识别mnist

1.使用方法:CNN卷积神经网络

2.网络架构：

输入 -> 卷积层1 -> relu激励层 -> 池化层 -> 完全层1 -> 完全层2 -> softmax -> 结果

3.初始化设置：

卷积层1，卷积层2，完全层1，完全层2的四个权重W初始化均为正态分布的随机值。
卷积层1，卷积层2，完全层1，完全层2的四个偏置b初始化均为常数0。
输入x是[num,784]的张量。输出y_conv是[num,10]的张量，每一维是每个数字的概率。

4.损失函数：交叉熵

Loss = y*real*log(y*conv)，描述预测值和真实值概率分布的差异。

5.优化方法：Adam优化算法

是一个寻找全局最优点的优化算法，引入了二次方梯度校正。不容易陷于局部优点。

```
#utf-8
import tensorflow as tf
import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)#导入minist数据集
sess = tf.InteractiveSession()

x = tf.placeholder("float", shape=[None, 784])#x是占位符，表示输入图片的数组形式，形状为[None,784]，第一位是图片数量，784是像素数
y_real = tf.placeholder("float", shape=[None, 10])#y_real是占位符，表示图片表达的真实数字，形状为[None,10],
                                                    第一位是图片数量，第二位正确的数作为下标的值为1，其余为0

def weight_variable(shape):#定义W权重，初始化为正态分布
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)

def bias_variable(shape):#定义b偏置量，初始化为0
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)

def conv2d(x, W):#定义卷积层，x是输入张量，W是卷积核，步长为1，没有边距
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max_pool_2x2(x):#定义池化层，模板为2X2，步长为2，没有边距
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],strides=[1, 2, 2, 1], padding='SAME')
W_conv1 = weight_variable([5, 5, 1, 32])#第一层卷积核，32个5X5的卷积核
b_conv1 = bias_variable([32])#第一层偏置量，32个常量

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

h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)#第一层激励层
h_pool1 = max_pool_2x2(h_conv1)#第一层池化层

W_conv2 = weight_variable([5, 5, 32, 64])#第二层卷积核，5X5X32 64个
b_conv2 = bias_variable([64])#第二层偏置量，64个常量

h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)#第二层激励层
h_pool2 = max_pool_2x2(h_conv2)#第二层池化层

W_fc1 = weight_variable([7 * 7 * 64, 1024])#完全层的权重
b_fc1 = bias_variable([1024])#完全层的偏置量

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)#计算

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

W_fc2 = weight_variable([1024, 10])#权重
b_fc2 = bias_variable([10])#偏置量

y_conv=tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)#网络输出结果，长度为10的向量，取得每个数字的概率。softmax使概率为0-1之间。

cross_entropy = -tf.reduce_sum(y_real*tf.log(y_conv))#损失函数，交叉熵=y_real*log(y_cov)
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)#训练过程，以AdamOptimizer的优化过程反向优化参数
correct_prediction = tf.equal(tf.argmax(y_realconv,1), tf.argmax(y_real,1))#取两者最大值的下标比较，预测是否正确？
                                                                          返回true则正确，返回false则错误
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))#将correct_prediction转换为浮点数，正确为1，错误为0

sess.run(tf.initialize_all_variables())#初始化所有参数

for i in range(20000):
     batch = mnist.train.next_batch(50)#以50个为一批取mnist数据
     train_step.run(feed_dict={x: batch[0], y_real: batch[1], keep_prob: 0.5})#训练
     if i%100 == 0:#i为100判断正确性
        train_accuracy = accuracy.eval(feed_dict={x:batch[0], y_real: batch[1], keep_prob: 1.0})#判断目前批次的正确率
        print ("step %d, training accuracy %g"%(i, train_accuracy))#打印正确率
```