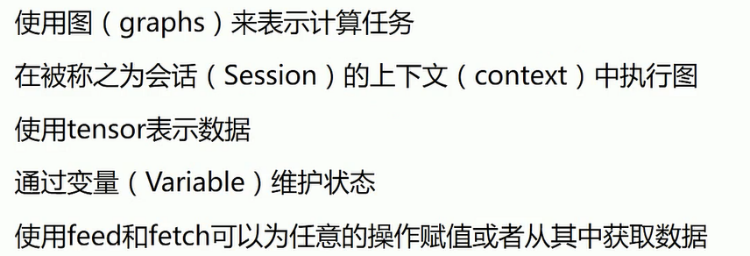
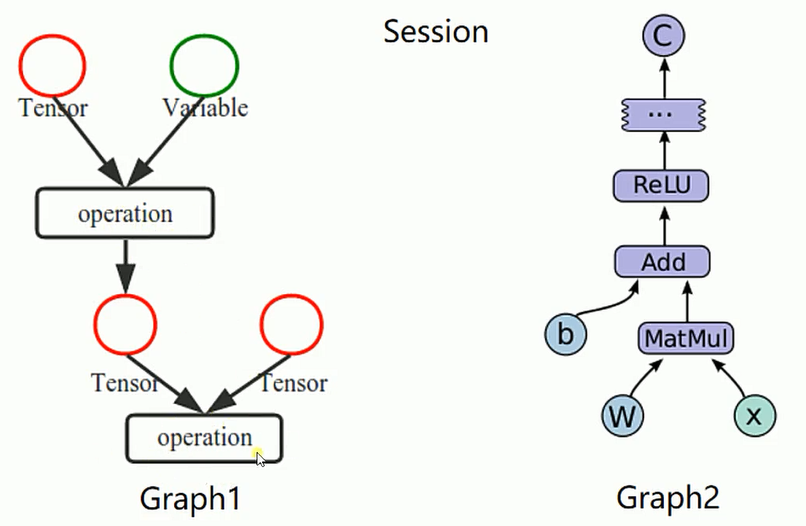
1. 基本概念
2. **概念**



**基本操作流程：（计算流程）**



1. **简单的创建会话：**

**（1）实例1：**

**import** tensorflow **as** tf  
*#创建常量op*m1 = tf.constant([[2,3]])  
m2 = tf.constant([[2],[3]])  
*#创建矩阵乘法op*matmul = tf.matmul(m1,m2)  
**with** tf.Session() **as** sess:  
 result = sess.run(matmul)  
 print(result)

#result = 13

**（2）实例2：**

*#实现1-5求和  
#初始化变量=0*state = tf.Variable(0,name = **'counter'**)  
*#创建op使state+1*newAdd = tf.add(state, 1)  
*#赋值op，state = state + 1*update = tf.assign(state, newAdd)  
*#初始化变量*init = tf.global\_variables\_initializer()  
**with** tf.Session() **as** sess:  
 sess.run(init)  
 **for** \_ **in** range(5):  
 sess.run(update)  
 print(sess.run(state))

1. **使用fetch来运行多个op**：使用[ ]，来运行多个op

eg:

sess.run( [ op1,op2 ] )

1. **创建占位符：placeholder**

**feed\_dict的数据需要用字典形式传入**

**eg：**

#None:可以使任意值，可用于批量梯度下降时，指定不同的训练数据集

m1 = tf.placeholder(tf.float32,[None,100])

eg：

#实现两数乘法

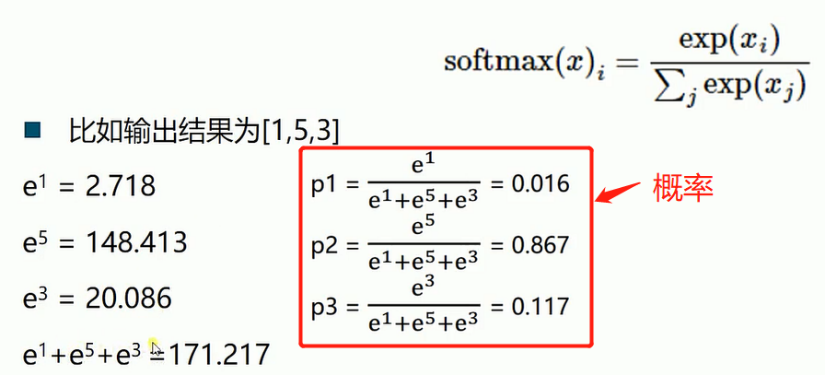
m1 = tf.placeholder(tf.float32)  
m2 = tf.placeholder(tf.float32)  
mul = tf.multiply(m1, m2)  
**with** tf.Session() **as** sess:  
 print(sess.run(mul, feed\_dict = { m1:[7.], m2:[2.] }))

1. **Tensorflow使用图模型的好处**
2. 图模型的最大好处是节约系统开销，提高资源的利用率，可以更加高效的进行运算。因为我们在图的执行阶段，只需要运行需要的op,这样提高了资源的利用率。
3. 这种结构有利于提取中间某些节点的结果，方便以后利用中间的节点去进行其它运算；
4. 这种结构对分布式运算更加友好，运算的过程可以分配给多个CPU或是GPU同时进行，提高运算效率；
5. 图模型把运算分解成了很多个子环节，所以这种结构使得求导更加方便。
7. **基于tensorflow的简单分类模型**
8. **使用tensorflow做简单的线性回归：**

*#生成随机数据100个*x = np.random.rand(100)  
noise = np.random.rand(100) / 10  
yData = x \* 1 + 0.5 + noise  
*#构造模型*k = tf.Variable(2.1)  
b = tf.Variable(1.1)  
y = k \* x + b  
*#代价函数，reduce\_mean：计算平均值*cost = tf.reduce\_mean(tf.square( y - yData ))  
*#定义梯度下降法优化器*optimizer = tf.train.GradientDescentOptimizer(0.2)  
*#最优化参数*fit = optimizer.minimize(cost)  
*#初始化变量*init = tf.global\_variables\_initializer()

**with** tf.Session() **as** sess:  
 sess.run(init)  
 **for** step **in** range(200):  
 sess.run(fit)  
 **if** step%20 == 0:  
 print(step, sess.run([k, b]))  
 plt.scatter(x,yData,color=**'r'**,s=20)  
 plt.plot(x, sess.run(k\*x+b),ls = **"-"**, lw=2 ,c = **'b'**)  
 plt.show()

1. **softmax模型：给不同对象分配概率：**（公式如下）



3. **Tensorboard**
4. **打开tensorboard的结果文件：**

在cmd里，使用命令：tensorboard --logdir=结果文件的路径

（复制网址，即可在浏览器中看到结果）

**eg：**

（1）*#参数概要，用于计算传入的参数的各种数值***def** variable\_summaries(var):  
 **with** tf.name\_scope(**'summaries'**):  
 mean = tf.reduce\_mean(var)  
 tf.summary.scalar(**'mean'**, mean)*#平均值* **with** tf.name\_scope(**'stddev'**):  
 stddev = tf.sqrt(tf.reduce\_mean(tf.square(var - mean)))  
 tf.summary.scalar(**'stddev'**, stddev)*#标准差* tf.summary.scalar(**'max'**, tf.reduce\_max(var))*#最大值* tf.summary.scalar(**'min'**, tf.reduce\_min(var))*#最小值* tf.summary.histogram(**'histogram'**, var)*#直方图*

（2）使用（1）的函数

**with** tf.name\_scope(**'layer'**):  
 **with** tf.name\_scope(**'weights'**):  
 weight = tf.Variable(tf.zeros([784,10]))  
 variable\_summaries(weight)

（3）使用summary.scalar()计算参数变量的和

**with** tf.name\_scope(**'cost'**):  
 cost = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits \

(labels=y,logits=prediction))  
 tf.summary.scalar(**'cost'**,cost)

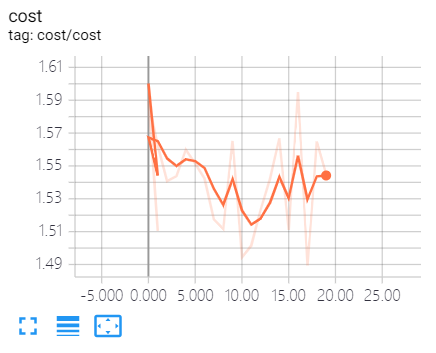
（4）*#合并所有的summary*merged = tf.summary.merge\_all()

（5）具体实现：

with tf.Session() as sess:

……

summary, \_ = sess.run([merged, fit], feed\_dict={x: batch\_x, y: batch\_y})  
writer.add\_summary(summary, fitNum)

(6)结果：

3. **常用函数**
4. **tf.argmax(input, axis=None, name=None, dimension=None)**  
   **功能：**对矩阵按行或列计算最大值，**输出最大值的下标（序号）**

input：输入Tensor（矩阵）  
axis：0表示按列，1表示按行  
name：名称  
dimension：和axis功能一样，默认axis取值优先。

**eg：**

tf.argmax( y, 1 )

1. **tf.equal()**

**功能：**对比这两个矩阵或者向量的相等的元素，如果是相等的那就返回True，反正返回False，即：预测结果正确的数量。

**eg：**

import tensorflow as tf

import numpy as np

A = [[1,3,4,5,6]]

B = [[1,3,4,3,2]]

with tf.Session() as sess:

print(sess.run(tf.equal(A, B)))

**结果**：[[ True True True False False]]

**（注：**一般来说tf.argmax与tf.equal联合使用

eg: tf.equal(tf.argmax(y, 1), tf.argmax(y\_, 1)) ）

1. **tf.cast(x, dtype, name=None)**

**功能：**实现数据类型转换

x：输入

dtype：转换目标类型

name：名称

**eg:**转换为float32类型的数据

tf.cast(result, tf.float32)

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

**功能：**占位符，可理解为形参，在实际使用中再进行赋值

* dtype：类型：tf.float32，tf.float64
* shape：形状。默认：一维，也可以是多维， [None, 3]表示列是3，行不定
* name：名称

eg：

x = tf.placeholder(tf.float32, [**None**,784])

1. **交叉熵代价函数**

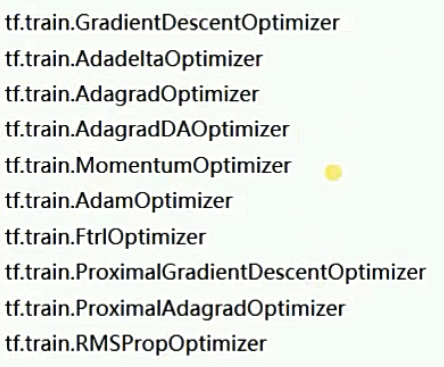
tf.nn.softmax\_cross\_entropy\_with\_logits(labels=y,logits=prediction)

1. **初始化参数的方式：**

初始化参数为正态分布的数：（stddev，设置标准差）

weight = tf.Variable(tf.truncated\_normal([784,2000],stddev=0.1))

1. **tensorflow中的优化器：**



1. **tensorflow保存函数、模型、权重参数**

……

saver = tf.train.Saver()

with tf.Session() as sess:

……

#保存路径:当前文件夹+ net/my\_net.ckpt

saver.restore(sess,'net/my\_net.ckpt')

1. inception-V3

Google的图像处理架构包

'http://download.tensorflow.org/models/image/imagenet/inception-2015-12-05.tgz'

附：讲解文章：

<https://blog.csdn.net/u012426298/article/details/80854852>

1. **保存、读取训练结果：**

使用Saver（）、restore（），仅用于保存训练的结果。

**import** tensorflow **as** tf  
  
var1 = tf.Variable([1,3],name=**'v1'**)  
var2 = tf.Variable([2,4],name=**'v2'**)  
init = tf.global\_variables\_initializer()  
*#保存数据*saver = tf.train.Saver()  
*#读取数据*modeFile = tf.train.latest\_checkpoint(**'./'**)  
**with** tf.Session() **as** sess:  
 *#sess.run(init)*

*#保存  
 # save\_path = saver.save(sess,"./save.ckpt")*

*#读取，*

*#1、要用和保存变量相同的Saver对象来恢复变量*

*#2、不需要事先对变量进行初始化* print(saver.restore(sess,modeFile))