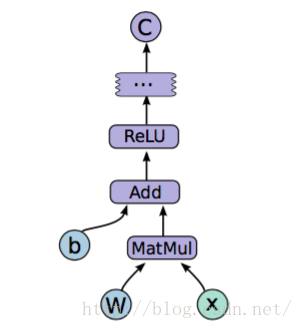
1.tensorflow的运行流程

tensorflow的运行流程主要有2步，分别是**构造模型**和**训练**。

在构造模型阶段，我们需要构建一个图(Graph)来描述我们的模型。所谓图，也可以理解为流程图，就是将数据的输入->中间处理->输出的过程表示出来，就像下面这样。



注意此时是不会发生实际运算的。而在模型构建完毕以后，会进入训练步骤。此时才会有实际的数据输入，梯度计算等操作。那么，**如何构建抽象的模型呢？**这里就要提到tensorflow中的几个概念:**Tensor,Variable,placeholder**,而在训练阶段，则需要介绍**Session**。下面先解释一些上面的几个概念

1.1概念描述

1.1.1 Tensor

Tensor的意思是张量，不过按我的理解，其实就是指矩阵。也可以理解为tensorflow中矩阵的表示形式。Tensor的生成方式有很多种，最简单的就如

import tensorflow as tf # 在下面所有代码中，都去掉了这一行，默认已经导入

a = tf.zeros(shape=[1,2])

* 1
* 2

不过要注意，因为在训练开始前，所有的数据都是抽象的概念，也就是说，此时a只是表示这应该是个1\*5的零矩阵，而没有实际赋值，也没有分配空间，所以如果此时print,就会出现如下情况:

print(a)

#===>Tensor("zeros:0", shape=(1, 2), dtype=float32)

* 1
* 2

只有在训练过程开始后，才能获得a的实际值

sess = tf.InteractiveSession()

print(sess.run(a))

#===>[[ 0. 0.]]

* 1
* 2
* 3

这边设计到Session概念，后面会提到

1.1.2 Variable

故名思议，是变量的意思。一般**用来表示图中的各计算参数**，包括矩阵，向量等。例如，我要表示上图中的模型，那表达式就是 

y=Relu(Wx+b)

（relu是一种激活函数，具体可见[这里](http://blog.csdn.net/u014595019/article/details/52562159)）这里W和b是我要用来训练的参数，那么此时这两个值就可以用Variable来表示。Variable的初始函数有很多其他选项，这里先不提，只输入一个Tensor也是可以的

W = tf.Variable(tf.zeros(shape=[1,2]))

* 1

注意，此时W一样是一个抽象的概念，而且**与Tensor不同，Variable必须初始化以后才有具体的值**。

tensor = tf.zeros(shape=[1,2])

variable = tf.Variable(tensor)

sess = tf.InteractiveSession()

# print(sess.run(variable)) # 会报错

sess.run(tf.initialize\_all\_variables()) # 对variable进行初始化

print(sess.run(variable))

#===>[[ 0. 0.]]

1.1.3 placeholder

又叫占位符，同样是一个抽象的概念。用于表示输入输出数据的格式。告诉系统：这里有一个值/向量/矩阵，现在我没法给你具体数值，不过我正式运行的时候会补上的！例如上式中的x和y。因为没有具体数值，所以只要指定尺寸即可

x = tf.placeholder(tf.float32,[1, 5],name='input')

y = tf.placeholder(tf.float32,[None, 5],name='input')

上面有两种形式，第一种x，表示输入是一个[1,5]的横向量。   
而第二种形式，表示输入是一个[?,5]的矩阵。那么什么情况下会这么用呢?就是需要输入一批[1,5]的数据的时候。比如我有一批共10个数据，那我可以表示成[10,5]的矩阵。如果是一批5个，那就是[5,5]的矩阵。tensorflow会自动进行批处理

1.1.4 Session

session，也就是会话。我的理解是，**session是抽象模型的实现者**。为什么之前的代码多处要用到session？因为模型是抽象的嘛，只有实现了模型以后，才能够得到具体的值。同样，具体的**参数训练，预测，甚至变量的实际值查询，都要用到session**,看后面就知道了

1.2 模型构建

这里我们使用官方tutorial中的mnist数据集的分类代码，公式可以写作 

z=Wx+ba=softmax(z)

那么该模型的代码描述为

# 建立抽象模型

x = tf.placeholder(tf.float32, [None, 784]) # 输入占位符

y = tf.placeholder(tf.float32, [None, 10]) # 输出占位符（预期输出）

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

b = tf.Variable(tf.zeros([10]))

a = tf.nn.softmax(tf.matmul(x, W) + b) # a表示模型的实际输出

# 定义损失函数和训练方法

cross\_entropy = tf.reduce\_mean(-tf.reduce\_sum(y \* tf.log(a), reduction\_indices=[1])) # 损失函数为交叉熵

optimizer = tf.train.GradientDescentOptimizer(0.5) # 梯度下降法，学习速率为0.5

train = optimizer.minimize(cross\_entropy) # 训练目标：最小化损失函数

可以看到这样以来，模型中的所有元素(图结构，损失函数，下降方法和训练目标)都已经包括在train里面。我们可以把train叫做**训练模型**。那么我们还需要**测试模型**

correct\_prediction = tf.equal(tf.argmax(a, 1), tf.argmax(y, 1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))

上述两行代码，tf.argmax表示找到最大值的位置(也就是预测的分类和实际的分类)，然后看看他们是否一致，是就返回true,不是就返回false,这样得到一个boolean数组。tf.cast将boolean数组转成int数组，最后求平均值，得到分类的准确率(怎么样，是不是很巧妙)

1.3 实际训练

有了训练模型和测试模型以后，我们就可以开始进行实际的训练了

sess = tf.InteractiveSession() # 建立交互式会话

tf.initialize\_all\_variables().run() # 所有变量初始化

for i in range(1000):

batch\_xs, batch\_ys = mnist.train.next\_batch(100) # 获得一批100个数据

train.run({x: batch\_xs, y: batch\_ys}) # 给训练模型提供输入和输出

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

可以看到，在模型搭建完以后，我们只要为模型提供输入和输出，模型就能够自己进行训练和测试了。中间的求导，求梯度，反向传播等等繁杂的事情，tensorflow都会帮你自动完成。

2. 实际代码

实际操作中，还包括了获取数据的代码

"""A very simple MNIST classifier.

See extensive documentation at

http://tensorflow.org/tutorials/mnist/beginners/index.md

"""

from \_\_future\_\_ import absolute\_import

from \_\_future\_\_ import division

from \_\_future\_\_ import print\_function

# Import data

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

import tensorflow as tf

flags = tf.app.flags

FLAGS = flags.FLAGS

flags.DEFINE\_string('data\_dir', '/tmp/data/', 'Directory for storing data') # 把数据放在/tmp/data文件夹中

mnist = input\_data.read\_data\_sets(FLAGS.data\_dir, one\_hot=True) # 读取数据集

# 建立抽象模型

x = tf.placeholder(tf.float32, [None, 784]) # 占位符

y = tf.placeholder(tf.float32, [None, 10])

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

b = tf.Variable(tf.zeros([10]))

a = tf.nn.softmax(tf.matmul(x, W) + b)

# 定义损失函数和训练方法

cross\_entropy = tf.reduce\_mean(-tf.reduce\_sum(y \* tf.log(a), reduction\_indices=[1])) # 损失函数为交叉熵

optimizer = tf.train.GradientDescentOptimizer(0.5) # 梯度下降法，学习速率为0.5

train = optimizer.minimize(cross\_entropy) # 训练目标：最小化损失函数

# Test trained model

correct\_prediction = tf.equal(tf.argmax(a, 1), tf.argmax(y, 1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))

# Train

sess = tf.InteractiveSession() # 建立交互式会话

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

for i in range(1000):

batch\_xs, batch\_ys = mnist.train.next\_batch(100)

train.run({x: batch\_xs, y: batch\_ys})

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

得到的分类准确率在91%左右