tensorflow, running many times in the same file with error :993] Not found: Key in checkpoint

I have trained a model using MNIST and I want to make a small app. And whenever I call my prediction function for more than 2 times, the error would occur.

```
W c:\tf_jenkins\home\workspace\release-
win\device\cpu\os\windows\tensorflow\core\framework\op_kernel.cc:993] Not found:
Key v1_1 not found in checkpoint

W c:\tf_jenkins\home\workspace\release-
win\device\cpu\os\windows\tensorflow\core\framework\op_kernel.cc:993] Not found:
Key v4_1 not found in checkpoint

W c:\tf_jenkins\home\workspace\release-
```

```
win\device\cpu\os\windows\tensorflow\core\framework\op_kernel.cc:993] Not found:
Key v2_1 not found in checkpoint

W c:\tf_jenkins\home\workspace\release-
win\device\cpu\os\windows\tensorflow\core\framework\op_kernel.cc:993] Not found:
Key v3_1 not found in checkpoint
```

so I track all variables,here is the first time I call the function:

```
v1:0
v2:0
v3:0
v4:0
```

There have four variables in my model, but when I call my function again, the variables become this:

```
v1:0
v2:0
v3:0
v4:0
v1_1:0
v2_1:0
v3_1:0
v4_1:0
```

I debugged my code, it seems that when I call the prediction function again, all the variables are assigned a new name and append to the old ones. I don't know how to fix it, please help me!

here is my prediction function(I have trained my model before and here I just restore it)

```
def predictint(imvalue):

#define the model
n_hidden_1 = 256
n_input = 784
n_classes = 10

# Inputs and Outputs
x = tf.placeholder(tf.float32, [None, n_input])
weights = {
    'w1': tf.Variable(tf.random_normal([n_input, n_hidden_1],
stddev=0.1),name='v1'),
    'out': tf.Variable(tf.zeros([n_hidden_1, n_classes]),name='v2')
}

biases = {
    'b1': tf.Variable(tf.zeros([n_hidden_1]),name='v3'),
    'out': tf.Variable(tf.zeros([n_classes]),name='v4')
}

def multilayer_perceptron(_X, _weights, _biases):   tf.nn.relu(tf.add(tf.matmul(_X,
_weights['w1']), _biases['b1']))
    return (tf.matmul(layer_1, _weights['out'] + _biases['out']))

pred = multilayer_perceptron(x, weights, biases)
#init=tf.global_variables_initializer()
saver=tf.train.Saver()
#saver=tf.train.Saver([weights['w1'],weights['out'],biases['b1'],biases['out']])

'''
Load the model.ckpt file which is stored in the same directory as this python
script
 '''
all_vars=tf.trainable_variables()
for v in all_vars:
    print(v.name)
with tf.Session() as sess:
    saver.restore(sess,"E:/Qt/haha/actual.ckpt")
    prediction=tf.argmax(pred,1)
    return prediction.eval(feed_dict={x:[imvalue]},session=sess)
```

python tensorflow

asked Apr 3 at 11:29

Ge Wang
1 1

1 Answer

This problem arises because each call to `predictint()` will add nodes (including new variable nodes) to the same TensorFlow graph, which accounts for the additional variables with the suffix `"_1"` that you see in the second execution.

The easiest solution is to wrap each call to `predictint()` as follows:

```
def predictint(imvalue)
  # This `with` statement ensures that a new, empty graph is used as the container
```

```
    # for all nodes created inside the following block.
    with tf.Graph().as_default():

      # Define the model
      n_hidden_1 = 256
      n_input = 784
      n_classes = 10

      # Inputs and Outputs
      x = tf.placeholder(tf.float32, [None, n_input])
      weights = {
          'w1': tf.Variable(tf.random_normal([n_input, n_hidden_1], stddev=0.1),
  name='v1'),
          'out': tf.Variable(tf.zeros([n_hidden_1, n_classes]), name='v2')
      }

      biases = {
          'b1': tf.Variable(tf.zeros([n_hidden_1]),name='v3'),
          'out': tf.Variable(tf.zeros([n_classes]),name='v4')
      }

      def multilayer_perceptron(_X, _weights, _biases):
        layer_1 = tf.nn.relu(tf.add(tf.matmul(_X, _weights['w1']), _biases['b1']))
        return (tf.matmul(layer_1, _weights['out'] + _biases['out']))

      pred = multilayer_perceptron(x, weights, biases)
      saver=tf.train.Saver()

      # This will only print four names: "v1:0", "v2:0", "v3:0", and "v4:0".
      all_vars = tf.trainable_variables()
      for v in all_vars:
        print(v.name)

      with tf.Session() as sess:
        saver.restore(sess, "E:/Qt/haha/actual.ckpt")
        prediction = tf.argmax(pred, 1)
        return prediction.eval(feed_dict={x:[imvalue]},session=sess)
```

The above solution should work fine. A more efficient solution would be to create a single graph and session, load the checkpoint once, and reuse it for all calls to `predictint()` .

answered Apr 3 at 15:35

mrry
46.8k 3 112 162

Thank you very much! your solutions perfectly solve my problem, I am a beginner in tensorflow, so much to learn~ thank you again~ – Ge Wang 2 days ago