**姓名：杨庆 学号：201822090316**

**题目.利用MNIST数据集，构建前馈神经网络，并画出运行曲线**

'''

**（1）导入数据和相关包及输出数据集相关信息等设置**

'''

# -\*- coding: utf-8 -\*-

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow.examples.tutorials.mnist import input\_data

mnist = input\_data.read\_data\_sets('MNIST\_data', one\_hot=True)

# 设置tensorflow对GPU使用按需分配

config = tf.ConfigProto()

config.gpu\_options.allow\_growth = True

sess = tf.InteractiveSession(config=config)

print(type(mnist)) # <class 'tensorflow.contrib.learn.python.learn.datasets.base.Datasets'>

print('Training data shape:', mnist.train.images.shape) # Training data shape: (55000, 784)

print('Test data shape:', mnist.test.images.shape) # Test data shape: (10000, 784)

print('Validation data shape:', mnist.validation.images.shape) # Validation data shape: (5000, 784)

print('Training label shape:', mnist.train.labels.shape) # Training label shape: (55000, 10)

'''

**（2）搭建前馈神经网络模型**

'''

# 初始化权值和偏重

def weight\_variable(shape):

# 使用正太分布初始化权值

initial = tf.truncated\_normal(shape, stddev=0.1) # 标准差为0.1

return tf.Variable(initial)

def bias\_variable(shape):

initial = tf.constant(0.1, shape=shape)

return tf.Variable(initial)

# input layer，None表示张量第一维度可以是任意长度的

x\_ = tf.placeholder(tf.float32, shape=[None, 784])

y\_ = tf.placeholder(tf.float32, shape=[None, 10])

# 隐藏层

w\_h = weight\_variable([784, 1024])

b\_h = bias\_variable([1024])

hidden = tf.nn.relu(tf.matmul(x\_, w\_h) + b\_h)

# 输出层

w\_o = weight\_variable([1024, 10])

b\_o = bias\_variable([10])

output = tf.nn.softmax(tf.matmul(hidden, w\_o) + b\_o)

'''

**（3）设置对数似然损失函数**

'''

# 代价函数 J =-(Σy.logaL)/n .表示逐元素乘

cost = tf.reduce\_mean(-tf.reduce\_sum(y\_ \* tf.log(output), axis=1))

'''

**（4）求解并训练**

'''

train = tf.train.AdamOptimizer(0.001).minimize(cost)

# 预测结果评估

# tf.argmax(output,1) 按行统计最大值得索引

correct = tf.equal(tf.argmax(output, 1), tf.argmax(y\_, 1)) # 返回一个数组 表示统计预测正确或者错误

accuracy = tf.reduce\_mean(tf.cast(correct, tf.float32)) # 求准确率

# 创建list 保存每一迭代的结果

training\_accuracy\_list = []

test\_accuracy\_list = []

training\_cost\_list = []

test\_cost\_list = []

# 使用会话执行图

sess.run(tf.global\_variables\_initializer()) # 初始化变量

# 开始迭代 使用Adam优化的随机梯度下降法

for i in range(5000): # 一个epoch需要迭代次数计算公式：测试集长度 / batch\_size

x\_batch, y\_batch = mnist.train.next\_batch(batch\_size=64)

# 开始训练

train.run(feed\_dict={x\_: x\_batch, y\_: y\_batch})

if (i + 1) % 200 == 0:

# 输出训练集准确率

# training\_accuracy = accuracy.eval(feed\_dict={x\_:mnist.train.images,y\_:mnist.train.labels})

training\_accuracy, training\_cost = sess.run([accuracy, cost],

feed\_dict={x\_: mnist.train.images, y\_: mnist.train.labels})

training\_accuracy\_list.append(training\_accuracy)

training\_cost\_list.append(training\_cost)

print('{0}:Training set accuracy {1},cost {2}.'.format(i + 1, training\_accuracy, training\_cost))

# 输出测试集准确率

# test\_accuracy = accuracy.eval(feed\_dict={x\_:mnist.test.images,y\_:mnist.test.labels})

test\_accuracy, test\_cost = sess.run([accuracy, cost], feed\_dict={x\_: mnist.test.images, y\_: mnist.test.labels})

test\_accuracy\_list.append(test\_accuracy)

test\_cost\_list.append(test\_cost)

print('{0}:Test set accuracy {1},cost {2}.'.format(i + 1, test\_accuracy, test\_cost))

'''

**（5）绘制训练集准确率，以及测试集准确率曲线**

'''

def plot\_overlay\_accuracy(training\_accuracy, test\_accuaracy):

# 迭代次数

num\_epochs = len(test\_accuaracy)

# 获取一个figure实例

fig = plt.figure()

# 使用面向对象的方式添加Axes实例，参数1：子图总行数 参数2：子图总列数 参数3：子图位置

ax = fig.add\_subplot(111)

ax.plot(np.arange(0, num\_epochs),

[accuracy \* 100.0 for accuracy in test\_accuaracy],

color='#2A6EA6',

label="Accuracy on the test data")

ax.plot(np.arange(0, num\_epochs),

[accuracy \* 100.0 for accuracy in training\_accuracy],

color='#FFA933',

label="Accuracy on the training data")

ax.grid(True)

ax.set\_xlim([0, num\_epochs])

ax.set\_xlabel('Epoch')

ax.set\_ylim([90, 100])

ax.legend(loc="lower right") # 右小角

plt.show()

'''

**（6）绘制训练集代价和测试集代价函数曲线**

'''

def plot\_overlay\_cost(training\_cost, test\_cost):

'''

training,test:训练集测试集代价 list类型

'''

# 迭代次数

num\_epochs = len(test\_cost)

fig = plt.figure()

ax = fig.add\_subplot(111)

ax.plot(np.arange(0, num\_epochs),

[cost for cost in test\_cost],

color='#2A6EA6',

label="Cost on the test data")

ax.plot(np.arange(0, num\_epochs),

[cost for cost in training\_cost],

color='#FFA933',

label="Cost on the training data")

ax.grid(True)

ax.set\_xlim([0, num\_epochs])

ax.set\_xlabel('Epoch')

# ax.set\_ylim([0, 0.75])

ax.legend(loc="upper right")

plt.show()

'''

**（6）绘制最终曲线图**

'''

plot\_overlay\_cost(training\_cost\_list, test\_cost\_list)

plot\_overlay\_accuracy(training\_accuracy\_list, test\_accuracy\_list)

'''

**（7）输出结果及曲线图**

'''

2018-10-31 14:29:48.745946: I tensorflow/stream\_executor/cuda/cuda\_gpu\_executor.cc:898] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero

2018-10-31 14:29:48.746191: I tensorflow/core/common\_runtime/gpu/gpu\_device.cc:1344] Found device 0 with properties:

name: GeForce GTX 1070 Ti major: 6 minor: 1 memoryClockRate(GHz): 1.683

pciBusID: 0000:01:00.0

totalMemory: 7.93GiB freeMemory: 7.53GiB

2018-10-31 14:29:48.746200: I tensorflow/core/common\_runtime/gpu/gpu\_device.cc:1423] Adding visible gpu devices: 0

2018-10-31 14:29:48.935325: I tensorflow/core/common\_runtime/gpu/gpu\_device.cc:911] Device interconnect StreamExecutor with strength 1 edge matrix:

2018-10-31 14:29:48.935347: I tensorflow/core/common\_runtime/gpu/gpu\_device.cc:917] 0

2018-10-31 14:29:48.935351: I tensorflow/core/common\_runtime/gpu/gpu\_device.cc:930] 0: N

2018-10-31 14:29:48.935491: I tensorflow/core/common\_runtime/gpu/gpu\_device.cc:1041] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 7272 MB memory) -> physical GPU (device: 0, name: GeForce GTX 1070 Ti, pci bus id: 0000:01:00.0, compute capability: 6.1)

<class 'tensorflow.contrib.learn.python.learn.datasets.base.Datasets'>

Training data shape: (55000, 784)

Test data shape: (10000, 784)

Validation data shape: (5000, 784)

Training label shape: (55000, 10)

200:Training set accuracy 0.9280181527137756,cost 0.2417505532503128.

200:Test set accuracy 0.9236000180244446,cost 0.24713367223739624.

400:Training set accuracy 0.9478181600570679,cost 0.172581285238266.

400:Test set accuracy 0.9451000094413757,cost 0.19024436175823212.

600:Training set accuracy 0.9652000069618225,cost 0.11855651438236237.

600:Test set accuracy 0.9614999890327454,cost 0.1311209499835968.

800:Training set accuracy 0.9697272777557373,cost 0.1019970178604126.

800:Test set accuracy 0.958299994468689,cost 0.1317475140094757.

1000:Training set accuracy 0.9782363772392273,cost 0.0748174712061882.

1000:Test set accuracy 0.9696999788284302,cost 0.10308928042650223.

1200:Training set accuracy 0.9798363447189331,cost 0.06998933106660843.

1200:Test set accuracy 0.9700999855995178,cost 0.0987396314740181.

1400:Training set accuracy 0.9821272492408752,cost 0.06052234396338463.

1400:Test set accuracy 0.9710999727249146,cost 0.09167661517858505.

1600:Training set accuracy 0.9841272830963135,cost 0.051619723439216614.

1600:Test set accuracy 0.9728000164031982,cost 0.08829327672719955.

1800:Training set accuracy 0.9869999885559082,cost 0.043218471109867096.

1800:Test set accuracy 0.9729999899864197,cost 0.08344998955726624.

2000:Training set accuracy 0.9875272512435913,cost 0.0430133081972599.

2000:Test set accuracy 0.9733999967575073,cost 0.08432697504758835.

2200:Training set accuracy 0.989799976348877,cost 0.0361553393304348.

2200:Test set accuracy 0.9758999943733215,cost 0.08136545121669769.

2400:Training set accuracy 0.9892908930778503,cost 0.03630650416016579.

2400:Test set accuracy 0.9760000109672546,cost 0.08358122408390045.

2600:Training set accuracy 0.9884181618690491,cost 0.036777254194021225.

2600:Test set accuracy 0.9745000004768372,cost 0.08053569495677948.

2800:Training set accuracy 0.993399977684021,cost 0.023497065529227257.

2800:Test set accuracy 0.9779999852180481,cost 0.0737122893333435.

3000:Training set accuracy 0.993254542350769,cost 0.022750070318579674.

3000:Test set accuracy 0.9789000153541565,cost 0.06917915493249893.

3200:Training set accuracy 0.9912545680999756,cost 0.02906673401594162.

3200:Test set accuracy 0.9754999876022339,cost 0.08160725235939026.

3400:Training set accuracy 0.9899636507034302,cost 0.030348634347319603.

3400:Test set accuracy 0.9746999740600586,cost 0.07778524607419968.

3600:Training set accuracy 0.9926000237464905,cost 0.023747967556118965.

3600:Test set accuracy 0.9753000140190125,cost 0.08161821216344833.

3800:Training set accuracy 0.9956363439559937,cost 0.016152694821357727.

3800:Test set accuracy 0.9801999926567078,cost 0.0673767477273941.

4000:Training set accuracy 0.9943636655807495,cost 0.019177671521902084.

4000:Test set accuracy 0.9789999723434448,cost 0.06859112530946732.

4200:Training set accuracy 0.9952545166015625,cost 0.016384044662117958.

4200:Test set accuracy 0.9804999828338623,cost 0.06996545195579529.

4400:Training set accuracy 0.9969817996025085,cost 0.01115241926163435.

4400:Test set accuracy 0.9807000160217285,cost 0.06920187175273895.

4600:Training set accuracy 0.9964908957481384,cost 0.012487773783504963.

4600:Test set accuracy 0.98089998960495,cost 0.07165580242872238.

4800:Training set accuracy 0.9954181909561157,cost 0.016384895890951157.

4800:Test set accuracy 0.9778000116348267,cost 0.07801597565412521.

5000:Training set accuracy 0.9948909282684326,cost 0.01720154657959938.

5000:Test set accuracy 0.9782999753952026,cost 0.07976825535297394.



