# 实验一

一、问题描述

学会使用学习到的线性回归的知识，手动使用adagrad梯度下降方法，通过给定的相关数据来完成PM2.5值的回归预测。

本次作业是使用某监控站的观测记录，分成train set 以及 test set，train set 是该站每个月前20天的所有资料，test set 是从该站剩下的资料中取样出来。即：

1）Train.csv每个月前20天的完整资料。

2）Test.csv从剩下资料中取样出连续的10小时为1笔数据，前9小时所有的观测数据当做feature，第10小时的PM2.5 当做answer，一共可以取出240笔不重复的test data，请根据feature预测这些240笔的PM2.5.

Data中含有18项污染物的观测数据AMB\_TEMP, CH4, CO, NHMC, NO, NO2, NOx, O3, PM10, PM2.5, RAINFALL, RH, SO2, THC, WD\_HR, WIND\_DIREC, WIND\_SPEED, WS\_HR。

二、设计简要描述：

1、从Train data 数据集中提取出有用的数据

2、从数据集中分离出train\_x,train\_y 数据

3、采用函数的形式，实现adagrad算法

4、给train\_x加偏置项，初始化w

5、带入train\_x,lrate,iteration到函数算法中得到结果

6、存下模型w

7、读取测试集，提取出test\_x数据集

8、将测试集于训练出的模型进行预测，生成最终结果 predict.csv

9、读入老师给的标准ans数据集

10、将自己做的结果和老师给的结果进行比对

三、程序清单：

import csv, os

import numpy as np

import matplotlib.pyplot as plt

from numpy.linalg import inv

import math

# Adagrad 算法实现：

def ada(X, Y, w, lrate, iteration):

    s\_grad = np.zeros(len(X[0]))

    list\_cost = []

    for i in range(iteration):

        hypo = np.dot(X,w)

        loss = hypo - Y

        cost = np.sum(loss\*\*2)/len(X)

        cost\_a = math.sqrt(cost)

        list\_cost.append(cost)

        grad = np.dot(X.T, loss) # 求微分 loss = (hypo - y)   --->   &gra/&w1 = (hypo - y) \* x1 = loss \* x1

        s\_grad += grad \*\* 2

        ada = np.sqrt(s\_grad)

        w = w - lrate\*grad/ada

        print('iteration: %d | Cost: %f  ' % (i, cost\_a))

    return w, list\_cost

# 提取数据：

# 每一个维度储存一种污染物的数据

data = []

for i in range(18):

    data.append([])

#data (18,1)

#read data

n\_row = 0

text = open('data/train\_zf.csv', 'r', encoding='big5')

row = csv.reader(text, delimiter=',')

for r in row:

    if n\_row != 0:

        for i in range(3,27):

            if r[i] != "NR":

                data[(n\_row-1)%18].append(float(r[i]))

            else:

                data[(n\_row-1)%18].append(float(0))

    n\_row = n\_row + 1

text.close

#parse data to trainX and trainY

x = []

y = []

for i in range(12):

    for j in range(471):

        x.append([])

        for t in range(18):

            for s in range(9):

                x[471\*i + j].append(data[t][480\*i+j+s])

        y.append(data[9][480\*i+j+9])

trainX = np.array(x) #每一行有9\*18个数 每9个代表9天的某一种污染物

trainY = np.array(y)

trainX = np.concatenate((np.ones((trainX.shape[0],1)), trainX), axis=1)# 加偏置 b

w = np.zeros(len(trainX[0]))

# 训练模型

w\_ada, cost\_list\_ada = ada(trainX, trainY, w, lrate=1, iteration=20000)

# save model

np.save('model.npy', w\_ada)

# read model

w = np.load('model.npy')

#parse test data

test\_x = []

n\_row = 0

text = open('data/test\_zf.csv' ,"r")

row = csv.reader(text , delimiter= ",")

for r in row:

    if n\_row %18 == 0:

        test\_x.append([])

        for i in range(2,11):

            test\_x[n\_row//18].append(float(r[i]) )

    else :

        for i in range(2,11):

            if r[i] !="NR":

                test\_x[n\_row//18].append(float(r[i]))

            else:

                test\_x[n\_row//18].append(0)

    n\_row = n\_row+1

text.close()

test\_x = np.array(test\_x)

test\_x = np.concatenate((np.ones((test\_x.shape[0],1)),test\_x), axis=1)#加上偏置

# 计算结果，并且存下来

ans = []

for i in range(len(test\_x)):

    ans.append(["id\_"+str(i)])

    a = np.dot(test\_x[i],w\_ada)

    ans[i].append(a)

filename = "data/predict\_zf.csv"

text = open(filename, "w+")

s = csv.writer(text,delimiter=',',lineterminator='\n')

s.writerow(["id","value"])

for i in range(len(ans)):

    s.writerow(ans[i])

text.close()

plt.figure()

plt.subplot(132)

plt.title('ada')

plt.xlabel('dataset')

plt.ylabel('pm2.5')

plt.plot(np.arange((len(ans\_y))), ans\_y, 'r,')

plt.plot(np.arange(240), y\_ada, 'g')

plt.tight\_layout()

plt.savefig(os.path.join(os.path.dirname(\_\_file\_\_), "figures/Compare"))

plt.show()

# 实验二

一、问题描述：

学会使用学习到的概率生成模型相关的知识，找出各类别最佳的高斯分布，从而达到通过输入测试，完成二分类任务，成功预测是否该用户年薪达到50k美元。

数据集：贝克尔(Barry Becker)从1994年的人口普查数据库中提取的数据。利用以下条件提取出一组相当清晰的记录:

((AGE>16) && (AGI>100) && (AFNLWGT>1) && (HRSWK>0)).

数据集展示：

二、设计简要描述：

1、从Train data 数据集中提取数据train\_x，进行编码，然后归一化，去除维度数据里面一个多余的属性。

2、从数据集中分离出train\_y

3、采用函数的形式，算出需要的平均值，方差，实现概率生成模型算法

4、得到模型结果 w，b

5、读取测试集，提取出test\_x数据集

1. 将test\_x使用训练出的模型进行预测，生成最终结果 PGModelpredict.csv
2. 随机抽取部分训练集进行预测与已有结果进行对比，分析正确率。

三、程序清单：

import pandas as pd

import numpy as np

from numpy.linalg import inv

import os

def dataProcess\_X(rawData):

#sex 只有两个属性 先drop之后处理

if "income" in rawData.columns:

Data = rawData.drop(["sex", 'income'], axis=1) #删掉性别，收入两行

else:

Data = rawData.drop(["sex"], axis=1)

listObjectColumn = [col for col in Data.columns if Data[col].dtypes == "object"] #读取非数字的column

listNonObjedtColumn = [x for x in list(Data) if x not in listObjectColumn] #数字的column

#提出那些列是对象的拿出来。

ObjectData = Data[listObjectColumn]

NonObjectData = Data[listNonObjedtColumn]

#insert set into nonobject data with male = 0 and female = 1

NonObjectData.insert(0 ,"sex", (rawData["sex"] == " Female").astype(np.int))# 首先 0 列，列名：sex，第三个参数：数值， 0代表男性，1代表女性。

#set every element in object rows as an attribute

ObjectData = pd.get\_dummies(ObjectData)# 独热编码。

Data = pd.concat([NonObjectData, ObjectData], axis=1)

Data\_x = Data.astype("int64")

# Data\_y = (rawData["income"] == " <=50K").astype(np.int)

#normalize

Data\_x = (Data\_x - Data\_x.mean()) / Data\_x.std() #每列求平均值，然后计算与平均值之间的距离 / 标准偏差。

#返回的是经过处理过的dataframe。独热编码

return Data\_x

def dataProcess\_Y(rawData):

df\_y = rawData['income']

Data\_y = pd.DataFrame((df\_y==' >50K').astype("int64"), columns=["income"]) #不知道多少行，一列的数据。（n,1)

return Data\_y

def sigmoid(z):

res = 1 / (1.0 + np.exp(-z))

return np.clip(res, 1e-8, (1-(1e-8))) #防止返回的数过小，过大

def \_shuffle(X, Y): #X and Y are np.array

randomize = np.arange(X.shape[0])

np.random.shuffle(randomize)

return (X[randomize], Y[randomize])

把x，y分开成训练集和验证集。

def split\_valid\_set(X, Y, percentage):

all\_size = X.shape[0]

valid\_size = int(floor(all\_size \* percentage))

X, Y = \_shuffle(X, Y)

X\_valid, Y\_valid = X[ : valid\_size], Y[ : valid\_size]

X\_train, Y\_train = X[valid\_size:], Y[valid\_size:]

return X\_train, Y\_train, X\_valid, Y\_valid

# 验证的函数

def valid(w，b，X):

X\_t = X.T

a = np.dot(w,X\_t) + b

y = sigmoid(a)

y\_ = np.around(y)

result = (np.squeeze(Y) == y\_)

print('Valid acc = %f' % (float(result.sum()) / result.shape[0]))

return

def train(X\_train, Y\_train):

train\_data\_size = X\_train.shape[0]

num1 = 0

num2 = 0

mu1 = np.zeros((106,))

mu2 = np.zeros((106,))

for i in range(train\_data\_size):

if Y\_train[i] == 1: # >50k

mu1 += X\_train[i]

num1 += 1

else:

mu2 += X\_train[i]

num2 += 1

# mu1 present >50k 平均值mu1、mu2

mu1 /= num1

mu2 /= num2

sigma1 = np.zeros((106, 106))

sigma2 = np.zeros((106, 106))

for i in range(train\_data\_size):

if Y\_train[i] == 1:

sigma1 += np.dot(np.transpose([X\_train[i] - mu1]), [X\_train[i] - mu1])

else:

sigma2 += np.dot(np.transpose([X\_train[i] - mu2]), [X\_train[i] - mu2])

sigma1 /= num1 #平均方差

sigma2 /= num2

#共享方差

shared\_sigma = (float(num1) / train\_data\_size) \* sigma1 + (float(num2) / train\_data\_size) \* sigma2

N1 = num1

N2 = num2

sigma\_inv = inv(shared\_sigma)

# 两个函数之间的差 \* 公共方差

w = np.dot((mu1 - mu2), sigma\_inv)

b = (-0.5) \* np.dot(np.dot(mu1.T, sigma\_inv), mu1) + (0.5) \* np.dot(np.dot(mu2.T, sigma\_inv), mu2) + np.log(

float(N1) / N2)

# 训练出最终的w,b

return w,b

if \_\_name\_\_ == "\_\_main\_\_":

trainData = pd.read\_csv("data/train.csv")

testData = pd.read\_csv("data/test.csv")

# ans = pd.read\_csv("data/correct\_answer.csv")

# here is one more attribute in trainData

x\_train = dataProcess\_X(trainData).drop(['native\_country\_ Holand-Netherlands'], axis=1).values

x\_test = dataProcess\_X(testData).values

y\_train = dataProcess\_Y(trainData).values

X\_t = x\_test.T

a = np.dot(w, X\_t) + b

y = sigmoid(a)

y\_ = np.around(y).astype(np.int)

output\_dir = "output/"

df = pd.DataFrame({"id": np.arange(1, 16282), "label": y\_})

if not os.path.exists(output\_dir):

os.mkdir(output\_dir)

df.to\_csv(os.path.join(output\_dir+'PGModelPredict.csv'), sep='\t', index=False)

# 实验三

1、从Train data 数据集中提取数据train\_x，进行编码，然后归一化，去除维度数据里面一个多余的属性。

2、从数据集中分离出train\_y，并且编码好

3、 写好打乱循序，拆分数据集一部分为trainData、一部分为validData

4、设定数据训练的训练的次数，每一次训练里面每一批数据的大小

5、对trainData进行训练，每一次训练完成之后，把损失函数还有正确率提出来，留着之后进行绘图。

1. 最终训练完，返回训练好的W
2. 将test\_x使用训练出的模型进行预测，生成最终结果 LRpredict.csv
3. 利用之前的数据，画出训练集以及验证集中lossfuncton的值 以及 准确率 的对比图形

三、程序清单：

import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
from math import floor, log  
import os  
  
output\_dir = "output/"  
# 处理数据 x  
def dataProcess\_X(rawData):  
  
 #sex 只有两个属性 先drop之后处理  
 if "income" in rawData.columns:  
 Data = rawData.drop(["sex", 'income'], axis=1)  
 else:  
 Data = rawData.drop(["sex"], axis=1)  
 listObjectColumn = [col for col in Data.columns if Data[col].dtypes == "object"] #读取非数字的column  
 listNonObjedtColumn = [x for x in list(Data) if x not in listObjectColumn] #数字的column  
  
 ObjectData = Data[listObjectColumn]  
 NonObjectData = Data[listNonObjedtColumn]  
 #insert set into nonobject data with male = 0 and female = 1  
 NonObjectData.insert(0 ,"sex", (rawData["sex"] == " Female").astype(np.int))  
 #set every element in object rows as an attribute  
 ObjectData = pd.get\_dummies(ObjectData)  
  
 Data = pd.concat([NonObjectData, ObjectData], axis=1)  
 Data\_x = Data.astype("int64")  
 #normalize  
 Data\_x = (Data\_x - Data\_x.mean()) / Data\_x.std()  
  
 return Data\_x  
# 处理数据 y  
def dataProcess\_Y(rawData):  
 df\_y = rawData['income']  
 Data\_y = pd.DataFrame((df\_y==' >50K').astype("int64"), columns=["income"])  
 return Data\_y  
  
def sigmoid(z):  
 res = 1 / (1.0 + np.exp(-z))  
 return np.clip(res, 1e-8, (1-(1e-8)))  
  
#洗牌函数，洗乱数据集  
def \_shuffle(X, Y):  
 randomize = np.arange(X.shape[0])  
 np.random.shuffle(randomize)  
 return (X[randomize], Y[randomize])  
  
def split\_valid\_set(X, Y, percentage):  
 all\_size = X.shape[0]  
 valid\_size = int(floor(all\_size \* percentage))  
 X, Y = \_shuffle(X, Y)  
 X\_valid, Y\_valid = X[ : valid\_size], Y[ : valid\_size]  
 X\_train, Y\_train = X[valid\_size:], Y[valid\_size:]  
 return X\_train, Y\_train, X\_valid, Y\_valid  
  
def valid(X, Y, w):  
 a = np.dot(w,X.T)  
 y = sigmoid(a)  
 y\_ = np.around(y)  
 result = (np.squeeze(Y) == y\_)  
 acc = (float(result.sum()) / result.shape[0])  
 print('Valid acc = %f' % (float(result.sum()) / result.shape[0]))  
 return y\_ , acc  
  
def train(X\_train, Y\_train):  
 valid\_set\_percentage = 0.2  
 w = np.zeros(len(X\_train[0]))  
 l\_rate = 0.001  
 #0.001  
 batch\_size = 32  
 X\_train, Y\_train, X\_valid, Y\_valid = split\_valid\_set(X\_train, Y\_train, valid\_set\_percentage)  
 train\_dataz\_size = len(X\_train)  
 step\_num = int(floor(train\_dataz\_size / batch\_size))  
 epoch\_num = 300  
 list\_cost = []  
 list\_cost\_v = []  
  
 accs\_train = []  
 accs\_valid = []  
  
 for epoch in range(1, epoch\_num):  
 total\_loss = 0.0  
 total\_loss\_v = 0.0  
 #X\_train, Y\_train = \_shuffle(X\_train, Y\_train)  
  
 for idx in range(1, step\_num):  
 X = X\_train[idx\*batch\_size:(idx+1)\*batch\_size]  
 Y = Y\_train[idx\*batch\_size:(idx+1)\*batch\_size]  
 z = np.dot(X, w)  
 y = sigmoid(z)  
  
 grad = np.sum(-1 \* X \* (np.squeeze(Y) - y).reshape((batch\_size, 1)), axis=0)  
 w = w - l\_rate \* grad  
 cross\_entropy = -1 \* (  
 np.dot(np.squeeze(Y.T), np.log(y)) + np.dot((1 - np.squeeze(Y.T)), np.log(1 - y))) / len(Y)  
 total\_loss += cross\_entropy  
  
 z\_v = np.dot(X\_valid, w)  
 y\_v = sigmoid(z\_v)  
 total\_loss\_v += -1 \* (np.dot(np.squeeze(y\_v.T), np.log(y\_v)) + np.dot((1 - np.squeeze(y\_v.T)),  
 np.log(1 - y\_v))) / len(y\_v)  
 list\_cost.append(total\_loss)  
 list\_cost\_v.append(total\_loss\_v)  
  
 result = valid(X\_train, Y\_train, w)  
 result\_v = valid(X\_valid, Y\_valid, w)  
 accs\_train.append(result[1])  
 accs\_valid.append(result\_v[1])  
  
 drawLoss(list\_cost,list\_cost\_v)  
  
 drawAccs(accs\_train,accs\_valid)  
  
 return w  
  
def drawLoss(list\_cost,list\_cost\_v):  
 plt.figure()  
 plt.plot(np.arange(len(list\_cost)), list\_cost)  
 plt.plot(np.arange(len(list\_cost\_v)), list\_cost\_v)  
 plt.legend(['train','dev'])  
 plt.title("Train Process")  
 plt.xlabel("epoch\_num")  
 plt.ylabel("Cost Function (Cross Entropy)")  
 plt.savefig(os.path.join(os.path.dirname(output\_dir), "TrainProcess"))  
 plt.show()  
  
def drawAccs(accs\_train,accs\_valid):  
 plt.figure()  
 plt.plot(np.arange(len(accs\_train)), accs\_train)  
 plt.plot(np.arange(len(accs\_valid)), accs\_valid)  
 plt.legend(['train','dev'])  
 plt.title("Train Process")  
 plt.xlabel("epoch\_num")  
 plt.ylabel("Accuracy of Function ")  
 plt.savefig(os.path.join(os.path.dirname(output\_dir), "TrainProcess\_accuracy"))  
 plt.show()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 trainData = pd.read\_csv("data/train.csv")  
 testData = pd.read\_csv("data/test.csv")  
  
 # here is one more attribute in trainData  
 x\_train = dataProcess\_X(trainData).drop(['native\_country\_ Holand-Netherlands'], axis=1).values  
 x\_test = dataProcess\_X(testData).values  
 y\_train = dataProcess\_Y(trainData).values  
  
 x\_test = np.concatenate((np.ones((x\_test.shape[0], 1)), x\_test), axis=1)  
 x\_train = np.concatenate((np.ones((x\_train.shape[0], 1)),x\_train), axis=1)  
 w = train(x\_train, y\_train)  
 y\_ = valid(x\_test, y\_train, w)  
 df = pd.DataFrame({"id": np.arange(1, 16282), "label": y\_[0]})  
 if not os.path.exists(output\_dir):  
 os.mkdir(output\_dir)  
 df.to\_csv(os.path.join(output\_dir + 'LR\_predict.csv'), sep='\t', index=False)

# 实验四

一、问题描述：

学会搭建Keras开发环境，掌握基于TensorFlow的高级API框架Keras的基本用法，通过MNIST手写数字体数据集，学会搭建基于Keras API的神经网络，并用来识别手写数字体。

可以在Ubuntu18.04操作系统上搭建开发环境，所使用的开发工具包括Anaconda、Tensorflow、Keras，使用Python语言。因为Keras是基于Tensorflow的高层API,所以需要先安装Tensorflow再安装Keras。

（我使用的是自己电脑macos pycharm环境）

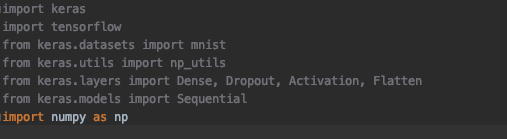
1. 设计简要描述：

安装tensorFlow、keras使用的是pip和老师给的实验指导书步骤不一样。之后都是按照实验指导书一样的步骤。每一步都有对应的截图：

2.3导入模块

本教程需要导入Keras里的模块。请学员编码实现从Keras的datasets模块中导入数据集mnist模块,从layer.core模块导入Dense和Activation模块,从optimizers 模块导入SGD模块，从utils模块导入np\_utils模块。

检测条件：导入模块没有报错



3.数据预处理

3.1加载训练集和测试集

请学员通过mnist模块的load\_data方法编程实现如下功能：把数据集中的训练数据和测试数据分别赋值给(X\_train,y\_train)和(X\_test,yest)两组变量，并通过shape方法查看四个变量的维度是否正确

执行结果如下：

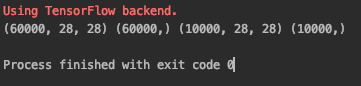
(60000,28,28)

(60000,)

(10000,28,28)

(10000)

检测条件：是否输出以上结果。



3.2重塑训练集和测试集的形状

X\_train和X\_test如上所示分别是60000×28×28和10000×28×28的形状的张量，请通过numpy模块的reshape方法编程实现将其形状重塑为60000×784，10000×784，并使用numpy模块的astype方法将X\_train和X\_test的元素数据类型改为float32。

然后请学员通过print方法结合X\_train/X\_test的shape和dtype属性编程实现查看形状和数据类型是否正确。

提示1：numpy模块的reshape方法形式如下：

a.reshape(shape) ： 不改变numpy数组a的元素，返回一个shape形状的数组，原数组不变

例：a = np.arange(20)

#原数组不变

In [1]: a.reshape([4,5])

Out[1]:

array([[ 0, 1, 2, 3, 4],

[ 5, 6, 7, 8, 9],

[10, 11, 12, 13, 14],

[15, 16, 17, 18, 19]])

提示2: numpy模块的数组对象的astype方法形式如下：

astype（dtype，order ='K'，cast ='unsafe'，subok = True，copy = True）

参数dytpe表示需要转化的数据类型，其他参数都有缺省值，这里用不到，可以不用管。

代码执行结果：

(60000,784)

(10000,784)

float32

float32

检测条件：是否输出以上结果



3.3 归一化

上面我们已经重塑了X\_train和X\_test的形状，但是mnist图片是通过矩阵存储在计算机里面，矩阵里的每一个元素都是一个像素点，每个像素都转换成了0～255的值，其中0代表白色，255代表黑色。为了方便神经网络的计算，我们需要对像素点进行归一化。

请学员通过编程把X\_train和X\_test里面的像素点归一化到0-1之间的值，并查看X\_train的第2个例子的第100个到150个像素点的值。

代码执行结果：

array([0. , 0. , 0. , 0. , 0. ,

0. , 0. , 0. , 0. , 0. ,

0. , 0. , 0. , 0. , 0. ,

0. , 0. , 0. , 0. , 0. ,

0. , 0. , 0. , 0. , 0. ,

0. , 0. , 0.2 , 0.62352943, 0.99215686,

0.62352943, 0.19607843, 0. , 0. , 0. ,

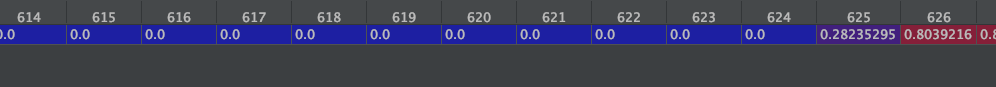
0. , 0. , 0. , 0. , 0. ,

0. , 0. , 0. , 0. , 0. ,

0. , 0. , 0. , 0. , 0. ,

0. ], dtype=float32)

检测条件：输出以上结果则正确。



3.4 one-hot编码

请学员基于Keras的utils模块的to\_categorical方法编程实现对Y\_train和Y\_test的one-hot编码，并打印Y\_train前五行看编码是否正确

提示：to\_categorical方法的形式如下：

to\_categorical(y, num\_classes=None)

其中，参数y表示要转换为one-hot编码的类向量，而参数num\_classes表示类别的总数。该方法返回的是输入的类向量中所有类别所对应的one-hot编码构成的矩阵。例如，以下代码将类别1和类别3分别转换成对应的one-hot编码。

示例代码：

ohl=keras.utils.to\_categorical([1,3],num\_classes=5)

print(ohl)

对应的输出结果

[[0. 1. 0. 0. 0.]

[0. 0. 0. 1. 0.]]

代码执行结果：

array([[0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],

[1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],

[0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],

[0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],

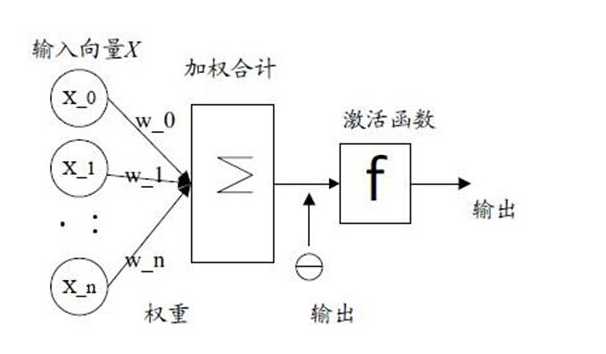
[0., 0., 0., 0., 0., 0., 0., 0., 0., 1.]], dtype=float32

检测条件：输出以上结果则正确

4搭建神经网络

4.1添加层

Keras的原始构造模块就是模型，最简单的模型称为序列模型，即Sequential模型。接下来，我们来通过序列模型构建最基本的神经网络----感知机。感知机的模型如下：



请学员编程实现如下功能：首先创建一个序列模型对象model；并调用model对象的add方法添加一个全连接层（全连接层在Keras中用Dense类来表示），其输出维度是10，输入维度是784；然后再通过add方法给输出层添加一个激活层（激活层在Keras中用Activation类来表示）得到最后输出；最后通过model的summary方法查看模型是否正确。

提示1：可以通过 Sequential类的构造方法来构建一个Keras序列模型对象。

Eg:model = Sequential()

提示2: 可通过如下add方法的形式来将一个个layer加入模型model中

model.add(layer)

其中Model是我们通过Sequential类构造的序列模型对象，然后调用add()方法添加层来搭建神经网络，其中layer可以是全连接层也可以是其它常用的神经网络层结构，层的具体定义后续会讲解。

提示3: Dense类的构造方法形式如下：

Dense(units,activation=None,use\_bias=True, kernel\_initializer='glorot\_uniform',bias\_initializer='zeros',kernel\_regularizer=None,bias\_regularizer=None,activity\_regularizer=None,kernel\_constraint=None,bias\_constraint=None)

虽然参数很多，但是我们实际中使用只需要注明几个显著的参数即可。

其中参数units表示该层的输出维度，为大于0的整数，其余的参数都有缺省值，在这里用不到。

use\_bias: 布尔值，是否使用偏置项

注意如果第一层就是全连接层需要使用参数input\_shape指定输入大小

提示4:Activation类的构造方法形式如下：：

Activation(activation)

其中参数activation表示用于神经网络的激活函数

代码执行结果:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

dense\_2 (Dense) (None, 10) 7850

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

activation\_2 (Activation) (None, 10) 0

=================================================================

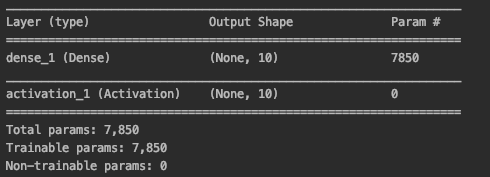
Total params: 7,850

Trainable params: 7,850

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

检测条件：输出以上内容则正确



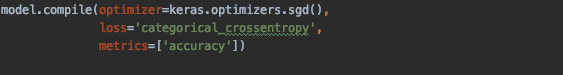
4.2 编译神经网络

在定义好神经网络模型后，需要对模型进行编译，已有经过编译，模型才能由Keras的后端（Theao或Tensorflow）来执行。

请学员编程实现以下功能：通过model对象的compile方法来对模型进行编译。

提示：compile方法有三个参数，其中参数loss表示损失函数，这里应该用categorical\_crossentropy作为损失函数；参数 optimizer表示所采用的优化算法，这里可以用SGD方法；参数metrics表示性能评估指标，这里可以用accuracy作为性能评估指标。

检测条件：需对代码进行判断



4.3 训练神经网络

一旦编译完模型，就可以用model对象的fit方法进行模型的训练。

请学员编程实现以下功能：用fit方法来训练神经网络。

提示：fit方法的形式如下：

fit(self, x=None, y=None, batch\_size=None, epochs=1, verbose=1, callbacks=None, validation\_split=0.0, validation\_data=None, shuffle=True, class\_weight=None, sample\_weight=None, initial\_epoch=0, steps\_per\_epoch=None, validation\_steps=None)

虽然参数很多，但是我们实际中使用只需要注明几个显著的参数即可。

x：输入数据。如果模型只有一个输入，那么x的类型是numpy数组，如果模型有多个输入，那么x的类型应当为list，list的元素是对应于各个输入的numpy array。

y：标签，numpy 数组。如果模型有多个输出，可以传入一个numpy 数组的list。

batch\_size：整数，指定进行梯度下降时每个batch包含的样本数。训练时一个batch的样本会被计算一次梯度下降，使目标函数优化一步。

epochs：整数，训练终止时的epoch值，训练将在达到该epoch值时停止

verbose：日志显示，0为不在标准输出流输出日志信息，1为输出进度条记录，2为每个epoch输出一行记录

validation\_split：0~1之间的浮点数，用来指定训练集的一定比例数据作为验证集。验证集将不参与训练，并在每个epoch结束后测试的模型的指标，如损失函数、精确度等。

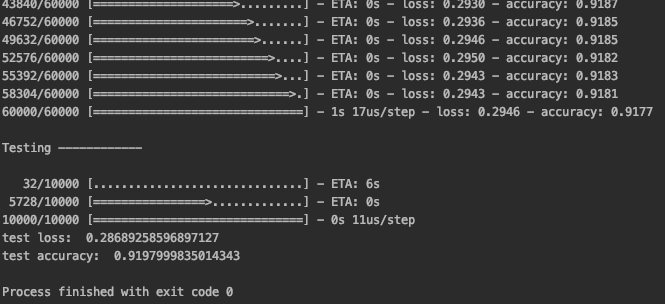
其中参数x,y分别对应X\_train, Y\_train,参数batch\_sze可设置为128，参数epochs可设置为200，参数verbose设置为1，参数validation\_split设置为0.2

代码执行结果：

Epoch 200/200

48000/48000 [==============================] - 1s 15us/step - loss: 0.2761 - acc: 0.9230 - val\_loss: 0.2756 - val\_acc: 0.9241

检测条件：输出的结果准确率acc和loss与上面接近即正确。



4.4 评估神经网络

一旦模型训练完成，我们就可以在全新的样本测试集上进行评估，来观察模型的好坏。

请学员编程实现以下功能：

1. 使用model对象的evaluate方法来评估模型，并将evaluate的返回值保存到变量score中
2. 然后打印score的第一项和第二项值，查看总的损失值和准确率。

提示：evaluate()定义形式如下：

evaluate(self, x, y, batch\_size=32, verbose=1, sample\_weight=None)

x：输入数据

y：标签

batch\_size：整数，含义同fit的同名参数

verbose：含义同fit的同名参数，但只能取0或1

evaluate方法前2个参数是测试集特征和标签（可以分别用X\_test和Y\_test），verbose,可设置为1，其它参数可以不用考虑。

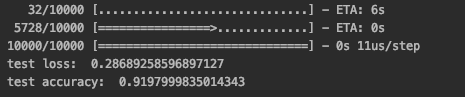
代码执行结果：

10000/10000 [==============================] - 0s 24us/step

Test score 0.27740466350317

Test accuracy 0.9213

检测条件：总的损失和准确率接近即正确。



5优化神经网络

5.1 模型改进

之前构建了一个最简单的神经网络。实际上，构建神经网络的时候可以同时添加层和激活函数，如下所示：

model.add(Dense(10,input\_shape = (784,))，activation = ’softmax’)

请学员编程实现以下功能：在上述最基本的神经网络基础上通过add方法加一个隐藏层，隐藏层的输出维度是128，激活函数是’relu’,同时输出层不再是一个单一的激活函数，而是用Dense定义全连接输出层，其中输出维度是10,激活函数是’softmax’，注意，此时我们的输入层的的输出的激活函数是’relu’.然后通过summary.model()方法查看构建是否正确

代码执行结果：

Layer (type) Output Shape Param #

=================================================================

dense\_3 (Dense) (None, 128) 100480

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_4 (Dense) (None, 128) 16512

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

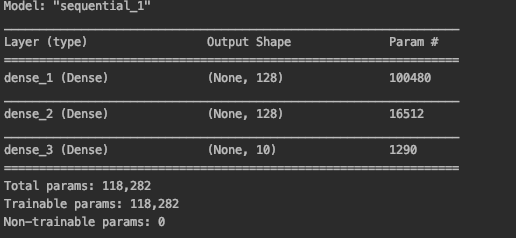
dense\_5 (Dense) (None, 10) 1290

=================================================================

Total params: 118,282

Trainable params: 118,282

Non-trainable params: 0



检测条件：得到如上左右结果则代表正确：

5.2 编译神经网络

请学员编程实现以下功能：通过model对象的compile方法来对模型编译，其中参数loss用categorical\_crossentropy，参数optimizer用SGD，参数metrics选用’accuracy’

检测条件：需对代码进行判断

5.3 训练神经网络

请学员编程实现以下功能：用fit方法来训练神经网络，其中batch\_sze设置为128，epochs设置为20，verbose设置为1，validation\_split设置为0.2。

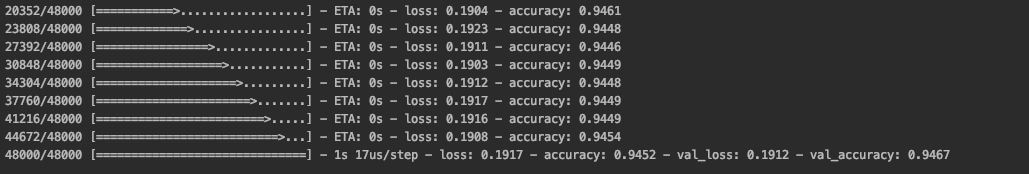
代码执行结果：

Epoch 20/20

48000/48000 [==============================] - 2s 32us/step - loss: 0.1906 - acc: 0.9453 - val\_loss: 0.1885 - val\_acc: 0.9476

10000/10000 [==============================] - 0s 30us/step

检测条件：输出的结果准确率acc和loss与上面接近即正确。



5.4 评估神经网络

请学员编程实现以下功能：使用evaluate函数来评估模型。

其中参数x,y分别对应测试集的数据和标签，verbose设置为1，其余参数可以忽略。并将evaluate的返回值赋值给score，然后打印score的第一项和第二项值，查看总的损失值和准确率。

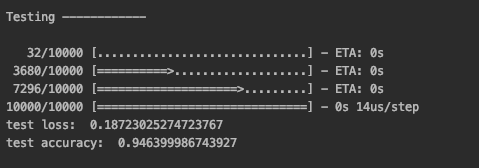
代码执行结果：

10000/10000 [==============================] - 0s 24us/step

Test score: 0.18817616567984224

Test accuracy: 0.9452

检测条件：输出的结果准确率acc和loss与上面接近即正确。



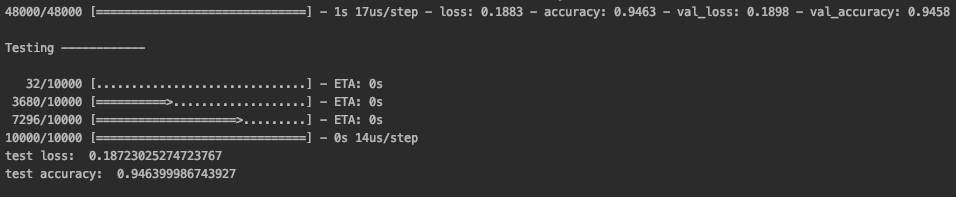
5.5总结：我们通过增加2个隐藏层，让准确率提升了2.2%，然而我们的迭代次数从200显著减少到了20，说明了我们的神经网络得到了很好的改善。

三、程序清单：

import keras  
from keras.datasets import mnist  
from keras.utils import np\_utils  
from keras.layers import Dense, Dropout, Activation, Flatten  
from keras.models import Sequential  
import matplotlib.pyplot as plt  
import numpy as np  
  
def load\_data(path='mnist.npz'):  
 f = np.load(path)  
 x\_train, y\_train = f['x\_train'], f['y\_train']  
 x\_test, y\_test = f['x\_test'], f['y\_test']  
 f.close()  
 return (x\_train, y\_train), (x\_test, y\_test)  
# 将数据读入  
(x\_train, y\_train), (x\_test, y\_test) = load\_data()  
  
print(x\_train.shape,y\_train.shape,x\_test.shape,y\_test.shape)  
  
# plot 4 images as gray scale  
# plt.subplot(221)  
# print(y\_train[4545],y\_train[1],y\_train[2],y\_train[3])  
# plt.imshow(x\_train[4545], cmap=plt.get\_cmap('gray'))  
# plt.subplot(222)  
# plt.imshow(x\_train[1], cmap=plt.get\_cmap('gray'))  
# plt.subplot(223)  
# plt.imshow(x\_train[2], cmap=plt.get\_cmap('gray'))  
# plt.subplot(224)  
# plt.imshow(x\_train[3], cmap=plt.get\_cmap('gray'))  
# # show the plot  
# plt.show()  
X\_train = x\_train.reshape(x\_train.shape[0], 784)  
X\_test = x\_test.reshape(x\_test.shape[0],784)  
X\_train = X\_train.astype('float32')  
X\_test = X\_test.astype('float32')  
  
# print(X\_train.shape,X\_test.shape)  
# print(X\_train.dtype,X\_test.dtype)  
# 归一化  
X\_train /= 255  
X\_test /= 255  
  
# 对于y 独热编码  
Y\_train = np\_utils.to\_categorical(y\_train, 10)  
Y\_test = np\_utils.to\_categorical(y\_test, 10)  
# 初始化序列模型：  
model = Sequential()  
#model.add(Dense(10,input\_dim=784))  
#model.add(Activation('softmax'))  
model.add(Dense(128,input\_dim=784,activation='relu'))  
model.add(Dense(128,activation='relu'))  
model.add(Dense(10,activation='softmax'))  
#model.summary()  
model.compile(optimizer=keras.optimizers.sgd(),  
 loss='categorical\_crossentropy',  
 metrics=['accuracy'])  
  
# 训练模型  
model.fit(X\_train, Y\_train, nb\_epoch=20, batch\_size=128,verbose=1,validation\_split=0.2)  
  
model.save('mnist\_kerasModel.h5')  
#测试模型  
print('\nTesting ------------')  
loss, accuracy = model.evaluate(X\_test, Y\_test,verbose=1)  
print('test loss: ', loss)  
print('test accuracy: ', accuracy)

四、结果分析

本次实验的结果输出如下：



最后得到的结果和老师给的实验指导书的结果相差无几。甚至有时候还会好一点。可以说是得到了正确的结果。

# 实验五

一、问题描述：

掌握基于TensorFlow的高级API框架Keras的基本用法，仍然通过MNIST手写数字体数据集，学会搭建给予Keras API的卷积神经网络，并用来识别手写数字体。

可以在Ubuntu18.04操作系统上搭建开发环境，所使用的开发工具包括Anaconda、Tensorflow、Keras，使用Python语言。因为Keras是基于Tensorflow的高层API,所以需要先安装Tensorflow再安装Keras。

（我使用的是自己电脑macos pycharm环境）

1. 设计简要描述：

实验内容：

1．1．准备工作

1.1． 导入包和数据集

首先请导入包，该步骤参考实验一，

1.1.1 安装h5py包

在本实验是在Ubuntu18.04下完成的，其中包h5py需要自己自行安装。操作步骤如下:

首先Ctrl+Alt+T打开终端，然后输入pip install h5py完成安装。

检测条件：环境终端下输入pip list(或者conda list)查看已安装的包有h5py则安装成功。

1.1.2 导入相关包

%matplotlib inline

import matplotlib.pyplot as plt

import tensorflow as tf

import numpy as np

import math

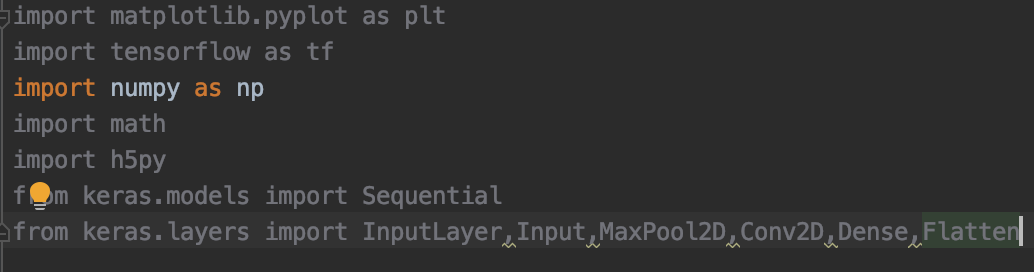
import h5py

本教程需要从Tensorflow导入一些Keras的包。请从keras的models模块中导入Sequential,从layers模块中导入InputLayer,Input,Reshape,MaxPooing2D,Conv2D,Dense,Flatten

提示：可参考如下导入方式：

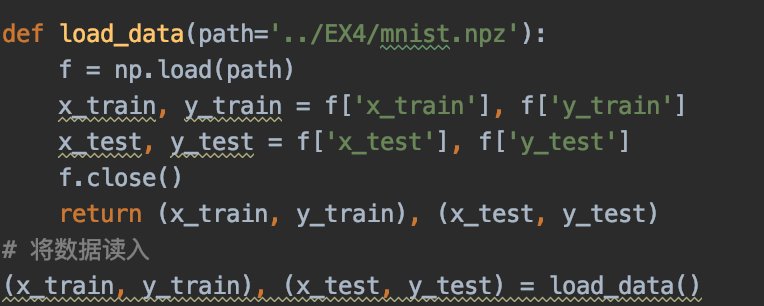
from tensorflow.python.keras.A import B（A,B代表2个模块)

检测条件：导入包没有报错信息



1.2. 载入数据

请加载MNIST数据集（该数据集大约有12MB，如果给定的地址里没有文件，它将自动下载）



1.3．配置神经网络

配置神经网络参数，包括定义图像中每个维度中的像素个数（img\_size）、图像的像素总个数（img\_size\_flat）、用来重塑图像的高度和宽度的元组（img\_shape表示）、用来重塑图像的高度，宽度和深度的元组（img\_shape\_full表示，用于在Keras重塑）、类别数量（num\_classes）、图像通道数（num\_channels）。请打印并查看上述定义和赋值的变量。

参考代码：

# 图像的每个维度中的像素数。

img\_size = data.img\_size

# 图像存储在一维矩阵中的总长度

img\_size\_flat = data.img\_size\_flat

# 用来重塑图像的高度和宽度的元组。

img\_shape = data.img\_shape

# 用来重塑图像的高度，宽度和深度的元组。

# 这用于在Keras重塑。.

img\_shape\_full = data.img\_shape\_full

#类别数量

num\_classes = data.num\_classes

# 图像的通道数

~~num\_channels = data.num\_channel~~

#打印显示变量

print(img\_size)

print(img\_size\_flat)

print(img\_shape)

print(img\_shape\_full)

print(num\_classes)

print(num\_channels)

代码执行结果：

28

784

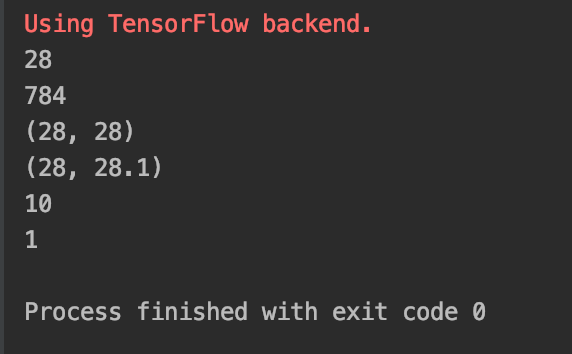
(28, 28)

(28, 28,1)

10

1

检测条件：是否输出以上结果



1.4.绘制图像的辅助函数

实现的绘制图像的辅助函数plot\_images并打印测试集前9张图片来检测该函数是否正确

参考代码：

def plot\_images(images, cls\_true, cls\_pred=None):

assert len(images) == len(cls\_true) == 9

# 创建3x3子图.

fig, axes = plt.subplots(3, 3)

fig.subplots\_adjust(hspace=0.3, wspace=0.3)

for i, ax in enumerate(axes.flat):

# 画图.

ax.imshow(images[i].reshape(img\_shape), cmap='binary')

# 显示真正的预测的类别.

if cls\_pred is None:

xlabel = "True: {0}".format(cls\_true[i])

else:

xlabel = "True: {0}, Pred: {1}".format(cls\_true[i], cls\_pred[i])

#将类别作为x轴的标签

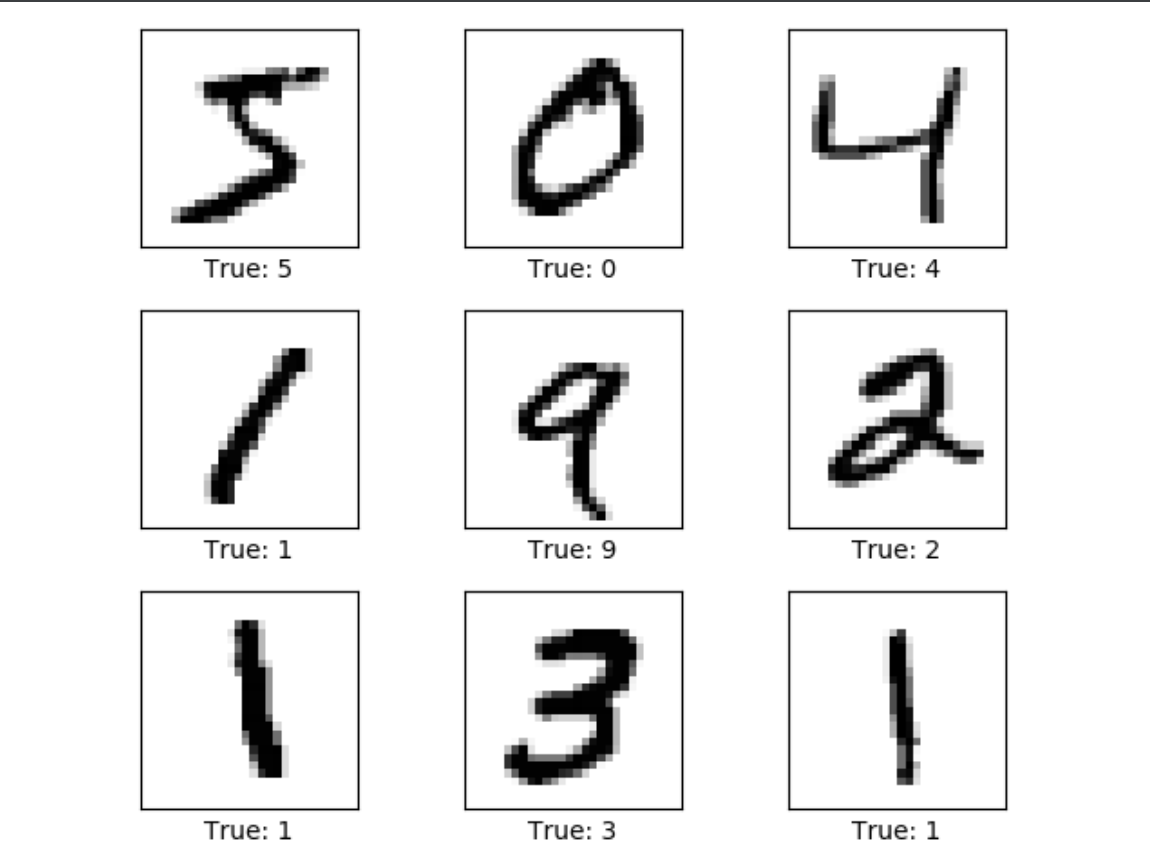
ax.set\_xlabel(xlabel)

# 去除图中的刻度线.

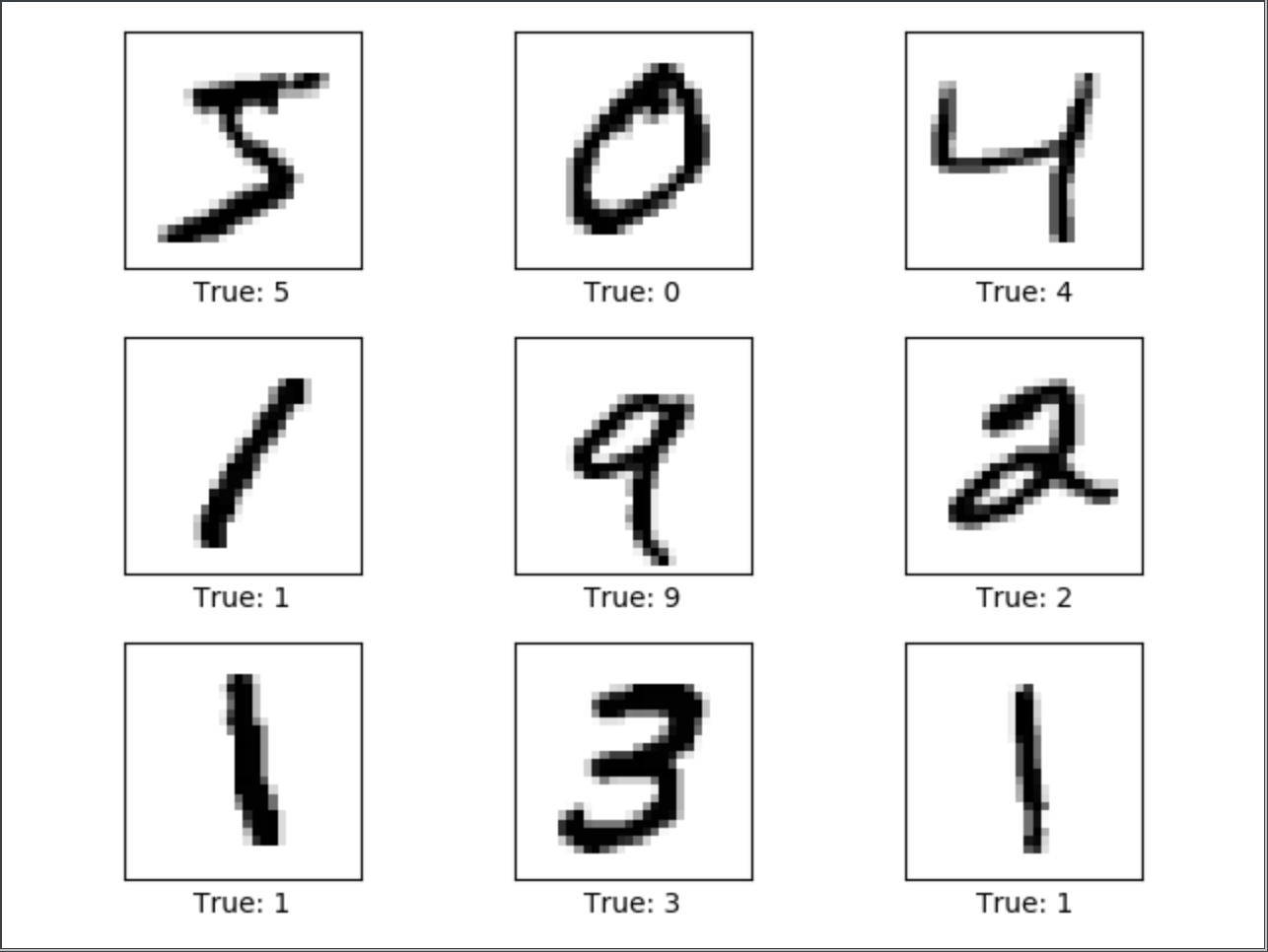
ax.set\_xticks([])

ax.set\_yticks([])

plt.show()



检测条件：输出结果如上即可。



1.5.绘制错误分类图像的辅助函数

参考代码：

绘制图像的辅助函数plot\_example\_errors

def plot\_example\_errors(cls\_pred, correct):

# 此函数从下面的print\_test\_accuracy()调用

# cls\_pred包含了测试集所有图像的预测类别

# correct是布尔数组，表示预测的类是否等于测试集中每个图像的真是类别.

# 布尔数组中未正确分类的图像

incorrect = (correct == False)

# 从测试集中获取未正确分类的图片

images = data.x\_test[incorrect]

# 获取这些图像的预测列表

cls\_pred = cls\_pred[incorrect]

# 获取这些图像的正确类别

cls\_true = data.y\_test\_cls[incorrect]

# 画出前9张图像

plot\_images(images=images[0:9],

cls\_true=cls\_true[0:9],）

cls\_pred=cls\_pred[0:9])

检测条件：需要对代码进行判断



2.序列模型

2.1模型框架

Keras API有两种构建神经网络的模式。最简单的是顺序模型，它只允许按顺序添加图层。如下是Keras配置序列模型介绍，提供给学员学习，不用实现代码。

参考代码：

# 开始构建Keras 序列模型。

model = Sequential()

# 添加一个输入层，类似于TensorFlow中的feed\_dict。

#输入形状input-shape 必须是包含图像大小image-size\_flat的元组。

model.add(InputLayer(input\_shape=(img\_size\_flat,)))

# 输入是一个包含784个元素的扁平数组，

# 但卷积层期望图像形状是（28,28,1）

model.add(Reshape(img\_shape\_full))

# 具有ReLU激活和最大池化的第一个卷积层。

model.add(Conv2D(kernel\_size=5, strides=1, filters=16, padding='same',

activation='relu', name='layer\_conv1'))

model.add(MaxPooling2D(pool\_size=2, strides=2))

# 具有ReLU激活和最大池化的第二个卷积层

model.add(Conv2D(kernel\_size=5, strides=1, filters=36, padding='same',

activation='relu', name='layer\_conv2'))

model.add(MaxPooling2D(pool\_size=2, strides=2))

# 将卷积层的4级输出展平为2级，可以输入到完全连接/密集层。

model.add(Flatten())

# 具有ReLU激活的第一个完全连接/密集层。

model.add(Dense(128, activation='relu'))

# 最后一个完全连接/密集层，具有softmax激活功能，用于分类

model.add(Dense(num\_classes, activation='softmax'))

2.2模型编译

神经网络现已定义，必须通过添加损失函数，优化器和性能指标来最终确定。这在Keras中被称为模型“编译”。我们可以使用字符串定义优化器，或者如果我们想要更多地控制其参数，那么我们需要实例化一个对象。 例如，我们可以设置学习率。

请从tensorflow.python.keras.optimizers包中导入Adam，并设置优化器optimizer的学习率为1e-3，然后编译模型，将参数optimizer设置为optimizer， loss设置为'categorical\_crossentropy'，矩阵 metrics设置为'accuracy'

检测条件：需对代码进行判断

model.compile(optimizer=adam(lr=0.001),  
 loss='categorical\_crossentropy',  
 metrics=['accuracy'])

2.3 训练

现在模型已经完全定义了损失函数和优化器，我们可以使用序列模型的fit（）函数进行训练，此函数采用numpy数组作为输入。并使用给定的批量大小执行给定数量的训练epoch。 一个epoch是对整个训练集的充分利用。因此，对于10个epoch，我们将在整个训练集上随机迭代10次。

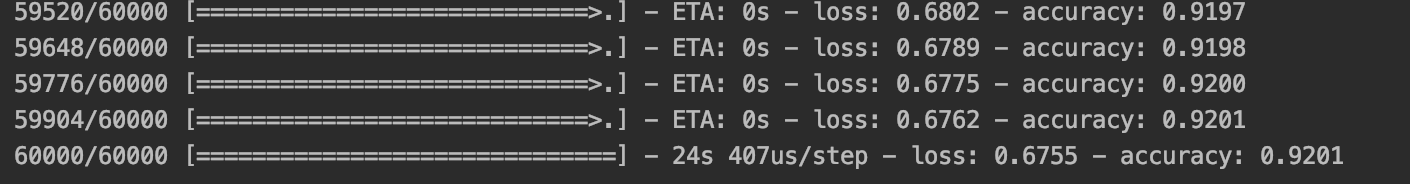
现在请拟合模型：epochs设置为1，batch\_size设置为128

代码执行结果：

Epoch 1/1

55000/55000 [==============================] - 27s 495us/step - loss: 0.2315 - acc: 0.9302

检测条件：总次数是55000且loss和acc数值接近即正确。



2.4评估与性能指标

现在模型已经过训练，我们可以在测试集上测试它的性能。在这里，也使用numpy数组作为输入。

请用model.evaluate方法来评估其在测试集上的性能，并将结果返回给result，然后查看result的性能指标。值得注意的是，result的返回值可以通过键值对的形式通过zip（key，value）函数访问，其中参数key可传入model的成员变量metric\_name作为键,而参数value可传入result作为值。提示:可参考如下列表解析式。

for a, b in zip(c, d):

print(a, b)

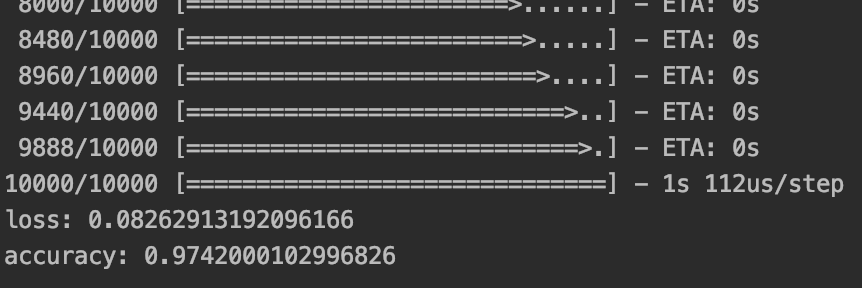
代码输出结果：

10000/10000 [==============================] - 3s 288us/step

loss 0.057774256201833483

acc 0.9797

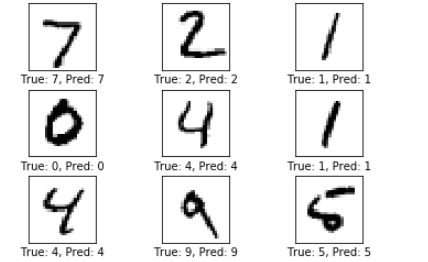
检测条件： 输出结果类似即可



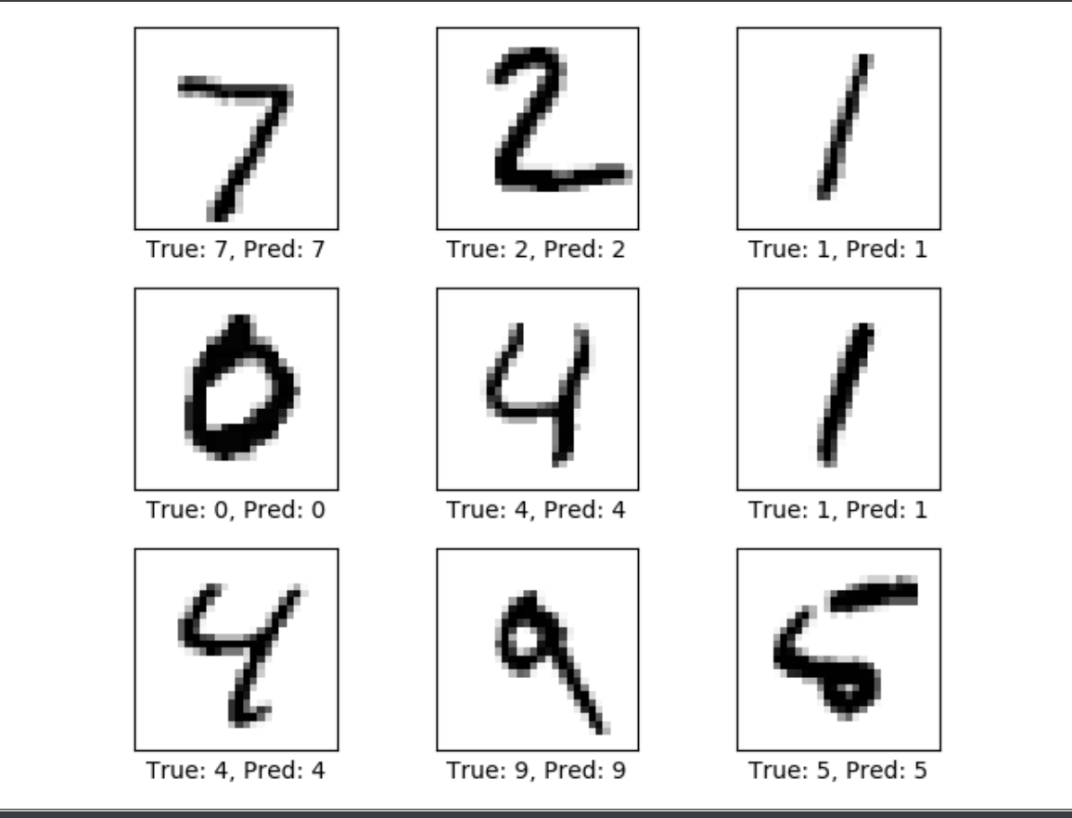
2.5 预测

我们还可以预测新图像的分类。 在这里，我们将只使用测试集中的一部分图像，即测试集的前九张图像。请对比一下预测的类别和真实的类别。提示，可以使用model类的predict（x）方法。该方法返回模型所预测的输入的图像x的类别。获取预测的结果后，请将预测的结果用plot\_images(images,cls\_true,cls\_pred)方法显示出来。

代码执行结果：



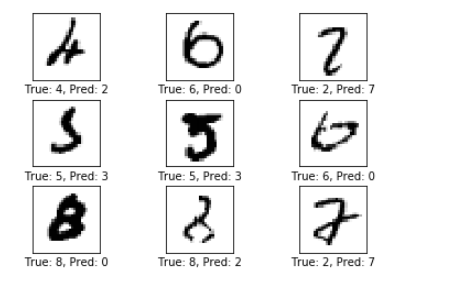
检测条件：输出上述结果即正确



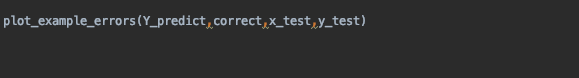
2.6 错分类的图片

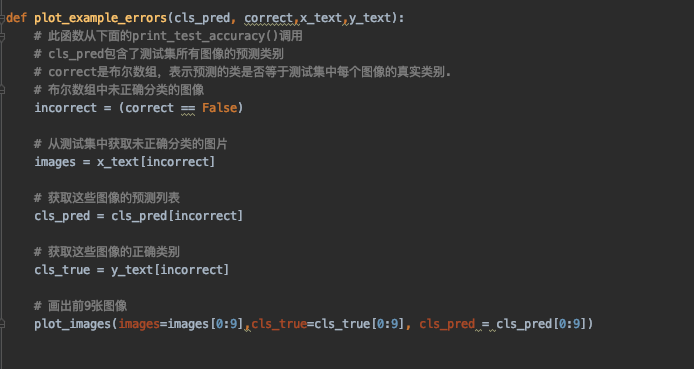
我们可以从测试集中绘制错误分类图像的一些示例。请从测试集中得到所有图像的预测类，参数返回给y\_pred，然后将预测的类数从One-Hot编码数组转换为整数赋值给cls\_pred，最后调用plot\_example\_errors绘制y\_pred的前九张错分类的图像

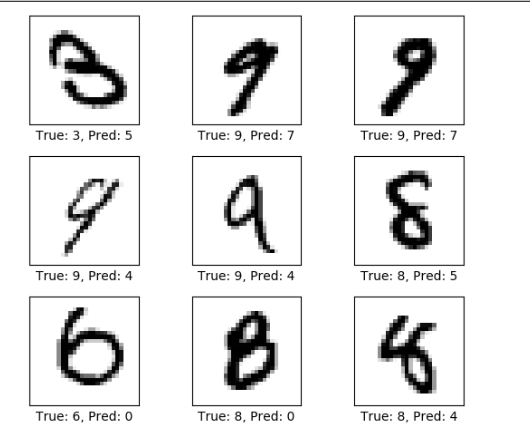
代码输出结果：



检测条件：需对代码进行判断，（深度学习本身学习机制导致同样的方式训练同样的次数结果也会有一点小小的不同，因为分类出来图片的随机性，所以图片可能每一次的结果都不一样，以下也是同样道理）







1. 功能模型

Keras API还可用于使用功能模型构建更复杂的网络。具体做法就是每次调用Keras 创建神经网络的API后都会创建并返回一个可被其他API作为参数调用的实例。这允许我们构建比序列模型所允许的模型更复杂的计算图。

* 1. 模型框架

如下是Keras配置功能模型介绍。

参考代码：

# 创建一个输入层，类似于TensorFlow中的feed\_dict。.

#输入形状input-shape 必须是包含图像大小image\_size\_flat的元组。.

inputs = Input(shape=(img\_size\_flat,))

#用于构建神经网络的变量。

net = inputs

# 输入是一个包含784个元素的扁平数组

# 但卷积层期望图像形状是（28,28,1）

net = Reshape(img\_shape\_full)(net)

# 具有ReLU激活和最大池化的第一个卷积层。

net = Conv2D(kernel\_size=5, strides=1, filters=16, padding='same',

activation='relu', name='layer\_conv1')(net)

net = MaxPooling2D(pool\_size=2, strides=2)(net)

# 具有ReLU激活和最大池化的第二个卷积层.

net = Conv2D(kernel\_size=5, strides=1, filters=36, padding='same',

activation='relu', name='layer\_conv2')(net)

net = MaxPooling2D(pool\_size=2, strides=2)(net)

# 将卷积层的4级输出展平为2级，可以输入到完全连接/密集层。

net = Flatten()(net)

# 具有ReLU激活的第一个完全连接/密集层。

net = Dense(128, activation='relu')(net)

# 最后一个完全连接/密集层，具有softmax激活功能，用于分类

net = Dense(num\_classes, activation='softmax')(net)

#神经网络输出

outputs = net

* 1. 模型编译

我们现在已经使用其输入和输出定义了模型的体系结构。我们现在必须创建一个Keras模型并使用损失函数和优化器对其进行编译，以便为训练做好准备。首先需用以下代码来完成导入model这个包（不要求学员实现）：

from tensorflow.python.keras.models import Model

请学员自行创建新的模型，并把模型赋值给model2,并参考2.2节完成model.compile的配置，其中optimizer选择rmsprop，其余不变。

检测条件：需对代码进行判断

def KerasAPI(x\_train,y\_train):  
 # 初始化数据，  
 x\_train = x\_train.reshape(60000, 28, 28, 1)  
 y\_train = to\_categorical(y\_train,10)  
 # 创建一个输入层，类似于TensorFlow中的feed\_dict。.  
 # 输入形状input-shape 必须是包含图像大小image\_size\_flat的元组。.  
 inputs = Input(shape=((28,28,1)))  
  
 # 用于构建神经网络的变量。  
 net = inputs  
  
 # 输入是一个包含784个元素的扁平数组  
 # 但卷积层期望图像形状是（28,28,1）  
 # net = Reshape(img\_shape\_full)(net)  
 # 具有ReLU激活和最大池化的第一个卷积层。  
 net = Conv2D(kernel\_size=5, strides=1, filters=16, padding='same',  
 activation='relu', name='layer\_conv1')(net)  
 net = MaxPooling2D(pool\_size=2, strides=2)(net)  
  
 # 具有ReLU激活和最大池化的第二个卷积层.  
 net = Conv2D(kernel\_size=5, strides=1, filters=36, padding='same',  
 activation='relu', name='layer\_conv2')(net)  
 net = MaxPooling2D(pool\_size=2, strides=2)(net)  
  
 # 将卷积层的4级输出展平为2级，可以输入到完全连接/密集层。  
 net = Flatten()(net)  
  
 # 具有ReLU激活的第一个完全连接/密集层。  
 net = Dense(128, activation='relu')(net)  
  
 # 最后一个完全连接/密集层，具有softmax激活功能，用于分类  
 net = Dense(num\_classes, activation='softmax')(net)  
  
 # 神经网络输出  
 outputs = net  
  
 model = Model(inputs=inputs, outputs=outputs)  
 model.compile(optimizer=tf.train.RMSPropOptimizer(0.001),  
 loss='categorical\_crossentropy',  
 metrics=['accuracy'])

### 训练

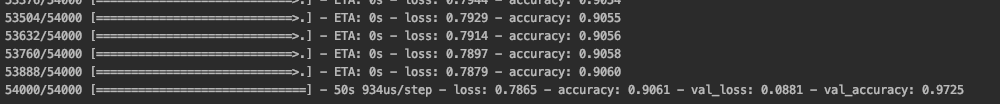
现在模型已经完全定义了损失函数和优化器，请使用和序列模型相同的fit（）函数进行训练，此函数采用numpy数组作为输入。其中epoches=1,batch\_siez = 128。

代码执行结果：

Epoch 1/1

55000/55000 [==============================] - 30s 552us/step - loss: 0.1918 - acc: 0.9399

检测条件：输出loss和acc与上类似即可，且总次数是55000.



* 1. 评估

现在模型已经过训练，我们可以在测试集上测试它的性能。 这也使用numpy数组作为输入。请参照2.4请用model.evaluate方法来评估其在测试集上的性能，并将结果返回给result，然后查看result的性能指标，result的返回值分别对应的名称是model.metrics\_names。最后，请查看直接打印出准确率的结果：

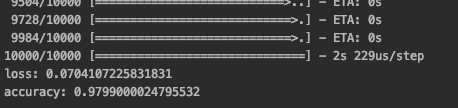
代码输出结果：

10000/10000 [==============================] - 3s 273us/step

loss 0.0590884289749898

acc 0.9807

检测条件：得到如上左右结果则代表正确：

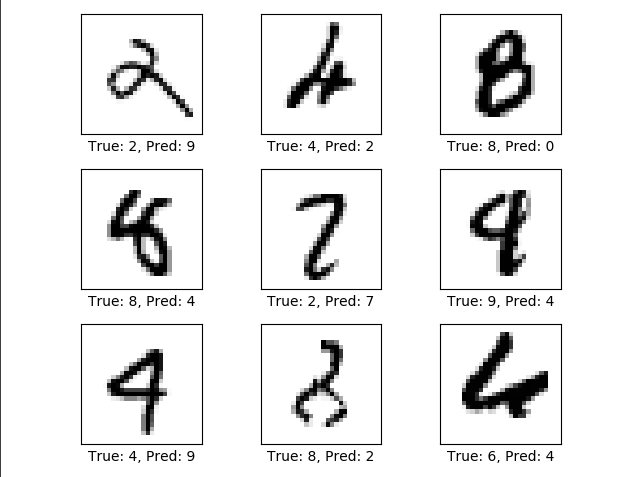


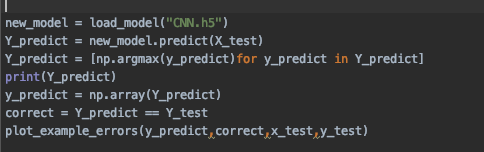
* 1. 错分类的图片

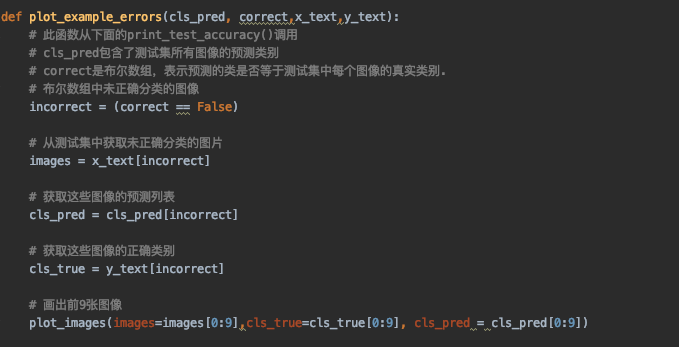
参照2.5和2.6绘制出错分类的前九张图像。

代码执行结果：

检测条件：对代码进行判断。因为深度学习机制所以每次预测的结果不一定相同，上一步的acc和loss类似的话此步也相应正确。







1. 保存和加载模型

4.1保存Keras模型

首先设置模型的保存路径（可自行设置）。实例代码：

path\_model = 'model.keras'

可以简单的使用model.save（）来保存模型，参数就是上一步的路径变量。请保存上述的模型model2到你自己设置的路径中（假设路径参数就是path\_model）

参考代码：

model2.save(path\_model)

检测条件：需对代码进行判断



4.2删除模型

Keras可使用model. del（）来删除模型。请用del来删除model2

参考代码：

del model2

检测条件：需对代码进行判断



4.3加载模型

我们需要导入如下包来加载模型：

from tensorflow.python.keras.models import load\_model

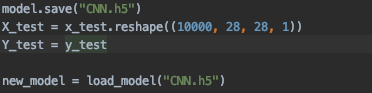
请加载上述保存的模型然后赋值给model3

参考代码：

model3 = load\_model(path\_model)

检测条件：需对代码进行判断



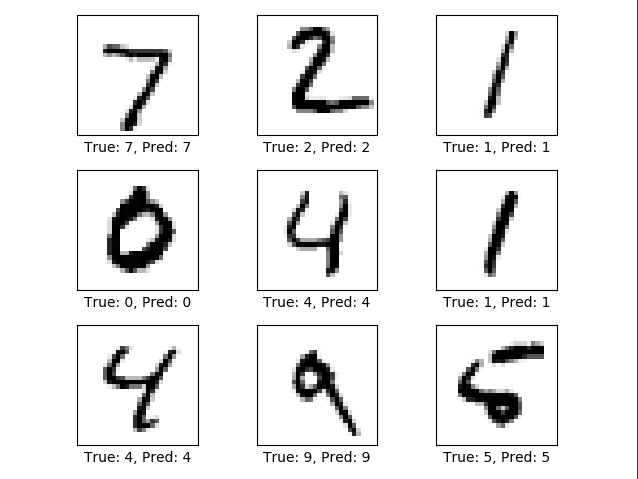


4.4用加载的模型来预测测试集的前9张图片

请用model3来完成测试集前九张图片的预测：

代码执行结果：

检测条件：输出如上结果则正确



5权重和输出的可视化

5.1画卷积权重的辅助函数：

请参考实验3的相应部分来完成绘制卷积权重的辅助函数plot\_conv\_weights(weights, input\_channel=0) ，输入参数weights是待可视化的权重，input\_channel是输入通道的数量，其缺省值为0。

参考代码：

def plot\_conv\_weights(weights, input\_channel=0):

#获取权重的最低值和最高值

# 这用于校正图像的颜色强度，以便可以相互比较.

w\_min = np.min(weights)

w\_max = np.max(weights)

# 卷积层中的卷积核数量

num\_filters = weights.shape[3]

#要绘制的网格数.

# 卷积核的平方根.

num\_grids = math.ceil(math.sqrt(num\_filters))

#创建带有网格子图的图像.

fig, axes = plt.subplots(num\_grids, num\_grids)

# 绘制所有卷积核的权重

for i, ax in enumerate(axes.flat):

# 仅绘制有限的卷积核权重

if i<num\_filters:

# 获取输入通道的第i个卷积核的权重

# 有关于４维张量格式的详细信息请参阅new\_conv\_layer()

img = weights[:, :, input\_channel, i]

# 画图

ax.imshow(img, vmin=w\_min, vmax=w\_max,

interpolation='nearest', cmap='seismic')

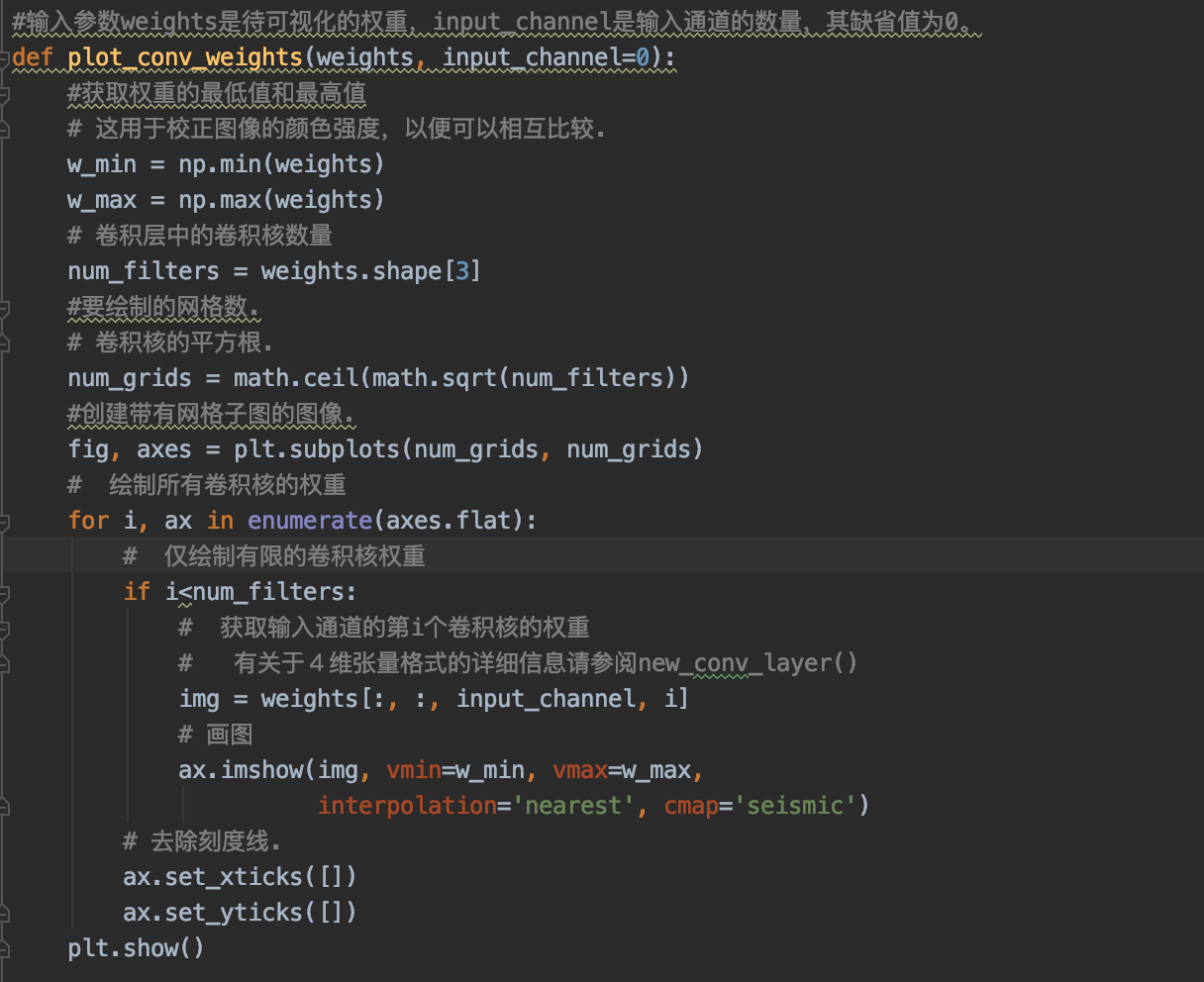
# 去除刻度线.

ax.set\_xticks([])

ax.set\_yticks([])

plt.show()

检测条件：需对代码进行判断



5.2得到层

Keras有一种简单的方法来列出模型中的图层。即调用summary()方法。请调用summary方法查看model3的所有层的信息，并把输入层赋值给layer\_input，第一层卷积层赋值给layer\_conv1，第二层卷积层辅助给layer\_conv2（提示:summary返回的是一个列表，可以通过列表索引完成）

参考代码：

model3.summary()

layer\_input = model3.layers[0]

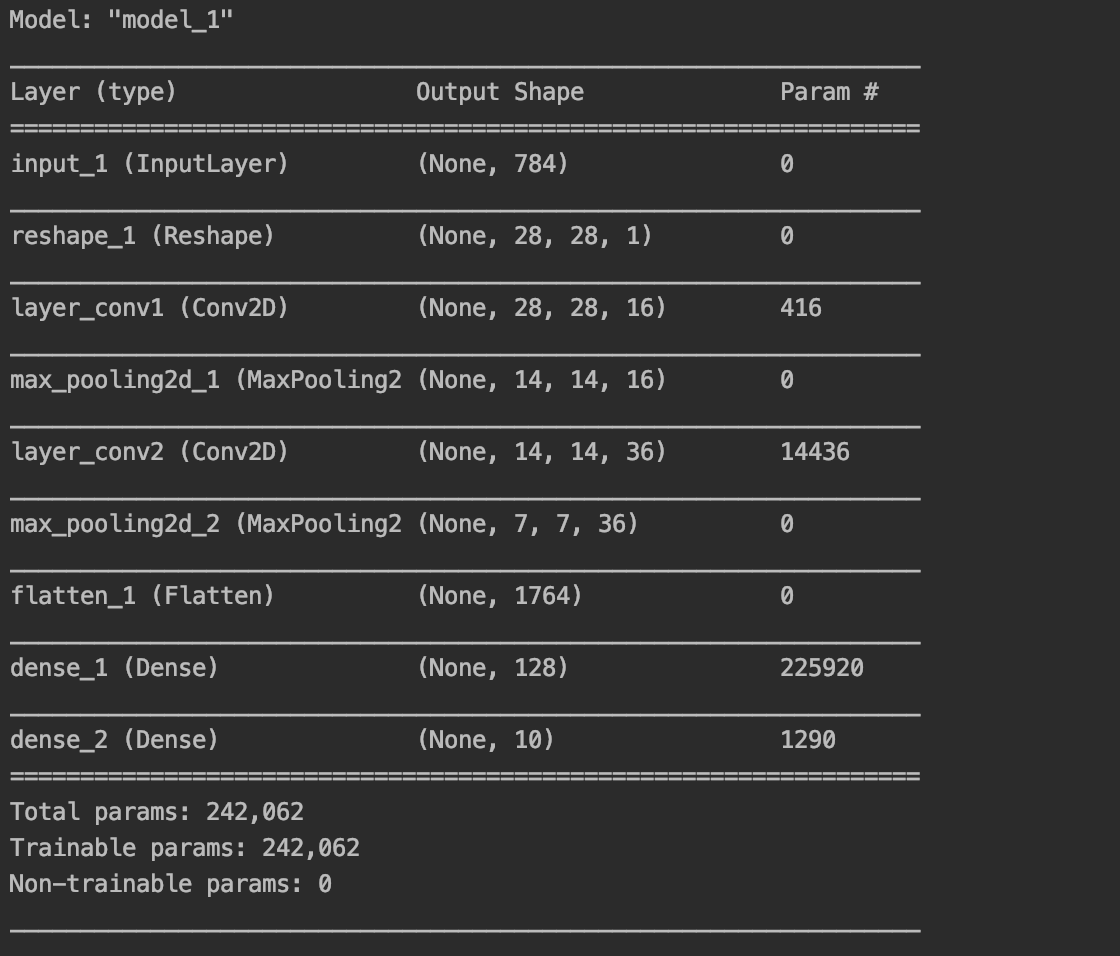
layer\_conv1 = model3.layers[2]

layer\_conv2 = model3.layers[4]

代码执行结果：

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

检测条件：输出结果如上所示即可



5.3卷积权重

有层了之后我们可以通过layer.get\_weights()获取各层权重，然后再通过plot\_conv\_weights函数来绘制权重图。请学员通过layer.get\_weights()函数获取卷积层1和卷积层2的权重并调用plot\_conv\_weights函数来绘制权重矩阵：

参考代码：

weights\_conv1 = layer\_conv1.get\_weights()[0]

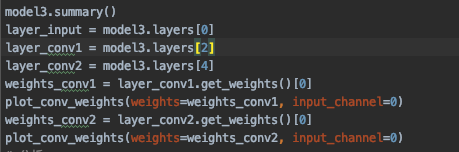
plot\_conv\_weights(weights=weights\_conv1, input\_channel=0)

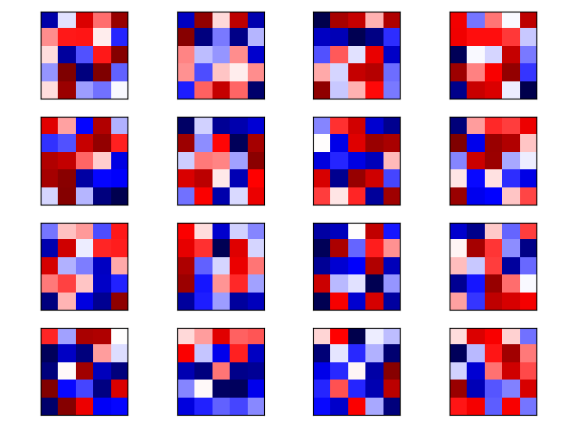
weights\_conv2 = layer\_conv2.get\_weights()[0]

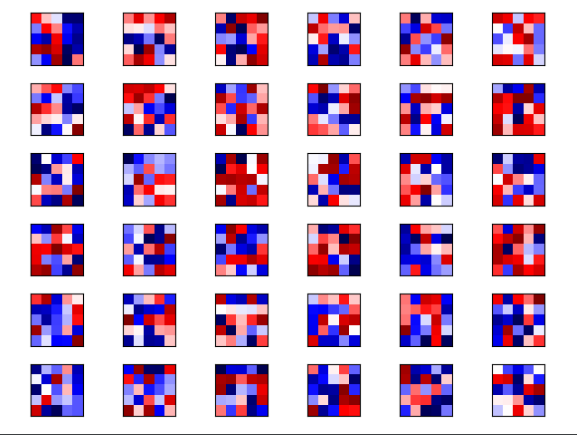
plot\_conv\_weights(weights=weights\_conv2, input\_channel=0)

代码执行结果：

检测条件：需对代码进行判断，且图类似即可。







5.4绘制出卷积层输出的帮助函数

请参考实验3 相应部分（4.2）完成绘制卷积层输出的帮助函数plot\_conv\_output(values)，其中参数values就代表了卷积层 （比如卷积层1传递给value来完成函数内部操作）

参考代码：

def plot\_conv\_output(values):

# 卷积层中的卷积核数量

num\_filters = values.shape[3]

# 要绘制的网格数

# 卷积核的平方根

num\_grids = math.ceil(math.sqrt(num\_filters))

# 创建带有网格子图的图像

fig, axes = plt.subplots(num\_grids, num\_grids)

#画出所有卷积核的输出图像

for i, ax in enumerate(axes.flat):

# 仅画出有效卷积核图像

if i<num\_filters:

# 获取第i个卷积核的输出图像

img = values[0, :, :, i]

# 画图e.

ax.imshow(img, interpolation='nearest', cmap='binary')

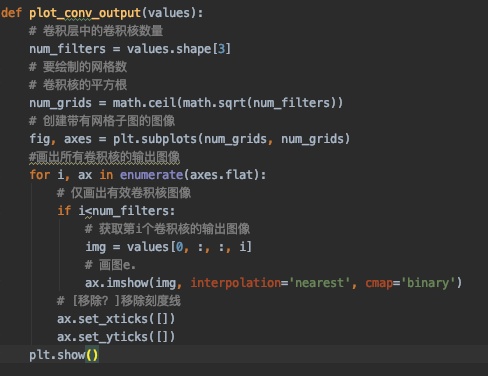
# 移除刻度线

ax.set\_xticks([])

ax.set\_yticks([])

plt.show()

检测条件：需对代码进行判断



5.5输入图像

请参考实验3中的4.3部分来完成plot\_image(image)函数（参数image表示图像）并绘制测试集的第一张图片。

参考代码:

def plot\_image(image):

plt.imshow(image.reshape(img\_shape),

interpolation='nearest',

cmap='binary')

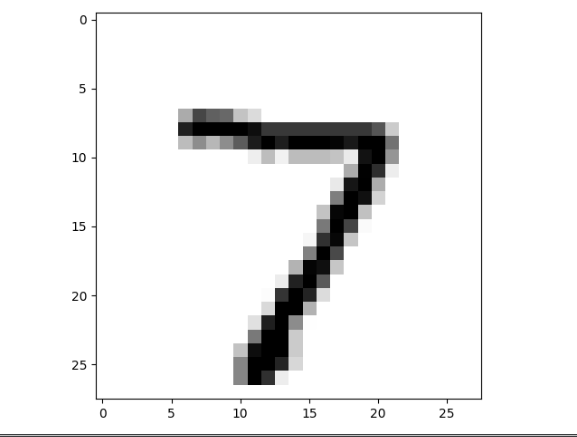
plt.show()

image1 = data.x\_test[0]

plot\_image(image1)

代码执行结果：

检测条件：输出结果如上即正确



5.6卷积层输出之方法一

5.6.1 基于K函数的模型转换函数

在Keras模型中有不同的方法来获取图层的输出。方法一是使用 K函数将Keras模型的一部分转换为函数。首先需要导入该函数：

from tensorflow.python.keras import backend as K

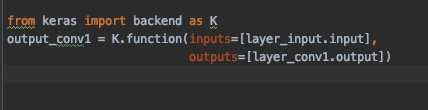
请使用K函数来获取图层输出（参数分别是输入层的输入和卷积层1的输出），并赋值给output\_conv1

参考代码:

output\_conv1 = K.function(inputs=[layer\_input.input],

outputs=[layer\_conv1.output])

检测条件：需对代码进行判断



5.6.2 获取卷积层1的输出

然后我们可以用输入图像调用经由K函数转换的output\_conv1函数

请在上一步已经通过得到的output\_conv1函数（该函数的输入是一张图片）的基础上，用图片image1输入到函数中，并取返回的第一个维度的值给layer\_output1。请输出查看一下layer\_output1的形状。

提示该函数的输入是image，它需要被嵌套在2个列表里面即[[image1]]来作为输入(因为改函数需要这个维度的数组)，同时该函数返回一个比我们想要的维度更多的数组，所以我们需要取出第一维的元素返回给layer\_output1。

参考代码：

layer\_output1 = output\_conv1([[image1]])[0]

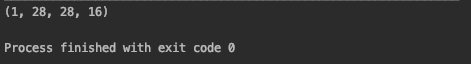
layer\_output1.shape

代码执行结果：

(1, 28, 28, 16)

检测条件：输出以上结果则正确

image1 = image1.reshape(1,784)#!!!很关键。  
layer\_output1 = output\_conv1([[image1]])[0]  
  
print(layer\_output1.shape)



5.6.3绘制输出

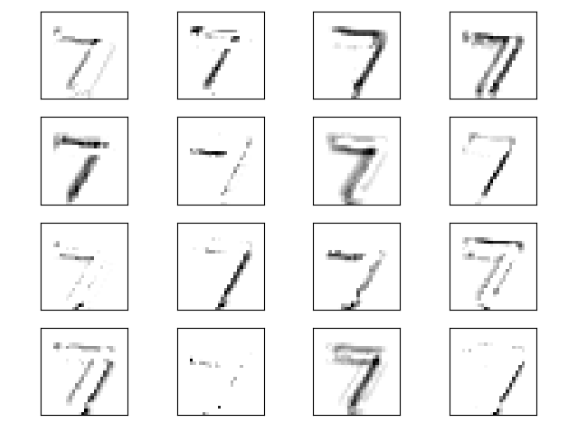
最后，我们可以绘制卷积层的所有16个通道的输出。请使用plot\_conv\_output函数绘制卷积层1的输出：

参考代码：

plot\_conv\_output(values=layer\_output1)

代码执行结果：

检测条件：输出结果类似即可



5.7卷积层输出之方法二

5.7.1 基于Model函数的图层输出获取

Keras还有另一种获取模型内层输出的方法。该方法使用与原始模型相同的输入创建另一个功能模型，但输出现在取自我们感兴趣的卷积层（即卷积层2的输出）。

请使用Model函数来获取图层输出（参数分别是输入层的输入和卷积层2的输出），并赋值给output\_conv2。

参考代码：

output\_conv2 = Model(inputs=layer\_input.input,

outputs=layer\_conv2.output)

检测条件：需对代码进行判断

# 卷积层输出方法二  
output\_conv2 = Model(inputs=layer\_input.input,  
 outputs=layer\_conv2.output)

5.7.2获取卷积层2的输出

Model函数的输出的是一个新建的模型对象。对于上一步得到的模型对象output\_conv2，我们可以通过该模型对象调用典型的Keras函数。为了得到卷积层的输出，我们通过output\_conv2调用predict()函数，predict()函数的输入是经过numpy数组转换的输入图像image1（predict函数的输入需要np数组）。

请调用output\_conv2的predict()函数得到卷积层2的输出，赋值给layer\_output2并查看其形状

参考代码：

layer\_output2 = output\_conv2.predict(np.array([image1]))

layer\_output2.shape

代码执行结果：

(1, 14, 14, 36)

检测条件：需对代码进行判断



layer\_output2 = output\_conv2.predict(np.array([image1]))  
print(layer\_output2.shape)

5.7.3绘制输出

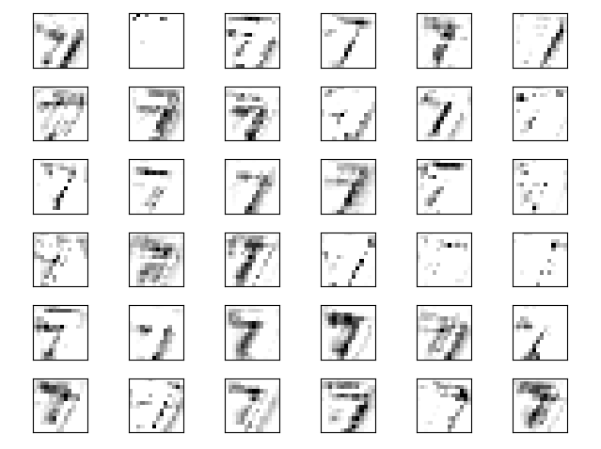
最后，我们可以绘制卷积层的所有36个通道的输出。请使用plot\_conv\_output函数绘制卷积层2的输出：

参考代码：

plot\_conv\_output(values=layer\_output2)

代码执行结果：

检测条件：输出结果类似即可



# 实验六

1. 问题描述：

文本是最常用的序列数据之一，可以理解为字符序列或单词序列，但是最常见的是单词级处理。本实验将会介绍单词和字符的one-hot编码，然后学会用Keras完成单词级的one-hot编码，然后介绍目前自然语言处理常用到的预处理方法词嵌入，并用Keras自带数据集完成简单的词嵌入模型。

1. 设计简要描述：

实验内容：

1. one-hot编码

1.1 one-hot编码简介

在最开始的实验2和实验3中，我们已经使用过简单的one-hot编码，one-hot编码是将标记转换为向量的最常用，最基本的方法。对于单词级别的one-hot编码，它将每个单词与一个唯一的整数索引相关联，然后将这个整数索引i转换为长度为N的二进制向量（N是词表的大小），这个向量只有第i个元素是1，其余都是0，对于字符级别的one-hot编码，它将每个字符与一个唯一的整数索引相关联，其余与单词级别的类似。

1.2 单词级的one-hot编码

初始数据：每个样本是列表的一个元素（本例中的样本是一个句子，但也可以是一篇文档）

现在我们通过以下方式来完成单词级的one-hot编码（该步骤不需要学员实现），请复制代码到.ipynb文件中并运行，代码如下：

import numpy as np

#自行创建的简单数据

samples = ['The cat sat on the mat.', 'The dog ate my homework.']

# 构建数据中所有标记的索引，用一个字典来存储

token\_index = {}

for sample in samples:

# 利用split方法对样本进行分词.

for word in sample.split():

if word not in token\_index:

#为每个唯一单词指定一个唯一索引

token\_index[word] = len(token\_index) + 1

#没有为索引编号0指定单词

#对样本进行分词

# 只考虑每个样本前max\_length个单词。

max\_length = 10

#结果返回给results:

results = np.zeros((len(samples), max\_length, max(token\_index.values()) + 1))

for i, sample in enumerate(samples):

for j, word in list(enumerate(sample.split()))[:max\_length]:

index = token\_index.get(word)

#指定唯一的元素为1

results[i, j, index] = 1.

#查看索引字典

print(token\_index)

print(results[1,1])#样本列表的第二个元素的第二个单词编码情况

代码执行结果：

{'The': 1,

'cat': 2,

'sat': 3,

'on': 4,

'the': 5,

'mat.': 6,

'dog': 7,

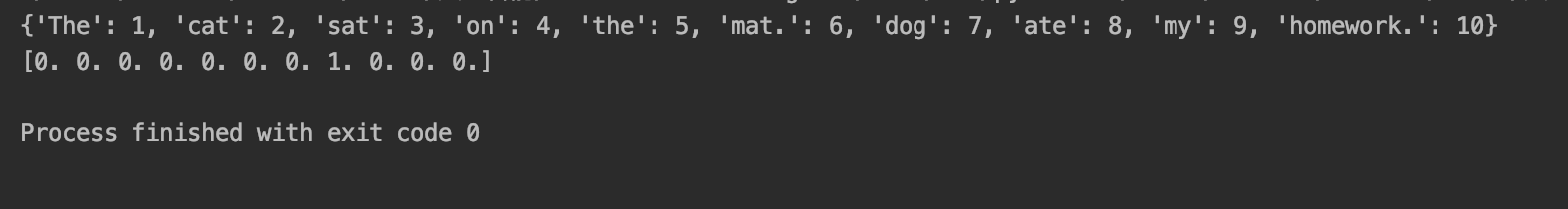
'ate': 8,

'my': 9,

'homework.': 10}

[0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]

检测条件：输出是否一致。



1.3 请学员完成以下任务

1）给上述samples列表新增一个元素‘a panda is sleeping’

2) 自行完成单词级别的分词任务，并查看索引字典和样本列表的第二个元素的第二个单词的编码情况。（注意索引是0开始）

参考代码：

import numpy as np

#自行创建的简单数据

samples = ['The cat sat on the mat.', 'The dog ate my homework.'，'a panda is sleeping.']

# 构建数据中所有标记的索引，用一个字典来存储

token\_index = {}

for sample in samples:

# 利用split方法对样本进行分词.

for word in sample.split():

if word not in token\_index:

#为每个唯一单词指定一个唯一索引

token\_index[word] = len(token\_index) + 1

#没有为索引编号0指定单词

#对样本进行分词

# 只考虑每个样本前max\_length个单词。

max\_length = 10

#结果返回给results:

results = np.zeros((len(samples), max\_length, max(token\_index.values()) + 1))

for i, sample in enumerate(samples):

for j, word in list(enumerate(sample.split()))[:max\_length]:

index = token\_index.get(word)

#唯一的元素为1

results[i, j, index] = 1.

#查看索引字典

token\_index

代码执行结果：

{'The': 1,

'cat': 2,

'sat': 3,

'on': 4,

'the': 5,

'mat.': 6,

'dog': 7,

'ate': 8,

'my': 9,

'homework.': 10,

'a.': 11,

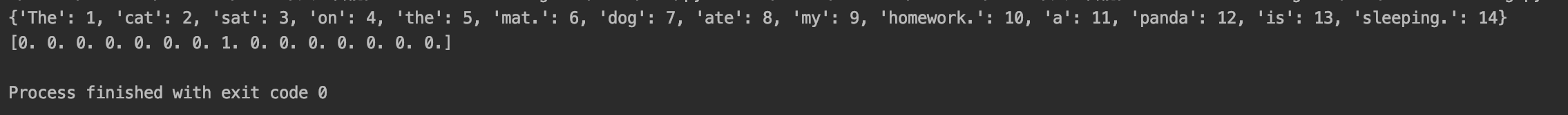
'panda.': 12,

'is.': 13 ,

'sleeping.': 14}

[0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]

检测条件：输出是否一致



1.4 字符级的one-hot编码

与上面类似现在我们通过以下方式来完成单词级的one-hot编码（该步骤不需要学员实现），请复制代码到.ipynb文件中并运行，代码如下：

import string

samples = ['The cat sat on the mat.', 'The dog ate my homework.']

characters = string.printable #所以可以打印的ASCII字符

#创建索引字典

token\_index = dict(zip(characters, range(1, len(characters) + 1)))

#为所有可能打印的字符创建一个字典

max\_length = 50

results = np.zeros((len(samples), max\_length, max(token\_index.values()) + 1))

for i, sample in enumerate(samples):

for j, character in enumerate(sample[:max\_length]):

index = token\_index.get(character)

results[i, j, index] = 1.

print(token\_index)#查看索引字典

print(results[1,1])#样本列表的第二个元素的第二个字符编码情况

代码执行结果：

array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,

0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,

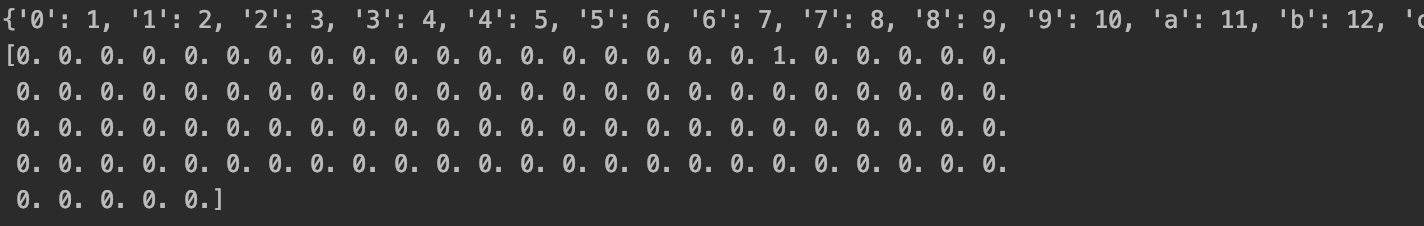
0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,

0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,

0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,

0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])

检测条件：输出是否一致。



1.5 请学员使用 1.3中新增的数据一个元素的数据集来重构索引字典并查看第三个元素中的第三个字符的编码情况

参考代码：

import string

samples = ['The cat sat on the mat.', 'The dog ate my homework.'，'a panda is sleeping.']

characters = string.printable #所以可以打印的ASCII字符

#创建索引字典

token\_index = dict(zip(characters, range(1, len(characters) + 1)))

#为所有可能打印的字符创建一个字典

max\_length = 50

results = np.zeros((len(samples), max\_length, max(token\_index.values()) + 1))

for i, sample in enumerate(samples):

for j, character in enumerate(sample[:max\_length]):

index = token\_index.get(character)

results[i, j, index] = 1.

print(token\_index)#查看索引字典

print(results[2,2])#样本列表的第3个元素的第3个字符编码情况

代码执行结果：

array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,

0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.,

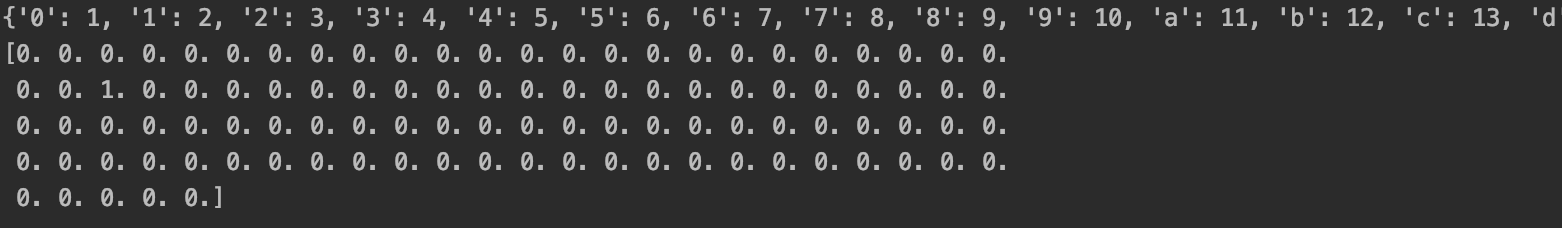
0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,

0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,

0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,

0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])

检测条件：输出是否一致。



1.6 用keras实现单词级的one-hot编码

我们可以通过以下简单的方式用keras来实现one-hot编码（该步骤不需要学员实现），请复制代码到.ipynb文件中并运行，代码如下：

from keras.preprocessing.text import Tokenizer

samples = ['The cat sat on the mat.', 'The dog ate my homework.']

#创建一个分词器

# 只考虑前1000个最常见的单词

tokenizer = Tokenizer(num\_words=1000)

# 构建单词索引

tokenizer.fit\_on\_texts(samples)

# 将字符串转换为整数索引的组成的列表

sequences = tokenizer.texts\_to\_sequences(samples)

# 可以直接得到one-hot编码二进制表示

# 分词器也支持除one-hot编码外的其他向量化模式

one\_hot\_results = tokenizer.texts\_to\_matrix(samples, mode='binary')

# 找回单词索引

word\_index = tokenizer.word\_index

word\_index

代码执行结果：

{'the': 1,

'cat': 2,

'sat': 3,

'on': 4,

'mat': 5,

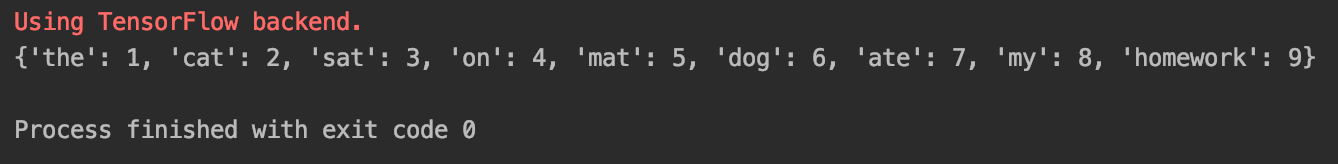
'dog': 6,

'ate': 7,

'my': 8,

'homework': 9}

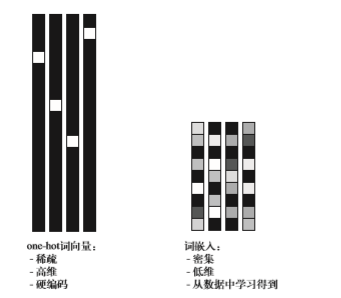
检测条件：输出是否一致。



1. 词嵌入模型

2.1 词嵌入简介

将单词与向量相关联的另外一个强大的方法就是词嵌入，one-hot编码得到的向量是二进制的、稀疏的（绝大部分你元素都是0）、维度很高（维度大小等于词表中的单词个数），而词嵌入是低微的浮点数向量（即密集向量，与稀疏向量相对），如下图所示，与one-hot编码得到的词向量不同，词嵌入是从数据中学习得到的。常见的词向量维度是256、512或1024（处理非常大的词表时）。与此相对，one-hot编码的词向量维度通常是20000或者更高（对应包含20000个标记的词表），因此词向量可以将更多的信息塞入更低的维度中。



2.2 利用Embedding层学习词嵌入

2.3 实例化Embedding层

可以将Embedding层理解为一个字典，将整数索引（表示特定单词）映射为密集向量。它接收整数作为输入，并在内部字典查找这些整数，然后返回相关联的向量。



在我们通过以下方式来实例化一个Embedding（该步骤不需要学员实现），请复制代码到.ipynb文件中并运行，代码如下：

from keras.layers import Embedding

# Embedding层至少需要2个参数

# 标记的个数（这里是1000，即最大单词索引+1）和嵌入维度（这里是64）

embedding\_layer = Embedding(1000, 64)

2.4 加载IMDB数据

本节使用IMDB数据集，它包含来自互联网电影数据库（IMDB）的50000条严重两极分化的评论。数据集被分为用于训练的25000条评论和用于测试的25000条评论，训练集和测试集都包含50%的正面评论和50%的负面评论。

可以通过以下方式构造我们的数据集（该步骤不需要学员实现），请复制代码到.ipynb文件中并运行，代码如下：

from keras.datasets import imdb

from keras import preprocessing

# 作为特征的单词，即选取出现频率最高的单词数量

max\_features = 10000

# 在这么多单词后截断文本

# (这些单词都属于前max\_features个最常见单词)

maxlen = 20

#将数据集加载为列表

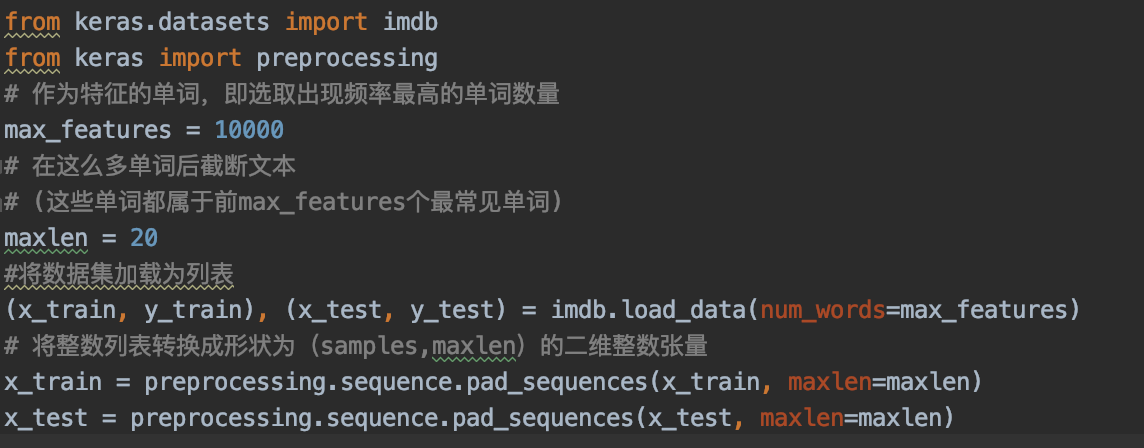
(x\_train, y\_train), (x\_test, y\_test) = imdb.load\_data(num\_words=max\_features)

# 将整数列表转换成形状为（samples,maxlen）的二维整数张量

x\_train = preprocessing.sequence.pad\_sequences(x\_train, maxlen=maxlen)

x\_test = preprocessing.sequence.pad\_sequences(x\_test, maxlen=maxlen)

检测条件：需要对代码进行判断。



2.5 构建keras模型并训练拟合模型

请学员完成以下步骤

1. 导入包Sequential,Flatten,Dense,Embedding
2. 定义一个序列模型
3. 添加一个Embedding层，标记个数10000，维度8，输入长度是maxlen
4. 添加一个Flatten层
5. 添加一个全连接层，输出维度是1，激活函数‘sigmoid’
6. 编译模型，优化器选取‘rmsprop’，损失函数选取‘binary\_crossentropy’,评估方式是‘acc’
7. 用.summary()方法查看模型架构
8. 拟合模型，epoch选取10，batch\_size选取32，validation\_split为0.2

参考代码：

from keras.models import Sequential

from keras.layers import Flatten, Dense

model = Sequential()

# 指定Embedding层的最大输入长度，以便后面将嵌入输入展平。

model.add(Embedding(10000, 8, input\_length=maxlen))

# 将三维的嵌入张量展平为(samples, maxlen \* 8)的二维张量

model.add(Flatten())

# 添加分类器

model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop', loss='binary\_crossentropy', metrics=['acc'])

model.summary()

history = model.fit(x\_train, y\_train,

epochs=10,

batch\_size=32,

validation\_split=0.2)

代码执行结果：

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

embedding\_2 (Embedding) (None, 20, 8) 80000

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten\_1 (Flatten) (None, 160) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_1 (Dense) (None, 1) 161

=================================================================

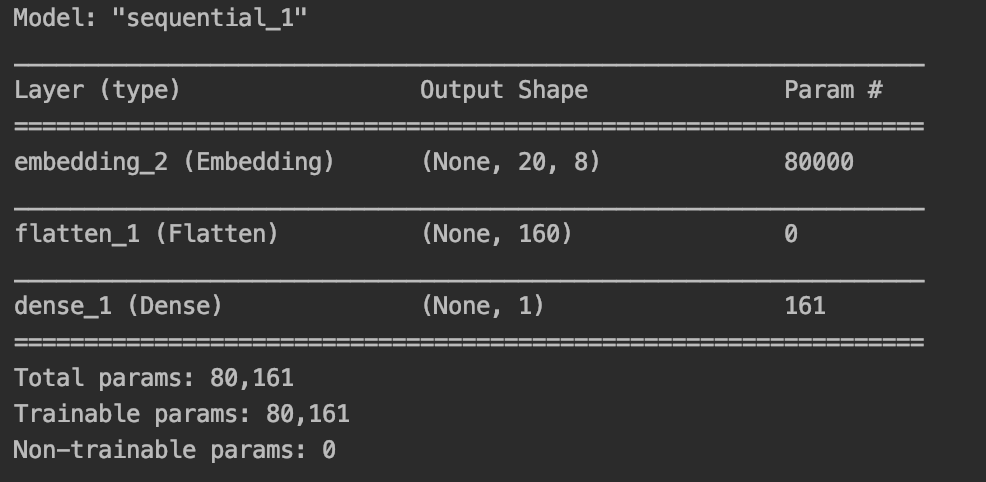
Total params: 80,161

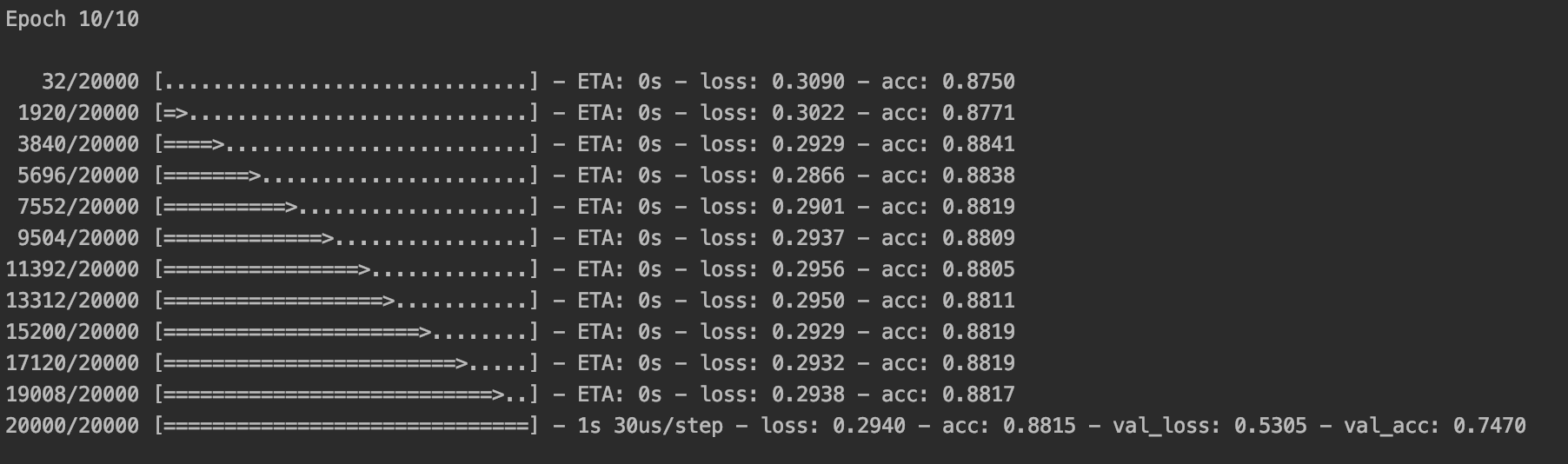
Trainable params: 80,161

Epoch 10/10

20000/20000 [==============================] - 1s 53us/step - loss: 0.2843 - acc: 0.8854 - val\_loss: 0.5301 - val\_acc: 0.7464

检测条件：输出模型架构一致且输出准确率接近即可。





# 实验七

1. 问题描述：

前见过的所以神经网络（比如全连接网络和卷积神经网络）都有一个主要特点，那就是它们都没有记忆。它们单独处理每个输入，在输入和输入之间没有保存任何状态。对于这样的网络，要想处理数据点的序列或者时间序列，你需要向网络同时展示整个序列，即将序列转换成单个数据点。例如，在实验九词嵌入中就是这么做的：将全部电影评论转换为一个大向量，然后一次性处理。

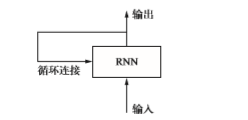
与此相反，当人在阅读这个句子时，是一个词一个词地阅读（或者说，眼睛一次扫视一次扫视地阅读），同时会记住之前的内容，这让你能够动态理解这个句子所传达的含义，以渐进的方式处理信息，同时保存一个关于所处理的内部模型，这就是循环神经网络，它广泛应用于自然语言处理中，本实验会介绍简单的循环神经网络（SimpleRNN）来处理文本数据，然后改进网络使用LSTM神经网络对比效果。

1. 设计简要描述：

1. 简单循环神经网络

* 1. 循环网络简介

RNN处理序列的方式是，遍历所有序列元素，并保存一个状态，其中包含与已查看内容相关的信息。实际上，RNN是一类具有内部环的的神经网络（如下图）。在处理两个不同的独立序列（比如两条不同的IMDB评论）之间，RNN状态会被重置，因此，你仍可以将一个序列看作单个数据点，即网络的单个输入。真正改变的是，数据点不再是在单个步骤中进行处理，相反，网络内部会对序列元素进行遍历。



* 1. Keras中的循环层

在Keras中能够通过以下代码实现一个简单的RNN层（该步骤不需要学员实现），请复制代码到.ipynb文件中并运行，代码如下：

from keras.models import Sequential

from keras.layers import Embedding, SimpleRNN

model = Sequential()

model.add(Embedding(10000, 32))

model.add(SimpleRNN(32))

model.summary()

代码执行结果：

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

embedding\_1 (Embedding) (None, None, 32) 320000

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

simple\_rnn\_1 (SimpleRNN) (None, 32) 2080

=================================================================

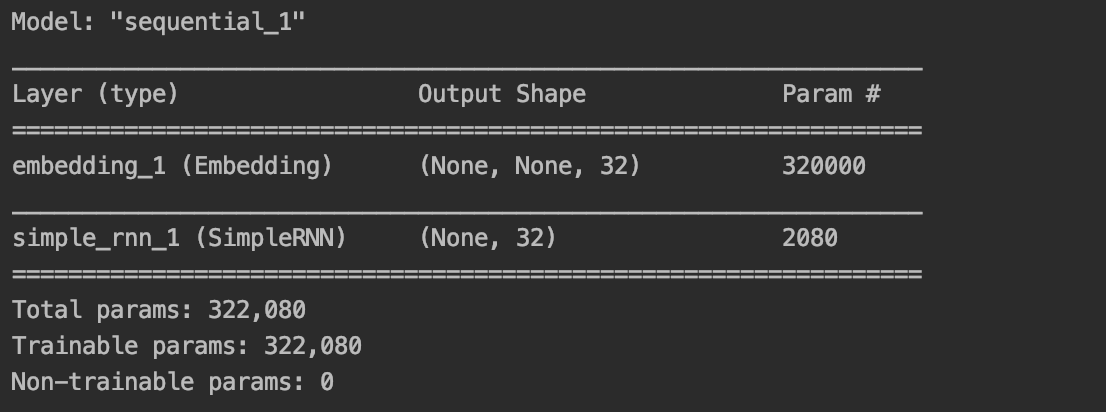
Total params: 322,080

Trainable params: 322,080

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

检测条件：输出是否一致



* 1. 数据处理

我们还是使用IMDB电影评论的数据，但是我们需要引入一个sequence模块把我们的数据格式化输入到神经网络中，（该步骤不需要学员实现），请复制代码到.ipynb文件中并运行，代码如下：

from keras.datasets import imdb

from keras.preprocessing import sequence

max\_features = 10000 #作为特征的单词个数

maxlen = 500

batch\_size = 32

#加载数据

print('Loading data...')

(input\_train, y\_train), (input\_test, y\_test) = imdb.load\_data(num\_words=max\_features)

print(len(input\_train), 'train sequences')

print(len(input\_test), 'test sequences')

print('Pad sequences (samples x time)')

input\_train = sequence.pad\_sequences(input\_train, maxlen=maxlen)

input\_test = sequence.pad\_sequences(input\_test, maxlen=maxlen)

print('input\_train shape:', input\_train.shape)

print('input\_test shape:', input\_test.shape)

代码执行结果：

Loading data...

25000 train sequences

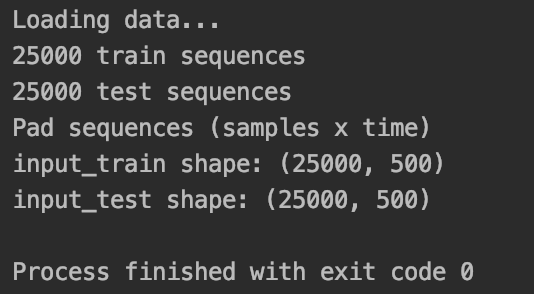
25000 test sequences

Pad sequences (samples x time)

input\_train shape: (25000, 500)

input\_test shape: (25000, 500)

检测条件：输出是否一致



* 1. 搭建自己RNN神经网络

请学员按照如下步骤搭建自己的神经网络并应用上面数据拟合模型

1. 从kaeras导入全连接层模块
2. 定义一个序列模型
3. 添加一个Embedding层，参数是（max\_features,32）
4. 添加一个SimpleRNN层，输出维度32
5. 添加一个全连接层，输出维度1，激活函数‘sigmoid’
6. 编译模型，参数分别是‘rmsprop’,’binary\_crossentropy’,[‘acc’]
7. 拟合模型，epochs=10.batch\_size=128,validation\_split=0.2，结果返回给history

参考代码：

from keras.layers import Dense

model = Sequential()

model.add(Embedding(max\_features, 32))

model.add(SimpleRNN(32))

model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop', loss='binary\_crossentropy', metrics=['acc'])

history = model.fit(input\_train, y\_train,

epochs=10,

batch\_size=128,

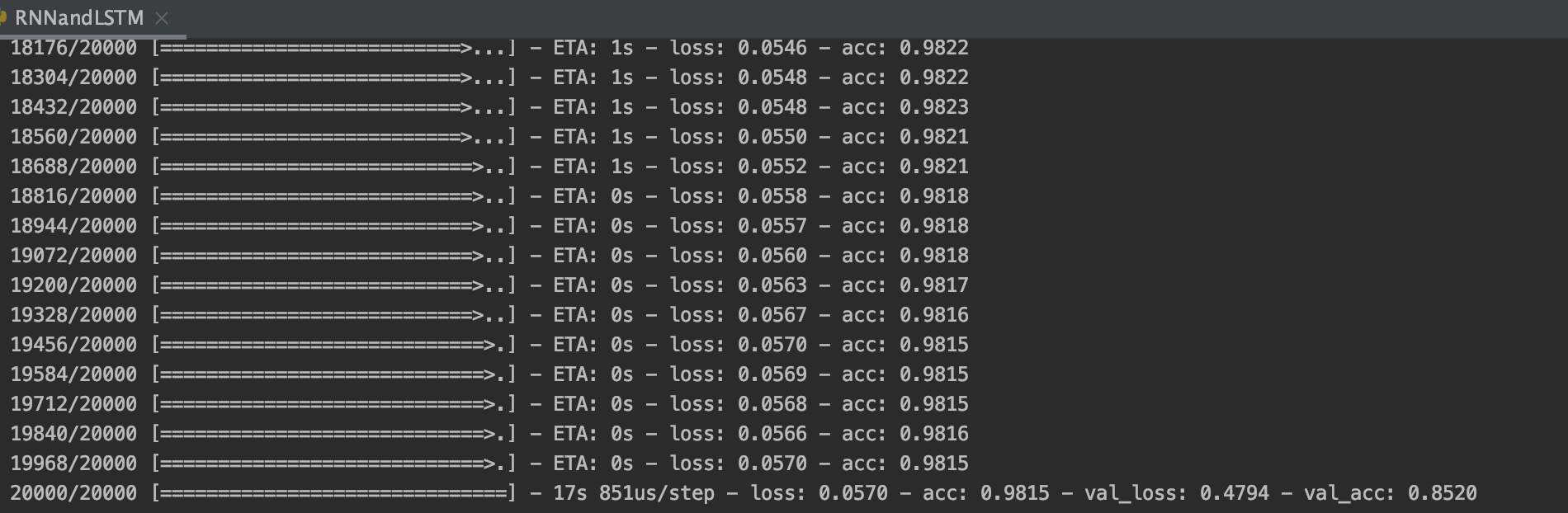
validation\_split=0.2)

代码执行结果：

Epoch 10/10

20000/20000 [==============================] - 19s - loss: 0.0256 - acc: 0.9925 - val\_loss: 0.6707 - val\_acc: 0.8054

检测条件：输出准确率接近即可



1.5 绘制结果

能够通过以下代码绘制训练损失和准确率率结果（该步骤不需要学员实现），请复制代码到.ipynb文件中并运行，代码如下：

import matplotlib.pyplot as plt

acc = history.history['acc']

val\_acc = history.history['val\_acc']

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'bo', label='Training acc')

plt.plot(epochs, val\_acc, 'b', label='Validation acc')

plt.title('Training and validation accuracy')

plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')

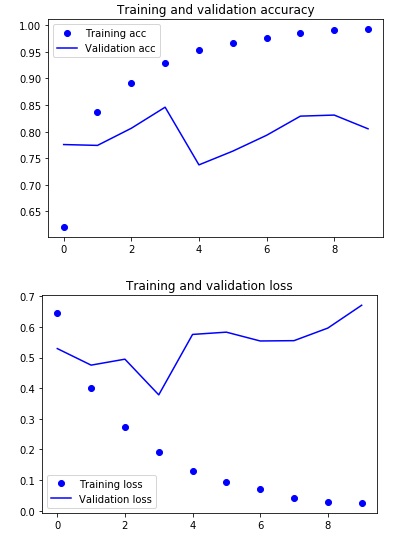
plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title('Training and validation loss')

plt.legend()

plt.show()

代码执行结果：



检测条件：输出图像趋势接近即可

2 LSTM循环神经网络

2.1 SimpleRNN的缺陷

SimpleRNN最大的问题是：在时刻t,理论上来说，它应该能够记住许多时间步之前见过的信息，但实际上它是不可能学到这种长期依赖的，其原因在于梯度消失问题，随着层数的增加，网络最终变得无法训练，LSTM层和GRU层都是为了解决这个问题而设计的。本实验主要介绍使用更为广泛的LSTM模型

2.2 LSTM层是SimpleRNN层的一种变体，它增加了一种携带信息跨越多个时间步的方法。假设设有一条传送带，其运行方向平行于你所处理的序列。序列中的信息可以在任意位置跳上传送带，然后被传送到更晚的时间步，并在需要时原封不动地跳回来。这实际上就是LSTM的原理：它保存信息以便后面使用，从而防止较早期的信号在处理过程中逐渐消失。

2.3 搭建自己的LSTM神经网络

类似1.4节请学员完成以下步骤

1）从kaeras导入LSTM模块

2）定义一个序列模型

3）添加一个Embedding层，参数是（max\_features,32）

4）添加一个LSTM层，输出维度32

5）添加一个全连接层，输出维度1，激活函数‘sigmoid’

6） 编译模型，参数分别是‘rmsprop’,’binary\_crossentropy’,[‘acc’]

7）拟合模型，epochs=10.batch\_size=128,validation\_split=0.2，结果返回给history

参考代码：

from keras.layers import LSTM

model = Sequential()

model.add(Embedding(max\_features, 32))

model.add(LSTM(32))

model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',

loss='binary\_crossentropy',

metrics=['acc'])

history = model.fit(input\_train, y\_train,

epochs=10,

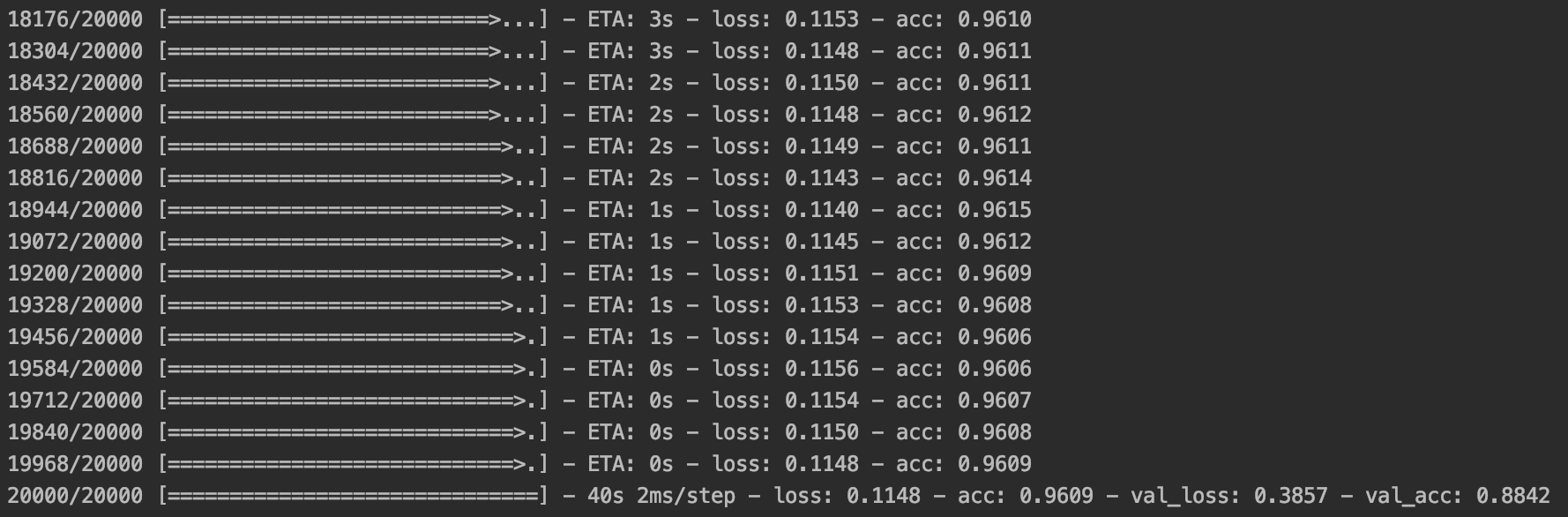
batch\_size=128,

validation\_split=0.2)

代码执行结果：

20000/20000 [==============================] - 106s - loss: 0.1092 - acc: 0.9628 - val\_loss: 0.3972 - val\_acc: 0.8758

检测条件：输出准确率接近即可。



2.4 绘制结果

请学员参考1.5节绘制损失值和准确率的结果图

参考代码：

acc = history.history['acc']

val\_acc = history.history['val\_acc']

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'bo', label='Training acc')

plt.plot(epochs, val\_acc, 'b', label='Validation acc')

plt.title('Training and validation accuracy')

plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')

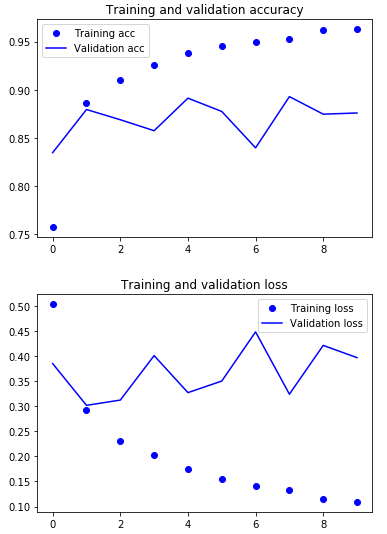
plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title('Training and validation loss')

plt.legend()

plt.show(）

代码执行结果：



检测条件：输出图像趋势接近即可

# 实验八

1. 问题描述：

自编码器是一种数据的压缩算法，其中数据的压缩和解压缩函数有如下几个特点：

1. 数据相关的
2. 有损的
3. 从样本中自动学习。

在大部分提到的自动编码器的场合，压缩和解压缩的函数是通过神经网络实现的。

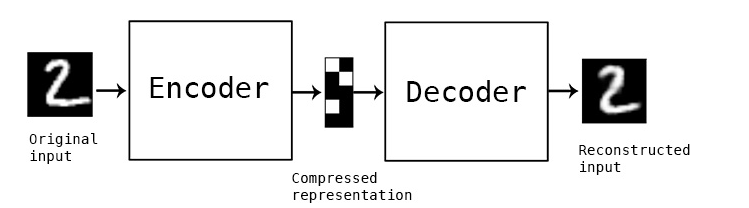
搭建一个自动编码器需要完成下面三洋工作：搭建编码器，搭建解码器，设定一个损失函数，用以衡量由于压缩而损失掉的信息。本实验会通过搭建一个简单的自编码器观测数据信息，并再搭建一个卷积自编码器作为对比，并学会使用自编码器进行降噪。

1. 设计简要描述：

1 自编码器

1.1 自编码器简介

自编码器是一类尝试使用反向传播算法重新创建输入数据作为输出的神经网络。自编码器包含两部分：编码器和解码器。编码器读取输入并把它压缩成紧凑表示，解码器则读取紧凑表示并用其重建输入。如下图所示：



1.2 搭建简单的自编码器模型

可以通过keras很快的创建如下的编码器和解码器并编译（该步骤不需要学员实现），请复制代码到.ipynb文件中并运行，代码如下：

from keras.layers import Input, Dense

from keras.models import Model

# 编码器维度

encoding\_dim = 32

input\_img = Input(shape=(784,))

# "encoded" 是把输入编码表示

encoded = Dense(encoding\_dim, activation='relu')(input\_img)

# "decoded" 是输入的有损重构

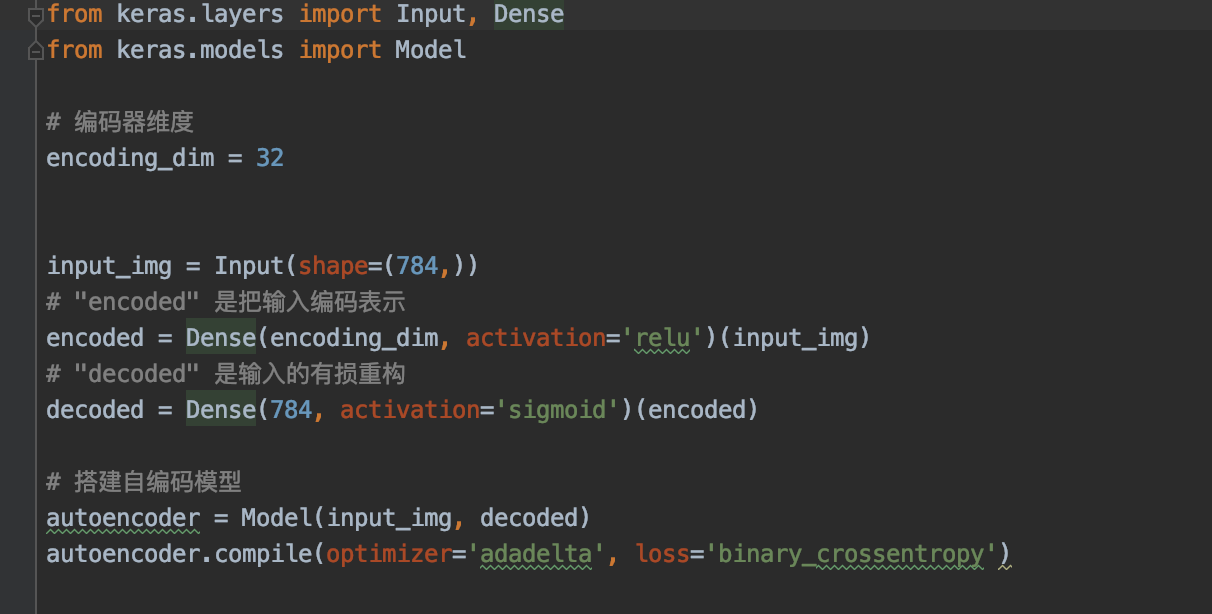
decoded = Dense(784, activation='sigmoid')(encoded)

# 搭建自编码模型

autoencoder = Model(input\_img, decoded)

autoencoder.compile(optimizer='adadelta', loss='binary\_crossentropy')

检测条件：需要对代码进行判断。



1.3 导入数据集

可通过以下代码导入mnist数据集（此处已实现多次，学员可直接复制到环境中使用）代码如下：

from keras.datasets import mnist

import numpy as np

(x\_train, \_), (x\_test, \_) = mnist.load\_data()

x\_train = x\_train.astype('float32') / 255.

x\_test = x\_test.astype('float32') / 255.

x\_train=x\_train.reshape((len(x\_train),np.prod(x\_train.shape[1:]))

x\_test =x\_test.reshape((len(x\_test), np.prod(x\_test.shape[1:])))

print(x\_train.shape)

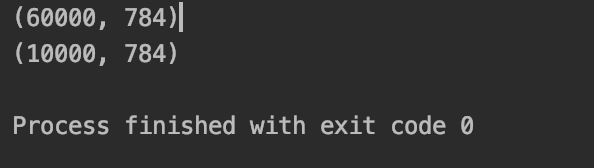
print(x\_test.shape)

代码执行结果：

(60000,784)

(10000,784)

检测条件：输出是否一致



1.4 拟合模型

请学员通过以上数据拟合模型autoencoder

提示：epochs =50,batch\_size=256,shuffle=True validation\_data=(x\_test,x\_test)

参考代码：

autoencoder.fit(x\_train, x\_train,

epochs=50,

batch\_size=256,

shuffle=True,

validation\_data=(x\_test, x\_test))

代码执行结果：

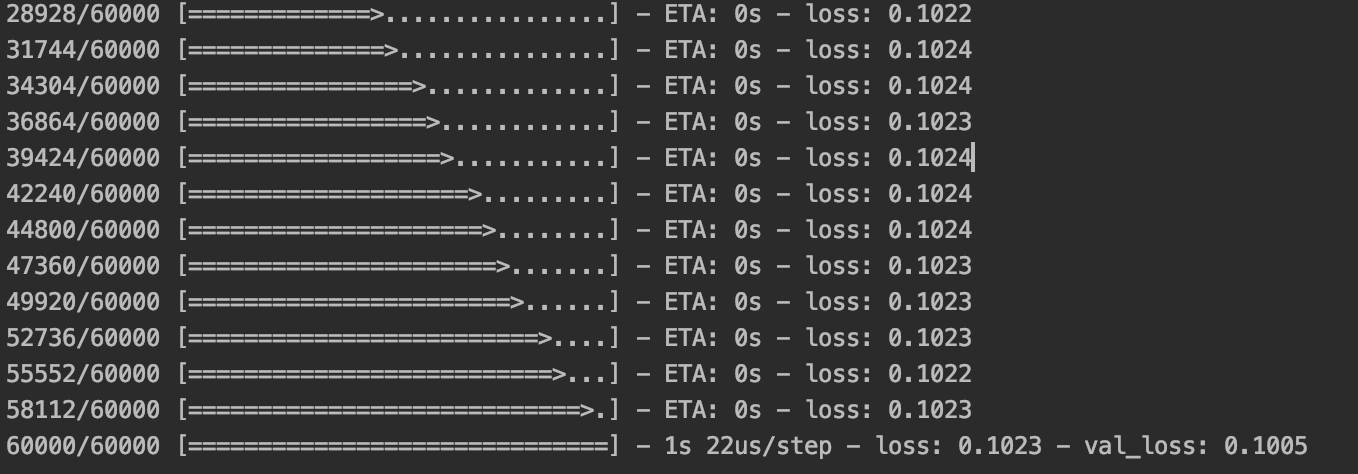
Epoch 50/50

60000/60000 [==============================] - 4s 75us/step - loss: 0.1052 - val\_loss: 0.1033

Out[9]:

<keras.callbacks.History at 0x7fbec2d6e518>

检测条件：输出loss与上述接近即可。



1.5 查看重构的输出与原来的输出对比

我们可以通过以下方式来查看（该步骤不需要学员实现），请复制代码到.ipynb文件中并运行，代码如下：

# 从测试集选取一些数据来编码和解码

encoded\_imgs = encoder.predict(x\_test)

decoded\_imgs = decoder.predict(encoded\_imgs)

import matplotlib.pyplot as plt

n = 10 # 打印的图片数量

plt.figure(figsize=(20, 4))

for i in range(n):

# 显示原来图像

ax = plt.subplot(2, n, i + 1)

plt.imshow(x\_test[i].reshape(28, 28))

plt.gray()

ax.get\_xaxis().set\_visible(False)

ax.get\_yaxis().set\_visible(False)

# 显示重构后的图像

ax = plt.subplot(2, n, i + 1 + n)

plt.imshow(decoded\_imgs[i].reshape(28, 28))

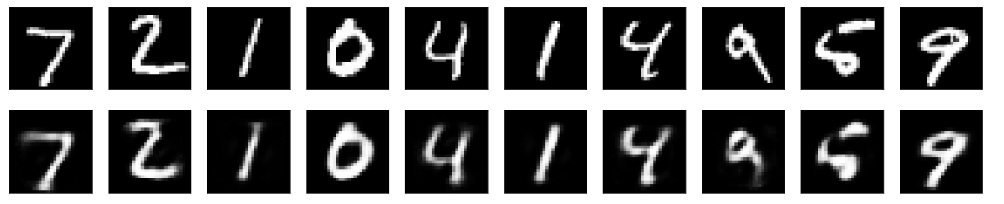
plt.gray()

ax.get\_xaxis().set\_visible(False)

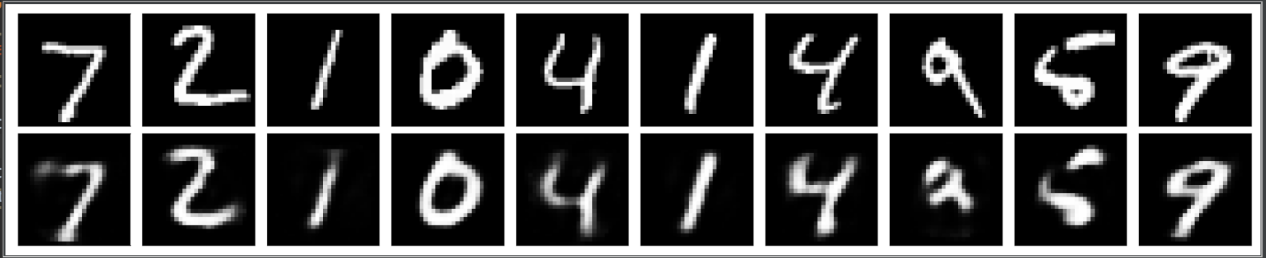
ax.get\_yaxis().set\_visible(False)

plt.show()

代码执行结果：



检测条件：是否输出以上结果。



2 卷积自编码器

2.1 搭建卷积自编码器

当输入是图像时，使用卷积神经网络基本上总是有意义的。在现实中，用于处理图像的自动编码器几乎都是卷积自动编码器——又简单又快

卷积自编码器的编码器部分由卷积层和MaxPooling层构成，MaxPooling负责空域下采样。而解码器由卷积层和上采样层构成。

我们可以通过以下方式来创建如下的卷积编码器和解码器并编译（该步骤不需要学员实现），请复制代码到.ipynb文件中并运行，代码如下：

from keras.layers import Input, Dense, Conv2D, MaxPooling2D, UpSampling2D

from keras.models import Model

from keras import backend as K

input\_img = Input(shape=(28, 28, 1)) #输入图像形状

#编码器

x=Conv2D(16, (3, 3), activation='relu', padding='same')(input\_img)

#卷积层

x = MaxPooling2D((2, 2), padding='same')(x)#空域下采样

x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)

x = MaxPooling2D((2, 2), padding='same')(x)

x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)

encoded = MaxPooling2D((2, 2), padding='same')(x)

#解码

x = Conv2D(8, (3, 3), activation='relu', padding='same')(encoded)

x = UpSampling2D((2, 2))(x)#上采样层

x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)

x = UpSampling2D((2, 2))(x)

x = Conv2D(16, (3, 3), activation='relu')(x)

x = UpSampling2D((2, 2))(x)

decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = Model(input\_img, decoded)

autoencoder.compile(optimizer='adadelta', loss='binary\_crossentropy')#编译

2.2加载数据并拟合模型

请学员参考1.3和1.4节加载数据并完成模型拟合

提示1：

x\_train用reshape重构时参数设置为x\_train = np.reshape(x\_train, (len(x\_train), 28, 28, 1)) ，x\_test类似

提示2：

autoencoder中的batch\_size设置为128，其他不变

参考代码：

from keras.datasets import mnist

import numpy as np

(x\_train, \_), (x\_test, \_) = mnist.load\_data()

x\_train = x\_train.astype('float32') / 255.

x\_test = x\_test.astype('float32') / 255.

x\_train = np.reshape(x\_train, (len(x\_train), 28, 28, 1))

x\_test = np.reshape(x\_test, (len(x\_test), 28, 28, 1))

autoencoder.fit(x\_train, x\_train,

epochs=50,

batch\_size=128,

shuffle=True,

validation\_data=(x\_test, x\_test),)

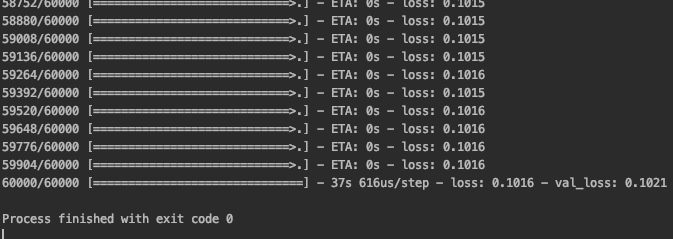
代码执行结果：

Epoch 50/50

60000/60000 [==============================] - 39s 644us/step - loss: 0.0979 - val\_loss: 0.0953

<keras.callbacks.History at 0x7fbec2d90128>

检测条件：输出loss与上述接近即可。



2.3查看重构的输出与原来的输出对比

请学员参考1.5节查看10张重构后的图像与原图像的对比

参考代码：

decoded\_imgs = autoencoder.predict(x\_test)

n = 10

plt.figure(figsize=(20, 4))

for i in range(n):

# 显示原图像

ax = plt.subplot(2, n, i+1)

plt.imshow(x\_test[i].reshape(28, 28))

plt.gray()

ax.get\_xaxis().set\_visible(False)

ax.get\_yaxis().set\_visible(False)

# 显示重构后的图像

ax = plt.subplot(2, n, i + n+1)

plt.imshow(decoded\_imgs[i].reshape(28, 28))

plt.gray()

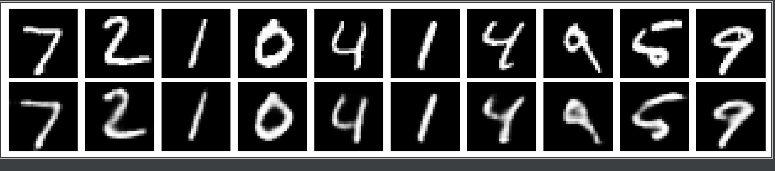
ax.get\_xaxis().set\_visible(False)

ax.get\_yaxis().set\_visible(False)

plt.show()

代码执行结果：

检测条件：是否输出以上结果。



3 自编码器的应用

3.1介绍

我们把训练样本用噪声污染，然后使用解码器解码出干净的照片，以获得去噪自动编码器。

3.2 把原图片加入高斯噪声

我们可以通过以下方式来加入噪声并查看加噪后的图片（该步骤不需要学员实现），请复制代码到.ipynb文件中并运行，代码如下：

from keras.datasets import mnist

import numpy as np

(x\_train, \_), (x\_test, \_) = mnist.load\_data()

x\_train = x\_train.astype('float32') / 255.

x\_test = x\_test.astype('float32') / 255.

x\_train = np.reshape(x\_train, (len(x\_train), 28, 28, 1))

x\_test = np.reshape(x\_test, (len(x\_test), 28, 28, 1))

noise\_factor = 0.5

x\_train\_noisy = x\_train + noise\_factor \* np.random.normal(loc=0.0, scale=1.0, size=x\_train.shape)

x\_test\_noisy = x\_test + noise\_factor \* np.random.normal(loc=0.0, scale=1.0, size=x\_test.shape)

x\_train\_noisy = np.clip(x\_train\_noisy, 0., 1.)

x\_test\_noisy = np.clip(x\_test\_noisy, 0., 1.)

n = 10

plt.figure(figsize=(20, 2))

for i in range(n):

ax = plt.subplot(1, n, i+1)

plt.imshow(x\_test\_noisy[i].reshape(28, 28))

plt.gray()

ax.get\_xaxis().set\_visible(False)

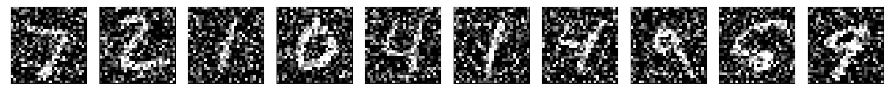
ax.get\_yaxis().set\_visible(False)

plt.show()

代码执行结果：



检测条件：是否输出以上结果。



3.3 构建自编码器

请学员查考2.1按如下步骤使用Keras函数模型搭建自编码器并拟合

1. 设置input\_img参数
2. 搭建编码器，添加Conv2D层，输入是input\_img，输出维度32，卷积核大小3×3，激活函数’‘relu’,padding设置为‘same’，返回给x
3. 添加Maxpooling2D层，大小是（2，2），padding设置为‘same’
4. 添加Conv2D层，输出维度32，卷积核大小3×3，激活函数’‘relu’,padding设置为‘same’
5. 添加Maxpooling2D层，大小是（2，2），padding设置为‘same’，返回给encoded
6. 设置解码器，添加Conv2D层，输入是encoded，输出维度32，卷积核大小3×3，激活函数’‘relu’,padding设置为‘same’，返回给x
7. 添加上采样层，大小是（2，2）
8. 添加Conv2D层，输出维度32，卷积核大小3×3，激活函数’‘relu’,padding设置为‘same’
9. 添加上采样层，大小是（2，2）
10. 添加Conv2D层，输入是encoded，输出维度1，卷积核大小3×3，激活函数’‘sigmoid’,padding设置为‘same’，返回给decoded
11. 设置自编码器，输入是input\_img,decoded
12. 编译自编码器，参数不变。
13. 拟合数据，参数与2.1节一致

参考代码：

input\_img = Input(shape=(28, 28, 1))

x = Conv2D(32, (3, 3), activation='relu', padding='same')(input\_img)

x = MaxPooling2D((2, 2), padding='same')(x)

x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)

encoded = MaxPooling2D((2, 2), padding='same')(x)

x = Conv2D(32, (3, 3), activation='relu', padding='same')(encoded)

x = UpSampling2D((2, 2))(x)

x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)

x = UpSampling2D((2, 2))(x)

decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = Model(input\_img, decoded)

autoencoder.compile(optimizer='adadelta', loss='binary\_crossentropy

autoencoder.fit(x\_train\_noisy, x\_train,

epochs=100,

batch\_size=128,

shuffle=True,

validation\_data=(x\_test\_noisy, x\_test),)

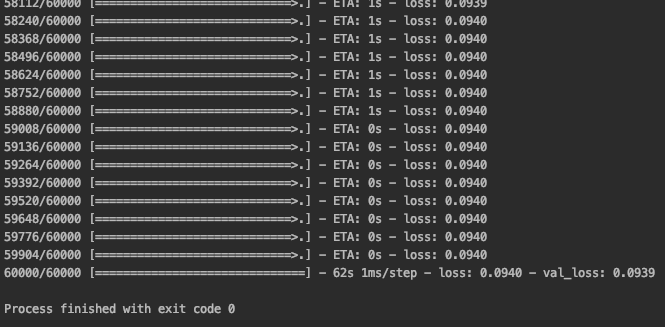
代码执行结果：

Epoch 50/50

60000/60000 [==============================] - 67s 1ms/step - loss: 0.0953 - val\_loss: 0.0946

<keras.callbacks.History at 0x7f685ce82390>

检测条件：输出loss与上述接近即可



2.3查看重构的输出与原来的输出对比

请学员参考1.5节查看10张重构后的图像与原图像的对比

参考代码：

decoded\_imgs = autoencoder.predict(x\_test)

n = 10

plt.figure(figsize=(20, 4))

for i in range(n):

ax = plt.subplot(2, n, i + 1)

plt.imshow(x\_test\_noisy[i].reshape(28, 28))

plt.gray()

ax.get\_xaxis().set\_visible(False)

ax.get\_yaxis().set\_visible(False)

ax = plt.subplot(2, n, i + n + 1)

plt.imshow(decoded\_imgs[i].reshape(28, 28))

plt.gray()

ax.get\_xaxis().set\_visible(False)

ax.get\_yaxis().set\_visible(False)

plt.show()

代码执行结果：

检测条件：是否输出以上结果。

