数字媒体技术作业 3

`罗剑杰 数字媒体 15331229`

作业内容

手写数字识别,用机器学习的方法将手写数字正确分类

实验过程

> 实验环境：
>
> - Ubuntu16.04 LTS
> - tensorflow 1.3.0

1. 下载mnist数据集
2. 使用Anaconda在Ubuntu16.04LTS上配置tensorflow的运行环境，并且配置cuda使得可以使用GPU来进行运算。
3. 参考tensorflow官方学习文档来进行学习对mnist数据集的初级处理以及高级处理。
4. 由于是初次涉及机器学习，一切从**0**开始，所以只是按部就班地对照这官方文档一步一步地去学习整个流程，整个实验过程以参考和动手实践为主。
5. 得到理想的实验结果，实验结束

实验过程与学习

本次实验主要以参考文献，初次一步一步地尝试为主，所以代码上基本使用了**tensorflow**官网的代码，下面是主要是描述每一步代码做了什么事情来体现自己的实验流程以及自己的学习过程。

首先，代码里面其实有很大一部分都是如何使用tensorflow的代码，包括怎么构建一个运算图，放进去运算等等，这些代码的步骤就直接跳过因为和手写数字识别的关系不大。我直接尝试说明建模的代码。

本次实验将tensorflow官网上给出的初级教程和深入教程都尝试了一遍，下面分情况叙述。

使用一层**softmax**回归的入门模型

为了得到一张给定图片属于某个特定数字类的证据（evidence），我们对图片像素值进行加权求和。如果这个像素具有很强的证据说明这张图片不属于该类，那么相应的权值为负数，相反如果这个像素拥有有利的证据支持这张图片属于这个类，那么权值是正数。同时加入一个额外的偏置量（bias），因为输入往往会带有一些无关的干扰量。

$$\text{evidence}_i = \sum_j W_{i,\,j} x_j + b_i$$

然后用softmax函数可以把这些证据转换成概率 **y**

$$y = softmax(evidence)$$

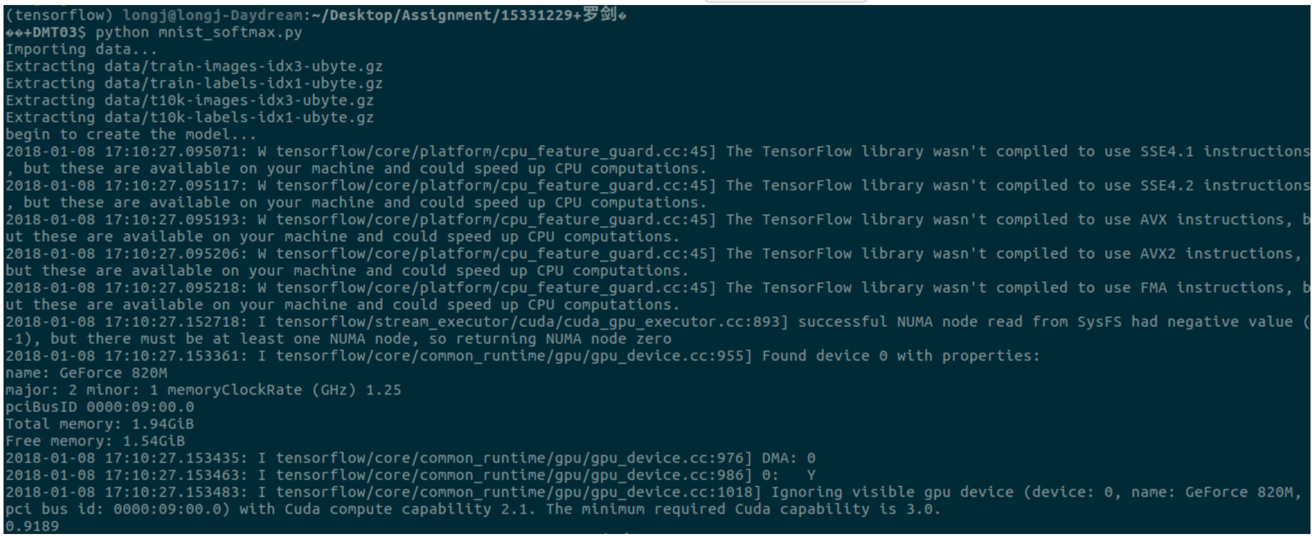
最后使用python的tensorflow的实现只需要下面这一行的代码：

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

训练的时候我们使用交叉熵来作为损失函数，然后使用GradientDescentOptimizer来进行训练。

```
cross_entropy = -tf.reduce_sum(y_*tf.log(y))
train_step = tf.train.GradientDescentOptimizer (0.01).minimize(cross_entropy )
```

每次训练都调用一次train_step，最后进行1000次训练，使用 `` `tf.argmax `` `来进行测试，出来的结果是0.92.



```
(tensorflow) longj@longj-Daydream:~/Desktop/Assignment/15331229+罗剑◆
◆◆+DMT03$ python mnist_softmax.py
Importing data...
Extracting data/train-images-idx3-ubyte.gz
Extracting data/train-labels-idx1-ubyte.gz
Extracting data/t10k-images-idx3-ubyte.gz
Extracting data/t10k-labels-idx1-ubyte.gz
begin to create the model...
2018-01-08 17:10:27.095071: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use SSE4.1 instructions
, but these are available on your machine and could speed up CPU computations.
2018-01-08 17:10:27.095117: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use SSE4.2 instructions
, but these are available on your machine and could speed up CPU computations.
2018-01-08 17:10:27.095193: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use AVX instructions, b
ut these are available on your machine and could speed up CPU computations.
2018-01-08 17:10:27.095206: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use AVX2 instructions,
but these are available on your machine and could speed up CPU computations.
2018-01-08 17:10:27.095218: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use FMA instructions, b
ut these are available on your machine and could speed up CPU computations.
2018-01-08 17:10:27.152718: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:893] successful NUMA node read from SysFS had negative value (
-1), but there must be at least one NUMA node, so returning NUMA node zero
2018-01-08 17:10:27.153361: I tensorflow/core/common_runtime/gpu/gpu_device.cc:955] Found device 0 with properties:
name: GeForce 820M
major: 2 minor: 1 memoryClockRate (GHz) 1.25
pciBusID 0000:09:00.0
Total memory: 1.94GiB
Free memory: 1.54GiB
2018-01-08 17:10:27.153435: I tensorflow/core/common_runtime/gpu/gpu_device.cc:976] DMA: 0
2018-01-08 17:10:27.153463: I tensorflow/core/common_runtime/gpu/gpu_device.cc:986] 0:   Y
2018-01-08 17:10:27.153483: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1018] Ignoring visible gpu device (device: 0, name: GeForce 820M,
pci bus id: 0000:09:00.0) with Cuda compute capability 2.1. The minimum required Cuda capability is 3.0.
0.9189
```

使用简单的卷积神经网络来进行训练

先构建4个方便使用的函数：

```
# 权重初始化
def weight_variable(shape):
  initial = tf.truncated_normal(shape, stddev=0.1)
  return tf.Variable(initial)

# 偏置项初始化
def bias_variable(shape):
  initial = tf.constant(0.1, shape=shape)
  return tf.Variable(initial)

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

# 池化
def max_pool_2x2(x):
  return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                        strides=[1, 2, 2, 1], padding='SAME')
```

构建一个多层卷积网络，模型构建过程如下，主要有四层：

1. 第一层由一个卷积接一个max pooling完成。卷积在每个5x5的patch中算出32个特征。卷积的权重张量形状是 `[5, 5, 1, 32]` ，前两个维度是patch的大小，接着是输入的通道数目，最后是输出的通道数目。 而对于每一个输出通道都有一个对应的偏置量。

```
W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])
x_image = tf.reshape(x, [-1,28,28,1])

# 把x_image和权值向量进行卷积，加上偏置项，然后应用ReLU激活函数，最后进行max pooling。
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)
```

2. 为了构建一个更深的网络，把几个类似的层堆叠起来，在第二层中，每个5x5的patch会得到64个特征。

```
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)
```

3. 现在，图片尺寸减小到7x7，我们加入一个有1024个神经元的全连接层，作为密集连接层，用于处理整个图片。我们把池化层输出的张量reshape成一些向量，乘上权重矩阵，加上偏置，然后对其使用ReLU。

```
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)
```

4. 最后，我们添加一个softmax层，就像前面的单层softmax regression一样

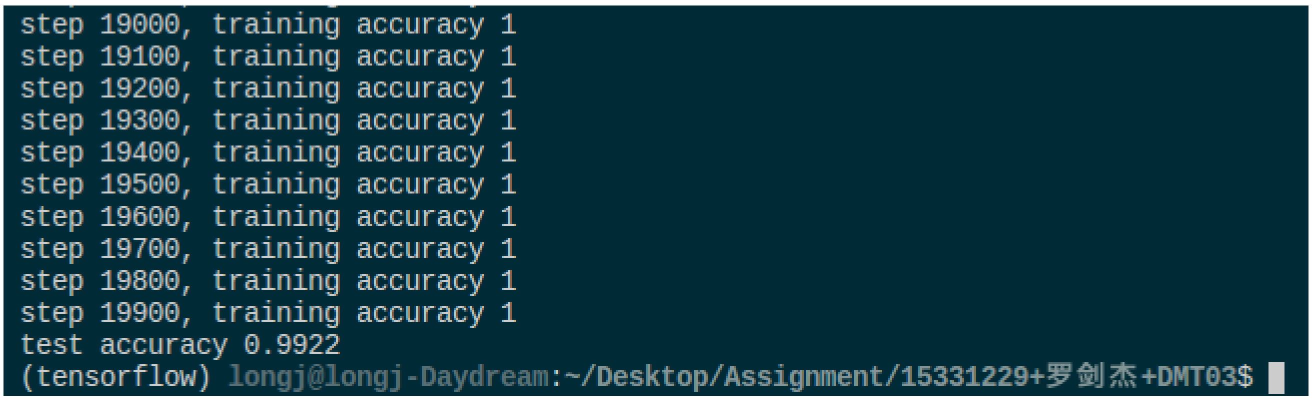
```
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])

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

除去模型的构建，其他的训练等都基本和基础版的一样，只不过是在训练的过程中要添加一个dropout来避免卷积神经网络的过拟合。

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

这次训练20000次出来的结果正确率是：0.992



```
step 19000, training accuracy 1
step 19100, training accuracy 1
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 0.9922
(tensorflow) longj@longj-Daydream:~/Desktop/Assignment/15331229+罗剑杰+DMT03$
```

参考资料

1. MNIST For ML Beginners
2. Deep MNIST for Experts
3. 极客学院官方翻译文档