

**1、tf.assign(A, new\_number):** 这个函数的功能主要是把 A 的值变为 new\_number

例如:

[python] [view plain](#) [copy](#)

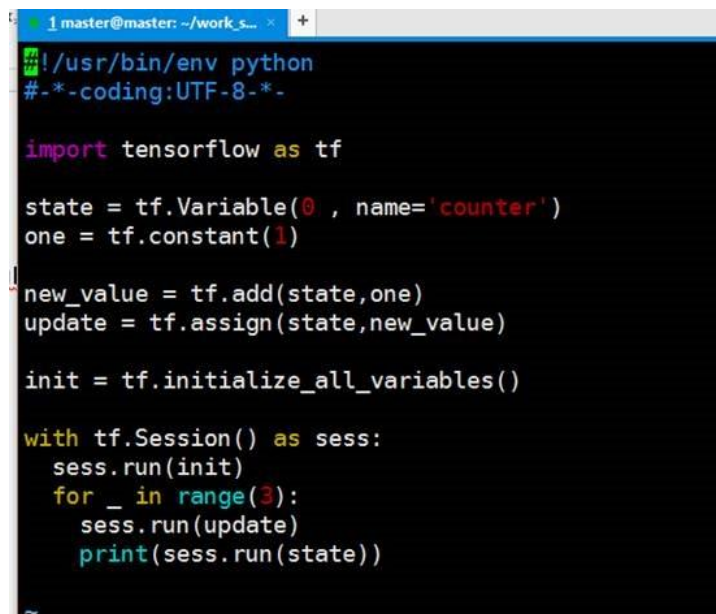
```
1. import tensorflow as tf;
2.
3. A = tf.Variable(tf.constant(0.0), dtype=tf.float32)
4. with tf.Session() as sess:
5.     sess.run(tf.initialize_all_variables())
6.     print sess.run(A)
7.     sess.run(tf.assign(A, 10))
8.     print sess.run(A)
```

输出:

0.0  
10.0

开始给 A 赋值为 0，经过 tf.assign 函数后，把 A 的值变为 10

## 2、tf.Variable()



```
1 master@master: ~/work_s... x +
! /usr/bin/env python
#-*-coding:UTF-8-*-

import tensorflow as tf

state = tf.Variable(0, name='counter')
one = tf.constant(1)

new_value = tf.add(state, one)
update = tf.assign(state, new_value)

init = tf.initialize_all_variables()

with tf.Session() as sess:
    sess.run(init)
    for _ in range(3):
        sess.run(update)
        print(sess.run(state))

~
```

实例讲解:

首先:

```
#!/usr/bin/env python
```

这句话是指定 python 的运行环境，这种指定方式有两种，一种是指定 python 的路径——#!/usr/bin/python（这里需要说明的是：“/usr/bin/python”是 python 的安装路径），我用的是 ubuntu14.0.4 这个版本中的含有 env 变量，记载着环境变量，所以也可以这样写。

```
 -*-coding:UTF-8 -*-
```

这句话是指定\*.py 的编码方式，如果文件中涉及到中文汉字的话，有必要写一下这句话。当然也可以这样写：encoding:UTF-8

```
import tensorflow as tf
```

这句话是导入 tensorflow 模块

```
state = tf.Variable(0, name='counter')
```

使用 tensorflow 在默认的图中创建节点，这个节点是一个变量。

```
one = tf.constant(1)
```

此处调用了 td 的一个函数，用于创建常量。

```
new_value = tf.add(state, one)
```

对常量与变量进行简单的加法操作，这点需要说明的是：在 TensorFlow 中，所有的操作 op，变量都视为节点，tf.add() 的意思就是在 tf 的默认图中添加一个 op，这个 op 是用来做加法操作的。

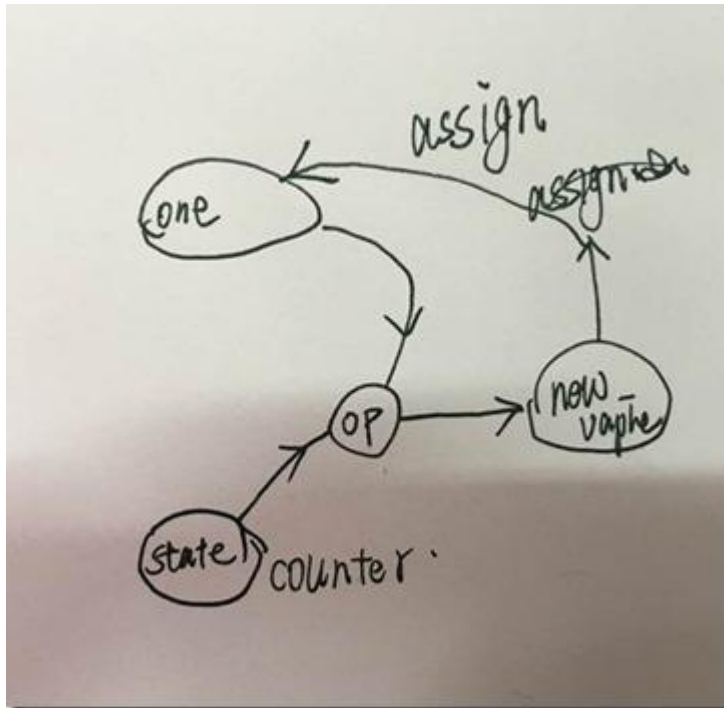
```
update = tf.assign(state, new_value)
```

这个操作是：赋值操作。将 new\_value 的值赋值给 update 变量。

好了，到此为止。我们的“图 flow”构建好了。

大致是这样的：

（注意流动 flow 的方向）



在这里，需要再次说明：我们此时只是定义好了图，并没有变量并没有初始化。目前只有 state 的值是 1。

```
init = tf.initialize_all_variables()
```

此处用于初始化变量。但是这句话仍然不会立即执行。需要通过 sess 来将数据流动起来。

切记：所有的运算都应在在 session 中进行：

```
with tf.Session() as sess:
```

此处自动开启一个 session

```
sess.run(init)
```

对变量进行初始化，执行 (run) init 语句

```
for _ in range(3):  
    sess.run(update)  
    print(sess.run(state))
```

循环 3 次，并且打印输出。

总结与体会：

(1)TensorFlow 与我们正常的编程思维略有不同：TensorFlow 中的语句不会立即执行；而是等到开启会话 session 的时候，才会执行 session.run() 中的语句。如果 run 中涉及到其他的节点，也会执行到。

(2)Tensorflow 模型中的所有的节点都是可以视为运算操作 op 或 tensor

输出的结果：

```
I tensorflow/core/common_runtime/gpu/gpu_init.cc:126] DMA: 0
I tensorflow/core/common_runtime/gpu/gpu_init.cc:136] 0: Y
I tensorflow/core/common_runtime/gpu/gpu_device.cc:755] Creating TensorFlow device (/gpu:0) -> (device: 0, name: Quadro K2000, pci bus id: 0000:05:00.0)
1
2
3
```

### 3、tf.Variable 和 tf.placeholder :

二者的主要区别在于：

- tf.Variable: 主要在于一些可训练变量 (trainable variables)，比如模型的权重 (weights, W) 或者偏执值 (bias)；
  - **声明时，必须提供初始值；**
  - 名称的真实含义，在于变量，也即在真实训练时，其值是会改变的，自然事先需要指定初始值；
  - weights = tf.Variable(
    - tf.truncated\_normal([IMAGE\_PIXELS, hidden1\_units],
    - stddev=1./math.sqrt(float(IMAGE\_PIXELS)),name='weights')
  - )
  - biases = tf.Variable(tf.zeros([hidden1\_units]), name='biases')
- tf.placeholder: 用于得到传递进来的真实的训练样本：
  - **不必指定初始值，可在运行时，通过 Session.run 的函数的 feed\_dict 参数指定；**
  - **这也是其命名的原因所在，仅仅作为一种占位符；**
- images\_placeholder = tf.placeholder(tf.float32, shape=[batch\_size, IMAGE\_PIXELS])  
labels\_placeholder = tf.placeholder(tf.int32, shape=[batch\_size])

当执行这些操作时，`tf.Variable` 的值将会改变，也即被修改，这也是其名称的来源（variable，变量）。

you use `tf.Variable` for trainable variables such as weights (W) and biases (B) for your model.

```
weights = tf.Variable(
    tf.truncated_normal([IMAGE_PIXELS, hidden1_units],
                        stddev=1.0 / math.sqrt(float(IMAGE_PIXELS))),
    name='weights')
biases = tf.Variable(tf.zeros([hidden1_units]), name='biases')
```

`tf.placeholder` is used to feed actual training examples.

```
images_placeholder = tf.placeholder(tf.float32, shape=(batch_size,
IMAGE_PIXELS))
labels_placeholder = tf.placeholder(tf.int32, shape=(batch_size))
```

This is how you feed the training examples during the training:

```
for step in xrange(FLAGS.max_steps):
    feed_dict = {
        images_placeholder: images_feed,
        labels_placeholder: labels_feed,
    }
    _, loss_value = sess.run([train_op, loss], feed_dict=feed_dict)
```

## 4、`np.newaxis`:

`np.newaxis` 的功能是插入新维度，看下面的例子：

```
a=np.array([1,2,3,4,5])
print a.shape
```

```
print a
```

输出结果

```
(5,)
[1 2 3 4 5]
```

可以看出 a 是一个一维数组,

```
x_data=np.linspace(-1,1,300)[: , np.newaxis]
a=np.array([1,2,3,4,5])
b=a[np.newaxis,:]
print a.shape,b.shape
print a
```

```
print b
```

输出结果:

```
(5,) (1, 5)
[1 2 3 4 5]
[[1 2 3 4 5]]
```

```
x_data=np.linspace(-1,1,300)[: , np.newaxis]
a=np.array([1,2,3,4,5])
b=a[:, np.newaxis]
print a.shape,b.shape
print a
print b
```

输出结果

```
(5,) (5, 1)
[1 2 3 4 5]
[[1]
 [2]
 [3]
 [4]
 [5]]
```

可以看出 np.newaxis 分别是在行或列上增加维度,原来是 (5, ) 的数组,在行上增加维度变成 (5,5) 的二维数组,在列上增加维度变为 (5,1) 的二维数组

