人工智能之深度学习

TensorFlow基础

Steven Tang

TensorFlow简介

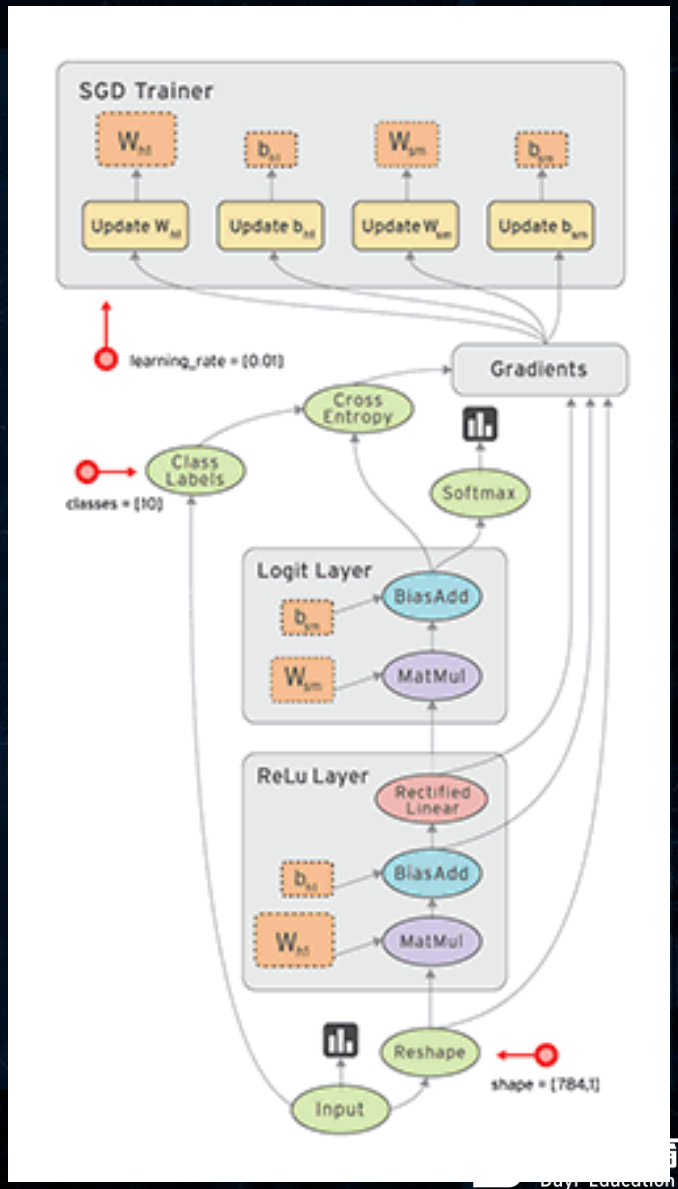
- 定义：TensorFlowTM is an open source software library for numerical computation using data flow graphs.
- GitHub：https://github.com/tensorflow/tensorflow
- Website：https://tensorflow.org/
- https://tensorflow.google.cn/

TensorFlow介绍

- TensorFlow™ 是一个采用数据流图（data flow graphs），用于数值计算的开源软件库。TensorFlow 最初由Google大脑小组（隶属于Google机器智能研究机构）的研究员和工程师们开发出来。2015年11月9日，Google发布人工智能系统TensorFlow并宣布开源。
- 其命名来源于本身的原理，Tensor（张量）意味着N维数组，Flow（流）意味着基于数据流图的计算。Tensorflow运行过程就是张量从图的一端流动到另一端的计算过程。张量从图中流过的直观图像是其取名为“TensorFlow”的原因。

What is Data Flow Graphs?

- 数据流图使用节点（Node）和线（Edges）的有向图描述数学计算；节点一般用来表示施加的数学操作，也可以表示数据输入(feed in)的起点和输出(push out)的终点，或者是读取/写入持久变量(persistent variable)的终点。线表示的是节点之间的输入/输出关系，这些线可以输运“size可动态调整”的多维数组，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。




TensorFlow
GANYMEDE AT
HALF PHASE

TensorFlow的特性

- 高度的灵活性：只要能够将计算表示成为一个数据流图，那么就可以使用TensorFlow。
- 可移植性：TensorFlow支持CPU和GPU的运算，并且可以运行在台式机、服务器、手机移动端设备等等。
- 自动求微分：TensorFlow内部实现了自动对于各种给定目标函数求导的方式。
- 多种语言支持：Python、C++
- 性能高度优化

CPU和GPU的区别

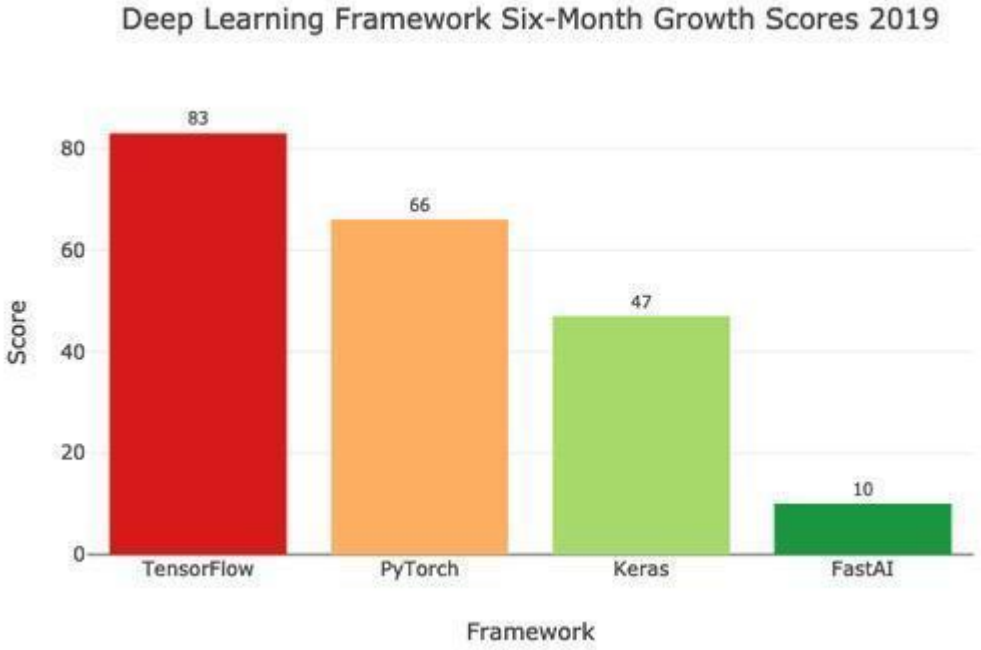
主流框架一览（为什么选择 Tensorflow？）

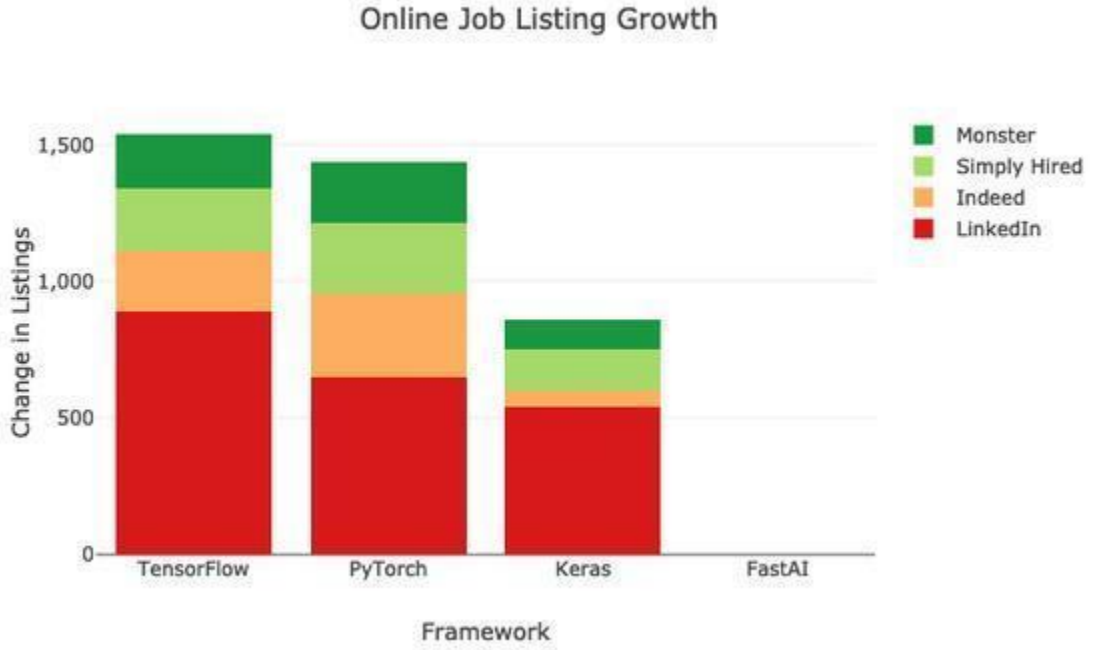
库名	发布者	支持语言	支持系统
TensorFlow	Google	Python/C++/ Java/Go	Linux/Mac/Windows OS/Android/iOS
Caffe	UC Berkeley	Python/C++/ Matlab	Linux/Mac OS/Windows
CNTK	Microsoft	Python/C++/ BrainScript	Linux/Windows
MXNet	DMLC (分布式机器学习社区)	Python/C++/Matlab/ Julia/Go/R/Scala	Linux/Mac OS/ Windows/Android/iOS
Pytorch	Facebook	Python	Linux/Mac OS/ Windows/Android/iOS
Theano	蒙特利尔大学	Python	Linux/Mac OS/Windows
Keras	François Chollet	Python	Linux/Windows

主流机器学习框架

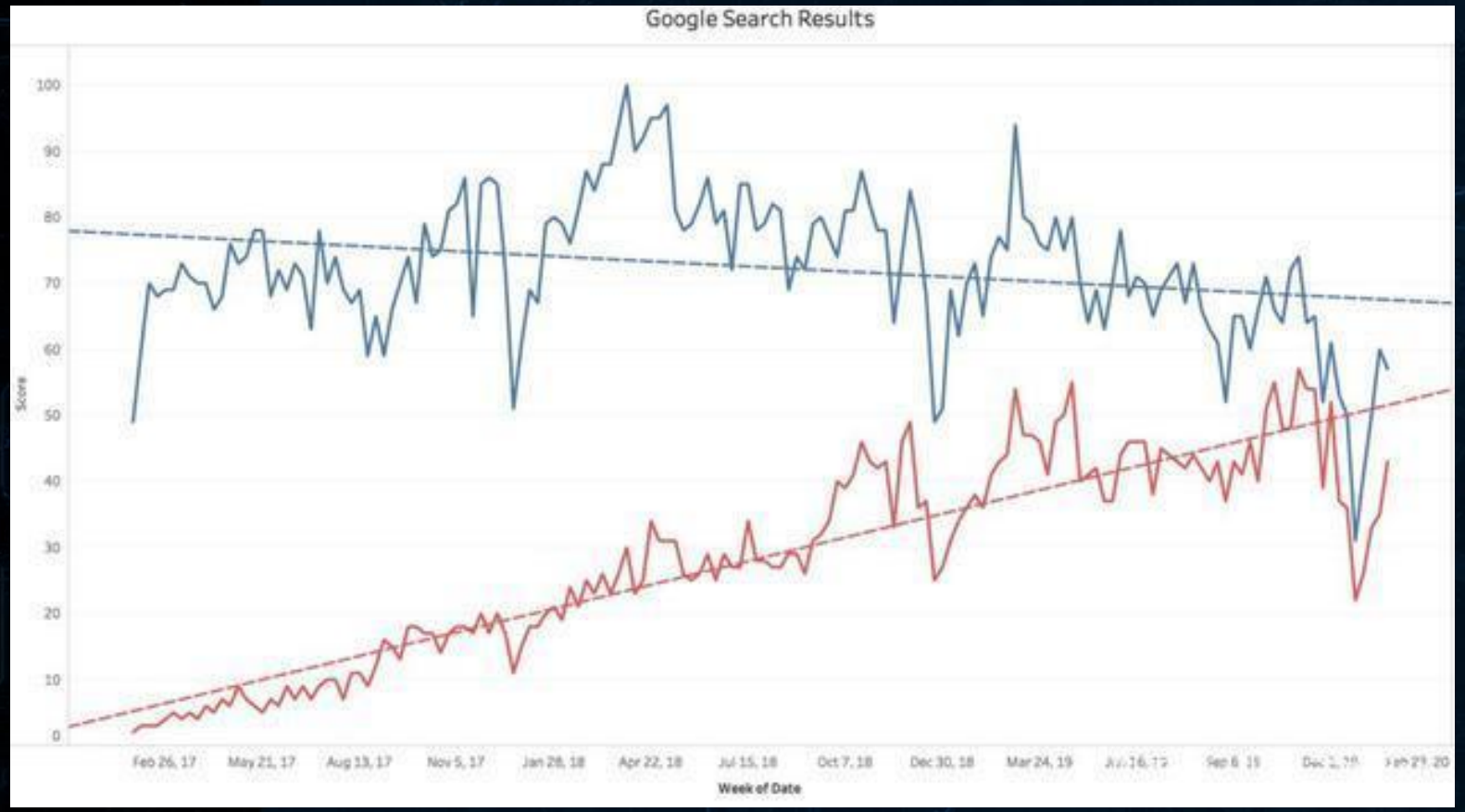
库名	学习材料丰富程度	CNN建模能力	RNN建模能力	易用程度	运行速度	多GPU支持程度
TensorFlow	★★★	★★★	★★	★★★	★★	★★
Caffe	★	★★	★	★	★	★
CNTK	★	★★★	★★★	★	★★	★
MXNet	★★	★★	★	★★	★★	★★★
Pytorch	★★	★★★	★★	★★	★★★	★★
Theano	★★	★★	★★	★	★★	★★
Keras	★★★	★★	★★★	★★★	★	★★

主流机器学习框架





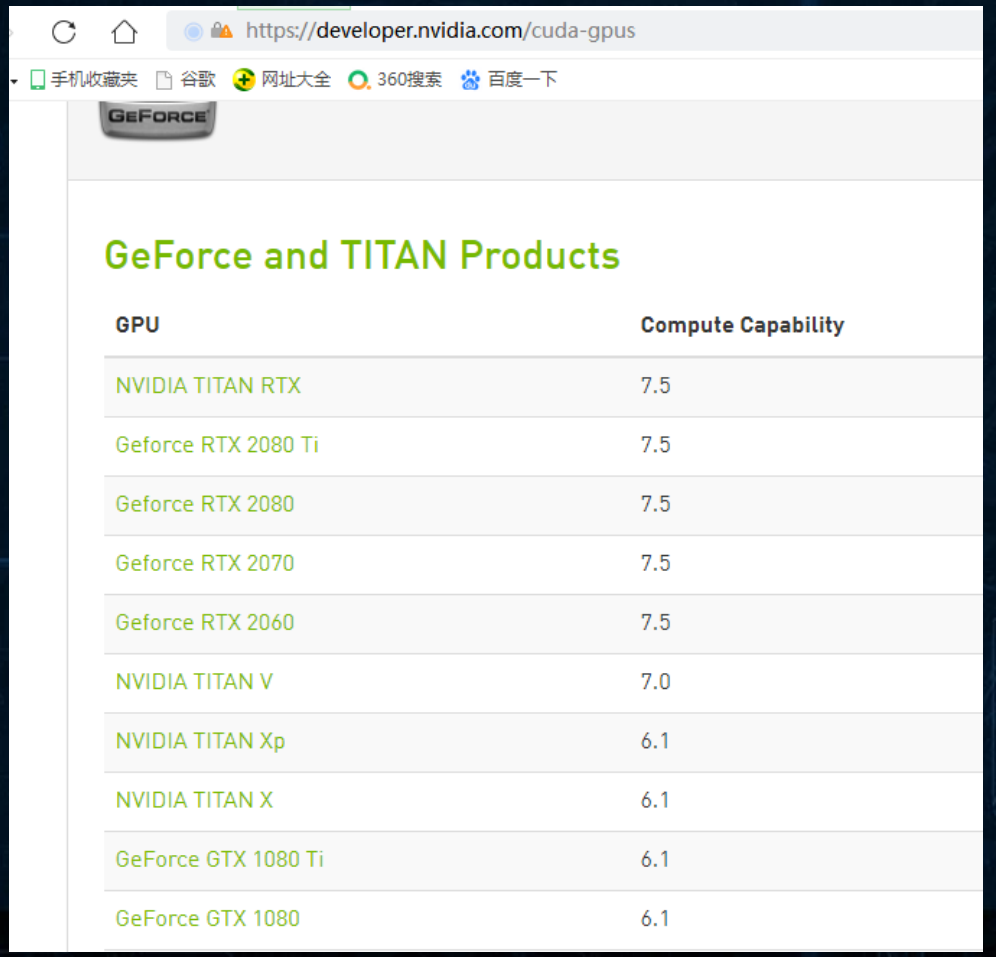
主流机器学习框架



Tensorflow 版本变迁

TensorFlow安装

- 首先确定是否能够安装GPU版本的tensorflow
- https://developer.nvidia.com/cuda-gpus



TensorFlow安装

- python -m pip install tensorflow_gpu -i https://pypi.douban.com/simple
- tensorflow_gpu默认镜像版本为2.2版本
- 建议安装python -m pip install tensorflow==1.13.1 -i https://pypi.douban.com/simple
- python -m pip install tensorflow-i https://pypi.douban.com/simple
- tensorflow默认镜像版本为1.15版本

TensorFlow基本概念

- 图（Graph）：图描述了计算的过程，TensorFlow使用图来表示计算任务。
- 张量（Tensor）：TensorFlow使用tensor表示数据。每个Tensor是一个类型化的多维数组。
- 操作（op）：图中的节点被称为op（opearation的缩写），一个op获得/输入0个或多个Tensor，执行计算，产生0个或多个Tensor。

TensorFlow基本概念

占位符(Placeholder)：数据输入的入口，运行中不会被改变。

变量（Variable）：运行过程中可以被改变，通常用来存放网络的权值和偏置。

会话（Session）：图必须在称之为“会话”的上下文中执行。会话将图的op分发到诸如CPU或GPU之类的设备上执行。

数据属性

数据类型	Python类型	描述
DT_FLOAT	tf.float32	32位浮点型
DT_DOUBLE	tf.float64	64位浮点型
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	用于量化操作的32位有符号整型
DT_QINT8	tf.qint8	用于量化操作的8位有符号整型
DT_QUINT8	tf.quint8	用于量化操作的8位无符号整型

节点

- 节点又称为算子，它代表一个操作，一般用来表示施加的数字运算，也可以表示数据输入的起点以及输出的重点，或者是读取/写出持久化变量的终点。

操作组	操作
Maths	Add, Sub, Mul, Div, Exp, Log, Greater, Less, Equal
Array	Concat, Slice, Split, Constant, Rank, Shape, Shuffle
Matrix	MatMul, MatrixInverse, MatrixDeterminant
Neuronal Network	SoftMax, Sigmoid, ReLU, Convolution2D, MaxPool
Checkpointing	Save, Restore
Queues and syncronizations	Enqueue, Dequeue, MutexAcquire, MutexRelease
Flow control	Merge, Switch, Enter, Leave, NextIteration

一个简单的例子

```
import tensorflow as tf
# 需要你编程：将下面转换成tensorflow
x = 10
y = 2
z = x/y - 1
x=tf.placeholder(tf.int32)
y=tf.placeholder(tf.int32)
z=tf.subtract(tf.div(x,y),tf.constant(1))
# 需要你编程：从session中打印 z
with tf.Session() as sess:
    output=sess.run(z,feed_dict={x:10,y:2})
    print(output)
```

TensorFlow基本用法

- 使用TensorFlow必须理解下列概念：
  - 使用图(graph)来表示计算任务；
  - 在会话(session)的上下文中执行图；
  - 使用placeholder来接收数据；
  - 通过变量(Variable)来维护状态；
  - 使用feed来喂数据，使用run来读数据。

TensorFlow程序结构

- TensorFlow的程序一般分为两个阶段：构建阶段和执行阶段；
- 构建阶段：op的执行步骤被描述称为一个图，然后使用TensorFlow提供的API构建这个图。
- 执行阶段：将构建好的执行图(Operation Graph)在给定的会话中执行，并得到执行结果。

TensorFlow图

- TensorFlow编程的重点是根据业务需求，使用TensorFlow的API将业务转换为执行图（有向无环图）；图中的节点是Tensor，节点之间的连线是节点之间的操作，连线前的节点可以认为是操作的输入，连线后的节点可以认为操作的输出；根据节点的特性（是否有输入输出），可以将节点分为源节点、中间节点和最终的结果节点。
- 图构建的第一步就是创建源op(source op)；源op不需要任何的输入。op构造器的返回值代表被构造出的op的输出，这些返回值可以传递给其它op构造器作为输入或者直接获取结果。
- TensorFlow库中有一个默认图(default graph)，op构造器可以直接为其添加节点，一般情况下，使用默认的Graph即可完成程序代码的实现。不过TensorFlow也支持通过Graph类管理多个图。

TensorFlow图

- 不使用默认图(Graph)，使用多个图来进行编程；但是注意：操作必须属于同一个图，不同图中的节点不能相连。

```
import tensorflow as tf
a=tf.constant(4.0)
print("a是否在默认图上:{}".format(a.graph == tf.get_default_graph()))

g=tf.Graph()
with g.as_default():
    b=tf.constant(3.0)
    c=tf.constant(2.0)
    print("b是否在默认图上:{}".format(b.graph == tf.get_default_graph()))
    print("c是否在默认图上:{}".format(c.graph == tf.get_default_graph()))

with tf.Graph().as_default() as g2:
    d=tf.constant(4.0)
    print("d是否在默认图上:{}".format(d.graph == g2))
    print("d是否在默认图上:{}".format(d.graph == g))

e=tf.add(tf.add(a,b),tf.add(c,d))
```

TensorFlow会话

- 当执行图构建完成后，才能给启动图，进入到执行阶段；启动图的第一步就是创建一个Session对象，如果无任何参数的情况下，会话构造器将启动默认图。

```
import tensorflow as tf

a=tf.constant(3)
b=tf.constant(4)
c=tf.multiply(a,b)

sess=tf.InteractiveSession()

print(c.eval())

with tf.Session() as sess:
    print(sess.run(c))
```

TensorFlow会话

- tf.Session在构建会话的时候，如果不给定任何参数，那么构建出来Session对应的内部的Graph其实就是默认Graph，不过我们可以通过参数给定具体对应的是那一个Graph以及当前Session对应的配合参数。Session的构造主要有三个参数，作用如下：
  - target：给定连接的url，只有当分布式运行的时候需要给定；
  - graph：给定当前Session对应的图，默认为TensorFlow中的默认图；
  - config：给定当前Session的相关参数，参数详见

TensorFlow会话

- 通过Session的config参数可以对TensorFlow的应用的执行进行一些优化调整，主要涉及到的参数如下：

属性	作用
gpu_options	GPU相关参数，主要参数：per_process_gpu_memory_fraction和allow_growth
allow_soft_placement	是否允许动态使用CPU和GPU，默认为False；当我们的安装方式为GPU的时候，建议该参数设置为True，因为TensorFlow中的部分op只能在CPU上运行。
log_device_placement	是否打印日志，默认为False，不打印日志
graph_options	Graph优化相关参数，一般不需要给定，默认即可，主要参数：optimizer_options(do_common_subexpression_elimination、do_constant_folding和opt_level)

TensorFlowGPU设置

```
import tensorflow as tf

a=tf.constant('1',tf.string)

b=tf.string_to_number(a,tf.float64)

c=tf.to_double(5.0)

d=tf.add(b,c)

gpu_options=tf.GPUOptions()
gpu_options.per_process_gpu_memory_fraction=0.5
gpu_options.allow_growth=True

configproto=tf.ConfigProto(gpu_options=gpu_options)

with tf.Session(config=configproto) as sess:

    print(sess.run(d))
```

TensorFlow会话

- 在TensorFlow中，除了可以使用Session表示这个会话外，还可以通过InteractiveSession来表示会话，InteractiveSession的意思是：交互式会话，使用交互式会话可以降低代码的复杂度。

```
import tensorflow as tf

a=tf.constant(3)


b=tf.constant(4)

c=tf.multiply(a,b)

sess=tf.InteractiveSession()

print(c.eval())


with tf.Session() as sess:
    print(sess.run(c))
```

Tensor张量

- TensorFlow使用Tensor数据结构来代表所有数据，计算图中，操作间传递的数据都是Tensor。Tensor可以看作是一个n维的数组或者列表，一个Tensor主要由一个静态数据类型和动态类型的维数(Rank、Shape)组成。Tensor可以在图中的节点之间流通。

阶	shape	数据	维数
0	()	标量，一个数：0	()
1	(d_0,)	一维向量[1,2,3]	(3,)
2	(d_0, d_1)	二维矩阵[[1,2,3],[2,3,4]]	(2,3)
3	(d_0, d_1, d_2)	三维张量[[[1,2],[1,2]],[[1,2],[1,2]]]	(2,2,2)
n	(d_0, d_1, d_2,.... d_{n-1})	...	...

TensorFlow变量

- 变量(Variables)是维护图执行过程中的状态信息。在训练模型过程中，可以通过变量来存储和更新参数。变量包含张量(Tensor)存放于内存的缓存区。建模的时候变量必须被明确的初始化，模型训练后变量必须被存储到磁盘。这些变量的值可以在之后的模型训练和分析中被加载。
- 在构建变量的时候，必须将一个张量或者可以转化为张量的Python对象作为初始值传入构造函数Variable中。
- 特别注意：变量在run之前需要进行初始化(tf.global_variables_initializer())。

TensorFlow变量

- 一个简单的使用变量的案例

```
import tensorflow as tf

v1=tf.Variable(tf.random_normal([10],stddev=0.5,dtype=tf.float32),name='v1')
v2=tf.Variable(tf.random_normal([5],stddev=10,dtype=tf.float32),name='v2')
v3=tf.Variable(4.0)
v4=tf.Variable([4,5])

init=tf.global_variables_initializer()

with tf.Session() as sess:

    sess.run(init)
  #  out1,out2=sess.run([v1,v2])
    out4=sess.run(v4)
```

TensorFlow Fetch

- 为了取回操作的输出内容，可以在使用Session对象的run方法调用执行图的时候，传入一些tensor，通过run方法就可以获取这些tensor对应的结果值。
- 如果需要获取多个tensor的值，那么尽量一次运行就获取所有的结果值，而不是采用逐个获取的方式。

```
import tensorflow as tf

a=tf.constant(3)
b=tf.constant(4)
c=tf.multiply(a,b)



with tf.Session() as sess:
    output=sess.run([a,b,c])
    output1, output2, output3=sess.run([a,b,c])

    print(output)
```

TensorFlow Feed

- Tensorflow还提供了填充机制(feed)，可以在构建图时使用placeholder类型的API临时替代任意操作的张量(占位符)，在调用Session对象的run()方法去执行图时，使用填充数据作为调用的参数，调用结束后，填充数据就消失了。
- feed使用一个tensor值临时替换一个操作的输出结果，在获取数据的时候必须给定对应的feed数据作为参数。feed只有在调用它的方法内有效，方法结束，feed就消失了。
- feed可以使用placeholder类型的API创建占位符，常见API：tf.placeholder、tf.placeholder_with_default

TensorFlow Feed

```
import tensorflow as tf

m1=tf.placeholder(tf.float32)
m2=tf.placeholder(tf.float32)
m3=tf.placeholder_with_default(4.0, shape=None)

output1=tf.multiply(m1,m2)
output2=tf.multiply(m1,m3)



with tf.Session() as sess:

    print(sess.run(output1,feed_dict={m1:2,m2:3}))
    print(sess.run(output2,feed_dict={m1:2}))
    print(sess.run(output2,feed_dict={m1:2,m3:3}))
```

TensorFlow 案例一

- 实现一个累加器，并且每一步均输出累加器的结果值。

TensorFlow 控制依赖

- 我们可以通过Variable和assign完成变量的定义和更新，但是如果在更新变量之前需要更新其它变量，那么会导致一个比较严重的问题：也就是需要多次调用sess.run方法来进行变量的更新。通过这种方式，代码复杂程度上升，同时也没有执行效率。
- 解决该问题的方案就是：控制依赖。通过TensorFlow中提供的一组函数来处理不完全依赖的情况下的操作排序问题(即给定哪个操作先执行的问题)， 通过tf.control_dependencies API完成。

TensorFlow 设备

- 设备是指一块可以用来运算并且拥有自己的地址空间的硬件，如CPU和GPU。Tensorflow为了在执行操作的时候，充分利用计算资源，可以明确指定操作在哪个设备上执行。
- 一般情况下，不需要显示指定使用CPU还是GPU，TensorFlow会自动检测。如果检测到GPU，TensorFlow会尽可能地利用第一个GPU来执行操作。注意：如果机器上有超过一个可用的GPU，那么除了第一个外其它GPU默认是不参与计算的。所以，在实际TensorFlow编程中，经常需要明确给定使用的CPU和GPU。
  - “/cpu:0”：表示使用机器CPU运算
  - “/gpu:0”：表示使用第一个GPU运算，如果有的话
  - “/gpu:1”：表示使用第二个GPU运算，以此类推

```
import tensorflow as tf

with tf.device('/gpu:0'):
    a=tf.constant([2,3,4])
    b=tf.constant(3)
    c=tf.multiply(a,b)


with tf.Session() as sess:
    print(sess.run(c))
```

TensorFlow 变量作用域（两套命名系统）

- Tensorflow中有两个作用域，一个是name_scope，另一个是variable_scope。
- 变量作用域机制在TensorFlow中主要通过两部分组成：
    - tf.get_variable：通过所给定的名字创建或者返回一个对应的变量
    - tf.variable_scope：为通过创建的变量或者操作Operation指定命名空间

```
import tensorflow as tf

tf.reset_default_graph()
with tf.Session() as sess:
    with tf.variable_scope('foo') as foo_scope:
        v=tf.get_variable('v',[1])
        w=tf.get_variable('w',[1])
        with tf.variable_scope('bar'):
            l=tf.get_variable('l',[2])
            with tf.variable_scope('test'):
                h=tf.get_variable('h',[1],\
                                  initializer=tf.random_normal_initializer())
                g=w+v+h+l[0]
    sess.run(tf.global_variables_initializer())
    print("{} {}".format(v.name,v.eval()))
    print("{} {}".format(w.name,w.eval()))
    print("{} {}".format(h.name,h.eval()))
    print("{} {}".format(g.name,g.eval()))
```

TensorFlow 变量作用域

```python
import tensorflow as tf

tf.reset_default_graph()
with tf.name_scope('duyi'):
    var_1 = tf.Variable(initial_value=[0], name='var_1')
    var_2 = tf.Variable(initial_value=[0], name='var_1')
    var_3 = tf.Variable(initial_value=[0], name='var_1')
print(var_1.name)
print(var_2.name)
print(var_3.name)

with tf.name_scope('AI'):
    var_1 = tf.Variable(initial_value=[0], name='var_1')

#with tf.variable_scope('test'):
print(var_1.name)

with tf.variable_scope('deeplearning'):
    var_2 = tf.get_variable(name='var_2', shape=[1, ])

    var_3 = tf.Variable(initial_value=[0], name='var_3')
    var_4 = tf.get_variable(name='var_4', shape=[1, ])

print(var_2.name)

print(var_3.name)
print(var_4.name)
```

TensorFlow 变量作用域

- tf.get_variable常用的initializer初始化器：

初始化器	描述
tf.constant_initializer(value)	初始化为给定的常数值value
tf.random_uniform_initializer(a, b)	初始化为从a到b的均匀分布的随机值
tf.random_normal_initializer(mean, stddev)	初始化为均值为mean、方差为stddev的服从高斯分布的随机值
tf.orthogonal_initializer(gini=1.0)	初始化一个正交矩阵，gini参数作用是最终返回的矩阵是随机矩阵乘以gini的结果
tf.identity_initializer(gini=1.0)	初始化一个单位矩阵，gini参数作用是最终返回的矩阵是随机矩阵乘以gini的结果

TensorFlow 中的条件判断

```
import tensorflow as tf
a=tf.constant(2)
b=tf.constant(3)
x=tf.constant(4)
y=tf.constant(5)
z = tf.multiply(a, b)
result = tf.cond(x < y, lambda: tf.add(x, z), lambda: tf.square(y))
#true_fn   ,false_fn
with tf.Session() as session:
    print(result.eval())
```

TensorFlow 中的条件判断

```
import tensorflow as tf
def f1():
    return tf.constant(17)
def f2():
    return tf.constant(23)
def f3():
    return tf.constant(-1)
x = 2
y = 3
r = tf.case({tf.less(x, y): f1, tf.greater(x, y): f2},\
              default=f3, exclusive=True)
sess = tf.Session()
print(sess.run(r))#17
```

TensorFlow 中的循环

```
import tensorflow as tf
a = tf.constant(10)
n = tf.constant(10)
def cond(a, n):
    return  a< n
def body(a, n):
    a = a + 1
    return a, n
a, n = tf.while_loop(cond, body, [a, n])
with tf.Session() as sess:
    tf.global_variables_initializer().run()
    res = sess.run([a, n])
print(res)
```

```
i= 0
n =10
while(i < n):
    i = i +1
```

谢谢