**TensorFlow入门**

**一、基础知识**

**张量（Tensor）**

TensorFlow 内部的计算都是基于张量的。张量是在标量、向量之上定义的，详细的定义比较复杂，可以先简单的将它理解为一个多维数组：

3 # 这个 0 阶张量就是标量，shape=[]

[1., 2., 3.] # 这个 1 阶张量就是向量，shape=[3]

[[1., 2., 3.], [4., 5., 6.]] # 这个 2 阶张量就是二维数组，shape=[2, 3]

[[[1., 2., 3.]], [[7., 8., 9.]]] # 这个 3 阶张量就是三维数组，shape=[2, 1, 3]

TensorFlow 内部使用tf.Tensor类的实例来表示张量，每个 tf.Tensor有两个属性：

dtype Tensor 存储的数据的类型，可以为tf.float32、tf.int32、tf.string…

shape Tensor 存储的多维数组中每个维度的数组中元素的个数，如上面例子中的shape

引入tensorflow模块：

import tensorflow as tf

创建一个整型常量，即 0 阶 Tensor：

t0 = tf.constant(3,dtype=tf.int32)

创建一个浮点数的一维数组，即 1 阶 Tensor：

t1 = tf.constant([3.,4.1,5.2],dtype=tf.float32)

创建一个字符串的2x2数组，即 2 阶 Tensor：

t2 = tf.constant([['Apple','Orange'],['Potato','Tomato']],dtype=tf.string)

创建一个 2x3x1 数组，即 3 阶张量，数据类型默认为整型：

t3 = tf.constant([[[5],[6],[7]],[[4],[3],[2]]])

打印上面创建的几个 Tensor：

print(t0)

print(t1)

print(t2)

print(t3)

print 一个 Tensor 只能打印出它的属性定义，并不能打印出它的值，要想查看一个 Tensor 中的值还需要经过Session 运行一下：

sess = tf.Session()

print(sess.run(t0))

print(sess.run(t1))

print(sess.run(t2))

print(sess.run(t3))

**数据流图(Dataflow Graph)**

数据流是一种常用的并行计算编程模型，数据流图是由**节点(nodes)**和**线(edges)**构成的有向图：

**节点(nodes)** 表示计算单元，也可以是输入的起点或者输出的终点

**线(edges)** 表示节点之间的输入/输出关系

在 TensorFlow 中，每个节点都是用 tf.Tensor的实例来表示的，即每个节点的输入、输出都是Tensor

TensorFlow 中的数据流图有以下几个优点：

**可并行** 计算节点之间有明确的线进行连接，系统可以很容易的判断出哪些计算操作可以并行执行

**可分发** 图中的各个节点可以分布在不同的计算单元(CPU、 GPU、 TPU等)或者不同的机器中，每个节点产生的数据可以通过明确的线发送的下一个节点中

**可优化** TensorFlow 中的 XLA 编译器可以根据数据流图进行代码优化，加快运行速度

**可移植** 数据流图的信息可以不依赖代码进行保存，如使用Python创建的图，经过保存后可以在C++或Java中使用

**Sesssion**

TensorFlow 底层是使用C++实现，这样可以保证计算效率，并使用 tf.Session类来连接客户端程序与C++运行。上层的Python、Java等代码用来设计、定义模型，构建的Graph，最后通过tf.Session.run()方法传递给底层执行。

**构建计算图**

Tensor 既可以表示输入、输出的端点，还可以表示计算单元，如下的代码创建了对两个 Tensor 执行 + 操作的 Tensor：

node1 = tf.constant(3.2)

node2 = tf.constant(4.8)

adder = node1+node2

print(adder)

sess = tf.Session()

print(sess.run(adder))

上面使用tf.constant()创建的 Tensor 都是常量，一旦创建后其中的值就不能改变了。有时还会需要从外部输入数据，这时可以用tf.placeholder 创建占位 Tensor，占位 Tensor 的值可以在运行的时候输入。如下就是创建占位 Tensor 的例子：

a = tf.placeholder(tf.float32)

b = tf.placeholder(tf.float32)

adder\_node = a+b

print(a)

print(b)

print(adder\_node)

sess = tf.Session()

print(sess.run(adder\_node,{a:3,b:4.5}))

print(sess.run(adder\_node,{a:[1,3],b:[2,4]}))

还可以添加其他操作构建复杂的 Graph:

add\_and\_triple = adder\_node\*3.

print(sess.run(add\_and\_triple,{a:3,b:4.5}))

**二、TensorFlow 应用实例**

**建立模型(Model)**

根据奥卡姆剃刀原则-若有多个假设与观察一致，则选最简单的那个-选择线性模型：

设计一个损失模型(loss model)，来评估一下哪个模型更合理一些，并找到一个最准确的模型：

损失模型里得到的loss越小，说明线性模型越准确。

**使用 TensorFlow 实现模型**

在线性模型 y=W×x+b中，输入x可以用占位 Tensor 表示，输出y可以用线性模型的输出表示，需要不断的改变W和b的值，来找到一个使loss最小的值。这里W和b可以用变量 Tensor 表示。使用tf.Variable()可以创建一个变量 Tensor，如下就是模型的实现代码：

W = tf.Variable([.1],dtype=tf.float32)

b = tf.Variable([-.1],dtype=tf.float32)

x = tf.placeholder(tf.float32)

linear\_model = W\*x+b

y = tf.placeholder(tf.float32)

loss = tf.reduce\_sum(tf.square(linear\_model - y))

sess = tf.Session()

通过tf.Variable()创建变量 Tensor 时需要设置一个初始值，但这个初始值并不能立即使用，变量 Tensor 需要经过下面的 init 过程后才能使用：

init = tf.global\_variables\_initializer()

sess.run(init)

print(sess.run(W))

变量初始化完之后，可以先用上面对W和b设置的初始值0.1和-0.1运行一下线性模型看看结果：

print(sess.run(linear\_model,{x:[1,2,3,6,8]}))

可以看到与实验的实际输出差距很大，再运行一下损失模型：

print(sess.run(loss,{x:[1,2,3,6,8],y:[4.8,8.5,10.4,21.0,25.3]}))

得到的损失值也很大。可以用tf.assign()对W和b变量重新赋值再检验一下：

fixW = tf.assign(W,[2.])

fixb = tf.assign(b,[1.])

sess.run([fixW,fixb]) # run 之后新值才会生效

print(sess.run(loss,{x:[1,2,3,6,8],y:[4.8,8.5,10.4,21.0,25.3]}))

TensorFlow 提供了训练模型的方法，自动进行繁琐的训练工作，即不断调整变量W和b的值，找到使损失值最小的W和b。

**使用 TensorFlow 训练模型**

TensorFlow 提供了很多优化算法来帮助训练模型。最简单的优化算法是**梯度下降(Gradient Descent)**算法，它通过不断的改变模型中变量的值，来找到最小损失值。 代码如下：

# 创建一个梯度下降优化器，学习率为0.001

optimizer = tf.train.GradientDescentOptimizer(0.001)

train = optimizer.minimize(loss)

# 用两个数组保存训练数据

x\_train = [1,2,3,6,8]

y\_train = [4.8,8.5,10.4,21.0,25.3]

# 训练10000次

for i in range(10000):

sess.run(train,{x:x\_train,y:y\_train})

# 打印训练后的结果

print('W: %s b: %s loss: %s'%(sess.run(W),sess.run(b),sess.run(loss,{x:x\_train,y:y\_train})))

**TensorBoard**

为了更方便 TensorFlow 的建模和调优，Google 还为 TensorFlow 开发了一款可视化的工具：TensorBoard，将第一个Demo的代码稍微改造一下，就可以使用 TensorBoard更加直观的理解 TensorFlow 的训练过程。主要改动的地方有以下几处：

# 损失模型隐藏到loss-model模块

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

loss = tf.reduce\_sum(tf.square(linear\_model - y))

# 给损失模型的输出添加scalar，用来观察loss的收敛曲线

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

# 调用 merge\_all() 收集所有的操作数据

merged = tf.summary.merge\_all()

# 模型运行产生的所有数据保存到 /tmp/tensorflow 文件夹供 TensorBoard 使用

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

# 训练10000次

for i in range(10000):

# 训练时传入merge

summary,\_ = sess.run([merged,train],{x:x\_train,y:y\_train})

# 收集每次训练产生的数据

writer.add\_summary(summary,i)

# 打印结果

curr\_W,curr\_b,curr\_loss = sess.run(

[W,b,loss],{x:x\_train,y:y\_train})

print('W: %s b: %s loss: %s'%(curr\_W,curr\_b,curr\_loss))

运行完上面的代码后，训练过程产生的数据就保存在 /tmp/tensorflow 文件夹了，可以在命令行终端运行下面的命令启动 TensorBoard：

# 通过 --logdir 参数设置存放训练数据的目录

$ tensorboard --logdir /tmp/tensorflow

然后在浏览器中打开 http://localhost:6006 页面就可以看到模型数据了。

在 SCALARS 页面可以看到通过 tf.summary.scalar("loss", loss)设置的loss收敛曲线，从曲线图中可以看出在训练了大概2000次的时候loss就已经收敛的差不多了。

在 GRAPHS 页面可以看到构建的模型的数据流图。

其中损失模型折叠到loss-model模块里了，双击该模块可以展开损失模型的内容。