Tnsorflow

## Tf参数共享机制

再造孪生网络的时候，最重要的是tensorflow的参数共享机制，tensorflow提供了Variable Scope这种独特机制来共享变量。这种机制涉及两个主要函数：

Tf.get\_variable(name, shape, initializer)

Tf.variable\_scope(scope\_name)

Tf.get\_variable()工作机制

当tf.get\_variable\_scope().reuse==False，调用该函数会创建新的变量

with tf.variable\_scope(‘test’):

v = tf.get\_variable(‘v’, [1])

v.name=”test/v:0”

当tf.get\_variable\_scope().reuse==True，调用该函数会重用已经创建的变量

with tf.variable\_scope(‘test’):

v = tf.get\_variable(‘v’, [1])

with tf.variable\_scope(‘test’, reuse=True):

v1 = tf.get\_variable(‘v’, [1])

print(v)

print(v1)

结果

<tf.Variable 'test/v:0' shape=(1,) dtype=float32\_ref>

<tf.Variable 'test/v:0' shape=(1,) dtype=float32\_ref>

v与v1是同一个变量

reuse还有继承的机制，例如下面例子，m1上层的test2继承了test中的reuse

with tf.variable\_scope('test'):

v = tf.get\_variable('v', [1])

with tf.variable\_scope('test2'):

m = tf.get\_variable('m', [1])

with tf.variable\_scope('test', reuse=True):

v1 = tf.get\_variable('v1', [1])

with tf.variable\_scope('test2'):

m1 = tf.get\_variable('m', [1])

print(v)

print(v1)

print(m)

print(m1)

结果

<tf.Variable 'test/v:0' shape=(1,) dtype=float32\_ref>

<tf.Variable 'test/v1:0' shape=(1,) dtype=float32\_ref>

<tf.Variable 'test/test2/m:0' shape=(1,) dtype=float32\_ref>

<tf.Variable 'test/test2/m:0' shape=(1,) dtype=float32\_ref>

v=v1 m=m1

tf.variable\_scope

tf.variable\_scope()用来指定变量的作用域，作为变量名的前缀，支持嵌套，

## tf.variable\_scope与tf.name\_scope的区别

tf.variable\_scope可以让变量有相同的命名，包括tf.get\_variable得到的变量，还有tf.variable的变量。

Tf.name\_scope可以让变量有相同的命名，只是限于tf.variable的变量。

例如

with tf.name\_scope('a\_name\_scope'):

initilizer = tf.constant\_initializer(value=1)

var1 = tf.get\_variable(name='var1', shape=[1], dtype=tf.float32, initializer=initilizer)

var2 = tf.Variable(name='var2', initial\_value=[2], dtype=tf.float32)

print(var1)

print(var2)

输出为：

<tf.Variable 'var1:0' shape=(1,) dtype=float32\_ref>

<tf.Variable 'a\_name\_scope/var2:0' shape=(1,) dtype=float32\_ref>

with tf.name\_scope('a\_name\_scope'):

initilizer = tf.constant\_initializer(value=1)

var1 = tf.get\_variable(name='var1', shape=[1], dtype=tf.float32, initializer=initilizer)

var2 = tf.Variable(name='var2', initial\_value=[2], dtype=tf.float32)

with tf.name\_scope('a\_name\_scope2'):

initilizer = tf.constant\_initializer(value=1)

var1 = tf.get\_variable(name='var1', shape=[1], dtype=tf.float32, initializer=initilizer)

var2 = tf.Variable(name='var2', initial\_value=[2], dtype=tf.float32)

运行报错

Variable var1 already exists, disallowed. Did you mean to set reuse=True or reuse=tf.AUTO\_REUSE in VarScope? Originally defined at:

with tf.variable\_scope('test'):

v1 = tf.get\_variable('v', [1])

with tf.variable\_scope('test2'):

m1 = tf.get\_variable('m', [1])

var22 = tf.Variable(name='var22', initial\_value=[2.2], dtype=tf.float32)

print(v1)

print(m1)

print(var22)

输出为

<tf.Variable 'test/test2/m:0' shape=(1,) dtype=float32\_ref>

<tf.Variable 'test/test2/m:0' shape=(1,) dtype=float32\_ref>

<tf.Variable 'test\_1/test2/var22:0' shape=(1,) dtype=float32\_ref>

总结

name\_scope 对 get\_variable新建变量的name属性无影响；对variable新建变量的name属性增加了“范围”标识。

variable\_scope对get\_variable新建变量的name属性和variable新建变量的name属性都增加了“范围”标识。

with tf.name\_scope('a\_name\_scope'):

initilizer = tf.constant\_initializer(value=1)

var2 = tf.Variable(name='var2', initial\_value=[2], dtype=tf.float32)

var3 = tf.Variable(name='var2', initial\_value=[2], dtype=tf.float32)

with tf.name\_scope('a\_name\_scope'):

initilizer = tf.constant\_initializer(value=1)

var4 = tf.Variable(name='var2', initial\_value=[2], dtype=tf.float32)

print(var2)

print(var3)

print(var4)

运行结果

<tf.Variable 'a\_name\_scope/var2:0' shape=(1,) dtype=float32\_ref>

<tf.Variable 'a\_name\_scope/var2\_1:0' shape=(1,) dtype=float32\_ref>

<tf.Variable 'a\_name\_scope\_1/var2:0' shape=(1,) dtype=float32\_ref>

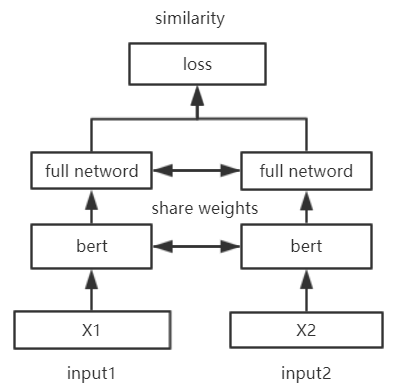
get\_variable新建变量如果遇见重复的name则会因为重复而报错。

variable新建的变量如果遇见重复的name则会自动修改前缀，以避免重复出现。

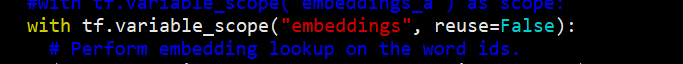
tf.Variable() variable 且以大写字母开头，该函数在于定义一个变量；tf.get\_variable()：可根据 name 值，返回该变量，如果该 name 不存在的话，当reuse=True的时候则会进行创建；当reuse=False的时候会报错。

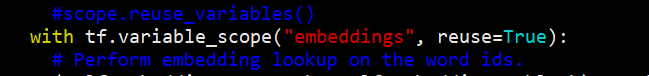
## 孪生网络改造

在改造孪生网络模型的时候，我们可以看到bert层与全连接层的网络会共享参数，这里使用的是get\_variable\_scope()函数来实现网络共享。



改造是在构建模型的时候，my\_modeling3.py文件里，



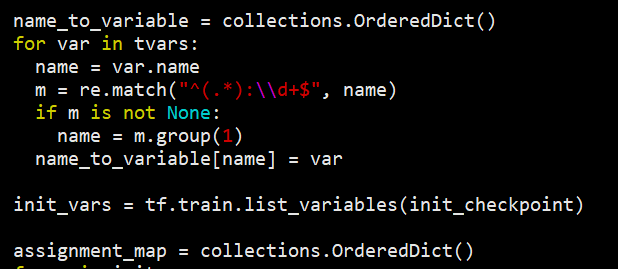


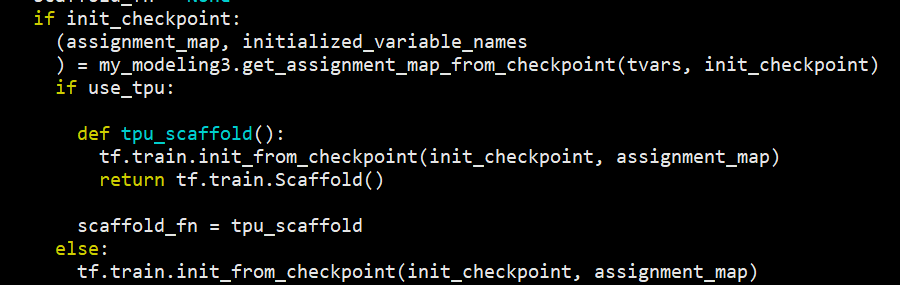




当reuse=True是寻找已经存在的变量，并不会重新构建

这里有一个坑，不要改变模型原来的层名字，比如encoder与embeddings，也不要添加层，否则会导致bert在初始化模型的时候，各个训练的原始模型会找不到参数，从而导致我们的网络参数随机初始化。下面是参数初始化的过程。





<https://blog.csdn.net/lucky7213/article/details/78967306>

<https://blog.csdn.net/lanchunhui/article/details/61914287>