

Python and Tensorflow

Python

Python 2.x? or Python 3.x?

Anaconda?

版本共存?

建立环境: <https://conda.io/docs/using/>

1. 环境的建立

```
conda create -n py3 python=3
```

2. 激活/关闭一个环境

```
source activate py3
```

```
source deactivate py3
```

3. 查看正在使用的环境

```
conda info --envs
```

4. 在当前环境中使用 pip

```
conda install -n py3 pip
source activate py3
pip install ipython
```

编辑器:

sublime? 推荐pycharm

推荐学习材料:

廖雪峰的wiki

<https://www.liaoxuefeng.com/wiki/0014316089557264a6b348958f449949df42a6d3a2e542c000/001431643484137e38b44e5925440ec8b1e4c70f800b4e2000>

第一个程序

```
print('Hello,World!')
```

思考一个问题：请输入你的姓名和email，利用程序如何做到？

```
print('please enter your name:')
name = input()
print('\nplease enter your email:')
email = input()
print('\nYour name is',name, 'and your email is', email)
```

Python基础

数据类型和变量

计算机处理不同类型的数据，需要不同的数据类型。Python常用的数据类型有以下几种。

整数和浮点数

```
1,2,100
0.1
1.2E-3
```

字符串

字符串是以单引号或者双引号扩起来的任意文本。单引号和双引号不是文本的一部分。如果一个文本中包含单引号或者双引号，可以使用转义字符。

```
'I am ok'
"I'm OK"
'I\'m OK'
```

格式化字符串

```
'Hi, %s, your score is %d.' % ('Michael', 95)
```

布尔值

它有两种值 True 和 False。它有三种运算： and, or 和 not。常用于条件判断中。

空值： None

变量和常量

变量用来存放数据。它指代的数据可以根据需要变化。常量是不变的。

```
name = input()
```

使用list 和 tuple

list 是python内置的一种数据类型。它是一种有序的集合，可以随时删除和添加其中的元素。

```
classmates = ['Michael', 'Bob', 'Tracy']
len(classmates)
classmates[0]
classmates[-1]
classmates[0:2]
classmates[:3]
classmates.append('Adam')
classmates.insert(1, 'Jack')
classmates.pop()
classmates.pop(1)
classmates[1] = 'Sarah'
L = ['Apple', 123, True]
s = ['python', 'java', ['asp', 'php'], 'scheme']
len(s)
L = []
len(L)
```

另一种有序列表叫元组：tuple。tuple和list非常类似，但是tuple一旦初始化就不能修改，比如同样是列出同学的名字：

```
classmates = ('Michael', 'Bob', 'Tracy')
```

条件判断

我们通过两个例子，来看条件判断的使用

```
birth = input('birth: ')
birth = int(birth)
if birth < 2000:
    print('00前')
else:
    print('00后')
```

```
age = 3
if age >= 18:
    print('adult')
elif age >= 6:
    print('teenager')
else:
    print('kid')
```

```
age = 20
if age >= 6:
    print('teenager')
elif age >= 18:
    print('adult')
else:
    print('kid')
```

循环

常用的有两个 for 循环和 while 循环。

```
names = ['Michael', 'Bob', 'Tracy']
for name in names:
    print(name)
```

```
sum = 0
for x in range(101):
    sum = sum + x
print(sum)
```

```
n = 1
while n <= 100:
    if n > 10: # 当n = 11时, 条件满足, 执行break语句
        break # break语句会结束当前循环
    print(n)
    n = n + 1
print('END')
```

```
print('Are you ready?')
s = input()
while True:
    check = (s == 'Y' or s == 'y' or s == 'yes' or s == 'Yes')
    if check:
        print('Game starts!\n')
        break
    else:
        print('Are you ready?')
        s = input()
```

```
n = 0
while n < 10:
    n = n + 1
    if n % 2 == 0: # 如果n是偶数，执行continue语句
        continue # continue语句会直接继续下一轮循环，后续的print()语句不会执行
    print(n)
```

使用dict

Python内置了字典：dict的支持，dict全称dictionary，使用键-值（key-value）存储，具有极快的查找速度。

```
d = {'Michael': 95, 'Bob': 75, 'Tracy': 85}
d['Michael']
d.get('Thomas')
d.get('Thomas', -1)
d.pop('Bob')
```

函数

调用函数

Python内置了很多有用的函数，我们可以直接调用。

要调用一个函数，需要知道函数的名称和参数，比如求绝对值的函数`abs`，只有一个参数。

```
max(2, 3, 1, -5)
```

定义函数

```
def my_abs(x):  
    if x >= 0:  
        return x  
    else:  
        return -x
```

```
import math  
def move(x, y, step, angle=0):  
    nx = x + step * math.cos(angle)  
    ny = y + step * math.sin(angle)  
    return nx, ny  
  
move(100,100,100,math.pi/4)
```

函数的参数

```
def power(x, n=2):  
    s = 1  
    while n > 0:  
        n = n - 1  
        s = s * x  
    return s
```

Tensorflow

安装:

pip install tensorflow

Tensors

多维数组

```
3 # a rank 0 tensor; a scalar with shape []  
[1., 2., 3.] # a rank 1 tensor; a vector with shape [3]  
[[1., 2., 3.], [4., 5., 6.]] # a rank 2 tensor; a matrix with shape [2, 3]  
[[[1., 2., 3.]], [[7., 8., 9.]]] # a rank 3 tensor with shape [2, 1, 3]
```

Computation Graph

由 数据 (node) 和 运算 (边) 构成。

```
node1 = tf.constant(3.0, dtype=tf.float32)
node2 = tf.constant(4.0) # also tf.float32 implicitly
print(node1, node2)
```

```
sess = tf.Session()
print(sess.run([node1, node2]))
```

```
node1 = tf.constant(3.0, dtype=tf.float32)
node2 = tf.constant(4.0) # also tf.float32 implicitly
print(node1, node2)
node3 = tf.add(node1, node2)
print("node3:", node3)
print("sess.run(node3):", sess.run(node3))
```

Placeholders

```
a = tf.placeholder(tf.float32)
b = tf.placeholder(tf.float32)
adder_node = a + b # + provides a shortcut for tf.add(a, b)
```


```
print(sess.run(adder_node, {a: 3, b: 4.5}))
print(sess.run(adder_node, {a: [1, 3], b: [2, 4]}))
```

```
add_and_triple = adder_node * 3.
print(sess.run(add_and_triple, {a: 3, b: 4.5}))
```

Variables [parameters]

```
W = tf.Variable([.3], dtype=tf.float32)
b = tf.Variable([-0.3], dtype=tf.float32)
x = tf.placeholder(tf.float32)
linear_model = W*x + b
```


注意：Constants are initialized when you call `tf.constant`, and their value can never change. By contrast, variables are not initialized when you call `tf.Variable`. To initialize all the variables in a TensorFlow program, you must explicitly call a special operation as follows:

```
init = tf.global_variables_initializer()
sess.run(init) 
```

```
print(sess.run(linear_model, {x: [1, 2, 3, 4]}))
```

Optimization

define loss

```
y = tf.placeholder(tf.float32)
squared_deltas = tf.square(linear_model - )
loss = tf.reduce_sum(squared_deltas)
print(sess.run(loss, {x: [1, 2, 3, 4], y: [0, -1, -2, -3]}))
```

```
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)
```

```
sess.run(init) # reset values to incorrect defaults.
print(sess.run([W, b]))
for i in range(1000):
    sess.run(train, {x: [1, 2, 3, 4], y: [0, -1, -2, -3]})
    print(sess.run([W, b]))

print(sess.run([W, b]))
```

Complete Program

```
import tensorflow as tf

# Model parameters
W = tf.Variable([.3], dtype=tf.float32)
b = tf.Variable([-0.3], dtype=tf.float32)
# Model input and output
x = tf.placeholder(tf.float32)
linear_model = W*x + b
y = tf.placeholder(tf.float32)

# loss
loss = tf.reduce_sum(tf.square(linear_model - y)) # sum of the squares
# optimizer
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)
```



```

# training data
x_train = [1, 2, 3, 4]
y_train = [0, -1, -2, -3]
# training loop
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init) # reset values to wrong
for i in range(1000):
    sess.run(train, {x: x_train, y: y_train})

# evaluate training accuracy
curr_W, curr_b, curr_loss = sess.run([W, b, loss], {x: x_train, y: y_train})
print("W: %s b: %s loss: %s"%(curr_W, curr_b, curr_loss))

```

Softmax Regression with Tensorflow

Download and Read in Data

```

from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

```

construct the model

```

import tensorflow as tf
x = tf.placeholder(tf.float32, [None, 784])

```

```

W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))

```

```

y = tf.nn.softmax(tf.matmul(x, W) + b)

```

```

y_ = tf.placeholder(tf.float32, [None, 10])

```

```

cross_entropy0 = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=
[1]))

cross_entropy =tf.reduce_mean( tf.nn.softmax_cross_entropy_with_logits(labels =
y_,logits = tf.matmul(x, W) + b))

## sess.run(cross_entropy0,feed_dict={x: mnist.test.images, y_: mnist.test.labels})
## sess.run(cross_entropy,feed_dict={x: mnist.test.images, y_: mnist.test.labels})

```

optimization

```

train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

```

```

sess = tf.InteractiveSession()
tf.global_variables_initializer().run()

```

```

for _ in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

```

Evaluation

```

correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))

```

```

accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

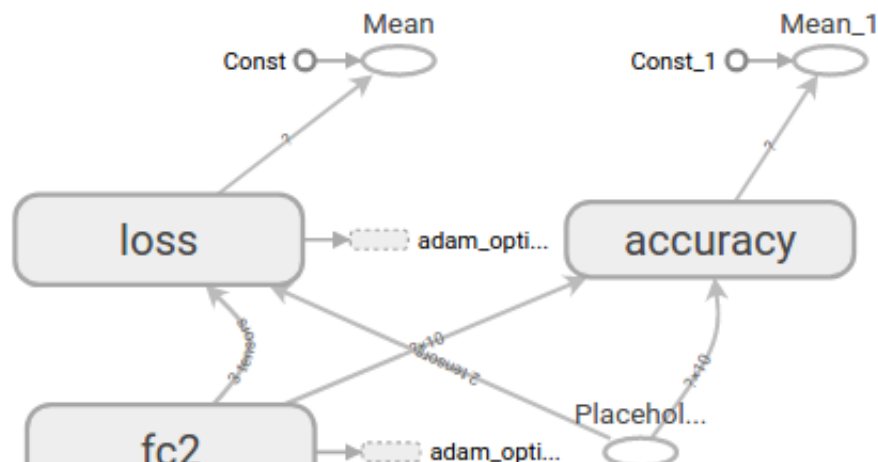
```

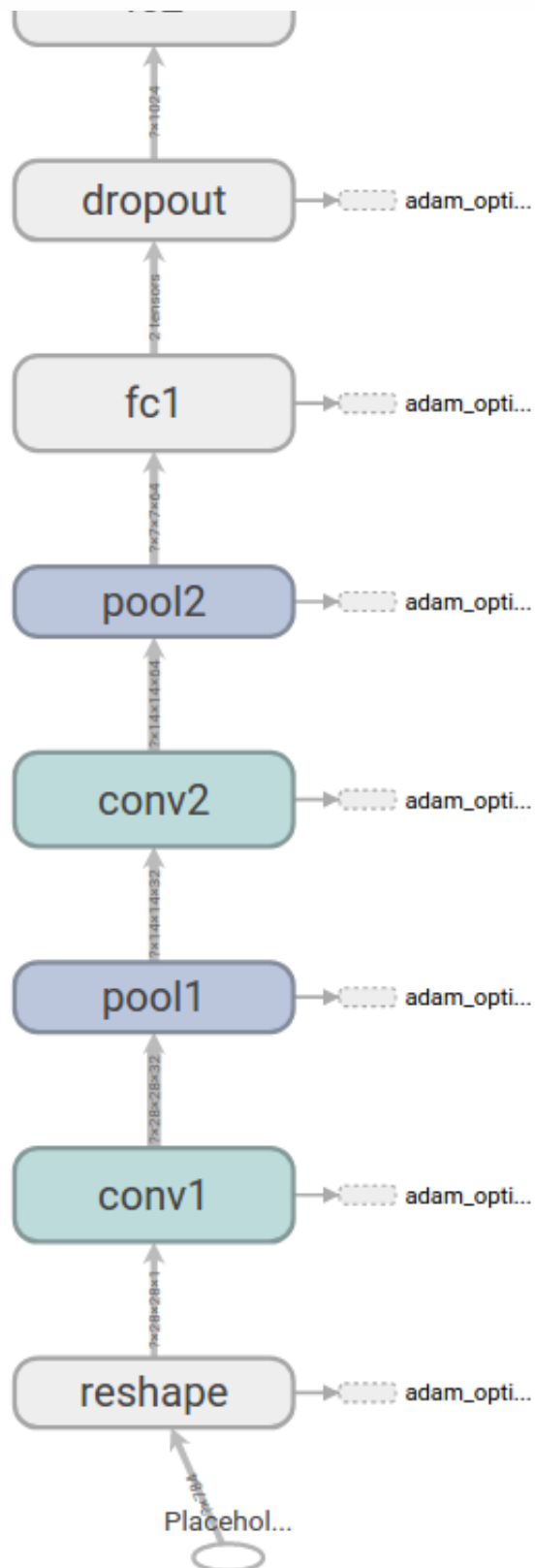
```

print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))

```

CNN with tensorflow





Prepare the data

```

from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)

x = tf.placeholder(tf.float32, shape=[None, 784])
y_ = tf.placeholder(tf.float32, shape=[None, 10])

```

Weight Initialization

```

def weight_variable(shape):
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)

def bias_variable(shape):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)

```

Convolution and Pooling

```

def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                          strides=[1, 2, 2, 1], padding='SAME')

```

First Convolutional Layer

```

W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])

```

Reshape the input image to a 4d tensor:

```

x_image = tf.reshape(x, [-1, 28, 28, 1])

```

First layer

```

h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)

```

Second layer

```

W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])

h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2)

```

FC — Full connected layer

```

W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])

h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

```

Dropout

```

keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

```

Output layer: the one before softmax

```

W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])

y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2

```

Softmax regression

```

cross_entropy = tf.reduce_mean(
    tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y_conv))
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(20000):
        batch = mnist.train.next_batch(50)
        if i % 100 == 0:
            train_accuracy = accuracy.eval(feed_dict={
                x: batch[0], y_: batch[1], keep_prob: 1.0})

```

```
    print('step %d, training accuracy %g' % (i, train_accuracy))
    train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob: 0.5})

print('test accuracy %g' % accuracy.eval(feed_dict={
    x: mnist.test.images, y_: mnist.test.labels, keep_prob: 1.0}))
```