# -\*- coding: utf-8 -\*-

from \_\_future\_\_ import print\_function

import tensorflow as tf

from tensorflow.examples.tutorials.mnist import input\_data

# number 1 to 10 data

mnist = input\_data.read\_data\_sets('MNIST\_data', one\_hot=True)

def compute\_accuracy(v\_xs, v\_ys):

global prediction

y\_pre = sess.run(prediction, feed\_dict={xs: v\_xs, keep\_prob: 1})

correct\_prediction = tf.equal(tf.argmax(y\_pre,1), tf.argmax(v\_ys,1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))

result = sess.run(accuracy, feed\_dict={xs: v\_xs, ys: v\_ys, keep\_prob: 1})

return result

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)

def conv2d(x, W):

# stride [1, x\_movement, y\_movement, 1]

# Must have strides[0] = strides[3] = 1

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

def max\_pool\_2x2(x):

# stride [1, x\_movement, y\_movement, 1]

return tf.nn.max\_pool(x, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')

# define placeholder for inputs to network

xs = tf.placeholder(tf.float32, [None, 784])/255. # 28x28

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

keep\_prob = tf.placeholder(tf.float32)

x\_image = tf.reshape(xs, [-1, 28, 28, 1])

# print(x\_image.shape) # [n\_samples, 28,28,1]

## conv1 layer ##

W\_conv1 = weight\_variable([5,5, 1,32]) # patch 5x5, in size 1, out size 32

b\_conv1 = bias\_variable([32])

h\_conv1 = tf.nn.relu(conv2d(x\_image, W\_conv1) + b\_conv1) # output size 28x28x32

h\_pool1 = max\_pool\_2x2(h\_conv1) # output size 14x14x32

## conv2 layer ##

W\_conv2 = weight\_variable([5,5, 32, 64]) # patch 5x5, in size 32, out size 64

b\_conv2 = bias\_variable([64])

h\_conv2 = tf.nn.relu(conv2d(h\_pool1, W\_conv2) + b\_conv2) # output size 14x14x64

h\_pool2 = max\_pool\_2x2(h\_conv2) # output size 7x7x64

## fc1 layer ##

W\_fc1 = weight\_variable([7\*7\*64, 1024])

b\_fc1 = bias\_variable([1024])

# [n\_samples, 7, 7, 64] ->> [n\_samples, 7\*7\*64]

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)

h\_fc1\_drop = tf.nn.dropout(h\_fc1, keep\_prob)

## fc2 layer ##

W\_fc2 = weight\_variable([1024, 10])

b\_fc2 = bias\_variable([10])

prediction = tf.nn.softmax(tf.matmul(h\_fc1\_drop, W\_fc2) + b\_fc2)

# the error between prediction and real data

cross\_entropy = tf.reduce\_mean(-tf.reduce\_sum(ys \* tf.log(prediction),

reduction\_indices=[1])) # loss

train\_step = tf.train.AdamOptimizer(1e-4).minimize(cross\_entropy)

sess = tf.Session()

# important step

# tf.initialize\_all\_variables() no long valid from

# 2017-03-02 if using tensorflow >= 0.12

if int((tf.\_\_version\_\_).split('.')[1]) < 12 and int((tf.\_\_version\_\_).split('.')[0]) < 1:

init = tf.initialize\_all\_variables()

else:

init = tf.global\_variables\_initializer()

sess.run(init)

for i in range(1000):

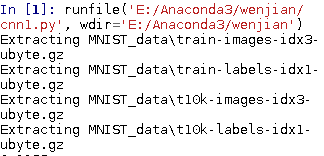
batch\_xs, batch\_ys = mnist.train.next\_batch(100)

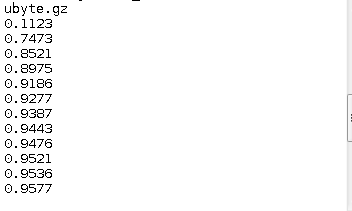
sess.run(train\_step, feed\_dict={xs: batch\_xs, ys: batch\_ys, keep\_prob: 0.5})

if i % 50 == 0:

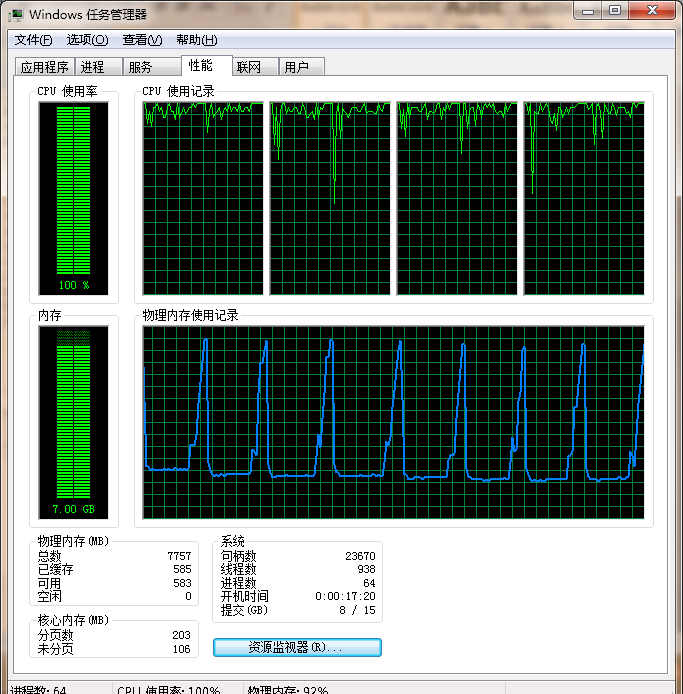
print(compute\_accuracy(

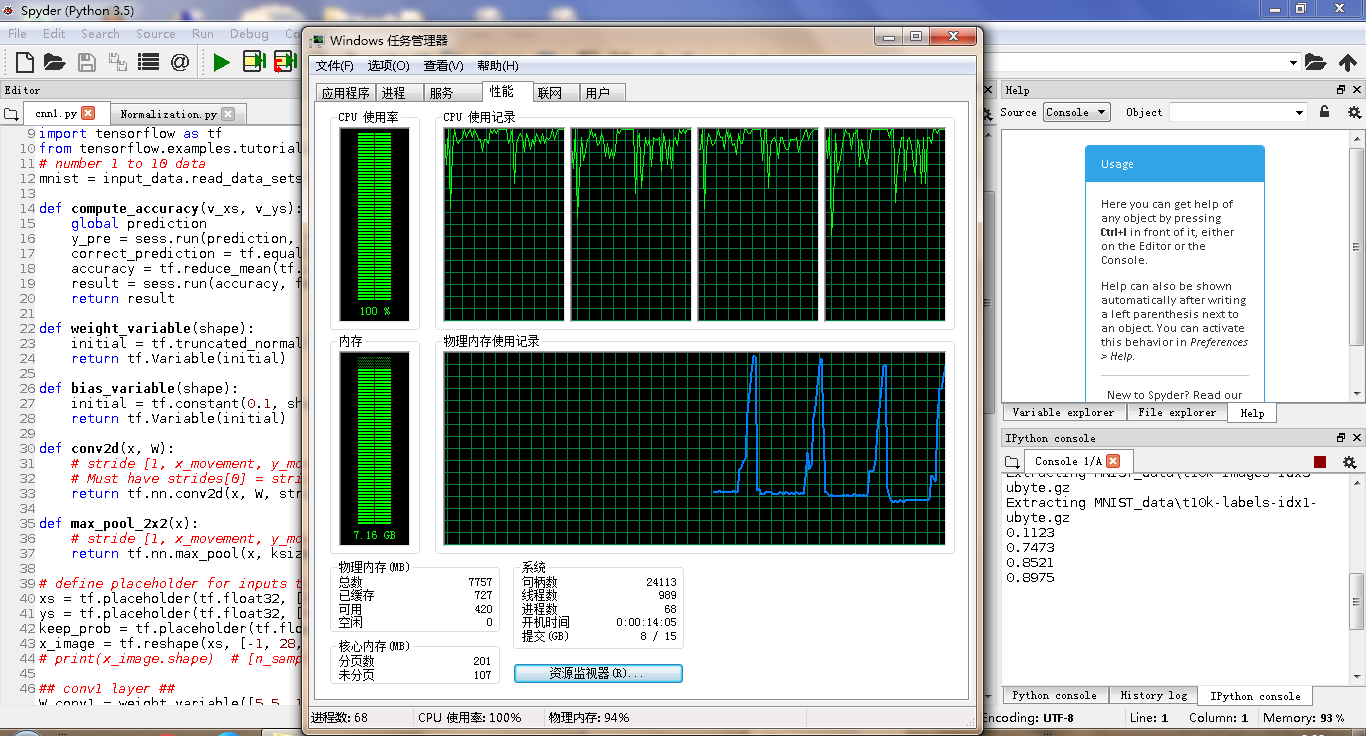
mnist.test.images, mnist.test.labels))





随便训练了下，训练时间比较久，





1.程序讲解

该程序采用CNN卷积神经网络识别训练1到10的数字。

##获取训练数据

from tensorflow.examples.tutorials.mnist import input\_data

# number 1 to 10 data

mnist = input\_data.read\_data\_sets('MNIST\_data', one\_hot=True)

##定义需要用到的函数，分别是验证正确率的compute\_accuracy、获取神经网络边上的权重weight\_variable，bias\_variable（偏置函数）、CNN卷积层函数conv2d、CNN池化层函数max\_pool\_2x2

**def compute\_accuracy(v\_xs, v\_ys):**

**global prediction**

#得到用测试数据算出来的预测值

**y\_pre = sess.run(prediction, feed\_dict={xs: v\_xs, keep\_prob: 1})**

#与标准值比较得偏差

# tf.argmax()返回行或列的最大值下标向量，1为行。

# tf.equal()两传入参数相等返回1，不等返回0.

**correct\_prediction = tf.equal(tf.argmax(y\_pre,1), tf.argmax(v\_ys,1))**

# tf.cast()转换参数值的类型

# tf.reduce\_mean()求平均值，没指定第二参数则所有值，指定1

#按行

**accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))**

#运行得到正确概率。

**result = sess.run(accuracy, feed\_dict={xs: v\_xs, ys: v\_ys, keep\_prob: 1})**

**return result**

**def weight\_variable(shape):**

**initial = tf.truncated\_normal(shape, stddev=0.1)**

# truncated\_normal()正态分布，均值为0，标准差0.1。如果随机

#出来的值偏离平均值超过两个标准差，这个数将被重新随机

**return tf.Variable(initial)**

**def bias\_variable(shape):**

**initial = tf.constant(0.1, shape=shape)**

# constant()产生一个给定定值的常量

**return tf.Variable(initial)**

**def conv2d(x, W):**

# stride [1, x\_movement, y\_movement, 1]

# Must have strides[0] = strides[3] = 1

#x 为四维矩阵，参数一传入batch的数量，后三位对应节点矩阵#W为过滤器规格 ，如[5,5,3,16]，过滤器5x5，当前层深度3，

#输出层深度16。步长1，填充0。

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

**def max\_pool\_2x2(x):**

# stride [1, x\_movement, y\_movement, 1]

#ksize过滤器尺寸2x2，步长2

**return tf.nn.max\_pool(x, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')**

#tf.placeholder()程序运行时才指定参数值，用于训练参数的传入。

**xs = tf.placeholder(tf.float32, [None, 784])/255. # 28x28**

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

**keep\_prob = tf.placeholder(tf.float32)**

# tf.reshape()改变矩阵维度，-1表示该维度的值随意。

**x\_image = tf.reshape(xs, [-1, 28, 28, 1])**

#接下来是建立训练神经网络，该神经网络的结构是卷积层池化层，#卷积层池化层，全连接层，全连接层，分别对应**conv1 layer，conv2** #**layer，#fc1 layer，fc2 layer**

#传入的batch（训练数据）的形式变化：[100,784]->[100,28,28,1]->

#[100,28,28,32]->[100,14,14,32]->[100,14,14,64]->[100,7,7,64]->

#[100,7\*7\*64]->[100,1024]->[100,10]

**## conv1 layer ##**

**W\_conv1 = weight\_variable([5,5, 1,32]) # patch 5x5, in size 1, out size 32**

**b\_conv1 = bias\_variable([32])**

#conv2d()矩阵乘，tf.nn.relu()激励函数去线性化。

**h\_conv1 = tf.nn.relu(conv2d(x\_image, W\_conv1) + b\_conv1) # output size 28x28x32**

**h\_pool1 = max\_pool\_2x2(h\_conv1) # output size 14x14x32**

**## conv2 layer ##**

**W\_conv2 = weight\_variable([5,5, 32, 64]) # patch 5x5, in size 32, out size 64**

**b\_conv2 = bias\_variable([64])**

**h\_conv2 = tf.nn.relu(conv2d(h\_pool1, W\_conv2) + b\_conv2) # output size 14x14x64**

**h\_pool2 = max\_pool\_2x2(h\_conv2) # output size 7x7x64**

**## fc1 layer ##**

**W\_fc1 = weight\_variable([7\*7\*64, 1024])**

**b\_fc1 = bias\_variable([1024])**

**# [n\_samples, 7, 7, 64] ->> [n\_samples, 7\*7\*64]**

**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)**

#tf.nn.dropout()一定概率将输出值保留，避免过拟合。

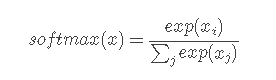
**h\_fc1\_drop = tf.nn.dropout(h\_fc1, keep\_prob)**

**## fc2 layer ##**

**W\_fc2 = weight\_variable([1024, 10])**

**b\_fc2 = bias\_variable([10])**

#tf.nn.softmax()将传入矩阵转化为概率值，按行，就是矩阵每行相#加值为1



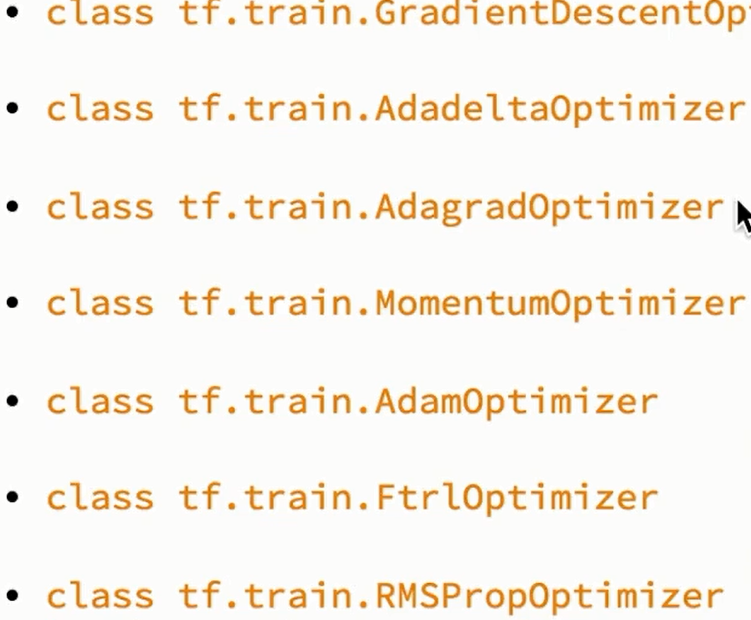
**prediction = tf.nn.softmax(tf.matmul(h\_fc1\_drop, W\_fc2) + b\_fc2)**

**cross\_entropy = tf.reduce\_mean(-tf.reduce\_sum(ys \* tf.log(prediction),**

**reduction\_indices=[1])) # loss**

# tf.train.AdamOptimizer（）优化器，学习效率0.0001，让损失值往

#减小趋势发展来改变参数值



**train\_step = tf.train.AdamOptimizer(1e-4).minimize(cross\_entropy)**

**sess = tf.Session()**

#根据tensorflow版本来选择初始化变量的方式

**if int((tf.\_\_version\_\_).split('.')[1]) < 12 and int((tf.\_\_version\_\_).split('.')[0]) < 1:**

**init = tf.initialize\_all\_variables()**

**else:**

**init = tf.global\_variables\_initializer()**

**sess.run(init)**

#训练1000次，把训练数据分成每份100个样值，50%的概率筛

#选训练值，这样循环输入的训练参数每次都会不一样。

**for i in range(1000):**

**batch\_xs, batch\_ys = mnist.train.next\_batch(100)**

**sess.run(train\_step, feed\_dict={xs: batch\_xs, ys: batch\_ys, keep\_prob: 0.5})**

# 每50次测试一下准确度，测试准确度的样本与训练的样本不是#同一个来源

**if i % 50 == 0:**

**print(compute\_accuracy(**

**mnist.test.images, mnist.test.labels))**