Tensorflow 模型保存、预测、finetuning

由于经常要使用 tensorflow 进行网络训练，但是在用的时候每次都要把模型重新跑一遍，这样就比较麻烦；另外由于某些原因程序意外中断，也会导致训练结果拿不到，而保存中间训练过程的模型可以以便下次训练时继续使用。所以学会 tensorflow 的 save model 和 load model 非常有用。
英文学习手册：A quick complete tutorial to save and restore Tensorflow models

1.什么是 Tensorflow 模型？

当你训练完一个神经网络后，你可能想保存它，方便将来用上，直接用的产品上，不用再经过漫长的训练。Tensorflow 模型主要包含两个内容：1）我们设计的网络图 2）我们训练过程中得到的网络图中的变量(variable)参数值。 因此，Tensorflow 主要由两个文件来描述：

a)**Meta graph**:这个文件主要保存整个 Tensorflow 图模型，包含所有的变量(variables)、操作(operations)、集合(collections)等。这个文件的扩展为**.meta**。

b)**Checkpoint file**:这个二进制文件包含所有的权重值、偏量值、梯度下降值以及其他变量值。这个文件的扩展为.ckpt。

2.如何保存 Tensorflow 模型

语句 1：saver=tf.train.Saver()

在 Tensorflow 中，如果你想保存图模型和所有的参数值，那么首先，我们应该使用 tf.train.Saver()创建一个模型保存类的实例。

语句 2：saver.save(sess,'my_model')

注意：Tensorflow 变量值只存在于会话执行中，因此保存模型的语句 saver.save(sess,'my_model')应该在 with tf.Session() as see:语句之后。

模型保存简单代码：

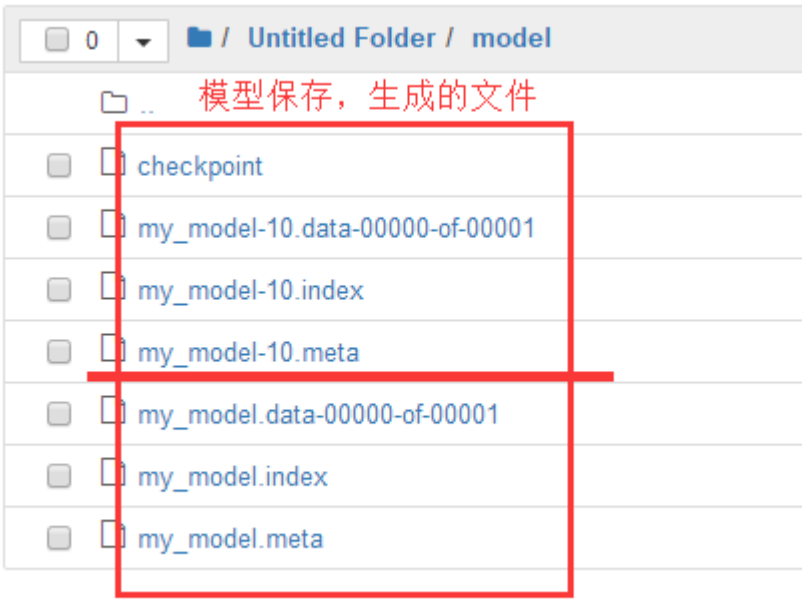
```
import tensorflow as tf
```

```
D:\ProgramData\Anaconda3\lib\site-packages\h5py\__init__.py:36: Fu
nt of issubdtype from `float` to `np.floating` is deprecated. In f
 np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
```

global_step可以设置为模型训练迭代次数，这样可以轻松使用

```
#声明两个变量
w1=tf.Variable(tf.random_normal(shape=[2]),name='w1')
w2=tf.Variable(tf.random_normal(shape=[5]),name='w2')
#保存模型
saver=tf.train.Saver()
with tf.Session() as sess:
    #变量初始化
    sess.run(tf.global_variables_initializer())
    saver.save(sess,'model/my_model')
    saver.save(sess,'model/my_model',global_step=10)
```

saver.save(sess,'模型名',global_step=epoch),保存在章迭代次数的模型参数。



注意：如果 tf.train.Saver()实例化，没有传入具体的参数时，它就会保存模型中所有的变量。如果，我们只想保存自己想要的一些变量，我们就可以把自己想要保存的变量名，以 list 的形式放入 tf.train.Save()中。如下：

```
import tensorflow as tf
```

```
D:\ProgramData\Anaconda3\lib\site-packages\h5py\__init__.
nt of issubdtype from `float` to `np.floating` is depreca
 np.dtype(float).type`.
  from ._conv import register_converters as _register_con
```

```
#声明两个变量
w1=tf.Variable(tf.random_normal(shape=[2]),name='w1')
w2=tf.Variable(tf.random_normal(shape=[5]),name='w2')

#保存模型
saver=tf.train.Saver([w1,w2])
with tf.Session() as sess:
    #变量初始化
    sess.run(tf.global_variables_initializer())
    saver.save(sess,'model/my_model')
```

把自己想要保存的变量名以list的形式传入

3.加载模型

模型的加载需要 2 步：

a）建立图模型

语句 1：saver=tf.train.import_meta_graph('my_model.meta')

我们已经在保存的时候，将模型的图保存在**.meta** 文件中了。因此使用语句：saver=tf.train.import_meta_graph('my_model.meta')。注意：此处虽然重新加载了图模型，但还没有将图中参数的实际数据放入。

b）加载参数

语句 2：saver.restore(sess,tf.train.latest_checkpoint('你的模型路径名')

使用 saver(它是 tf.train.Saver()类实例化的对象)调用 restore()方法，就能够加载模型参数。

模型加载简单代码：

4.使用存储好的模型进行工作

前面已经学会了模型保存 save 和模型加载 restore。此处要学习利用提前训练好的模型的参数，去预测、调试、或进一步训练。

无论什么时候使用 Tensorflow，自己定义的模型图，都需要放入训练数据集和一些参数。标准的方式是，将训练集和参数使用 placeholder 占位符进行填充。**注意：当模型**

保存的时候，其实使用 placeholder 占位符填充的的数据是没有保存的。

```
import tensorflow as tf
```

```
D:\ProgramData\Anaconda3\lib\site-packages\h5py\__init__.py:36: Fu
nt of issubdtype from `float` to `np.floating` is deprecated. In f
 np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
```

```
#预定义一些占位符
w1=tf.placeholder("float",name="w1")
w2=tf.placeholder("float",name="w2")
b1=tf.Variable(2.0,name="bias")
feed_dict={w1:4,w2:8}

#定义图
w3=tf.add(w1,w2)
w4=tf.multiply(w3,b1,name='op_to_restore')

#执行图
with tf.Session() as sess:
    #变量初始化
    sess.run(tf.global_variables_initializer())
    #实例化一个保存模型的对象，它可以保存所有的变量
    saver=tf.train.Saver()
    #打印（w1+w2）*b1
    print(sess.run(w4,feed_dict))

    #保存图
    saver.save(sess,'model/predict_model',global_step=1)
```

注意：在模型参数保存是，占位符的数据是不保存的

保存模型必须的两个语句

```
24.0
```

A.原始模型，新数据

利用原始的网络结构，不同的数据集，我们只需将新的数据集通过 feed_dict 传入原始模型中就行。具体步骤如下：

1）创建会话 tf.Session()

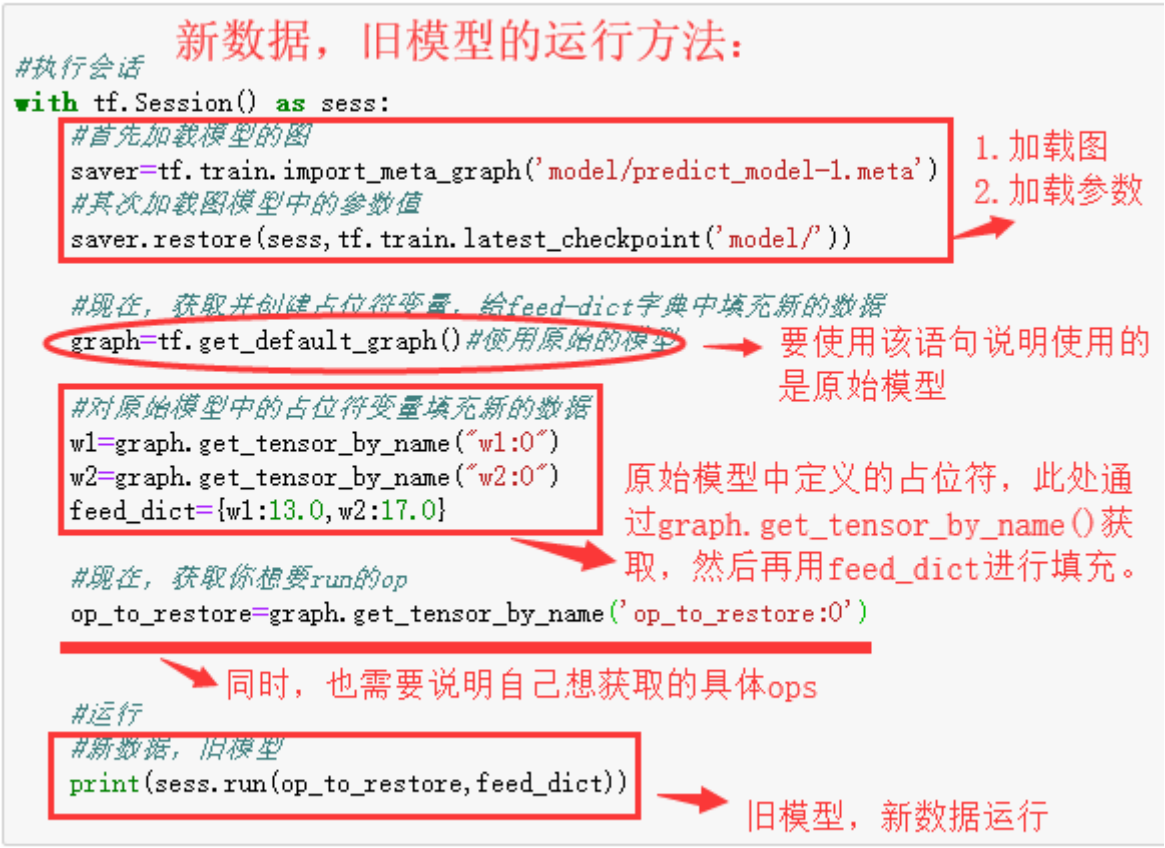
2）加载模型 (加载图和加载参数)

3）使用 graph=tf.get_default_graph()语句说明使用原始模型

4）获取原始模型中的 placeholder 和自己想要的 ops，然后通过 feed_dict 字典，对占位符填充新的数据集。

5）使用 sess.run()运行自己想要 op 和输入新数据集

```
import tensorflow as tf
```




```
INFO:tensorflow:Restoring parameters from model/predict_model-1
60.0
```

B.原始模型变新模型，新数据集

如果想要在原始训练好的模型的基础上添加更多的操作，然后再训练这个新模型，同样也是可以做到的。具体步骤如下：

1）创建会话 tf.Session()

2）加载模型 (加载图和加载参数)

3）使用 graph=tf.get_default_graph()语句说明使用原始模型

4）获取原始模型中的 placeholder，然后通过 feed_dict 字典，对占位符填充新的数据集。

5）获取原始模型中你想要的 ops，然后添加新的 ops，就构成了新模型。

6）使用 sess.run()运行自己想要 op 和输入新数据集

```
import tensorflow as tf
```

```
#执行会话
with tf.Session() as sess:
    #首先，加载模型的图
    saver=tf.train.import_meta_graph('model/predict_model-1.meta')
    #其次，加载图模型中的参数值
    saver.restore(sess,tf.train.latest_checkpoint('model/'))

    #获取原始模型图，
    graph=tf.get_default_graph()

    #对原始模型中的占位符变量填充新的数据
    w1=graph.get_tensor_by_name("w1:0")
    w2=graph.get_tensor_by_name("w2:0")
    feed_dict={w1:12.0,w2:18.0}

    #获取你想从原始模型中，想得到的ops
    op_to_restore=graph.get_tensor_by_name("op_to_restore:0")

    #在原始模型图中，添加新的ops
    add_on_op=tf.multiply(op_to_restore,2)

    #运行
    #新数据，新模型
    print(sess.run(add_on_op,feed_dict))
```

1. 加载图
2. 加载参数

说明自己使用原始模型图

对原始模型中的placeholder填充上新的数据集

从原始模型中，获取你想要的ops，然后再添加新的ops，则形成了新模型

运行，新数据，新模型

```
INFO:tensorflow:Restoring parameters from model/predict_model-1
120.0
```

模型预测

```
154
155 def predict():
156
157     with tf.Session() as sess:
158         #首先，加载模型的图
159         saver=tf.train.import_meta_graph('my_model/my-model-2.meta')
160         #其次，加载模型中的参数
161         saver.restore(sess,tf.train.latest_checkpoint('my_model/'))
162
163         #获取原始图模型
164         graph=tf.get_default_graph()
165
166         #新数据集准备
167         train_data = open("dataset/s_formated.txt", "r", encoding='utf-8').readlines()
168         predict_data = open("dataset/s_formated_test.txt", "r", encoding='utf-8').readline
169         predict_data_ed = data_pipeline(predict_data)
170         train_data_ed = data_pipeline(train_data)
171
172         # 对应成索引
173         word2index, index2word, slot2index, index2slot, intent2index, index2intent = \
174             get_info_from_training_data(train_data_ed)
175
176         # 需要预测数据集的索引已经得到了
177         index_predict = to_index(predict_data_ed, word2index, slot2index, intent2index)
178
176         # 需要预测数据集的索引已经得到了
177         index_predict = to_index(predict_data_ed, word2index, slot2index, intent2index)
178
179         #获取原始模型中的占位符变量
180         encoder_inputs = graph.get_tensor_by_name('encoder_inputs:0')
181         encoder_inputs_actual_length = graph.get_tensor_by_name('encoder_inputs_actual_le
182         decoder_targets = graph.get_tensor_by_name('decoder_targets:0')
183         intent_targets = graph.get_tensor_by_name('intent_targets:0')
184
185         #获取自己想在原始模型中需要的op
186         decode = graph.get_tensor_by_name('decode:0')
187         intent = graph.get_tensor_by_name('intent:0')
188         decoder_prediction = decode.sample_id
189
190         #分批放入新数据
191         pred_slots = []
192         slot_accs = []
193         intent_accs = []
194         for j, batch in enumerate(getBatch(batch_size, index_predict)):
195             print(j, batch)
196             unziped = list(zip(*batch))
197             output_feeds = [decoder_prediction, intent]
198             feed_dict = {encoder_inputs: np.transpose(unziped[0], [1, 0]),
199                          encoder_inputs_actual_length: unziped[1]}
200             decoder_prediction, intent = sess.run(output_feeds, feed_dict=feed_dict)
```

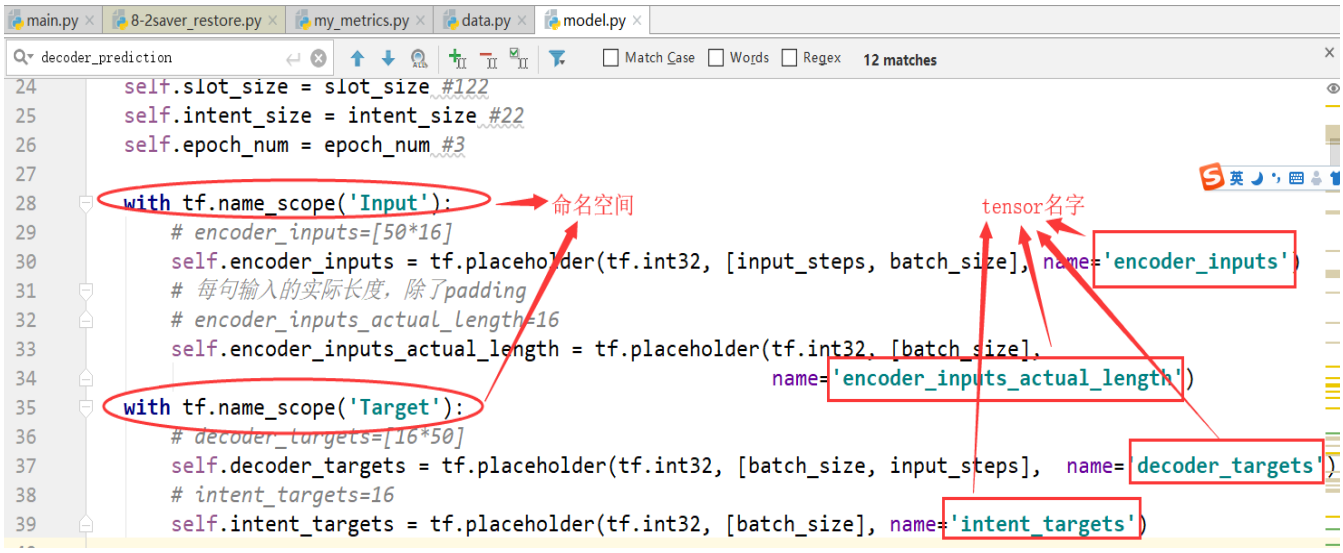
```
    for j, batch in enumerate(getBatch(batch_size, index_predict)):
        print(j, batch)
        unziped = list(zip(*batch))
        output_feeds = [decoder_prediction, intent]
        feed_dict = {encoder_inputs: np.transpose(unziped[0], [1, 0]),
                     encoder_inputs_actual_length: unziped[1]}
        decoder_prediction, intent = sess.run(output_feeds, feed_dict=feed_dict)
        decoder_prediction = np.transpose(decoder_prediction, [1, 0])

        slot_pred_length = list(np.shape(decoder_prediction))[1]
        pred_padded = np.lib.pad(decoder_prediction, ((0, 0), (0, input_steps - slot_pred_length)),
                                 mode="constant", constant_values=0)
        pred_slots.append(pred_padded)
        true_slot = np.array((list(zip(*batch))[2]))
        true_length = np.array((list(zip(*batch))[1]))
        true_slot = true_slot[:, :slot_pred_length]
        slot_acc = accuracy_score(true_slot, decoder_prediction, true_length)
        intent_acc = accuracy_score(list(zip(*batch))[3], intent)
        slot_accs.append(slot_acc)
        intent_accs.append(intent_acc)
        print('slot_acc_min_batch:', slot_acc)
        print('intent_acc_min_batch:', intent_acc)

    pred_slots_a = np.vstack(pred_slots)
    true_slots_a = np.array(list(zip(*index_predict))[2])[:pred_slots_a.shape[0]]
    print("Intent predict accuracy is: {}".format(np.average(intent_accs)))
    print("Slot predict accuracy is: {}".format(np.average(slot_accs)))
    print("Slot predict F1 score is: {}".format(f1_for_sequence_batch(true_slots_a, pred_slots_a)))
```

出现问题：KeyError: "The name 'encoder_inputs:0' refers to a Tensor which does not exist. The operation, 'encoder_inputs', does not exist in the graph."　错误提示：我保存的模型中，占位符 encoder_inputs 没有存在我当前保存的图中。

出现该问题的原因：这个 placeholder，被放在了自定义命名空间 Input 下了，所以在获取这个变量名时，路径就没有写对。



问题解决办法：修改 graph.get_tensor_by_name('tensor 正确的路径名')。

出现问题：KeyError: "The name 'decode:0' refers to a Tensor which does not exist. The operation, 'decode', does not exist in the graph." 错误提示：我想获取的 decode 其实不是一个简单的 op，并不存在于保存的图模型中。

其实 decode 在程序中是一个自定义函数。可以使用 tf.identity 给 op 命名。具体改动：

```
self.decoder_prediction = outputs.sample_id
tf.identity(outputs.sample_id,name='yaojuan')
```

使用 tf.identity()重新命名 tensor 名后，就能够很好的使用 graph.get_tensor_by_name()获取 op。

```
#分批放入新数据
pred_slots = []
slot_accs = []
intent_accs = []
for j, batch in enumerate(getBatch(batch_size, index_predict)):
    # print(j, batch)
    unziped = list(zip(*batch))
    #获取原始模型中,自己想要的op
    intent = graph.get_tensor_by_name('intent:0')
    decoder_prediction = graph.get_tensor_by_name('yaojuan:0')

    output_feeds = [decoder_prediction,intent]
    feed_dict = {encoder_inputs: np.transpose(unziped[0], [1, 0]),
                 encoder_inputs_actual_length: unziped[1]}
    decoder_prediction, intent = sess.run(output_feeds, feed_dict=feed_dict)
    decoder_prediction = np.transpose(decoder_prediction, [1, 0])
```

注意，这里由于是分批训练，所以每次都要从原始模型中去获取op　　如果获取op放在for循环外面，就会报错。