化成张量形式

imgpath = tf.convert\_to\_tensor(imgpath)

# 产生一个队列每次随机产生一张图片地址

# 注意这里要放在数组里面

image = tf.train.slice\_input\_producer([imgpath])

# 得到一个batch的图片地址

# 由于tf.train.slice\_input\_producer()函数默认是随机产生一个实例

# 所以在这里直接使用tf.train.batch()直接获得一个batch的数据即可

# 没有必要再去使用tf.trian.shuffle\_batch() 速度会慢

img\_batch = tf.train.batch([image],batch\_size=20,capacity=100)

with tf.Session() as sess:

coord = tf.train.Coordinator()

thread = tf.train.start\_queue\_runners(sess,coord)

i = 0

try:

while not coord.should\_stop():

imgs = sess.run(img\_batch)

print(imgs)

fig = plt.figure()

for i,path in enumerate(imgs):

img = plt.imread(path[0].decode('utf-8'))

axes = fig.add\_subplot(5,4,i+1)

axes.imshow(img)

axes.axis('off')

plt.ion()

plt.show()

time.sleep(1)

plt.close()

i+=1

if i%10==0:

break

except tf.errors.OutOfRangeError:

pass

finally:

coord.request\_stop()

coord.join(thread)

注意路径此时被加载成二进制编码格式了。

二.批量读取图片数据

    使用tf.train.slice\_input\_producer([path]),也可以批量读取图片，得到每个图片的路径后，我们可以加载图片并解码成三维数组的形式（图像的深度必须是3通道或者4通道，笔者实验灰度图像，一直不成功）。当使用tf.train.slice\_input\_producer()时，加载图片数据的reader使用tf.read\_file(filename),直接读取。注意图片记得resize().见下面代码：

#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Tue Mar 27 14:18:34 2018

@author: wsw

"""

# 用于通过读取图片的path,然后解码成图片数组的形式，最后返回batch个图片数组

import glob

import tensorflow as tf

import matplotlib.pyplot as plt

path\_list=r'/media/wsw/pythonfile\_withpycharm/SVMLearning/faceLibrary/人脸库/Yale2/'

img\_path = glob.glob(path\_list+'\*.bmp')

img\_path = tf.convert\_to\_tensor(img\_path,dtype=tf.string)

# 这里img\_path,不放在数组里面

# num\_epochs = 1,表示将文件下所有的图片都使用一次

# num\_epochs和tf.train.slice\_input\_producer()中是一样的

# 此参数可以用来设置训练的 epochs

image = tf.train.slice\_input\_producer([img\_path],num\_epochs=1)

# load one image and decode img

def load\_img(path\_queue):

# 创建一个队列读取器，然后解码成数组

# reader = tf.WholeFileReader()

# key,value = reader.read(path\_queue)

file\_contents = tf.read\_file(path\_queue[0])

img = tf.image.decode\_bmp(file\_contents,channels=1)

# 这里很有必要，否则会出错

# 感觉这个地方貌似只能解码3通道以上的图片

img = tf.image.resize\_images(img,size=(100,100))

# img = tf.reshape(img,shape=(50,50,4))

return img

img = load\_img(image)

print(img.shape)

image\_batch = tf.train.batch([img],batch\_size=20)

with tf.Session() as sess:

# initializer for num\_epochs

tf.local\_variables\_initializer().run()

coord = tf.train.Coordinator()

thread = tf.train.start\_queue\_runners(sess=sess,coord=coord)

try:

while not coord.should\_stop():

imgs = sess.run(image\_batch)

print(imgs.shape)

except tf.errors.OutOfRangeError:

print('done')

finally:

coord.request\_stop()

coord.join(thread)

三.使用tf.train.string\_input\_producer()

    tf.train.string\_input\_producer(path),传入路径时，不需要放入list中。然后加载图片的reader是tf.WholeFileReader(),其他地方和tf.train.slice\_input\_producer()函数用法基本类似。见代码：

#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Tue Mar 27 14:18:34 2018

@author: wsw

"""

# 用于通过读取图片的path,然后解码成图片数组的形式，最后返回batch个图片数组

import glob

import tensorflow as tf

path\_list = r'/media/wsw/文档/pythonfile\_withpycharm/SVMLearning/faceLibrary/人脸库/Yale2/'

img\_path = glob.glob(path\_list+'\*.bmp')

img\_path = tf.convert\_to\_tensor(img\_path,dtype=tf.string)

# 这里img\_path,不放在数组里面

# num\_epochs = 1,表示将文件下所有的图片都使用tf.decode\_raw

￼

tf.decode\_raw函数的意思是将原来编码为字符串类型的变量重新变回来，这个方法在数据集dataset中很常用，因为制作图片源数据一般写进tfrecord里用to\_bytes的形式，一次

# num\_epochs和tf.train.slice\_input\_producer()中是一样的

# 此参数可以用来设置训练的 epochs

image = tf.train.string\_input\_producer(img\_path,num\_epochs=1)

# load one image and decode img

def load\_img(path\_queue):

# 创建一个队列读取器，然后解码成数组

reader = tf.WholeFileReader()

key,value = reader.read(path\_queue)

img = tf.image.decode\_bmp(value,channels=3)

# 这里很有必要，否则会出错

# 感觉这个地方貌似只能解码3通道以上的图片

# img = tf.image.resize\_images(img,size=(100,100))

img = tf.reshape(img,shape=(224,224,3))

return img

img = load\_img(image)

print(img.shape)

image\_batch = tf.train.batch([img],batch\_size=20)

with tf.Session() as sess:

# initializer for num\_epochs

tf.local\_variables\_initializer().run()

coord = tf.train.Coordinator()

thread = tf.train.start\_queue\_runners(sess=sess,coord=coord)

try:

while not coord.should\_stop():

imgs = sess.run(image\_batch)

print(imgs.shape)

except tf.errors.OutOfRangeError:

print('done')

finally:

coord.request\_stop()

coord.join(thread)

---------------------

作者：AmibitionWei

来源：CSDN

原文：https://blog.csdn.net/qq\_30666517/article/details/79715045

版权声明：本文为博主原创文章，转载请附上博文链接！

# tf.decode\_raw和tf.cast

f.decode\_raw函数的意思是将原来编码为字符串类型的变量重新变回来，这个方法在数据集dataset中很常用，因为制作图片源数据一般写进tfrecord里用to\_bytes的形式，也就是字符串。这里将原始数据取出来 必须制定原始数据的格式，原始数据是什么格式这里解析必须是什么格式，要不然会出现形状的不对应问题！

例如元数据是tf.float64然后to\_bytes，但是用tf.decode\_raw解析的时候使用了tf.float32，那么形状跟值都会跟原始数据有差别，后面传入网络的时候一定会报tensorflow : Input to reshape is a tensor with 16384 values, but the requested shape has 49152 这种错误

tf.cast

这个函数主要用于数据类型的转变，不会改变原始数据的值还有形状的，

retyped\_images = tf.cast(decoded\_images, tf.float32)

labels = tf.cast(features['label'],tf.int32)

这里retyped\_images原来是tf.float64形状 labels是tf.uint8。tf.cast还可以用于将numpy数组转化为tensor.

# tensorflow中的协调器，tf.train.Coordinator和入队线程启动器tf.train.start\_queue\_runners

TensorFlow的Session对象是支持多线程的，可以在同一个会话（Session）中创建多个线程，并行执行。在Session中的所有线程都必须能被同步终止，异常必须能被正确捕获并报告，会话终止的时候， 队列必须能被正确地关闭。  
  
TensorFlow提供了两个类来实现对Session中多线程的管理：tf.Coordinator和 tf.QueueRunner，这两个类往往一起使用。

Coordinator类用来管理在Session中的多个线程，可以用来同时停止多个工作线程并且向那个在等待所有工作线程终止的程序报告异常，该线程捕获到这个异常之后就会终止所有线程。使用 tf.train.Coordinator()来创建一个线程管理器（协调器）对象。

QueueRunner类用来启动tensor的入队线程，可以用来启动多个工作线程同时将多个tensor（训练数据）推送入文件名称队列中，具体执行函数是 tf.train.start\_queue\_runners ， 只有调用 tf.train.start\_queue\_runners 之后，才会真正把tensor推入内存序列中，供计算单元调用，否则会由于内存序列为空，数据流图会处于一直等待状态。

1. 调用 tf.train.slice\_input\_producer，从 本地文件里抽取tensor，准备放入Filename Queue（文件名队列）中;
2. 调用 tf.train.batch，从文件名队列中提取tensor，使用单个或多个线程，准备放入文件队列;
3. 调用 tf.train.Coordinator() 来创建一个线程协调器，用来管理之后在Session中启动的所有线程;
4. 调用tf.train.start\_queue\_runners, 启动入队线程，由多个或单个线程，按照设定规则，把文件读入Filename Queue中。函数返回线程ID的列表，一般情况下，系统有多少个核，就会启动多少个入队线程（入队具体使用多少个线程在tf.train.batch中定义）;
5. 文件从 Filename Queue中读入内存队列的操作不用手动执行，由tf自动完成;
6. 调用sess.run 来启动数据出列和执行计算;
7. 使用 coord.should\_stop()来查询是否应该终止所有线程，当文件队列（queue）中的所有文件都已经读取出列的时候，会抛出一个 OutofRangeError 的异常，这时候就应该停止Sesson中的所有线程了;
8. 使用coord.request\_stop()来发出终止所有线程的命令，使用coord.join(threads)把线程加入主线程，等待threads结束。