# 第二周 Tensorflow的基本概念

## 1、基本概念

* 使用图（graphs）来表示计算任务
* 在被称之为会话（Session）的上下文（context）中执行图
* 使用tensor表示数据
* 通过变量Variable维护状态
* 使用feed和fetch可以为任意的操作赋值或者从其中获取数据

Tensorflow是一个编程系统，使用图（graphs）来表示计算任务，图（graphs）中的节点称之为operation，（add，mat之类），一个op获得0个或者多个tensor，执行计算，产生0个或多个tensor，tensor看作是一个n维的数组或列表。图必须在会话里被启动。



## 2、基本代码（变量、常量、会话、op节点、图）

# 声明变量

x = tf.Variable([1,2])

# 声明常量

a = tf.constant([3,3])

# 增加一个减法op

sub = tf.subtract(x,a)

# 增加一个加法op

add = tf.add(a,sub)

# 全局变量初始化

init = tf.global\_variables\_initializer()

# 在图内运行代码

with tf.Session() as sess:

sess.run(init)

print(x.value)

print(sess.run(sub))

print(sess.run(add))

# 赋值op

#update = tf.assign(state,new\_value)

## 3、Fetch and Feed

Fetch可以在一个会话中同时执行多个op,也就是说可以同事进行多个节点的计算

Feed 可以每次为网络传入不同的数据

代码示例：

# Fetch

input1 = tf.constant(3.0)

input2 = tf.constant(2.0)

input3 = tf.constant(5.0)

add = tf.add(input2,input3)

mul = tf.multiply(input1,add)

with tf.Session() as sess:

result = sess.run([mul, add])

print((result))

输出了两个数字

# Feed

# 创建占位符

input1 = tf.placeholder(tf.float32)

input2 = tf.placeholder(tf.float32)

output = tf.multiply(input1, input2) #元素级别相乘

with tf.Session() as sess:

# feed数据以字典形式传入

print(sess.run(output, feed\_dict={input1:[2.0],input2:[6.0]}))

有了占位符和feed，网络可以每次传入不同的数据

## 4、简单示例

import tensorflow as tf

import numpy as np

# 生成随机点

x\_data = np.random.rand(100)

y\_data = x\_data\*0.1 + 0.2

# 构建线性模型

b = tf.Variable(0.)

k = tf.Variable(0.)

y = k\*x\_data + b

# 定义损失函数

loss = tf.reduce\_mean(tf.square(y\_data-y))

# 定义一个梯度下降法进行训练的优化器,学习率是0.2

optimizer = tf.train.GradientDescentOptimizer(0.2)

# 最小化代价函数

train = optimizer.minimize(loss)

# 变量初始化

init = tf.global\_variables\_initializer()

with tf.Session() as sess:

sess.run(init)

for step in range(201):

sess.run(train)

if step%20 == 0:

print(step+1, sess.run([k,b]))

# 第三周 Tensorflow非线性回归以及分类的简单使用，softmax介绍

1、非线性回归简单示例

import tensorflow as tf

import numpy as np

import matplotlib.pyplot as plt

# 生成200个随机点，并改变其形状为200\*1

x\_data = np.linspace(-0.5, 0.5, 200)[:,np.newaxis]

noise = np.random.normal(0,0.02,x\_data.shape)

y\_data = np.square(x\_data) + noise

#查看一下数据形状

print(x\_data.shape)

type(x\_data)

print(noise.shape)

# 定义两个placeholder

x = tf.placeholder(tf.float32, [None,1])

y = tf.placeholder(tf.float32, [None,1])

# 定义中间层

Weights\_L1 = tf.Variable(tf.random\_normal([1,10]))

bias\_L1 = tf.Variable(tf.zeros([1,10]))

Wx\_plus\_b\_L1 = tf.matmul(x, Weights\_L1) + bias\_L1

# 激活函数

L1 = tf.nn.tanh(Wx\_plus\_b\_L1)

# 定义输出层

Weights\_L2 = tf.Variable(tf.random\_normal([10,1]))

bias\_L2 = tf.Variable(tf.zeros([1,1]))

Wx\_plus\_b\_L2 = tf.matmul(L1,Weights\_L2) + bias\_L2

prediction = tf.nn.tanh(Wx\_plus\_b\_L2)

# 二次代价函数（损失函数）

loss = tf.reduce\_mean(tf.square(y-prediction))

# 梯度下降法

train\_step = tf.train.GradientDescentOptimizer(0.1).minimize(loss)

with tf.Session() as sess:

# 变量的初始化

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

for \_ in range(2000):

sess.run(train\_step, feed\_dict={x:x\_data, y:y\_data})

# 获得预测值

prediction\_value = sess.run(prediction,feed\_dict={x:x\_data})

plt.figure()

plt.scatter(x\_data, y\_data)

plt.plot(x\_data, prediction\_value,'r-', lw=5)

plt.show()

# 第四周 交叉熵(cross-entropy)，过拟合，dropout以及Tensorflow中各种优化器的介绍

## 1、代价函数的选择（关于交叉熵代价函数和对数释然代价函数

如果输出神经元是线性的，二次代价函数是合适的选择，

如果输出神经元是S型函数，适用交叉熵代价函数。（这里指的是输出层被sigmoid函数激活的情况）

如果输出层被sigmoid激活，可采用交叉熵代价函数，如果使用softmax作为网络最后一层，此时常用对数释然代价函数。

对数似然代价函数与softmax的组合，跟交叉熵与sigmoid函数的组合非常相似。在二分类问题中，对数释然代价函数简化为交叉熵代价函数的形势。

在Tensorflow中

Tf.sigmoid\_cross\_entropy\_with\_logits()表示跟sigmoid搭配使用的交叉熵

Tf.softmax\_corss\_entropy\_with\_logits()表示跟softmax搭配使用的交叉熵。

## 2、过拟合

解决过拟合的操作

增加数据集、

正则化方法、减小loss，从而减小学习率

Dropout、随机丢弃某些神经节点，使网络不能过度依赖某些节点

## 3、Optimizer

各种优化器对比

标准梯度下降法：

计算所有样本汇总误差，根据总误差来更新权值；

缺点：太慢；

随机梯度下降法：

随机抽取一个样本来计算误差，然后更新权值；

缺点：对噪声过于敏感

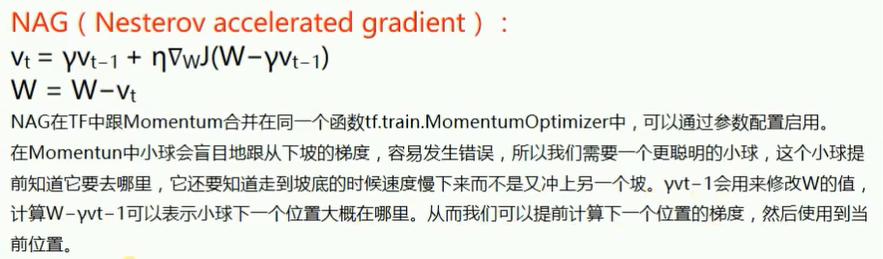
批量梯度下降法：

算是一种折中方案，从总样本中选取一个批次，计算该批次数据误差，来更新权值；

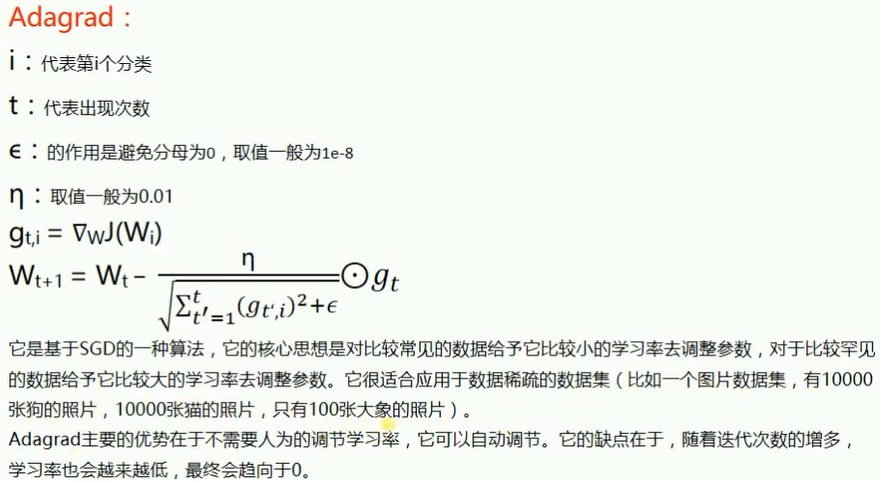
### SGD Momentum



### NAG

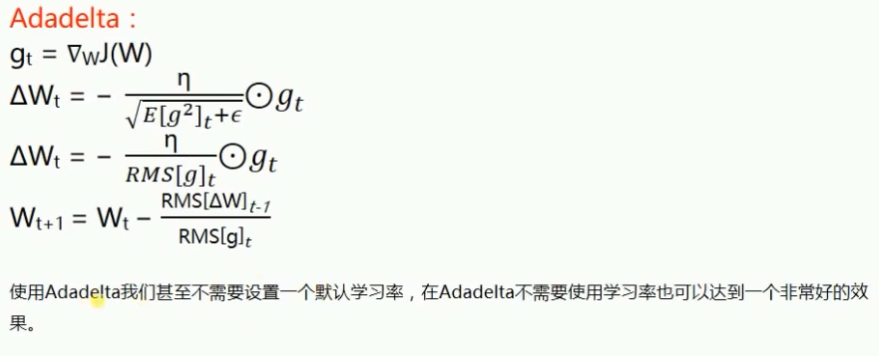


### Adagrad

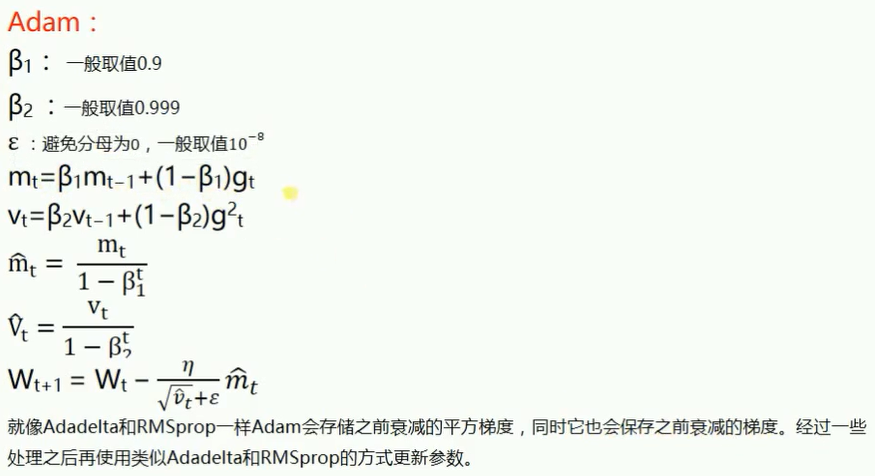


### RMSprop

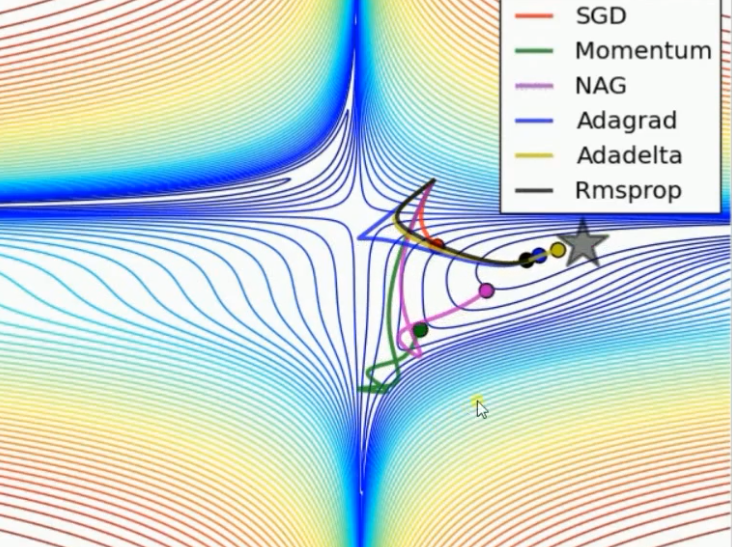
### Adadelta



### Adam



### 效果比较1



效果图：

排名：

Adadelta

Adagrade

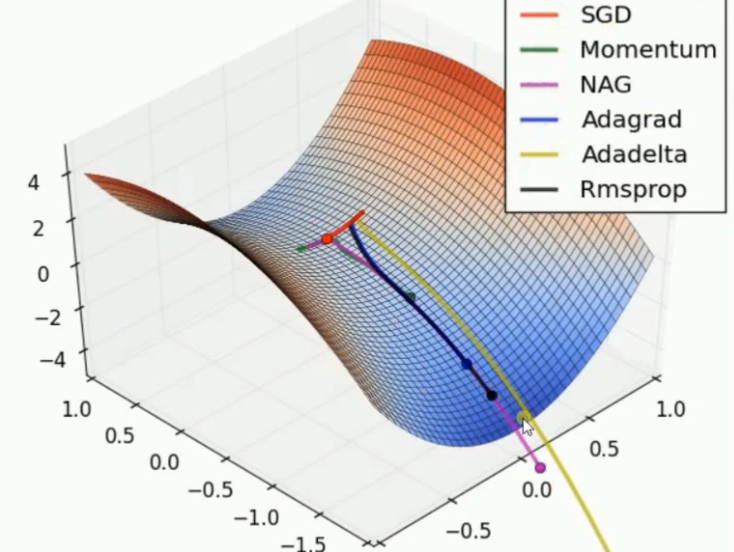
Rmsprop

NAG

Momentum

SGD

### 效果比较：鞍点问题



排名：

Adadelta以很快的速度冲下来

NAG在最初的迷茫之后快速走下鞍点，速度很快

Momentum也在迷茫之后走下鞍点，但是没有NAG

Rmsprop没有迷茫，但是下降速度有点慢

Adagrad 也没有迷茫，但是速度更慢

SGD，直接在鞍点下不来了

### 效果比较：总结

优化器各有优缺点

别的优化器收敛速度会比较快，但是SGD最后的结果一般来说很好

以下网址有不错的总结，可以参考

<https://blog.csdn.net/g11d111/article/details/76639460>

# 第五周 使用Tensorboard进行结构可视化，以及网络运算过程可视化

## 1、打开tensorboard

### 1给网络输入值加上命名空间

# 命名空间

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

x = tf.placeholder(tf.float32, [None,784],name="X-input")

y = tf.placeholder(tf.float32, [None, 10],name='y-input')

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

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

W = tf.Variable(tf.zeros([784,10]),name='W')

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

b = tf.Variable(tf.zeros([1, 10]),name='b')

with tf.name\_scope('xw\_plus\_b'):

wx\_plus\_b = tf.matmul(x,W) + b

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

prediction = tf.nn.softmax(wx\_plus\_b)

### 在运行网络时候加入日志

writer = tf.summary.FileWriter('logs/',sess.graph)

命令行：tensorboard –logdir = E:\pywork\DL\TensorFLow-Learning\logs

## 2、查看网络参数变化情况

主要用方法 tf.summary.scalar(‘name’,value)来记录

在运行网络之前，使用以下方法来汇总要记录的变量

Merged = tf.summary.merge\_all()

sess.run()里包含上面的变量并记录。看下面标准代码：

import tensorflow as tf

from tensorflow.examples.tutorials.mnist import input\_data

# 载入数据

mnist = input\_data.read\_data\_sets('MNIST\_data', one\_hot=True)

# 批次的大小

batch\_size = 128

n\_batch = mnist.train.num\_examples // batch\_size

# 参数概要

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

# 命名空间

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

x = tf.placeholder(tf.float32, [None,784],name="X-input")

y = tf.placeholder(tf.float32, [None, 10],name='y-input')

# 创建一个简单的神经网络

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

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

W = tf.Variable(tf.zeros([784,10]),name='W')

variable\_summaries(W)

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

b = tf.Variable(tf.zeros([1, 10]),name='b')

variable\_summaries(b)

with tf.name\_scope('xw\_plus\_b'):

wx\_plus\_b = tf.matmul(x,W) + b

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

prediction = tf.nn.softmax(wx\_plus\_b)

# 代价函数

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

loss = tf.reduce\_mean(tf.square(y-prediction))

tf.summary.scalar('loss', loss)

# 梯度下降法

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

train\_step = tf.train.GradientDescentOptimizer(0.2).minimize(loss)

# 初始化变量

init = tf.global\_variables\_initializer()

# 得到一个布尔型列表，存放结果是否正确

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

with tf.name\_scope('correct\_prediction'):

correct\_prediction = tf.equal(tf.argmax(y,1), tf.argmax(prediction,1)) #argmax 返回一维张量中最大值索引

# 求准确率

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

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction,tf.float32)) # 把布尔值转换为浮点型求平均数

tf.summary.scalar('accuracy', accuracy)

# 合并所有summary

merged = tf.summary.merge\_all()

with tf.Session() as sess:

sess.run(init)

writer = tf.summary.FileWriter('logs/',sess.graph)

for epoch in range(51):

for batch in range(n\_batch):

# 获得批次数据

batch\_xs, batch\_ys = mnist.train.next\_batch(batch\_size)

# 运行网络并记录log

summary,\_ = sess.run([merged, train\_step], feed\_dict={x:batch\_xs, y:batch\_ys})

# 记录变量

writer.add\_summary(summary,epoch)

acc = sess.run(accuracy, feed\_dict={x:mnist.test.images,y:mnist.test.labels})

print("Iter " + str(epoch) + " Testing Accuracy: " + str(acc))

## 3、补充资料

5-3的代码里更新了两部分代码，一部分是查看网络数据的更优美的代码，另一部分是莫烦大神的关于可视化梯度下降的部分的代码。链接及代码都在5-3的notebook里。

## 4、发现一个重要问题，

计算loss的时候，如果使用tf.nn.softmax\_cross\_entropy\_with\_logit

那么直接把函数值传进去就好，该函数会对logits先计算softmax，再计算交叉熵。

笔者写的代码已经都更正过来。

关于函数验证，详见

<https://blog.csdn.net/mao_xiao_feng/article/details/53382790>

## 5、关于tensorboard 可视化

能够利用tensorboard查看训练时参数变化情况，但是tensorboard可视化的部分代码运行不了会异常中止。

确定问题在于 metadata上,猜测跟tensorflow版本相关，没有进行进一步验证

# 第六周 卷积神经网络CNN

1、

经验之谈：样本数量最好是参数数量的5-30倍。数据量小而模型参数过的多容易出现过拟合现象。

定义weight、bias，

卷积、激活、池化、下一层

然后接2个全连接层，softmax，交叉熵、loss

# 第七周 LSTM

（这部分先跳过，打算先看一下吴恩达的课程，学习一下理论。如果需要的话，再回来补这部分）

感谢 @RickySam 提供本节的代码，git仓已经合并

# 第八周 保存和载入模型，使用Google的图像识别网络inception-v3进行图像识别

## 1、关键代码

Saver = tf.train.Saver()

保存模型： Saver.save(sess, “\*.ckpt”)

载入模型: Saver.restore(sess, “\*.ckpt”)

## 2、inception-v3

下面的代码包含了，下载、解压、使用tensorflow载入模型、保存tensorborad的log文件。

代码部分

inception\_pretrain\_model\_url = 'http://download.tensorflow.org/models/image/imagenet/inception-2015-12-05.tgz'

inception\_pretrain\_model\_dir = "inception\_model"

if not os.path.exists(inception\_pretrain\_model\_dir):

os.makedirs(inception\_pretrain\_model\_dir)

filename = inception\_pretrain\_model\_url.split('/')[-1]

filepath = os.path.join(inception\_pretrain\_model\_dir,filename)

if not os.path.exists(filepath):

print('download:',filename)

r = requests.get(inception\_pretrain\_model\_url, stream=True)

with open(filepath, 'wb') as f:

for chunk in r.iter\_content(chunk\_size=1024):

if chunk:

f.write(chunk)

print('finish: ',filename)

#解压文件

tarfile.open(filepath, 'r:gz').extractall(inception\_pretrain\_model\_dir)

log\_dir = 'logs/inception\_log'

if not os.path.exists(log\_dir):

os.makedirs(log\_dir)

# classfy\_image\_graph\_def.pb 为google训练好的模型

inception\_graph\_def\_fiel = os.path.join(inception\_pretrain\_model\_dir,'classify\_image\_graph\_def.pb')

with tf.Session() as sess:

#创建一个图来存放google训练好的模型

with tf.gfile.FastGFile(inception\_graph\_def\_fiel, 'rb') as f:

graph\_def = tf.GraphDef()

# graph\_def.ParseFromSring(f.read())

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

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

# 保存图的结构

writer = tf.summary.FileWriter(log\_dir, sess.graph)

writer.close()

## 3、使用inception-v3

关键代码：

# 创建一个图来存放google训练好的模型

with tf.gfile.FastGFile('inception\_model/classify\_image\_graph\_def.pb','rb') as f:

graph\_def = tf.GraphDef()

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

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

# 运行网络

with tf.Session() as sess:

# 拿到softmax的op

# 'softmax:0'这个名字，可以在网络中找到这个节点，它的名字就'(softmax)',

softmax\_tensor = sess.graph.get\_tensor\_by\_name('softmax:0')

for root,dirs,files in os.walk('images/'):

for file in files:

image\_data = tf.gfile.FastGFile(os.path.join(root,file),'rb').read()

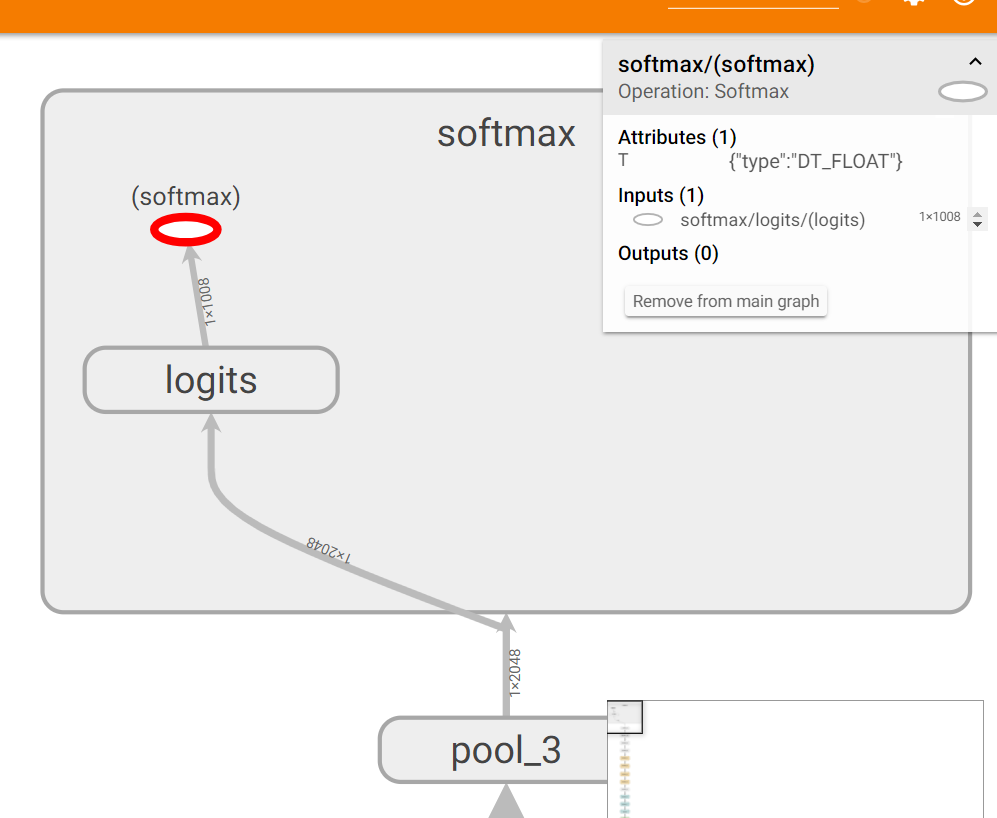
# 运行softmax节点，向其中feed值

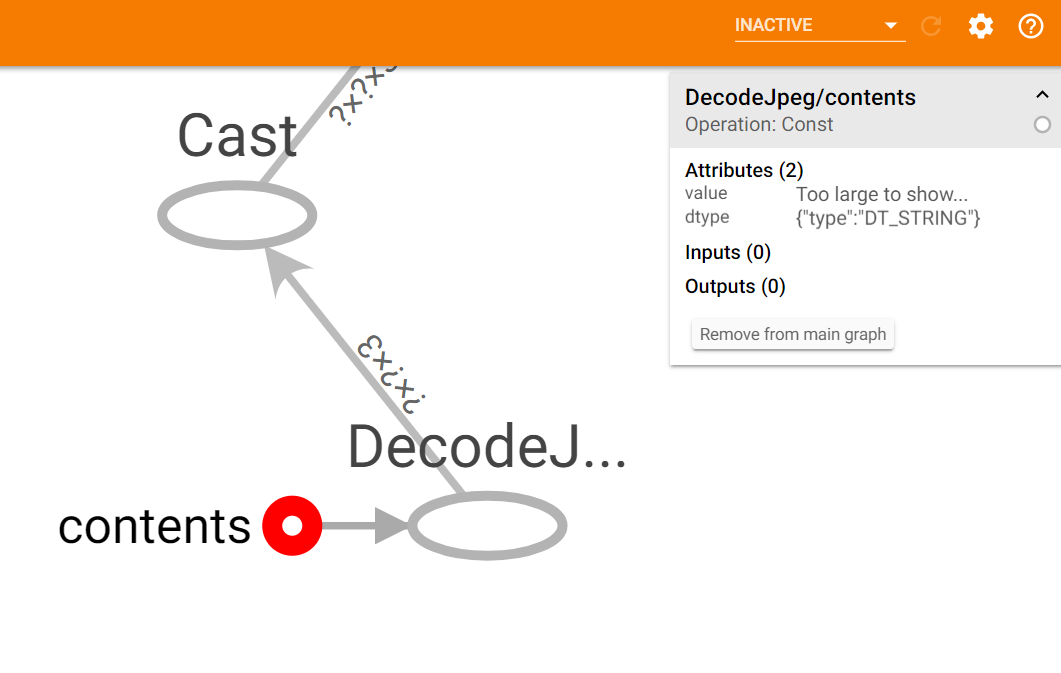
# 可以在网络中找到这个名字，DecodeJpeg/contents，

predictions =sess.run(softmax\_tensor,{'DecodeJpeg/contents:0':image\_data})

predictions = np.squeeze(predictions)# 把结果转化为1维数据

关于取op及向网络feed值用到得名字，如下图：





据此可以发现，根据名字取网络中op时，如果其名字带括号，就用括号内的名字，如果不带括号，就用右上角介绍的名字。

而带个0，是默认情况，如果网络中出现同名节点，这个编号会递增

# 第九周 tensorflow-gpu安装，设计并训练自己的网络模型

## 1、安装

略

## 2、训练自己的网络

三个办法，

自己写自己从头构建网络，从头训练；

用一个现成的质量比较好的模型，固定前面参数，在后面添加几层，训练后面的参数

改造现成的质量比较好的模型，训练整个网络的模型（初始层的学习率比较低）；

9-2这个部分的质量不是很实用，还是从网上找一些迁移学习的例子来学习比较实用，略。

# 十、多任务学习及验证码识别

## 1、关于tfrecord文件

于Tensorflow读取数据，官网给出了三种方法：

供给数据(Feeding)： 在TensorFlow程序运行的每一步， 让Python代码来供给数据。

从文件读取数据： 在TensorFlow图的起始， 让一个输入管线从文件中读取数据。

预加载数据： 在TensorFlow图中定义常量或变量来保存所有数据(仅适用于数据量比较小的情况)。

对于数据量较小而言，可能一般选择直接将数据加载进内存，然后再分batch输入网络进行训练（tip:使用这种方法时，结合yield 使用更为简洁，）。但是，如果数据量较大，这样的方法就不适用了，因为太耗内存，所以这时最好使用tensorflow提供的队列queue，也就是第二种方法 从文件读取数据。使用tensorflow内定标准格式——TFRecords。

从宏观来讲，tfrecord其实是一种数据存储形式。使用tfrecord时，实际上是先读取原生数据，然后转换成tfrecord格式，再存储在硬盘上。而使用时，再把数据从相应的tfrecord文件中解码读取出来。那么使用tfrecord和直接从硬盘读取原生数据相比到底有什么优势呢？其实，Tensorflow有和tfrecord配套的一些函数，可以加快数据的处理。实际读取tfrecord数据时，先以相应的tfrecord文件为参数，创建一个输入队列，这个队列有一定的容量（视具体硬件限制，用户可以设置不同的值），在一部分数据出队列时，tfrecord中的其他数据就可以通过预取进入队列，并且这个过程和网络的计算是独立进行的。也就是说，网络每一个iteration的训练不必等待数据队列准备好再开始，队列中的数据始终是充足的，而往队列中填充数据时，也可以使用多线程加速。

相关资料

<https://blog.csdn.net/happyhorizion/article/details/77894055>

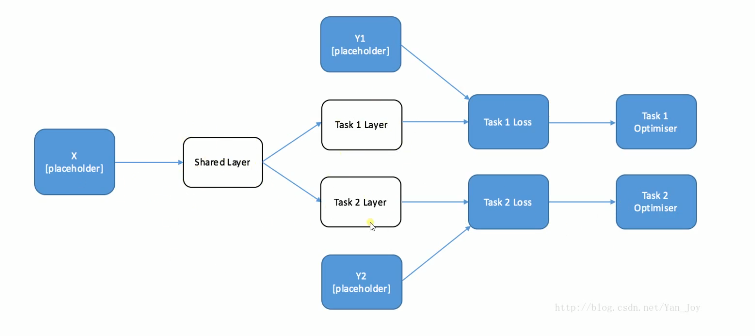
<https://blog.csdn.net/u010358677/article/details/70544241>

<https://blog.csdn.net/Best_Coder/article/details/70146441>

本目录下tfrecord文件夹下，意义10.1节都有相关实现代码

## 2、多任务学习

多个相似任务有多个数据集，交替训练



多个相似任务只有一个数据集，联合训练