TensorFlow初学者在使用过程中可能遇到的问题及解决办法

由 多多 于2016-11-02 08:30:33发表

注：本文为“小米安全中心”原创，转载请联系“小米安全中心”

上期回顾：Android逆向随笔之遇见MultiDex

TensorFlow是什么

官方的定义–TensorFlow是一个使用数据流图来进行数值计算的开源软件库。简单来说，TensorFlow是Google开源的深度学习框架。

TensorFlow初学者在使用过程中可能遇到的问题及解决办法

1. 出现问题：

```
tensorflow.python.framework.errors.FailedPreconditionError: Attempting
to use uninitialized value Variable
```

运行sess.run()前要记得忘了初始化所有的变量：

```
init_op = tf.initialize_local_variables()
sess.run(init_op)
```

2. 类似Cannot feed value of shape (500,) for Tensor '*', which has shape '(?, 500)'的问题。
   这种一般是给的数据的shape不匹配问题，一维的tensor数据TensorFlow给出的shape会类似（?,500），在确认传入数据无误的情况下，只要reshape成（1,500）就可以，当然你也可以确定1维的维度，然后另1维直接写-1就好了，TensorFlow会自动帮你计算。

```
inference_correct_prediction_value = sess.run(inference_correct_predict
ion, feed_dict={inference_op1: np.reshape(inference_op_value, (1,-1))})
```

3. 在训练过程中，每次运行sess.run(x)时的返回结果不一样。
   Tensorflow中如果直接打印tensor对象，会输出tensor对象的一些属性信息，而不是直接给出tensor对象的值：

```
tensorflow.python.ops.variables.Variable object at 0x4c63f90>
```

   如果需要查看Variable或Constant的值，需要运行sess.run(x)。首先我们开一个交互式的Session，假设x是我们要查看的对象：

```
import tensorflow as tf
x = tf.Variable([1.0,2.0])
sess = tf.InteractiveSession()
x.initializer.run()
print sess.run(x)
```

   假设x有输入要求，那么在查看其值之前需要使用feed操作，填充数据：

```
inputdata = ****
x_value = sess.run(x,feed_dict=inputdata)
print(x_value)
```

训练的时候，每执行一次sess.run(x)就会执行一次训练，像神经网络这种模型，有可能会导致不一样的结果，所以可以在同一个sess.run()中返回多个值，例如

```
inference_correct_prediction_value,inference_accuracy_value = sess.run(
[inference_correct_prediction, inference_accuracy], feed_dict=
{inference_op1: np.reshape(inference_op_value, (-1,1))})
```

run的参数里前面是操作的列表，后面依赖的数据统一放在feed_dict中，这样sess.run()返回的不是tensor对象，而是numpy的ndarray，处理起来就会比较方便了。

4. 出现问题：

```
Ran out of memory trying to allocate 625.0KiB
```

这种建议在运行之前先用gpustat 来查看一下GPU的状态，看是否还有空间，或者哪台GPU有空间。因为如果是使用GPU，TensorFlow默认会在第一块GPU上执行，

```
# gpustat -cup
Sun Oct 30 19:37:09 2016
[0] ***       | 35'C,   0 % |    22 / 11519 MB |
[1] ***       | 36'C,   0 % |    22 / 11519 MB |
```

然后可以通过命令指定GPU来运行：

```
CUDA_VISIBLE_DEVICES='1 2' python ***.py #注意等号前后没有空格
```

还可以在程序中使用device参数来指定在哪块GPU上运行，比如"/cpu:0"代表机器的CPU，"/gpu:0"代表机器的第一个GPU，"/gpu:1"以此类推：

```
with tf.Session() as sess:
    with tf.device("/gpu:1"):
        var1 = tf.constant([[1., 2.]])
        var2 = tf.constant([[3.],[5.]])
        product = tf.matmul(var1, var2)
```

5. 如何保存模型并在模型训练完后查看模型的训练参数？
   TensorFlow的checkpoint机制使得其能够同时支持Online Learning和Continuous Learning，首先，通过tf.train.Saver()将训练好的或者训练过程中的模型保存成checkpoint:

```
_, loss_value, step = sess.run([train_op, loss, global_step])
saver.save(sess,"./checkpoint/checkpoint.ckpt", global_step=step)
```

然后通过restore()函数从本地的checkpoint文件中恢复模型，当然也可以从该点开始继续运行，也就是所谓的Continuous Learning：

```
ckpt = tf.train.get_checkpoint_state("./checkpoint/")
if ckpt and ckpt.model_checkpoint_path:
    print("Continue training from the model {}".format(ckpt.model_check
point_path))
    saver.restore(sess, ckpt.model_checkpoint_path)
    _, loss_value, step = sess.run([train_op, loss, global_step])
```

最后通过tf.trainable_variables()获取返回模型中所训练的参数：

```
for var in tf.trainable_varisbles():
    print var.name
```

6. 如何处理训练数据量太大的情况？

   TensorFlow支持从csv文件和TFRecords文件读取数据，如果从二进制的TFRecords文件读取，可以采用QueueRunner和Coordinator的方式进行多线程读取，通过设置epoch参数控制训练数据文件迭代训练的次数，通过设置batch_size的大小来控制一次训练中从训练数据中取得的样本数量，还可以设置随机选取，有利于加快训练速度。

```
def read_and_decode(filename_queue):#从TFRecords中读取数据
    reader = tf.TFRecordReader()
    _, serialized_example = reader.read(filename_queue)
    features = tf.parse_single_example(serialized_example,
    features={
    "label": tf.FixedLenFeature([], tf.float32),
    "features": tf.FixedLenFeature([FEATURE_SIZE], tf.float32),
    })
    label = features["label"]
    features = features["features"]
return label, features

filename_queue = tf.train.string_input_producer(tf.train.match_filename
s_once(trainFile), num_epochs=epoch_number)
label, features = read_and_decode(filename_queue)
batch_labels, batch_features = tf.train.shuffle_batch([label, features]
, batch_size=batch_size, num_threads=thread_number, capacity=capacity,
min_after_dequeue=min_after_dequeue)
```

这里的trainFile可以是一个文件名的列表：

```
trainFile = ['./data/train_1.tfrecords','./data/train_2.tfrecords']
```

还可以是一个正则表达式：

```
trainFile = './data/*.tfrecords'
```

使用Coordinator来管理队列：

```
coord = tf.train.Coordinator()
threads = tf.train.start_queue_runners(coord=coord, sess=sess)
try:
    while not coord.should_stop():
    _, loss_value, step = sess.run([train_op, loss, global_step])
    saver.save(sess, "./checkpoint/checkpoint.ckpt",global_step=step)
except tf.errors.OutOfRangeError:
    print("Done training after reading all data")
finally:
    coord.request_stop()
```

这里经常会碰到的一个问题是在没有训练之前队列就关闭了，类似“get ‘OutOfRange’, the queue will be closed”的问题，这是因为epoch设置过小，在开始训练前就把数据读完退出了，可以把epoch设置的大一些，如果设置成Nnoe，程序会无限制地一直跑下去，当然你可以在结果足够好的时候手动中断程序的运行。这里就是我的问题啦，有没有什么好的方法来设置epoch参数？

7. 如何让程序分布式运行？

```
python ***.py --ps_hosts=127.0.0.1:2222,127.0.0.1:2223 --
worker_hosts=127.0.0.1:2224,127.0.0.1:2225 --job_name=ps --task_index=0
```

```
python ***.py --ps_hosts=127.0.0.1:2222,127.0.0.1:2223 --
worker_hosts=127.0.0.1:2224,127.0.0.1:2225 --job_name=ps --task_index=1
python ***.py --ps_hosts=127.0.0.1:2222,127.0.0.1:2223 --
worker_hosts=127.0.0.1:2224,127.0.0.1:2225 --job_name=worker --
task_index=0
python ***.py --ps_hosts=127.0.0.1:2222,127.0.0.1:2223 --
worker_hosts=127.0.0.1:2224,127.0.0.1:2225 --job_name=worker --
task_index=1
```

其中，ps是整个训练集群的参数服务器，保存模型的Variable，worker是计算模型梯度的节点，得到的梯度向量会交付给ps更新模型。ps_hosts代表有几个ps，worker_hosts代表有几个worker，ps跟worker都要启动，而且要先启动ps，再启动worker，要启动多个进程，像上面就要启动4个进程，否则会出现下面这个错误：

```
E0830 09:34:30.845674045   51986 tcp_client_posix.c:173]     failed to
connect to 'ipv4:127.0.0.1:2222': socket error: connection refused
```

当然，保险起见的话，前面也是可以指定GPU运行的。

8. 如何给TensorFlow的分布式程序传参？
   分布式通过tf.app.run()运行 ， main()调用的时候有一个下划线的，即：

```
def main(_):#这里的下划线不要忘掉
   ps_hosts = FLAGS.ps_hosts.split(",")
   worker_hosts = FLAGS.worker_hosts.split(",")
   cluster = tf.train.ClusterSpec({"ps": ps_hosts, "worker": worker_hos
ts})
   server = tf.train.Server(cluster, job_name=FLAGS.job_name, task_inde
x=FLAGS.task_index)
   if FLAGS.job_name == "ps":
```

```
        server.join()
    elif FLAGS.job_name == "worker":
       with tf.device(tf.train.replica_device_setter(worker_device="/job
:worker/task:%d" % FLAGS.task_index,
     cluster=cluster)):
       print 'start training'
       ......

if __name__ == "__main__":
     tf.app.run()
```

这里是如何实现传参呢？我们看一下tf.app.run()的源码：

```
 """Generic entry point script."""
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
import sys
from tensorflow.python.platform import flags
def run(main=None):
    f = flags.FLAGS
    f._parse_flags()
    main = main or sys.modules['__main__'].main
    sys.exit(main(sys.argv))
```

执行main函数之前首先进行flags的解析，也就是说TensorFlow通过设置flags来传递tf.app.run()所需要的参数，我们可以直接在程序运行前初始化flags，也可以在运行程序的时候设置命令行参数来达到传参的目的。

```
flags = tf.app.flags
FLAGS = flags.FLAGS
```

```
flags.DEFINE_float('learning_rate', 0.01, 'Initial learning rate.')
flags.DEFINE_integer('epoch_number', None, 'Number of epochs to run tra
iner.')
flags.DEFINE_integer("thread_number", 10 , "Number of thread to read da
ta")
flags.DEFINE_string("mode", "train", "Option mode: train, train_from_sc
ratch, inference")
```

比如我们运行程序的时候可以：

```
python ***.py --mode train
python ***.py --mode inference
```

基本上就这些了，针对TensorFlow应用的系统介绍，推荐@tobe迪豪的文章，希望大家多指正，多交流。