# Tensorflow模型的保存与恢复

## 方法一

Tensorflow提供了一个tf.train.Saver类来保存和恢复网络模型，简单介绍模型的使用方法。

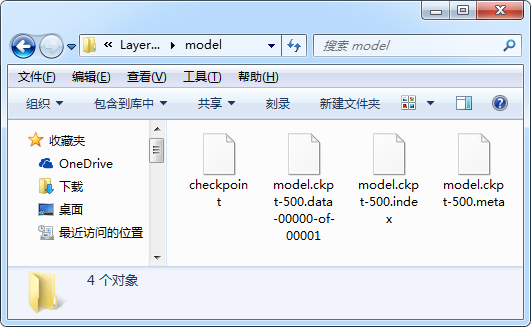
### 1.1模型的保存

在定义好网络的结构后，使用saver=tf.train.Saver()创建一个Saver对象，在训练结束后使用saver.save(sess, 'path/to/model/model.ckpt')来保存模型，在相应文件夹下会生成四个文件：

Checkpoint:保存了目录下所有模型文件的列表

Model.ckpt.data和Model.ckpt.index:保存模型参数

Model.ckpt.meta:保存了计算图的结构



### 1.2模型的恢复

恢复模型分为恢复模型结构和恢复模型参数两个步骤，也可以不从文件中恢复模型结构而重新实现与原结构一样的计算图。

使用saver=tf.train.import\_meta\_graph('path/to /model.ckpt.meta')来恢复网络结构。

使用saver.restore(sess, 'path/to/model.ckpt')来恢复网络参数。

除此之外，为了运行网络，还需要从图中读取到网络的输入tensor和输出tensor，这可以使用tf.get\_tensor\_by\_name获取对应的tensor，tensor的名称形式为<op\_name>:<output\_idx>，例如恢复网络的输入和输出可以实现：

|  |
| --- |
| x = graph.get\_tensor\_by\_name('x:0') y\_true = graph.get\_tensor\_by\_name('y\_true:0') accuracy = graph.get\_tensor\_by\_name('accuracy:0') |

另外，要注意的是，从文件中恢复模型后，不要再像训练模型阶段那样，使用sess.run(tf.global\_variables\_initializer())来初始化参数，否则参数值会被冲掉，特别注意！

## 方法二

在上面基本方法中，将模型与参数保存为几个文件，下面的方法将模型和参数保存到同一文件，比较好用。

### 2.1模型的保存

在模型训练结束后，使用convert\_variables\_to\_constants方法将指定节点转换为常数，通常为网络的输出，使用tf.train.write\_graph将图保存。以下代码演示了这些操作：

|  |
| --- |
| **import** tensorflow **as** tf **from** tensorflow.python.framework.graph\_util **import** convert\_variables\_to\_constants  input1 = tf.placeholder(tf.float32, name=**'input1'**) input2 = tf.placeholder(tf.float32, name=**'input2'**) output = tf.add(input1, input2, name = **'output'**) **with** tf.Session() **as** sess:  sess.run(tf.global\_variables\_initializer())  print(sess.run(output, feed\_dict={input1:1,input2:2}))  graph = convert\_variables\_to\_constants(sess, sess.graph\_def, [**"output"**])  tf.train.write\_graph(graph, **'model'**, **'graph.pb'**, as\_text=**False**) |

输出：



这里write\_graph函数的参数as\_text设置为False，模型以二进制的形式保存，如果设置为True,将以文本形式保存。在恢复模型时，要将读取方式设置为相应的形式。

### 2.2模型的恢复

使用open函数以指定的形式（二进制或文本）打开模型，使用ParseFromString将模型读取到graph\_def文件，使用import\_graph\_def从graph\_def文件中导出指定节点和placeholder，以下程序演示了该方法：

|  |
| --- |
| **import** tensorflow **as** tf new\_input1 = tf.placeholder(tf.float32, shape=()) new\_input2 = tf.placeholder(tf.float32, shape=()) **with** tf.Session() **as** sess:  **with** open(**'model/graph.pb'**, **'rb'**) **as** f:  graph\_def = tf.GraphDef()  graph\_def.ParseFromString(f.read())  new\_output = tf.import\_graph\_def(graph\_def, input\_map={**'input1:0'**:new\_input1, **'input2:0'**: new\_input2},  return\_elements=[**'output:0'**])  print(sess.run(new\_output, feed\_dict={new\_input1:4, new\_input2:3})) |

输出：



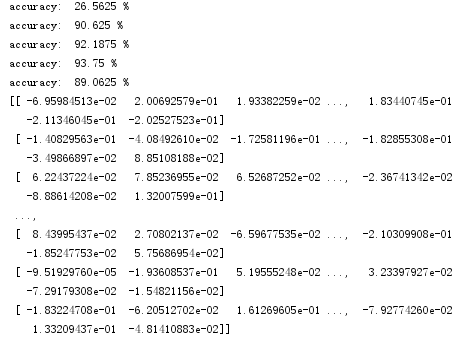
## 实例

### 3.1 方法一

#### 3.1.1 训练模型

|  |
| --- |
| **import** tensorflow **as** tf **from** tensorflow.examples.tutorials.mnist **import** input\_data data=input\_data.read\_data\_sets(**'F:/tmp/data'**, one\_hot=**True**)  IMG\_SIZE=28 IMG\_SIZE\_FLAT=IMG\_SIZE\*IMG\_SIZE IMG\_SHAPE=(IMG\_SIZE,IMG\_SIZE) NUM\_CHANNEL=1 NUM\_CLASSES=10 BATCH\_SIZE = 64 NUM\_ITERATIONS=500  **def** get\_weights\_variable(layer\_name):  **with** tf.variable\_scope(layer\_name, reuse=**True**):  variable=tf.get\_variable(**'kernel'**)  **return** variable  x=tf.placeholder(tf.float32,shape=[**None**, IMG\_SIZE\_FLAT], name=**'x'**) x\_image=tf.reshape(x, [-1, IMG\_SIZE, IMG\_SIZE, NUM\_CHANNEL]) y\_true=tf.placeholder(tf.float32,shape=[**None**, NUM\_CLASSES], name=**'y\_true'**) y\_true\_cls=tf.argmax(y\_true, dimension=1)  net=x\_image net=tf.layers.conv2d(inputs=net, name=**'layer\_conv1'**, padding=**'same'**, filters=16, kernel\_size=5,activation=tf.nn.relu) layer1\_conv1=net net=tf.layers.conv2d(inputs=net, name=**'layer\_conv2'**, padding=**'same'**, filters=36, kernel\_size=5,activation=tf.nn.relu) layer2\_conv2=net net=tf.layers.max\_pooling2d(inputs=net, pool\_size=2,strides=2) net=tf.contrib.layers.flatten(net) net=tf.layers.dense(inputs=net,name=**'layer\_fc1'**, units=128, activation=tf.nn.relu) net=tf.layers.dense(inputs=net,name=**'layer\_out'**, units=NUM\_CLASSES, activation=**None**) logits=net y\_pred=tf.nn.softmax(logits=logits) y\_pred\_cls=tf.argmax(y\_pred,dimension=1) cross\_entropy=tf.nn.softmax\_cross\_entropy\_with\_logits(labels=y\_true, logits=logits) loss=tf.reduce\_mean(cross\_entropy)  optimizer=tf.train.AdamOptimizer(learning\_rate=0.0001).minimize(loss) correct\_prediction=tf.equal(y\_pred\_cls,y\_true\_cls) accuracy=tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32), name=**'accuracy'**) saver = tf.train.Saver() **with** tf.Session() **as** sess:  sess.run(tf.global\_variables\_initializer())  **for** i **in** range(NUM\_ITERATIONS):  x\_batch, y\_true\_batch = data.train.next\_batch(BATCH\_SIZE)  feed\_dict\_train={x:x\_batch,y\_true:y\_true\_batch}  sess.run(optimizer, feed\_dict=feed\_dict\_train)  **if** i%100 == 0:  acc=sess.run(accuracy, feed\_dict=feed\_dict\_train)  print(**'accuracy: '**, acc \* 100, **'%'**)  saver.save(sess, **'F:/tensorflow\_project/LayerAPITest/model/model.ckpt'**, global\_step = NUM\_ITERATIONS)  w2=get\_weights\_variable(**'layer\_out'**)  print(sess.run(w2)) |

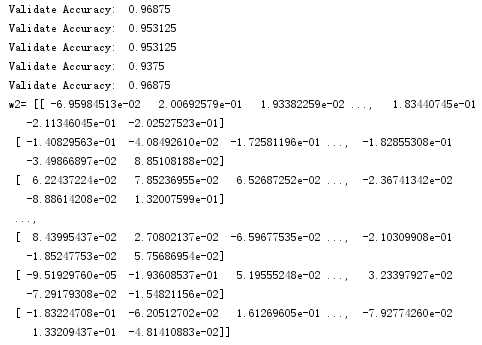
输出：



#### 3.1.2 恢复模型

|  |
| --- |
| *#coding=utf-8* **import** tensorflow **as** tf **from** tensorflow.examples.tutorials.mnist **import** input\_data data=input\_data.read\_data\_sets(**'F:/tmp/data'**, one\_hot=**True**)  BATCH\_SIZE = 64 saver=tf.train.import\_meta\_graph(**'F:/tensorflow\_project/LayerAPITest/model/model.ckpt.meta'**) **with** tf.Session() **as** sess:  saver.restore(sess, **'F:/tensorflow\_project/LayerAPITest/model/model.ckpt'**)  graph = tf.get\_default\_graph()  x = graph.get\_tensor\_by\_name(**'x:0'**)  y\_true = graph.get\_tensor\_by\_name(**'y\_true:0'**)  accuracy = graph.get\_tensor\_by\_name(**'accuracy:0'**)   w2=graph.get\_tensor\_by\_name(**'layer\_out/kernel:0'**)  **for** i **in** range(5):  x\_val, y\_val = data.validation.next\_batch(BATCH\_SIZE)  validate\_feed = {x: x\_val, y\_true: y\_val}  acc = sess.run(accuracy, feed\_dict=validate\_feed)  print(**"Validate Accuracy: "**, acc)  print(**"w2="**, sess.run(w2)) |

输出：

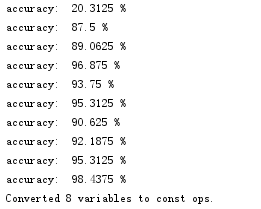


### 3.2 方法二

#### 3.2.1 训练模型

|  |
| --- |
| **import** tensorflow **as** tf **from** tensorflow.examples.tutorials.mnist **import** input\_data data=input\_data.read\_data\_sets(**'F:/tmp/data'**, one\_hot=**True**)  IMG\_SIZE=28 IMG\_SIZE\_FLAT=IMG\_SIZE\*IMG\_SIZE IMG\_SHAPE=(IMG\_SIZE,IMG\_SIZE) NUM\_CHANNEL=1 NUM\_CLASSES=10 BATCH\_SIZE = 64 NUM\_ITERATIONS=1000  input\_x=tf.placeholder(tf.float32,shape=[**None**, IMG\_SIZE\_FLAT], name=**'x'**) x\_image=tf.reshape(input\_x, [-1, IMG\_SIZE, IMG\_SIZE, NUM\_CHANNEL]) input\_y=tf.placeholder(tf.float32,shape=[**None**, NUM\_CLASSES], name=**'y\_true'**) y\_true\_cls=tf.argmax(input\_y, dimension=1)  net=x\_image net=tf.layers.conv2d(inputs=net, name=**'layer\_conv1'**, padding=**'same'**, filters=16, kernel\_size=5,activation=tf.nn.relu) layer1\_conv1=net net=tf.layers.conv2d(inputs=net, name=**'layer\_conv2'**, padding=**'same'**, filters=36, kernel\_size=5,activation=tf.nn.relu) layer2\_conv2=net net=tf.layers.max\_pooling2d(inputs=net, pool\_size=2,strides=2) net=tf.contrib.layers.flatten(net) net=tf.layers.dense(inputs=net,name=**'layer\_fc1'**, units=128, activation=tf.nn.relu) net=tf.layers.dense(inputs=net,name=**'layer\_out'**, units=NUM\_CLASSES, activation=**None**) logits=net y\_pred=tf.nn.softmax(logits=logits) y\_pred\_cls=tf.argmax(y\_pred,dimension=1) cross\_entropy=tf.nn.softmax\_cross\_entropy\_with\_logits(labels=input\_y, logits=logits) loss=tf.reduce\_mean(cross\_entropy)  optimizer=tf.train.AdamOptimizer(learning\_rate=0.0001).minimize(loss) correct\_prediction=tf.equal(y\_pred\_cls,y\_true\_cls) accuracy=tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32), name=**'accuracy'**)  **with** tf.Session() **as** sess:  sess.run(tf.global\_variables\_initializer())  **for** i **in** range(NUM\_ITERATIONS):  x\_batch, y\_true\_batch = data.train.next\_batch(BATCH\_SIZE)  feed\_dict\_train={input\_x:x\_batch,input\_y:y\_true\_batch}  sess.run(optimizer, feed\_dict=feed\_dict\_train)  **if** i%100 == 0:  acc=sess.run(accuracy, feed\_dict=feed\_dict\_train)  print(**'accuracy: '**, acc \* 100, **'%'**)  graph = tf.graph\_util.convert\_variables\_to\_constants(sess, sess.graph\_def,[**"accuracy"**])  tf.train.write\_graph(graph, **'model'**, **'graph.pb'**, as\_text=**False**) |

输出：



#### 3.2.2 恢复模型

|  |
| --- |
| *#coding=utf-8* **import** tensorflow **as** tf **from** tensorflow.examples.tutorials.mnist **import** input\_data data=input\_data.read\_data\_sets(**'F:/tmp/data'**, one\_hot=**True**)  IMG\_SIZE=28 IMG\_SIZE\_FLAT=IMG\_SIZE\*IMG\_SIZE NUM\_CLASSES=10 BATCH\_SIZE = 64  input\_x=tf.placeholder(tf.float32,shape=[**None**, IMG\_SIZE\_FLAT]) input\_y=tf.placeholder(tf.float32,shape=[**None**, NUM\_CLASSES])  **with** tf.Session() **as** sess:  **with** open(**'model/graph.pb'**, **'rb'**) **as** f:  graph\_def = tf.GraphDef()  graph\_def.ParseFromString(f.read())  accuracy = tf.import\_graph\_def(graph\_def, input\_map={**'x:0'**: input\_x, **'y\_true:0'**: input\_y},  return\_elements=[**'accuracy:0'**])  **for** i **in** range(5):  x\_val, y\_val = data.validation.next\_batch(BATCH\_SIZE)  validate\_feed = {input\_x: x\_val, input\_y: y\_val}  acc = sess.run(accuracy, feed\_dict=validate\_feed)  print(**"Validate Accuracy: "**, acc) |

输出：

