

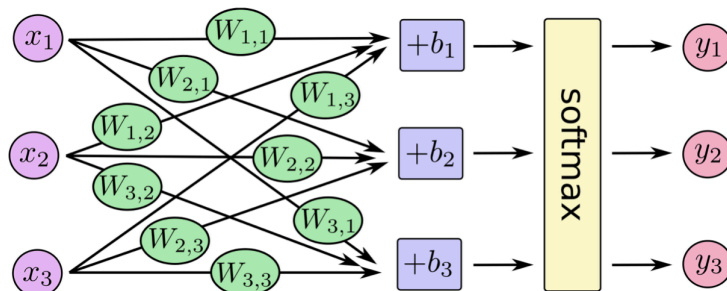
## 干货 | tensorflow 一步一步教你手写体数字识别

本文的主要目的是教会大家运用google开源的深度学习框架tensorflow来实现手写体数字识别，给出两种模型，一种是利用机器学习中的softmax regression作分类器，另一种将是搭建一个深度神经网络以达到99%正确率的手写体数字识别模型。其次，理解tensorflow工作流程和运行机制。最后，也会让大家了解一下深度神经网络的搭建过程。

### 第一种模型

我们知道，做机器学习的项目离不开数据，行业里有句话叫做“得数据者，得天下”，虽有点言过其实，但事实上，足够的样本数据是你获得较好模型的前提之一。这里，用到的是大名鼎鼎的mnist，其官网是<http://yann.lecun.com/exdb/mnist/>，大家可以自由下载。google给出的下载源码是input\_data.py，可以从[https://github.com/xupeng082008/tensorflow\\_mnist](https://github.com/xupeng082008/tensorflow_mnist)下载。下载后的数据集分为训练集、验证集、测试集(也就是train\_data,validation\_data,test\_dasta，记住，这样的划分很重要，它可以检验我们得到的模型在真实场景下的识别能力)。无论是训练、验证还是测试集，单个样本都是一帧包含手写数字的28x28图像以及其对应的标签。有了数据，我们开始建立模型。这里的模型是机器学习经常用到的softmax regression。

简单地用图片来表示一下：



用乘积形式表示如下：

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax} \begin{bmatrix} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{bmatrix}$$

在tensorflow中，它是这样实现的：

```
import tensorflow as tf
```

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

**x**是输入 (这里可以理解为28x28的图像拉伸为一个1x784向量)，**w**是网络权重系数，**b**是偏置。

代码如下：  
47888

代码相当简单，好理解。

下面贴出该模型所有的代码并作出解释

```
import tensorflow as tf
import input_data

#加载数据，注意一下one_hot 在tensorflow中，
#表示一个列向量，其中一个不为0，剩下的为0
mnist=input_data.read_data_sets("Mnist_data/",one_hot=True)
#在tensorflow中，tf.placeholder表示一个占位符，x不是一个特定的值，
#在运行时，才有值
x = tf.placeholder("float",[None,784])
#在tensorflow中，tf.Variable表示变量，且可以用不同的方式初始化
w = tf.Variable(tf.zeros([784,10]))
b = tf.Variable(tf.zeros([10]))
#softmax模型
y = tf.nn.softmax(tf.matmul(x,w) + b)
y_ = tf.placeholder("float",[None,10])
#模型的损失是交叉熵，详情可以见原创 干货 | 深度学习——损失函数不同，
收敛效率不同
cross_entropy = -tf.reduce_sum(y_*tf.log(y))
#随机梯度下降优化算法
#主要：TensorFlow 在这里实际上所做的是，它会在后台给述你的计算的那张图
#里面增加一系列新的计算操作单元用于实现反向传播算法和梯度下降算法。然后，
#它返回给你的只是一个单一的操作，当运行这个操作时，它用梯度下降算法训练
#你的模型，微调你的变量，不断减少成本
train_step =
tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)
#这里的代码跟tensorflow运行机制相关
init = tf.initialize_all_variables()
sess = tf.Session()
sess.run(init)
for i in range(1000):#1000 or more
    batch_xs,batch_ys = mnist.train.next_batch(100)
    sess.run(train_step,feed_dict={x:batch_xs,y_:batch_ys})

correct_prediction = tf.equal(tf.argmax(y,1),tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction,"float"))
print sess.run(accuracy,feed_dict=
```

```
{x:mnist.test.images,y_:mnist.test.labels})
```

这里简单说说tensorflow运行机制相关，让读者有一个感性的认识：

tensorflow使用图(graphs)来表示模型的整个计算过程

tensorflow在会话(Session)中执行图所表示的运行过程

tensorflow使用张量(tensors)来代表数据，可以表示为输入向量，权重系数等等

tensorflow通过变量(Variables)维护变量的运行状态

tensorflow使用供给(feeds)和取回(fetches)将数据传入或传出等操作

总之，tensorflow是以图来表示整个程序运行系统，图中以节点为组织，在会话中执行运行，然后产生张量，这个张量是一个四维向量，在图中流动起来。可以理解前面部分代码是在构建，后面是在执行。

上面的代码执行情况如下：

迭代次数	正确率
1000	91.19%
5000	92.21%
20000	92.27%

可以，softmax regression在这里达到的最好成绩92%左右，跟理想的还差很远。

## 第二种模型

这里，我们的模型是一个有着两个卷积层，2个全联接层，一个输出层组成，继续使用交叉熵损失函数，激活函数是relu，并且使用dropout技术，结构图如下：

Input	conv1	pool1	conv2	pool2	fc1	dropout	fc2	softn
-------	-------	-------	-------	-------	-----	---------	-----	-------

下面，我们来看看在tensorflow如何实现上面表格实现的深度神经网络

```
import tensorflow as tf
```

```
import input_data
```

```
mnist=input_data.read_data_sets("Mnist_data/",one_hot=True)
```

```
sess = tf.InteractiveSession()
```

```
x = tf.placeholder("float",shape=[None,784])
```

```
y_ = tf.placeholder("float",shape=[None,10])
```

```
w = tf.Variable(tf.zeros([784,10]))
```

```
b = tf.Variable(tf.zeros([10]))
```

```
#init weights and biases
```

```
#初始化权重系数、偏置，注意这里初始化不是用0来填充，以打破对称性
```

```
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)

#define convolution and pool function
#定义卷积、池化, tensorflow中有实现, 具体实践, 我们自行修改参数
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')

#conv and pool
#the first conv_pool
#patchsize(5x5) input_channels(1) output_channels(32
feature_maps)
w_conv1 = weight_variable([5,5,1,32])
b_conv1 = bias_variable([32])
x_image = tf.reshape(x, [-1,28,28,1])

#1st hidden outputs
h_conv1 = tf.nn.relu(conv2d(x_image,w_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)

#the second conv_pool
w_conv2 = weight_variable([5,5,32,64])
b_conv2 = bias_variable([64])

#2nd hidden outputs
h_conv2 = tf.nn.relu(conv2d(h_pool1,w_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2)

#fc全联接层
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使用dropout技术可以防止过拟合`

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

`#readout layer`

```
w_fc2 = weight_variable([1024,10])
b_fc2 = bias_variable([10])
```

`#softmax output`

```
y_conv = tf.nn.softmax(tf.matmul(h_fc1_drop,w_fc2) + b_fc2)
```

`#define loss function`

```
cross_entropy = -tf.reduce_sum(y_*tf.log(y_conv))
```

`#optimizer method choice`

```
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,"float"))
```

```
sess.run(tf.initialize_all_variables())
```

```
for i in range(10000):
```

```
    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)
```

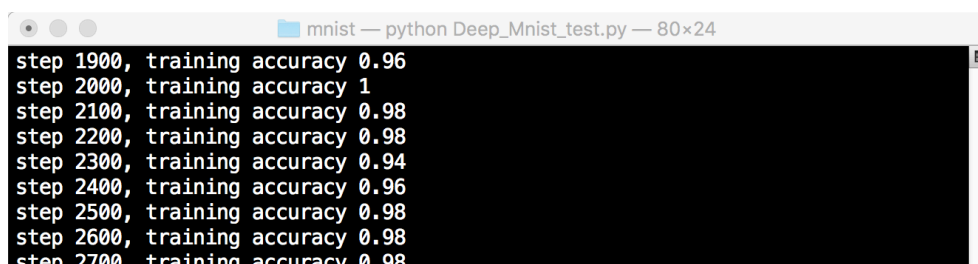
```
        train_step.run(feed_dict=
```

```
{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})
```

下面给出一些训练过程中间图：



```
mnist — python Deep_Mnist_test.py — 80x24
step 1900, training accuracy 0.96
step 2000, training accuracy 1
step 2100, training accuracy 0.98
step 2200, training accuracy 0.98
step 2300, training accuracy 0.94
step 2400, training accuracy 0.96
step 2500, training accuracy 0.98
step 2600, training accuracy 0.98
step 2700, training accuracy 0.98
```

```
step 2700, training accuracy 0.98
step 2800, training accuracy 1
step 2900, training accuracy 0.98
step 3000, training accuracy 0.96
step 3100, training accuracy 1
step 3200, training accuracy 0.98
step 3300, training accuracy 1
step 3400, training accuracy 1
step 3500, training accuracy 1
step 3600, training accuracy 1
step 3700, training accuracy 0.98
step 3800, training accuracy 1
step 3900, training accuracy 0.98
step 4000, training accuracy 0.98
step 4100, training accuracy 0.96
```

可以看出，训练识别率还是很不错的，我们看下最终迭代10000次的训练和测试结果：

```
mnist -- -bash -- 80x24
step 7800, training accuracy 0.98
step 7900, training accuracy 0.98
step 8000, training accuracy 1
step 8100, training accuracy 1
step 8200, training accuracy 1
step 8300, training accuracy 1
step 8400, training accuracy 1
step 8500, training accuracy 0.98
step 8600, training accuracy 1
step 8700, training accuracy 1
step 8800, training accuracy 1
step 8900, training accuracy 1
step 9000, training accuracy 1
step 9100, training accuracy 1
step 9200, training accuracy 1
step 9300, training accuracy 1
step 9400, training accuracy 1
step 9500, training accuracy 1
step 9600, training accuracy 1
step 9700, training accuracy 1
step 9800, training accuracy 1
step 9900, training accuracy 1
test accuracy 0.9905
xupengs-MacBook-Pro:mnist xupeng$
```

测试集上的正确率是99%左右。

可见深度神经网络，在手写体识别项目上表现地相比于softmax regression，效果会好的多的多。

## 总结

以上，

我们学习了在tensorflow中实现softmax regression、一种深度神经网络的过程；简单了解了tensorflow的运行机制和内部参数、函数机构，相信看完大家可以手动设计一个神经网络将识别率继续提高。

所有的代码可以从[https://github.com/xupeng082008/tensorflow\\_mnist](https://github.com/xupeng082008/tensorflow_mnist)下载。