江西财经大學

实验报告

课程名称：Python 程序设计

项目名称：python 期末大作业

组　　号：第十六组

组　　长：余拔金

组　　员：昌国根 高枫 杨博涵

时间：2018 年 1 月

1、 需求介绍及操作要求

1. 参考课件，将 CNN 中的 Pooling 改为 Autoencoder

1.1. 选择你感兴趣的图像数据集（注意样本数要足够大）进行辨识，比较 Pooling 与Autoencoder 运行速度与准确率的差异。

2. 参考课件，使用 Autoencoder 模拟 PCA（Principal Component Analysis）对 MNIST 手

写辨识数据集进行降维（例如，从 784 维度降至 150）。

2.1. 展示降维后的图片

2.2. 使用降维后的图片进行辨识

2.2.1. 比较降维前后辨识准确率的差异

2、技术描述

2.1、CNN

定义：卷积神经网络包含输入层、隐藏层和输出层，隐藏层又包含卷积层和pooling层，图像输入到卷积神经网络后通过卷积来不断的提取特征，每提取一个特征就会增加一个feature map，所以会看到卷积是的立方体不断的增加厚度，那么为什么厚度增加了但是却越来越瘦了呢，这就是pooling层的作用，pooling层也就是下采样，通常采用的是最大值pooling和平均值pooling，因为参数太多，所以通过pooling来稀疏参数，使网络不至于太复杂，下面通过代码来实现一个基于MNIST数据集的简单卷积神经网络。

定义卷积层的 weight bias

2.1.1、首先导入tensorflow,并且为了方便引用，别名命名为tf

```
2    import tensorflow as tf
```

2.1.2、采用的数据集是tensorflow里面的mnist数据集， 需要先导入它

```
3    from tensorflow.examples.tutorials.mnist import input_data
```

2.1.3、代码用到的数据集就是来自于它

```
5    mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
```

2.1.4、接着定义 Weight 变量，输入 shape，返回变量的参数。其中使用 tf.truncted_normal 产生随机变量来进行初始化：

```
15      def weight_variable(shape):
16          initial = tf.truncated_normal(shape, stddev=0.1)
17          return tf.Variable(initial)
```

2.1.5、同样的定义 biase 变量，输入 shape ，返回变量的一些参数。其中使用 tf.constant 常量函数来进行初始化：

```
19      def bias_variable(shape):
20          initial = tf.constant(0.1, shape=shape)
21          return tf.Variable(initial)
```

2.1.6、定义卷积，tf.nn.conv2d 函数是 tensoflow 里面的二维的卷积函数，x 是图片的所有参数，W 是此卷积层的权重，然后定义步长 strides=[1,1,1,1]值，strides[0]和 strides[3]的两个 1 是默认值，中间两个 1 代表 padding 时在 x 方向运动一步，y 方向运动一步，padding 采用的方式是 SAME。

```
23      def conv2d(x, W):
24          # stride [1, x_movement, y_movement, 1]
25          # Must have strides[0] = strides[3] = 1
26          return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')
```

定义 pooling

2.1.7、接着定义池化 pooling，为了得到更多的图片信息，padding 时选的是一次一步，也就是 strides[1]=strides[2]=1，这样得到的图片尺寸没有变化，而希望压缩一下图片也就是参数能少一些从而减小系统的复杂度，因此采用 pooling 来稀疏化参数，也就是卷积神经网络中所谓的下采样层。pooling 有两种，一种是最大值池化，一种是平均值池化，本例采用的是最大值池化 tf.max_pool()。池化的核函数大小为 2x2，因此 ksize=[1,2,2,1]，步长为 2，因此 strides=[1,2,2,1]：

```
28      def max_pool_2x2(x):
29          # stride [1, x_movement, y_movement, 1]
30          return tf.nn.max_pool(x, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')
```

图片处理

2.1.8、首先呢，定义一下输入的placeholder

```
xs = tf.placeholder(tf.float32, [None, 784])/255.   # 28x28
ys = tf.placeholder(tf.float32, [None, 10])
```

2.1.9、还定义了dropout的placeholder，它是解决过拟合的有效手段

```
keep_prob = tf.placeholder(tf.float32)
```

2.1.10、接着呢，需要处理xs，把xs的形状变成[-1,28,28,1]，-1代表先不考虑输入的图片例子多少这个维度，后面的1是channel的数量，因为输入的图片是黑白的，因此channel是1，如果是RGB图像，那么channel就是3。

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

建立卷积层

2.1.11、接着定义第一层卷积,先定义本层的Weight,本层卷积核patch的大小是5x5，因为黑白图片channel是1所以输入是1，输出是32个featuremap

```
W_conv1 = weight_variable([5,5, 1,32]) # patch 5x5, in size 1, out size 32
```

2.1.12、接着定义bias，它的大小是32个长度，因此传入它的shape为[32]

```
b_conv1 = bias_variable([32])
```

2.1.13、定义好了Weight和bias，就可以定义卷积神经网络的第一个卷积层h_conv1=conv2d(x_image,W_conv1)+b_conv1,同时对h_conv1进行非线性处理，也就是激活函数来处理，这里用的是tf.nn.relu（修正线性单元）来处理，要注意的是，因为采用了SAME的padding方式，输出图片的大小没有变化依然是28x28，只是厚度变厚了，因此现在的输出大小就变成了28x28x32

```
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1) # output size 28x28x32
```

2.1.14、最后再进行pooling处理，经过pooling的处理，输出大小就变为了14x14x32

```
h_pool1 = max_pool_2x2(h_conv1)
```

2.1.15、接着，同样的形式定义第二层卷积，本层的输入就是上一层的输出，本层的卷积核patch的大小是5x5，有32个featuremap所以输入就是32，输出定为64

```
## conv2 layer ##
W_conv2 = weight_variable([5,5, 32, 64]) # patch 5x5, in size 32, out size 64
b_conv2 = bias_variable([64])
```

2.1.16、接着就可以定义卷积神经网络的第二个卷积层，这时的输出的大小就是14x14x64

```
48    h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2) # output size 14x14x64
```

2.1.17、最后也是一个pooling处理，输出大小为7x7x64

```
49    h_pool2 = max_pool_2x2(h_conv2)                    # output size 7x7x64
50
```

建立全连接层

2.1.18、接下来定义 fully connected layer,进入全连接层时，通过 tf.reshape()将 h_pool2 的输出值从一个三维的变为一维的数据，-1 表示先不考虑输入图片例子维度，将上一个输出结果展平.

```
55    h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
```

2.1.19、此时 weight_variable 的 shape 输入就是第二个卷积层展平了的输出大小：7x7x64，后面的输出 size 继续扩大，定为 1024

```
51    ## fc1 layer ##
52    W_fc1 = weight_variable([7*7*64, 1024])
53    b_fc1 = bias_variable([1024])
```

2.1.20、然后将展平后的 h_pool2_flat 与本层的 W_fc1 相乘（注意这个时候不是卷积）

```
56    h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
```

2.1.21、考虑过拟合问题，可以加一个 dropout 的处理

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

2.1.22、接下来进行最后一层的构建，输入是 1024，最后的输出是 10 个（因为 mnist 数据集就是[0-9]十个类），prediction 就是最后的预测值

```
60    W_fc2 = weight_variable([1024, 10])
```

2.1.23、用 softmax 分类器（多分类，输出是各个类的概率）,对输出进行分类

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

选优化方法

2.1.24、利用交叉熵损失函数来定义 cost function

```
66    cross_entropy = tf.reduce_mean(-tf.reduce_sum(ys * tf.log(prediction),
67                                                  reduction_indices=[1]))       # loss
```

2.1.25、用 tf.train.AdamOptimizer()作为优化器进行优化，使 cross_entropy 最小

```
68    train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
```

2.1.26、定义 Session，初始化变量，训练数据，假定训练 1000 步，每 50 步输出一下准确率， 注意 sess.run() 时记得要用 feed_dict 给众多 placeholder feed 数据.

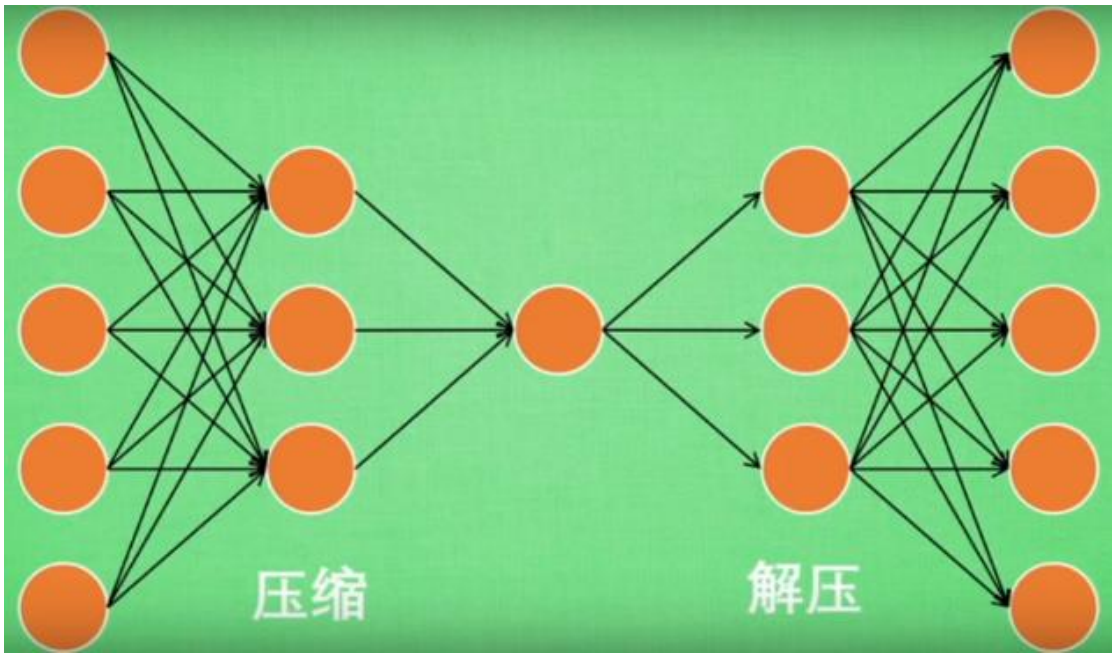
运行结果:

```
C:\Users\Administrator\Anaconda3\python.exe C:/Users/Adminis
Extracting MNIST_data\train-images-idx3-ubyte.gz
Extracting MNIST_data\train-labels-idx1-ubyte.gz
Extracting MNIST_data\t10k-images-idx3-ubyte.gz
Extracting MNIST_data\t10k-labels-idx1-ubyte.gz
0.265625
0.703125
0.796875
0.867188
0.867188
0.882813
0.875
0.882813
0.898438
0.914063
0.867188
0.929688
0.914063
0.9375
```

```
0.90625
0.929688
0.945313
0.953125
0.929688
0.96875
0.945313
0.984375
0.90625
0.976563
0.96875
0.953125
0.96875
```

2.2、自编码 Autoencoder (非监督学习)

定义：Autoencoder将有很多Feature的数据进行压缩，之后再进行解压的过程。 本质上来说，它也是一个对数据的非监督学习,,**PCA** (Principal component analysis)， 与 Autoencoder 相类似，它的主要功能即对数据进行非监督学习，并将压缩之后得到的“特征值”，这一中间结果正类似于PCA的结果。 之后再将压缩过的“特征值”进行解压，得到的最终结果与原始数据进行比较，对此进行非监督学习。大概过程如下图所示



2.2.1、**Autoencoder**

```
14  # Visualize decoder setting
15  # Parameters
16  learning_rate = 0.01
17  training_epochs = 5
18  batch_size = 256
19  display_step = 1
20  examples_to_show = 10
21
```

2.2.2、MNIST数据，每张图片大小是 28x28 pix，即 784 Features：

```
22  # Network Parameters
23  n_input = 784  # MNIST data input (img shape: 28*28)
```

2.2.3、在压缩环节：要把Features不断压缩，经过第一个隐藏层压缩至256个Features，再经过第二个隐藏层压缩至128个。在解压环节：将128个Features还原至256个，再经过一步还原至784个。在对比环节：比较原始数据与还原后

的拥有 784 Features 的数据进行 cost 的对比，根据 cost 来提升 Autoencoder 的准确率，下图是两个隐藏层的 weights 和 biases 的定义：

```
# hidden layer settings
n_hidden_1 = 256 # 1st layer num features
n_hidden_2 = 128 # 2nd layer num features
weights = {
    'encoder_h1': tf.Variable(tf.random_normal([n_input, n_hidden_1])),
    'encoder_h2': tf.Variable(tf.random_normal([n_hidden_1, n_hidden_2])),
    'decoder_h1': tf.Variable(tf.random_normal([n_hidden_2, n_hidden_1])),
    'decoder_h2': tf.Variable(tf.random_normal([n_hidden_1, n_input])),
}
biases = {
    'encoder_b1': tf.Variable(tf.random_normal([n_hidden_1])),
    'encoder_b2': tf.Variable(tf.random_normal([n_hidden_2])),
    'decoder_b1': tf.Variable(tf.random_normal([n_hidden_1])),
    'decoder_b2': tf.Variable(tf.random_normal([n_input])),
}
```

2.2.4、下面来定义 Encoder 和 Decoder ，使用的 Activation function 是 sigmoid， 压缩之后的值应该在 [0,1] 这个范围内。在 decoder 过程中，通常使用对应于 encoder 的 Activation function：

```
def encoder(x):
    # Encoder Hidden layer with sigmoid activation #1
    layer_1 = tf.nn.sigmoid(tf.add(tf.matmul(x, weights['encoder_h1']),
                                   biases['encoder_b1']))
    # Decoder Hidden layer with sigmoid activation #2
    layer_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer_1, weights['encoder_h2']),
                                   biases['encoder_b2']))
    return layer_2

# Building the decoder
def decoder(x):
    # Encoder Hidden layer with sigmoid activation #1
    layer_1 = tf.nn.sigmoid(tf.add(tf.matmul(x, weights['decoder_h1']),
                                   biases['decoder_b1']))
    # Decoder Hidden layer with sigmoid activation #2
    layer_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer_1, weights['decoder_h2']),
                                   biases['decoder_b2']))
    return layer_2
```

2.2.5、来实现 Encoder 和 Decoder 输出的结果：

```
123     # Construct model
124     encoder_op = encoder(X)
125     decoder_op = decoder(encoder_op)
126
127     # Prediction
128     y_pred = decoder_op
129     # Targets (Labels) are the input data.
130     y_true = X
131
```

2.2.6、再通过非监督学习进行对照，即对 “原始的有 784 Features 的数据集” 和 “通过 ‘Prediction’ 得出的有 784 Features 的数据集” 进行最小二乘法的计算，并且使 cost 最小化：

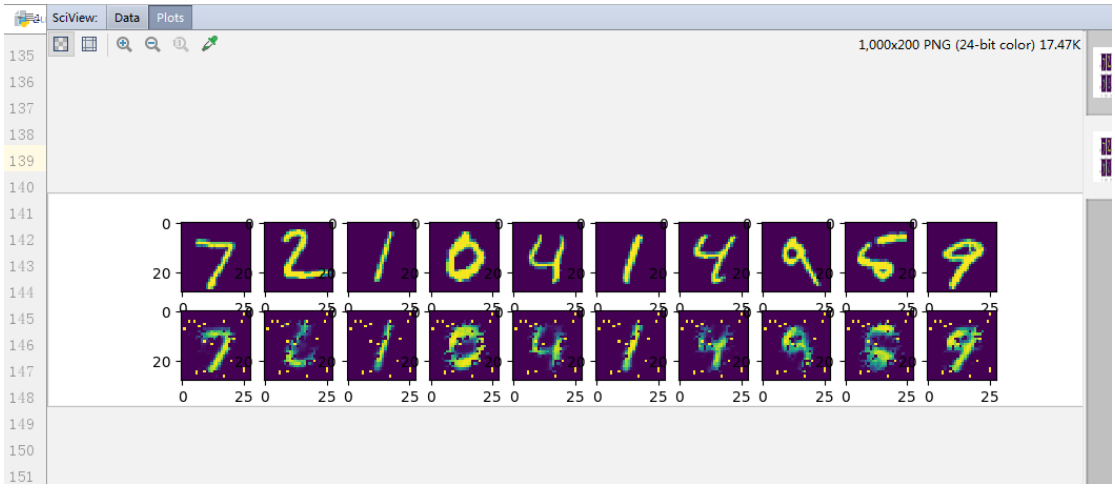
```
132     # Define loss and optimizer, minimize the squared error
133     cost = tf.reduce_mean(tf.pow(y_true - y_pred, 2))
134     optimizer = tf.train.AdamOptimizer(learning_rate).minimize(cost)
135
```

2.2.7、最后，通过 Matplotlib 的 pyplot 模块将结果显示出来， 注意在输出时MNIST数据集经过压缩之后 x 的最大值是1，而非255：

```
137 # Launch the graph
138 with tf.Session() as sess:
139     # tf.initialize_all_variables() no long valid from
140     # 2017-03-02 if using tensorflow >= 0.12
141     if int((tf.__version__).split('.')[1]) < 12 and int((tf.__version__).split('.')[0]) < 1:
142         init = tf.initialize_all_variables()
143     else:
144         init = tf.global_variables_initializer()
145     sess.run(init)
146     total_batch = int(mnist.train.num_examples/batch_size)
147     # Training cycle
148     for epoch in range(training_epochs):
149         # Loop over all batches
150         for i in range(total_batch):
151             batch_xs, batch_ys = mnist.train.next_batch(batch_size)  # max(x) = 1, min(x) = 0
152             # Run optimization op (backprop) and cost op (to get loss value)
```

```
153             _, c = sess.run([optimizer, cost], feed_dict={X: batch_xs})
154         # Display logs per epoch step
155         if epoch % display_step == 0:
156             print("Epoch:", '%04d' % (epoch+1),
157                   "cost=", "{:.9f}".format(c))
158 
159     print("Optimization Finished!")
160 
161     # # Applying encode and decode over test set
162     encode_decode = sess.run(
163         y_pred, feed_dict={X: mnist.test.images[:examples_to_show]})
164     # Compare original images with their reconstructions
165     f, a = plt.subplots(2, 10, figsize=(10, 2))
166     for i in range(examples_to_show):
167         a[0][i].imshow(np.reshape(mnist.test.images[i], (28, 28)))
168         a[1][i].imshow(np.reshape(encode_decode[i], (28, 28)))
169     plt.show()
```

2.2.8、通过5个 Epoch 的训练，结果如下：



上面一行是真实数据，下面一行是经过 encoder 和 decoder 之后的数据

3. 总结

通过做 python 大作业，让我们深刻的掌握本学期 python 的学习内容，特别是对 tensorflow 的学习和理解，为了完成这个对于我们组来说是一个非常艰巨的大作业来说，可谓是花了很多功夫，把老师的 ppt 从头到尾翻译啃了下来，将老师的代码一个字母一个字母敲到自己电脑运行，不断调试修改，虽然最后 CNN_Pooling.py 还有难以修复的 bug 存在，但这并不影响我们对 tensorflow 的学习，根据老师教案的思想，我们查阅最新相关资料，独立敲出能够运行的代码，了解每步代码的作用；通过这次学习，不但让我们更加深入的学习了 tensorflow，增强了我们每一个人自学和动手实践去解决问题的能力；并且让我们意识到团队协作的重要性，相信这对我们以后的工作，学习，生活，都有着重要的作用。

一个好的项目在于我们不断的改进与优化，我们的项目中还存在许多可以改进的地方，我们也会在之后用我们不断的学习做到更好。

4、附件（程序代码及相关材料）

见文件附件