

TensorFlow学习笔记2：构建CNN模型

2016-01-31

[上篇博文](#)主要是TensorFlow的一个简单入门，并介绍了如何实现Softmax Regression模型，来对MNIST数据集中的数字手写体进行识别。

然而，由于Softmax Regression模型相对简单，所以最终的识别准确率并不高。下面将针对MNIST数据集构建更加复杂精巧的模型，以进一步提高识别准确率。

深度学习模型

TensorFlow很适合用来进行大规模的数值计算，其中也包括实现和训练深度神经网络模型。下面将介绍TensorFlow中模型的基本组成部分，同时将构建一个CNN模型来对MNIST数据集中的数字手写体进行识别。

基本设置

在我们构建模型之前，我们首先加载MNIST数据集，然后开启一个TensorFlow会话(session)。

加载MNIST数据集

TensorFlow中已经有相关脚本，来自动下载和加载MNIST数据集。（脚本会自动创建MNIST_data文件夹来存储数据集）。下面是脚本程序：

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

这里mnist是一个轻量级的类文件，存储了NumPy格式的训练集、验证集和测试集，它同样提供了数据中mini-batch迭代的功能。

开启TensorFlow会话

TensorFlow后台计算依赖于高效的C++，与后台的连接称为一个会话(session)。TensorFlow中的程序使用，通常都是先创建一个图(graph)，然后在一个会话(session)里运行它。

这里我们使用了一个更为方便的类，`InteractiveSession`，这能让你在构建代码时更加灵活。`InteractiveSession`允许你做一些交互操作，通过创建一个计算流图([computation graph](#))来部分地运行图计算。当你在一些交互环境（例如Python）中使用时将更加方便。如果你不是使用`InteractiveSession`，那么你要在启动一个会话和运行图计算前，创建一个整体的计算流图。

下面是如何创建一个`InteractiveSession`：

```
import tensorflow as tf
sess = tf.InteractiveSession()
```

计算流图(Computation Graph)

为了在Python中实现高效的数值运算，通常会使用一些Python以外的库函数，如NumPy。但是，这样做会造成转换Python操作的开销，尤其是在GPUs和分布式计算的环境下。TensorFlow在这一方面（指转化操作）做了优化，它让我们能够在Python之外描述一个包含各种交互计算操作的整体流图，而不是每次都独立地在Python之外运行一个单独的计算，避免了许多转换开销。这样的优化方法同样用在了Theano和Torch上。

所以，以上这样的Python代码的作用是简历一个完整的计算流图，然后指定图中的哪些部分需要运行。关于计算流图的更多具体使用见[这里](#)。

Softmax Regression模型

见[上篇博文](#)。

CNN模型

Softmax Regression模型在MNIST数据集上91%的准确率，其实还是比较低的。下面我们将使用一个更加精巧的模型，一个简单的卷积神经网络模型(CNN)。这个模型能够达到99.2%的准确率，尽管这不是最高的，但已经足

够接受了。

权值初始化

为了建立模型，我们需要先创建一些权值(w)和偏置(b)等参数，这些参数的初始化过程中需要加入一小部分的噪声以破坏参数整体的对称性，同时避免梯度为0.由于我们使用ReLU激活函数（[详细介绍](#)），所以我们通常将这些参数初始化为很小的正值。为了避免重复的初始化操作，我们可以创建下面两个函数：

```
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)
```

卷积(Convolution)和池化(Pooling)

TensorFlow同样提供了方便的卷积和池化计算。怎样处理边界元素？怎样设置卷积窗口大小？在这个例子中，我们始终使用vanilla版本。这里的卷积操作仅使用了滑动步长为1的窗口，使用0进行填充，所以输出规模和输入的一致；而池化操作是在2 * 2的窗口内采用最大池化技术(max-pooling)。为了使代码简洁，同样将这些操作抽象为函数形式：

```
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')
```

其中，padding='SAME'表示通过填充0，使得输入和输出的形状一致。

第一层：卷积层

第一层是卷积层，卷积层将要计算出32个特征映射(feature map)，对每个5 * 5的patch。它的权值tensor的大小为[5, 5, 1, 32]。前两维是patch的大

小，第三维时输入通道的数目，最后一维是输出通道的数目。我们对每个输出通道加上了偏置(bias)。

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

为了使得图片与计算层匹配，我们首先reshape输入图像x为4维的tensor，第2、3维对应图片的宽和高，最后一维对应颜色通道的数目。（? 第1维为什么是-1? ）

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

然后，使用weight tensor对x_image进行卷积计算，加上bias，再应用到一个ReLU激活函数，最终采用最大池化。

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

第二层：卷积层

为了使得网络有足够深度，我们重复堆积一些相同类型的层。第二层将会有64个特征，对应每个5 * 5的patch。

```
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)
```

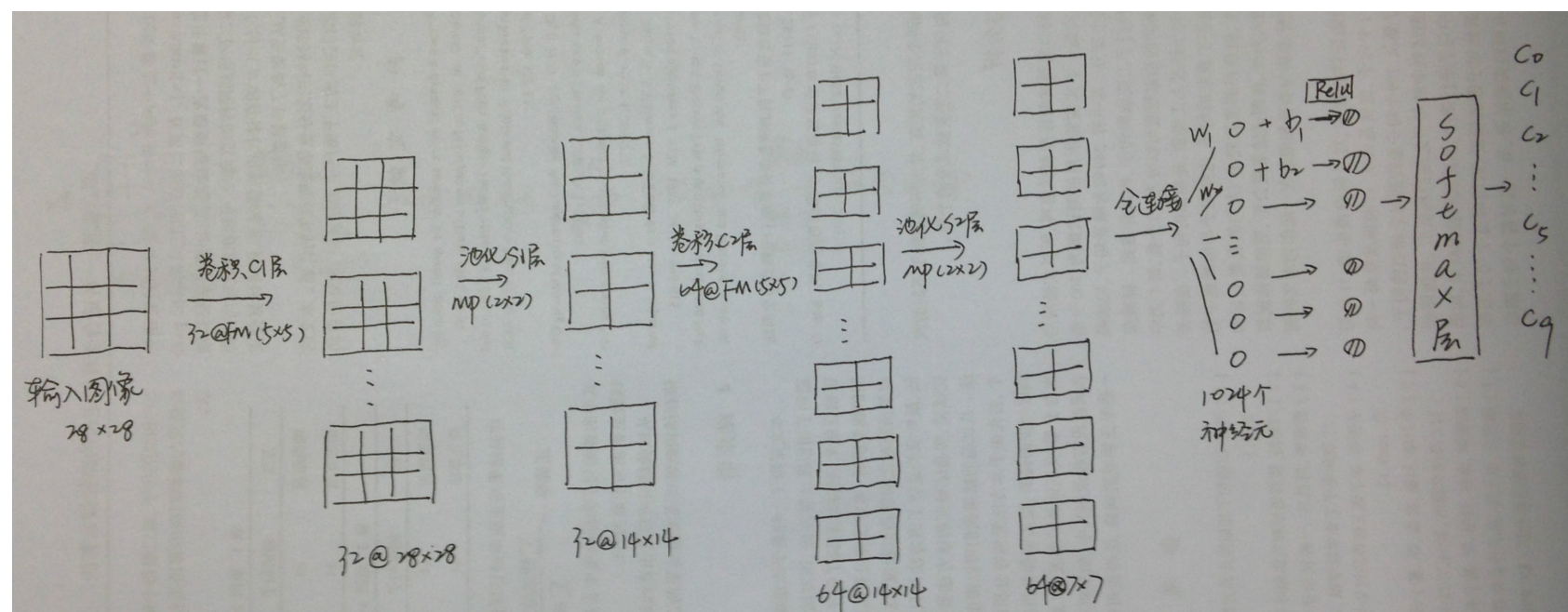
全连接层

到目前为止，图像的尺寸被缩减为7 * 7，我们最后加入一个神经元数目为1024的全连接层来处理所有的图像上。接着，将最后的pooling层的输出reshape为一个一维向量，与权值相乘，加上偏置，再通过一个ReLU函数。

```
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)
```

整个CNN的网络结构如下图：



Dropout

为了减少过拟合程度，在输出层之前应用dropout技术（即丢弃某些神经元的输出结果）。我们创建一个placeholder来表示一个神经元的输出在dropout时不被丢弃的概率。Dropout能够在训练过程中使用，而在测试过程中不使用。TensorFlow中的`tf.nn.dropout`操作能够利用mask技术处理各种规模的神经元输出。

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

输出层

最终，我们用一个softmax层，得到类别上的概率分布。（与之前的Softmax Regression模型相同）。

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

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


模型训练和测试

为了测试模型的性能，需要先对模型进行训练，然后应用在测试集上。和之前Softmax Regression模型中的训练、测试过程类似。区别在于：

1. 用更复杂的ADAM最优化方法代替了之前的梯度下降；
2. 增了额外的参数keep_prob在feed_dict中，以控制dropout的几率；
3. 在训练过程中，增加了log输出功能（每100次迭代输出一次）。

下面是程序：

```
cross_entropy = -tf.reduce_sum(y_*tf.log(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))
sess.run(tf.initialize_all_variables())
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}))
```

最终，模型在测试集上的准确率大概为99.2%，性能上要优于之前的Softmax Regression模型。

完整代码及运行结果

利用CNN模型实现手写体识别的完整代码如下：

```
__author__ = 'chapter'

import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data

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):
    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], padd

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
print("Download Done!")

sess = tf.InteractiveSession()

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

x = tf.placeholder(tf.float32, [None, 784])
x_image = tf.reshape(x, [-1, 28, 28, 1])

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

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)

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)

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

W_fc2 = weight_variable([1024, 10])

```

```

b_fc2 = bias_variable([10])

y_conv = tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)
y_ = tf.placeholder(tf.float32, [None, 10])

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

sess.run(tf.initialize_all_variables())

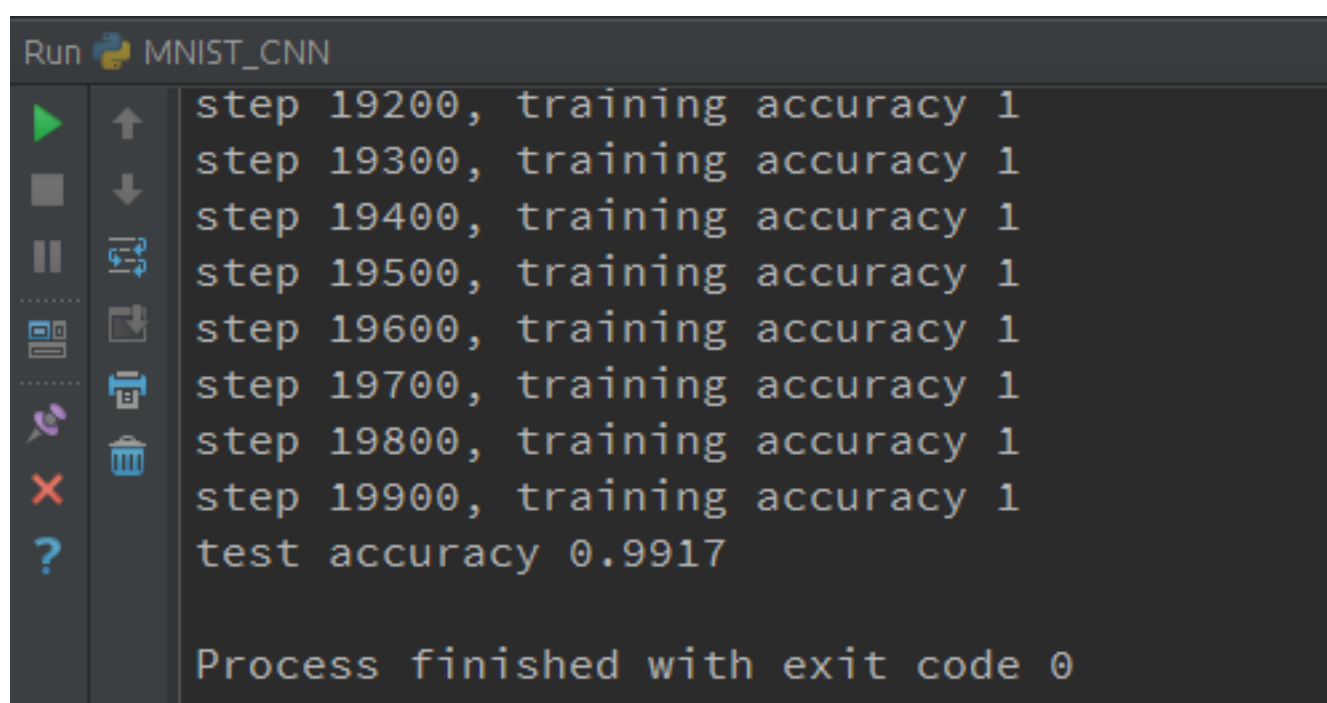
for i in range(20000):
    batch = mnist.train.next_batch(50)

    if i % 100 == 0:
        train_accuacy = accuracy.eval(feed_dict={x: batch[0], y_: batch[1],
        print("step %d, training accuracy %g"%(i, train_accuacy))
        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_

```

运行结果如下图：



```

Run MNIST_CNN
step 19200, training accuracy 1
step 19300, training accuracy 1
step 19400, training accuracy 1
step 19500, training accuracy 1
step 19600, training accuracy 1
step 19700, training accuracy 1
step 19800, training accuracy 1
step 19900, training accuracy 1
test accuracy 0.9917

Process finished with exit code 0

```

本文结束，感谢欣赏。

欢迎转载，请注明本文的链接地址：

<http://www.jeyzhang.com/tensorflow-learning-notes-2.html>

参考资料

[TensorFlow: Deep MNIST for Experts](#)