使用模拟集群运行分布式tensorflow

Created by zhaomingxing, last modified on Feb 10, 2017

第一步：创建多个docker container，我这里创建了2个参数服务器，ps0和ps1；2个worker，w0和w1。参数服务器ps和worker可以设置一个或多个。使用的是docker官方的镜像，最新版的tensorflow/tensorflow的镜像。

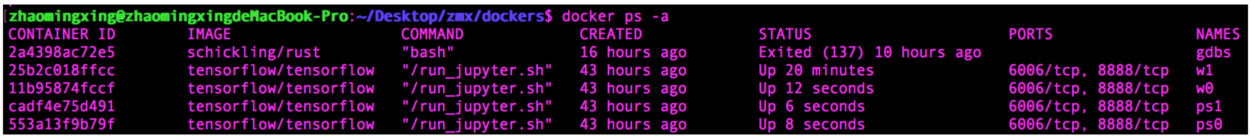
docker run -d -v ~/Desktop/zmx/dockers:/tf/mnist --name ps0 tensorflow/tensorflow

docker run -d -v ~/Desktop/zmx/dockers:/tf/mnist --name ps1 tensorflow/tensorflow

docker run -d -v ~/Desktop/zmx/dockers:/tf/mnist --name w0 tensorflow/tensorflow

docker run -d -v ~/Desktop/zmx/dockers:/tf/mnist --name w1 tensorflow/tensorflow

这里的~/Desktop/zmx/dockers是本地存放分布式mnist示例代码（distribute_version_MNIST.py）的目录，也可以不进行目录映射，但是要把同一份代码分别拷贝到每个docker container中，因为后面每个docker container都要运行这份代码，只是运行命令稍有区别。执行完上述命令之后查看已经生成的docker container如下：



```
zhaomingxing@zhaomingxingdeMacBook-Pro:~/Desktop/zmx/dockers$ docker ps -a
CONTAINER ID        IMAGE                   COMMAND               CREATED             STATUS                     PORTS                NAMES
2a4398ac72e5        schickling/rust         "bash"                16 hours ago        Exited (137) 10 hours ago                       gdbs
25b2c018ffcc        tensorflow/tensorflow   "/run_jupyter.sh"     43 hours ago        Up 20 minutes              6006/tcp, 8888/tcp   w1
11b95874fccf        tensorflow/tensorflow   "/run_jupyter.sh"     43 hours ago        Up 12 seconds              6006/tcp, 8888/tcp   w0
cadf4e75d491        tensorflow/tensorflow   "/run_jupyter.sh"     43 hours ago        Up 6 seconds               6006/tcp, 8888/tcp   ps1
553a13f9b79f        tensorflow/tensorflow   "/run_jupyter.sh"     43 hours ago        Up 8 seconds               6006/tcp, 8888/tcp   ps0
```

第二步：使用ifconfig命令查看上述的四个docker container的ip地址。

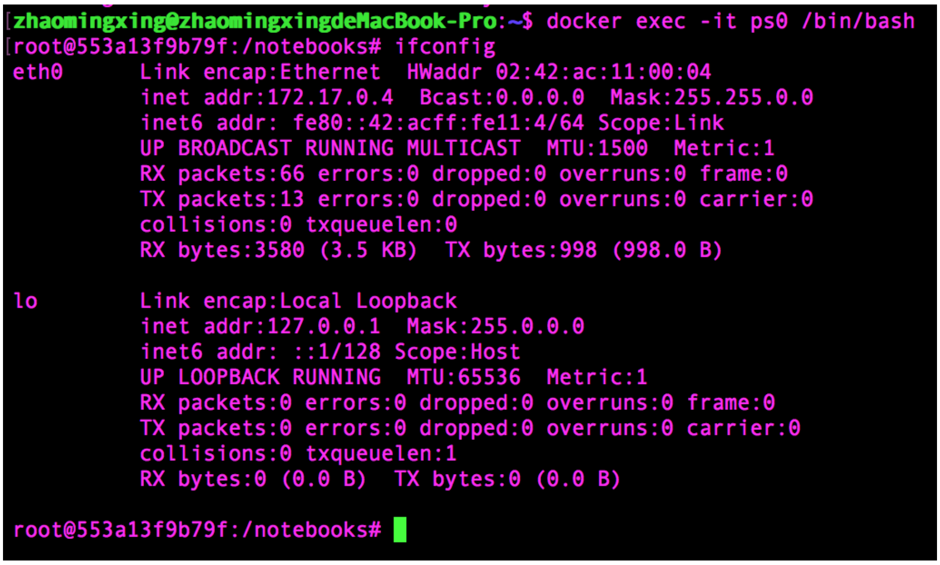
ps0：172.17.0.4

ps1：172.17.0.5

w0：172.17.0.3

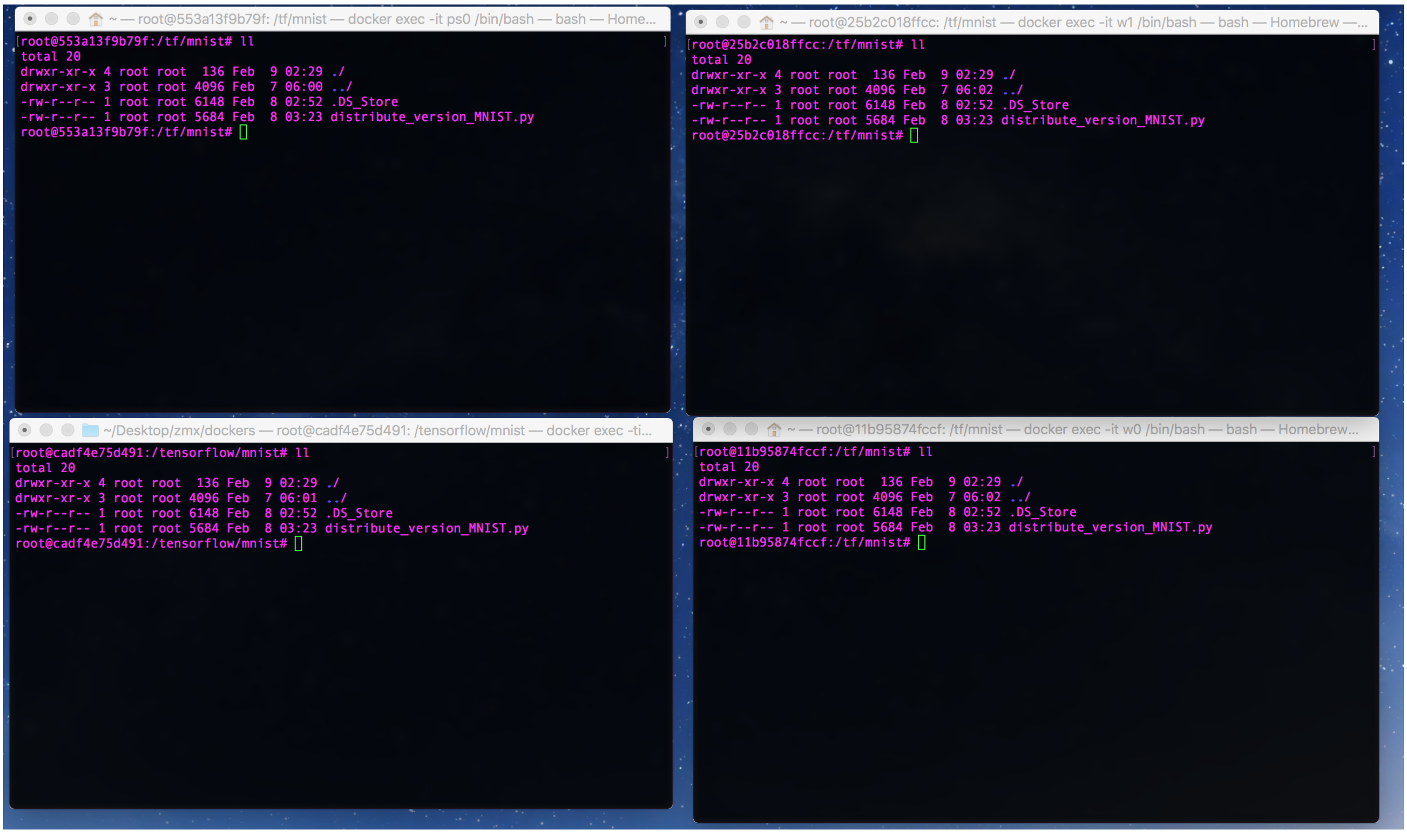
w1: 172.17.0.2

以ps0运行ifconfig的结果为例：



```
zhaomingxing@zhaomingxingdeMacBook-Pro:~$ docker exec -it ps0 /bin/bash
root@553a13f9b79f:/notebooks# ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:ac:11:00:04
          inet addr:172.17.0.4  Bcast:0.0.0.0  Mask:255.255.0.0
          inet6 addr: fe80::42:acff:fe11:4/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:66 errors:0 dropped:0 overruns:0 frame:0
          TX packets:13 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:3580 (3.5 KB)  TX bytes:998 (998.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@553a13f9b79f:/notebooks#
```

第三步：使用docker exec -ti ps1(docker container的name或ID) /bin/bash分别打开每个docker container的终端，切换到分布式MNIST代码所在的目录。


~ — root@553a13f9b79f: /tf/mnist — docker exec -it ps0 /bin/bash — bash — Home...
~ — root@25b2c018ffcc: /tf/mnist — docker exec -it w1 /bin/bash — bash — Homebrew — ...
~/Desktop/zmx/dockers — root@cadf4e75d491: /tensorflow/mnist — docker exec -ti...
~ — root@11b95874fccf: /tf/mnist — docker exec -it w0 /bin/bash — bash — Homebrew...

```
root@553a13f9b79f:/tf/mnist# ll
total 20
drwxr-xr-x 4 root root  136 Feb  9 02:29 ./
drwxr-xr-x 3 root root 4096 Feb  7 06:00 ../
-rw-r--r-- 1 root root 6148 Feb  8 02:52 .DS_Store
-rw-r--r-- 1 root root 5684 Feb  8 03:23 distribute_version_MNIST.py
root@553a13f9b79f:/tf/mnist#
```

```
root@25b2c018ffcc:/tf/mnist# ll
total 20
drwxr-xr-x 4 root root  136 Feb  9 02:29 ./
drwxr-xr-x 3 root root 4096 Feb  7 06:02 ../
-rw-r--r-- 1 root root 6148 Feb  8 02:52 .DS_Store
-rw-r--r-- 1 root root 5684 Feb  8 03:23 distribute_version_MNIST.py
root@25b2c018ffcc:/tf/mnist#
```

```
root@cadf4e75d491:/tensorflow/mnist# ll
total 20
drwxr-xr-x 4 root root  136 Feb  9 02:29 ./
drwxr-xr-x 3 root root 4096 Feb  7 06:01 ../
-rw-r--r-- 1 root root 6148 Feb  8 02:52 .DS_Store
-rw-r--r-- 1 root root 5684 Feb  8 03:23 distribute_version_MNIST.py
root@cadf4e75d491:/tensorflow/mnist#
```

```
root@11b95874fccf:/tf/mnist# ll
total 20
drwxr-xr-x 4 root root  136 Feb  9 02:29 ./
drwxr-xr-x 3 root root 4096 Feb  7 06:02 ../
-rw-r--r-- 1 root root 6148 Feb  8 02:52 .DS_Store
-rw-r--r-- 1 root root 5684 Feb  8 03:23 distribute_version_MNIST.py
root@11b95874fccf:/tf/mnist#
```

第四步：在上述的四个终端中分别运行分布式MNIST的代码distribute_version_MNIST.py：

ps0执行：

```
python distribute_version_MNIST.py  --ps_hosts=172.17.0.4:2222,172.17.0.5:2222 --worker_hosts=172.17.0.3:2222,172.17.0.2:2222 --job_name=ps --task_index=0
```

ps1执行：

```
python distribute_version_MNIST.py  --ps_hosts=172.17.0.4:2222,172.17.0.5:2222 --worker_hosts=172.17.0.3:2222,172.17.0.2:2222 --job_name=ps --task_index=1
```

w0执行：

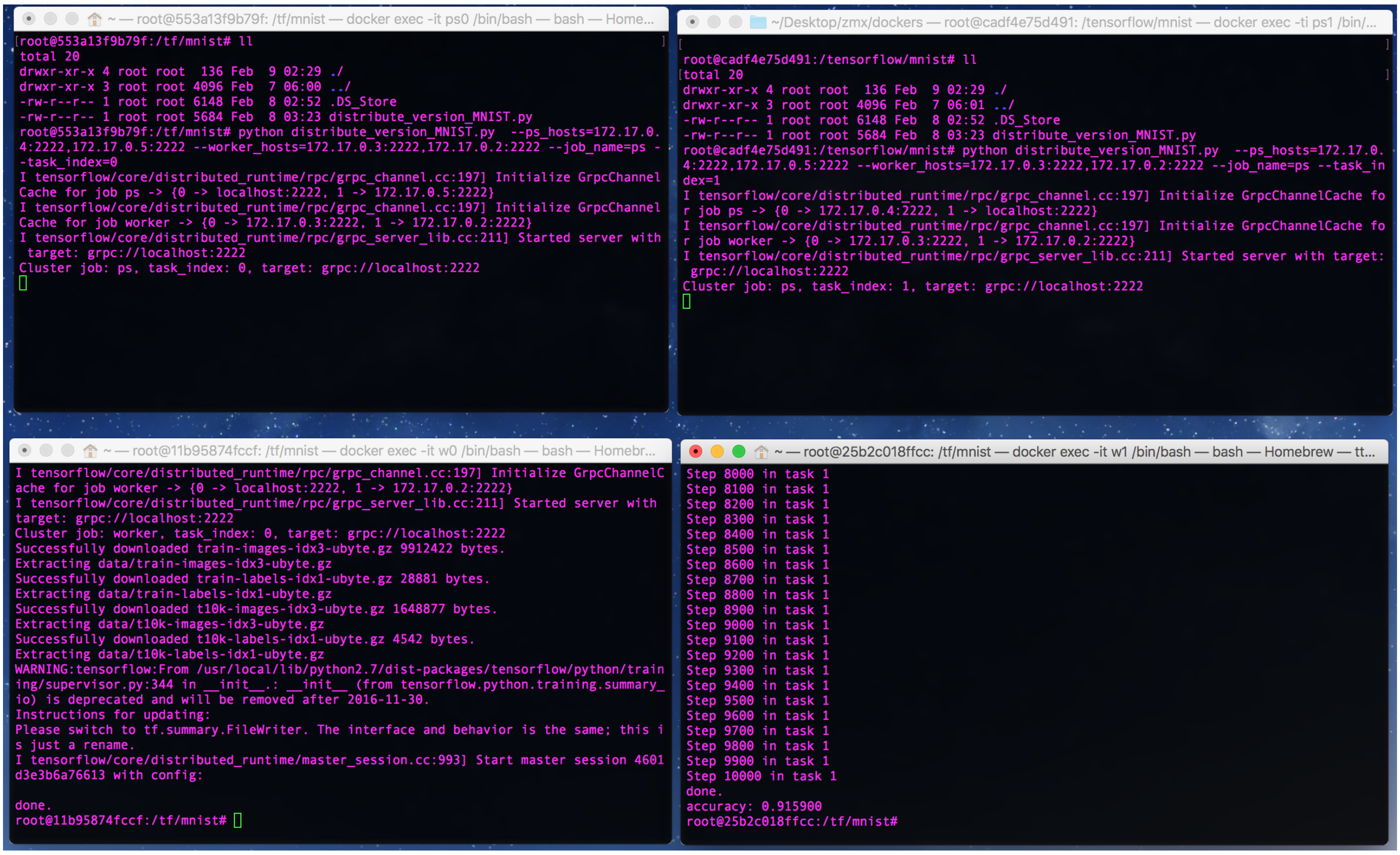
```
python distribute_version_MNIST.py  --ps_hosts=172.17.0.4:2222,172.17.0.5:2222 --worker_hosts=172.17.0.3:2222,172.17.0.2:2222 --job_name=worker --task_index=0
```

w1执行：

```
python distribute_version_MNIST.py  --ps_hosts=172.17.0.4:2222,172.17.0.5:2222 --worker_hosts=172.17.0.3:2222,172.17.0.2:2222 --job_name=worker --task_index=1
```

注意：w1执行几秒后会退出一次，再执行一次相同的命令即可。上述端口号(2222)是可以随便指定的，只要是docker container上的空闲端口号即可。

运行结果如下：


~ — root@553a13f9b79f: /tf/mnist — docker exec -it ps0 /bin/bash — bash — Home...
~/Desktop/zmx/dockers — root@cadf4e75d491: /tensorflow/mnist — docker exec -ti ps1 /bin/...
~ — root@11b95874fccf: /tf/mnist — docker exec -it w0 /bin/bash — bash — Homebr...
~ — root@25b2c018ffcc: /tf/mnist — docker exec -it w1 /bin/bash — bash — Homebrew — tt...

```
root@553a13f9b79f:/tf/mnist# ll
total 20
drwxr-xr-x 4 root root  136 Feb  9 02:29 ./
drwxr-xr-x 3 root root 4096 Feb  7 06:00 ../
-rw-r--r-- 1 root root 6148 Feb  8 02:52 .DS_Store
-rw-r--r-- 1 root root 5684 Feb  8 03:23 distribute_version_MNIST.py
root@553a13f9b79f:/tf/mnist# python distribute_version_MNIST.py  --ps_hosts=172.17.0.
4:2222,172.17.0.5:2222 --worker_hosts=172.17.0.3:2222,172.17.0.2:2222 --job_name=ps -
-task_index=0
I tensorflow/core/distributed_runtime/rpc/grpc_channel.cc:197] Initialize GrpcChannel
Cache for job ps -> {0 -> localhost:2222, 1 -> 172.17.0.5:2222}
I tensorflow/core/distributed_runtime/rpc/grpc_channel.cc:197] Initialize GrpcChannel
Cache for job worker -> {0 -> 172.17.0.3:2222, 1 -> 172.17.0.2:2222}
I tensorflow/core/distributed_runtime/rpc/grpc_server_lib.cc:211] Started server with
 target: grpc://localhost:2222
Cluster job: ps, task_index: 0, target: grpc://localhost:2222
```

```
root@cadf4e75d491:/tensorflow/mnist# ll
total 20
drwxr-xr-x 4 root root  136 Feb  9 02:29 ./
drwxr-xr-x 3 root root 4096 Feb  7 06:01 ../
-rw-r--r-- 1 root root 6148 Feb  8 02:52 .DS_Store
-rw-r--r-- 1 root root 5684 Feb  8 03:23 distribute_version_MNIST.py
root@cadf4e75d491:/tensorflow/mnist# python distribute_version_MNIST.py  --ps_hosts=172.17.0.
4:2222,172.17.0.5:2222 --worker_hosts=172.17.0.3:2222,172.17.0.2:2222 --job_name=ps --task_in
dex=1
I tensorflow/core/distributed_runtime/rpc/grpc_channel.cc:197] Initialize GrpcChannelCache fo
r job ps -> {0 -> 172.17.0.4:2222, 1 -> localhost:2222}
I tensorflow/core/distributed_runtime/rpc/grpc_channel.cc:197] Initialize GrpcChannelCache fo
r job worker -> {0 -> 172.17.0.3:2222, 1 -> 172.17.0.2:2222}
I tensorflow/core/distributed_runtime/rpc/grpc_server_lib.cc:211] Started server with target:
 grpc://localhost:2222
Cluster job: ps, task_index: 1, target: grpc://localhost:2222
```

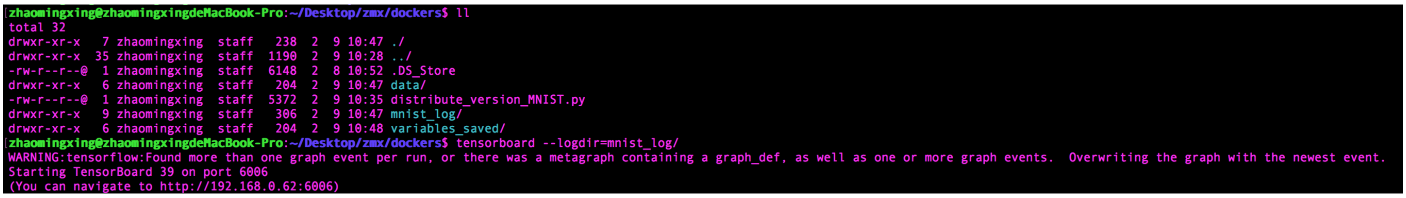
```
I tensorflow/core/distributed_runtime/rpc/grpc_channel.cc:197] Initialize GrpcChannelC
ache for job worker -> {0 -> localhost:2222, 1 -> 172.17.0.2:2222}
I tensorflow/core/distributed_runtime/rpc/grpc_server_lib.cc:211] Started server with
target: grpc://localhost:2222
Cluster job: worker, task_index: 0, target: grpc://localhost:2222
Successfully downloaded train-images-idx3-ubyte.gz 9912422 bytes.
Extracting data/train-images-idx3-ubyte.gz
Successfully downloaded train-labels-idx1-ubyte.gz 28881 bytes.
Extracting data/train-labels-idx1-ubyte.gz
Successfully downloaded t10k-images-idx3-ubyte.gz 1648877 bytes.
Extracting data/t10k-images-idx3-ubyte.gz
Successfully downloaded t10k-labels-idx1-ubyte.gz 4542 bytes.
Extracting data/t10k-labels-idx1-ubyte.gz
WARNING:tensorflow:From /usr/local/lib/python2.7/dist-packages/tensorflow/python/train
ing/supervisor.py:344 in __init__.: __init__ (from tensorflow.python.training.summary_
io) is deprecated and will be removed after 2016-11-30.
Instructions for updating:
Please switch to tf.summary.FileWriter. The interface and behavior is the same; this i
s just a rename.
I tensorflow/core/distributed_runtime/master_session.cc:993] Start master session 4601
d3e3b6a76613 with config:

done.
root@11b95874fccf:/tf/mnist#
```

```
Step 8000 in task 1
Step 8100 in task 1
Step 8200 in task 1
Step 8300 in task 1
Step 8400 in task 1
Step 8500 in task 1
Step 8600 in task 1
Step 8700 in task 1
Step 8800 in task 1
Step 8900 in task 1
Step 9000 in task 1
Step 9100 in task 1
Step 9200 in task 1
Step 9300 in task 1
Step 9400 in task 1
Step 9500 in task 1
Step 9600 in task 1
Step 9700 in task 1
Step 9800 in task 1
Step 9900 in task 1
Step 10000 in task 1
done.
accuracy: 0.915900
root@25b2c018ffcc:/tf/mnist#
```

从w1的输出结果来看，运行10000步之后，测试集上的准确率达到了0.915900.

第五步：使用tensorboard观察数据流图和程序执行过程中的accuracy的变化情况。

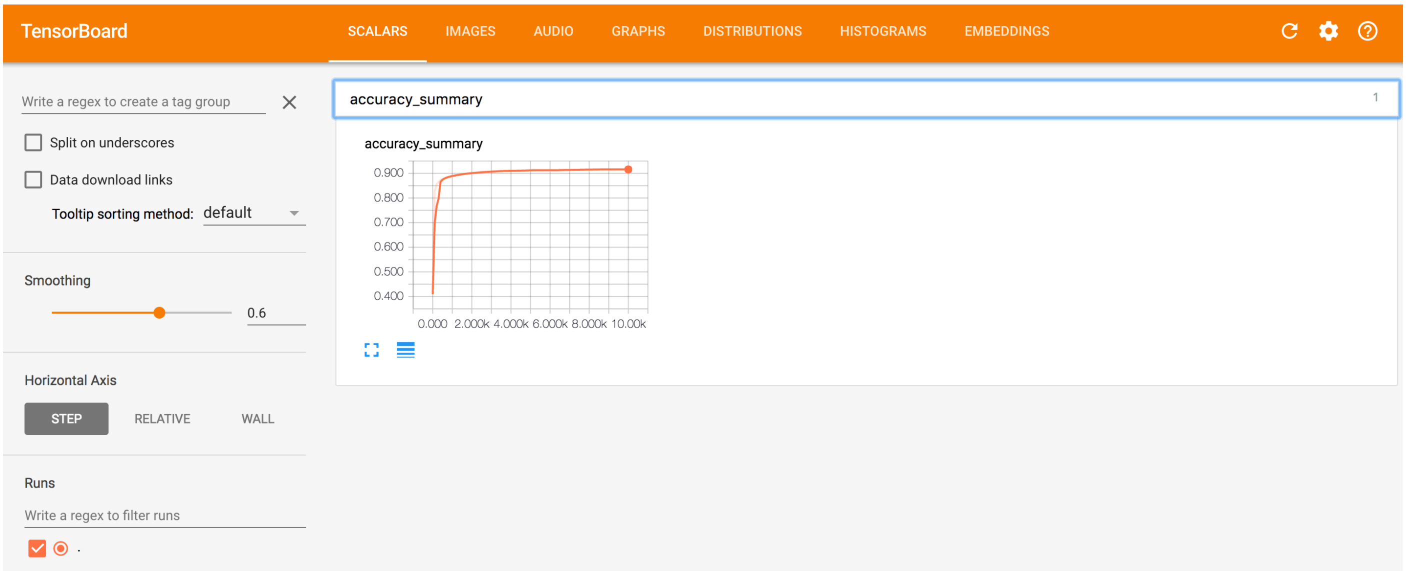
tensorboard的执行如下：



```
zhaomingxing@zhaomingxingdeMacBook-Pro:~/Desktop/zmx/dockers$ ll
total 32
drwxr-xr-x   7 zhaomingxing  staff   238  2  9 10:47 ./
drwxr-xr-x  35 zhaomingxing  staff  1190  2  9 10:28 ../
-rw-r--r--@  1 zhaomingxing  staff  6148  2  8 10:52 .DS_Store
drwxr-xr-x   6 zhaomingxing  staff   204  2  9 10:47 data/
-rw-r--r--@  1 zhaomingxing  staff  5372  2  9 10:35 distribute_version_MNIST.py
drwxr-xr-x   9 zhaomingxing  staff   306  2  9 10:47 mnist_log/
drwxr-xr-x   6 zhaomingxing  staff   204  2  9 10:48 variables_saved/
zhaomingxing@zhaomingxingdeMacBook-Pro:~/Desktop/zmx/dockers$ tensorboard --logdir=mnist_log/
WARNING:tensorflow:Found more than one graph event per run, or there was a metagraph containing a graph_def, as well as one or more graph events.  Overwriting the graph with the newest event.
Starting TensorBoard 39 on port 6006
(You can navigate to http://192.168.0.62:6006)
```

按照图片中的指示：在浏览器地址栏中输入http://192.168.0.62:6006即可。

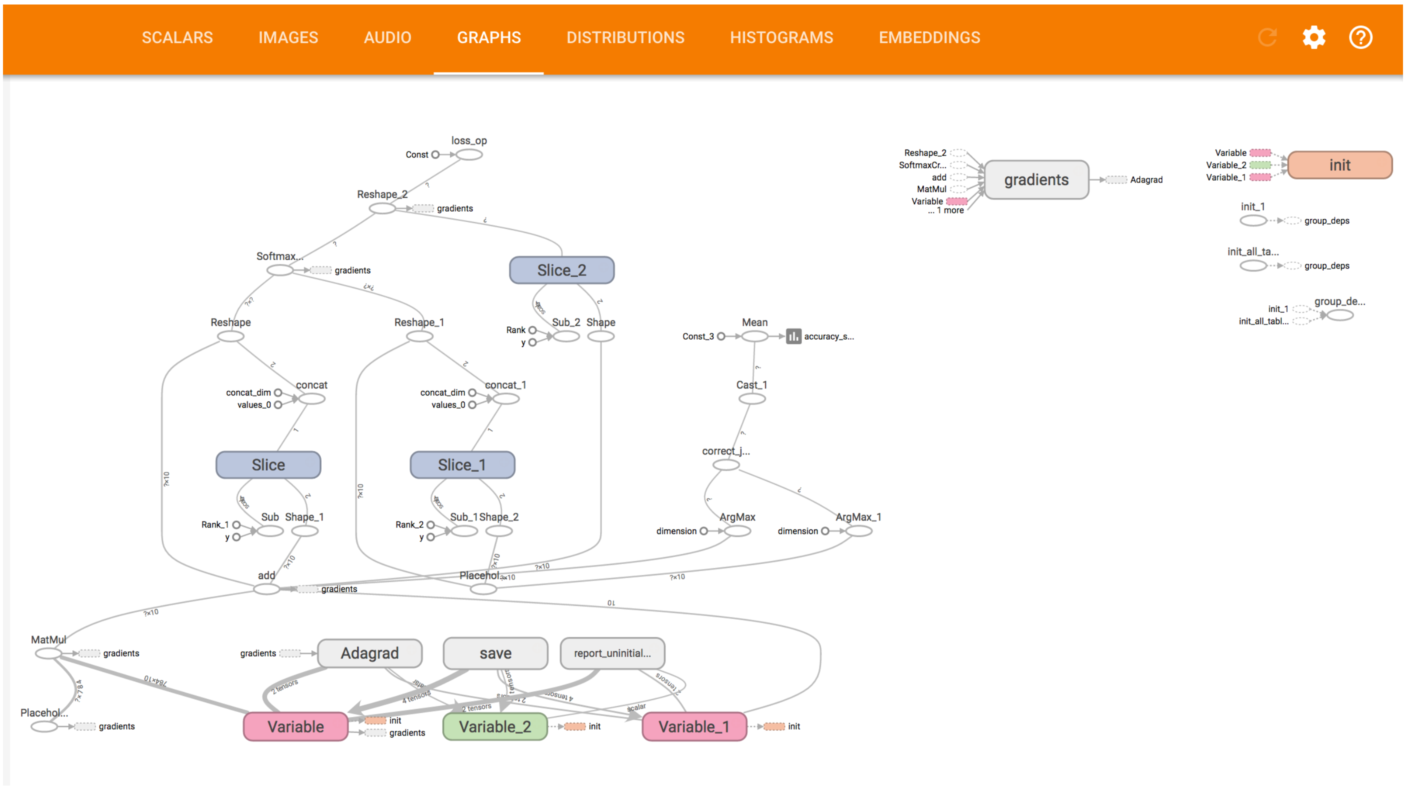
选择“SCALARS”栏，点击accuracy_summary可见下图：



□ Split on underscores
□ Data download links
Tooltip sorting method: default
Smoothing 0.6
☑ ○ .

上面的折线图就是训练过程中，测试集上的预测准确率的变化。

选择GRAPHS栏，可以看到下图：



最后附上分布式MNIST的源码：

distribute_version_MNIST.py

```python

#!/usr/bin/python
# -*- coding:utf-8 -*-
"""
================================================================================
author: 赵明星
desc:   分布式tensorflow实现手写数字识别MNIST。
================================================================================
"""
import sys
reload(sys)
sys.setdefaultencoding('utf-8')

import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
import os

tf.app.flags.DEFINE_string("ps_hosts",
                           "",
                           "Comma-separated list of hostname:port pairs")
tf.app.flags.DEFINE_string("worker_hosts",
                           "",
                           "Comma-separated list of hostname:port pairs")
tf.app.flags.DEFINE_string("job_name", "", "One of 'ps', 'worker'")
tf.app.flags.DEFINE_integer("task_index", 0, "Index of task within the job")
FLAGS = tf.app.flags.FLAGS
def main(_):
    ps_hosts = FLAGS.ps_hosts.split(",")
    worker_hosts = FLAGS.worker_hosts.split(",")
    # Create a cluster from the parameter server and worker hosts.
    cluster = tf.train.ClusterSpec({"ps": ps_hosts, "worker": worker_hosts})
    # Create and start a server for the local task.
    server = tf.train.Server(cluster,
                             job_name=FLAGS.job_name,
                             task_index=FLAGS.task_index)
    print("Cluster job: %s, task_index: %d, target: %s" % (FLAGS.job_name,
                                                            FLAGS.task_index,
                                                            server.target))
    if FLAGS.job_name == "ps":
        server.join()
    elif FLAGS.job_name == "worker":
        # Assigns ops to the local worker by default.
        with tf.device(tf.train.replica_device_setter(
                worker_device="/job:worker/task:%d" % FLAGS.task_index,
                cluster=cluster)):
            # Build model ...
            mnist = input_data.read_data_sets("data", one_hot=True)

            # Create the model
            x = tf.placeholder(tf.float32, [None, 784])
            W = tf.Variable(tf.zeros([784, 10]))
            b = tf.Variable(tf.zeros([10]))
            y = tf.matmul(x, W) + b
            # Define loss and optimizer
            y_ = tf.placeholder(tf.float32, [None, 10])
            cross_entropy = tf.reduce_mean(
                    tf.nn.softmax_cross_entropy_with_logits(y, y_),
                    name="loss_op")
            # loss_summary = tf.summary.scalar("loss", cross_entropy)
            global_step = tf.Variable(0)
            train_op = tf.train.AdagradOptimizer(0.01).minimize(
                cross_entropy, global_step=global_step)
            # Test trained model
            correct_prediction = tf.equal(tf.argmax(y, 1),
                                          tf.argmax(y_, 1),
                                          name="correct_judge_op")
            accuracy = tf.reduce_mean(tf.cast(correct_prediction,
                                              tf.float32))
            accuracy_summary = tf.summary.scalar("accuracy_summary",
                                                 accuracy)

            saver = tf.train.Saver()
            summary_op = tf.summary.merge_all()
            init_op = tf.global_variables_initializer()
        if not os.path.exists("mnist_log"):
            os.mkdir("mnist_log")
        # Create a "Supervisor", which oversees the training process.
        sv = tf.train.Supervisor(is_chief=(FLAGS.task_index == 0),
                                 logdir="mnist_log",
                                 init_op=init_op,
                                 summary_op=summary_op,
                                 saver = saver,
                                 global_step=global_step,
                                 save_model_secs=600)
        # The supervisor takes care of session initialization
        # and restoring from a checkpoint.
        sess = sv.prepare_or_wait_for_session(server.target)
        writer = tf.summary.FileWriter("mnist_log", sess.graph)
        # Start queue runners for the input pipelines (if ang).
        sv.start_queue_runners(sess)
        # Loop until the supervisor shuts down (or 2000 steps have completed).
        step = 0
        while not sv.should_stop() and step < 10000:
            batch_xs, batch_ys = mnist.train.next_batch(100)
            _, step = sess.run([train_op, global_step],
                               feed_dict={x: batch_xs, y_: batch_ys})
            if step % 100 == 0 and FLAGS.task_index != 0:
                res = sess.run(summary_op,
                               feed_dict={x: mnist.test.images,
                                          y_: mnist.test.labels})
                writer.add_summary(res, step)
                print("Step {0} in task {1}".format(step, FLAGS.task_index))
        print("done.")
        if not os.path.exists("variables_saved"):
            os.mkdir("variables_saved")
        saver.save(sess, "variables_saved/variables")
        if FLAGS.task_index != 0:
            print("accuracy: %f" % sess.run(accuracy,
                                            feed_dict={x: mnist.test.images,
                                                       y_: mnist.test.labels}))
if __name__ == "__main__":
    tf.app.run()
```

Like Be the first to like this

No labels