# 面向对象神经网络的tensorflow实现

## 神经网络类的实现

这里定义一个fullNet类对象，在其初始化函数\_\_init\_\_中完成变量的创建，网络的构建及变量的初始化。

\_\_init\_\_函数是类的初始化函数，在创建类时会自动调用，传入的参数有网络的shape和优化器，其中shape为一个整型列表，用来记录网络层节点个数，如[784,300,50,10]代表输入节点数为784，输出节点数为10，另两个隐层节点数分别为300和50，优化器为tf.train下的各种优化器。

假设网络shape的长度为4，则网络应该有3层，为此使用一个for循环来动态创建网络，网络参数为各层权值和偏置值，这可以方便的使用python的列表来保存，使用方法是先使用list()创建列表，然后使用append方法将新成员加入，因为list可以方便的使用索引，因此不使用dict来保存。

除\_\_init\_\_外，还定义了一个train\_net函数，该函数完成了一次训练，并返回正确率。

在使用该类时，首先定义了一个类对象net，然后使用循环不断的将训练数据传给net. train\_net函数，同时返回训练的准确率。

从代码中可见，我们可以方便的改变网络结构参数来尝试不同的网络，同时，在使用者角度来看，也看不到tf.Session，代码也十分明了。

## 2.代码

|  |
| --- |
| **import** tensorflow **as** tf **from** tensorflow.examples.tutorials.mnist **import** input\_data mnist=input\_data.read\_data\_sets(**"F:/tmp/data"**,one\_hot=**True**)  **class** fullNet(object):  **def** \_\_init\_\_(self, netshape, optimizer):  self.x\_input = tf.placeholder(tf.float32, [**None**, netshape[0]])  self.y\_input = tf.placeholder(tf.float32, [**None**, netshape[-1]])  self.optimizer = optimizer   deep = len(netshape)  self.W = list()  self.b = list()  self.out = list()  **for** i **in** range(deep - 1):  self.W.append(tf.Variable(tf.truncated\_normal([netshape[i], netshape[i + 1]], stddev=0.1)))  self.b.append(tf.Variable(tf.truncated\_normal(shape=[netshape[i + 1]], stddev=0.1)))  **if** i == 0:  self.out.append(tf.nn.relu(tf.matmul(self.x\_input, self.W[i]) + self.b[i]))  **else**:  self.out.append(tf.nn.relu(tf.matmul(self.out[i - 1], self.W[i]) + self.b[i]))   self.prediction = tf.nn.softmax(self.out[-1])  self.loss = tf.reduce\_mean(tf.square(self.y\_input - self.prediction))  self.train\_step = self.optimizer.minimize(self.loss)  correct\_prediction = tf.equal(tf.arg\_max(self.y\_input, 1), tf.arg\_max(self.prediction, 1))  self.acurracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))   self.sess = tf.Session()  init = tf.global\_variables\_initializer()  self.sess.run(init)   **def** train\_net(self, X, Y):  acc, \_ = self.sess.run([self.acurracy, self.train\_step], feed\_dict={self.x\_input: X, self.y\_input: Y})  **return** acc  net = fullNet([784, 300, 50, 10], tf.train.AdamOptimizer(0.0001)) batch\_size = 50 n\_batch = mnist.train.num\_examples // batch\_size  **for** epoch **in** range(51):  **for** batch **in** range(n\_batch):  batch\_xs, batch\_ys = mnist.train.next\_batch(batch\_size)  acc = net.train\_net(batch\_xs, batch\_ys)  print(**"epoch: "**, epoch, **", accuracy: "**, acc) print(**"finish!"**) |