**Practical No.1**

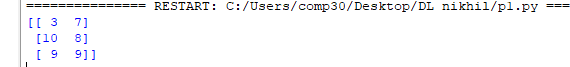
**[A] Aim: Perform basic mathematics operations in python**

**[1] Add matrix using dot()**

**Source Code:**

**import** numpy **as** np  
A = np.array([[1, 2], [3, 4], [5, 6]])  
A  
B = np.array([[2, 5], [7, 4], [4, 3]])  
B  
*# Add matrices A and B*C = A + B  
print(C)

**Output**

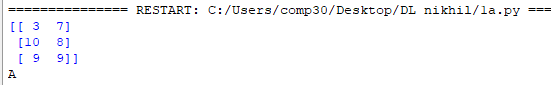


**[1a] Add matrix using add()**

**Source Code:**

**import** numpy **as** np  
A = np.array([[1, 2], [3, 4], [5, 6]])  
  
B = np.array([[2, 5], [7, 4], [4, 3]])  
  
*# Add matrices A and B*C = np.add(A, B)  
print(C)

**Output**

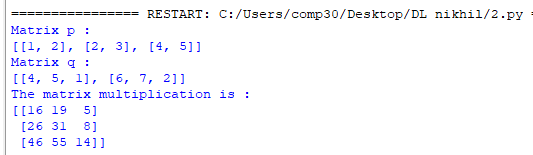


**[2]** **Multiply using dot()**

**Source Code:**

*# importing the module***import** numpy **as** np  
  
*# creating two matrices*p = [[1, 2], [2, 3], [4, 5]]  
q = [[4, 5, 1], [6, 7, 2]]  
print(**"Matrix p :"**)  
print(p)  
print(**"Matrix q :"**)  
print(q)  
  
*# computing product*result = np.dot(p, q)  
  
*# printing the result*print(**"The matrix multiplication is :"**)  
print(result)

**Output**

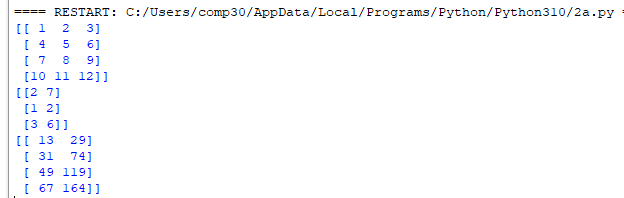


**[2a] Multiply using dot()**

**Source Code:**

**import** numpy **as** np  
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])  
print(A)  
B = np.array([[2, 7], [1, 2], [3, 6]])  
print(B)  
C = A.dot(B)  
print(C)

**Output:**



**[3a] Linear Combination**

**Source Code:**

**import** numpy **as** np  
x = np.array([[0, 1, 1],  
 [1, 1, 0],  
 [1, 0, 1]])  
y = ([3.65, 1.55, 3.42])  
scalars = np.linalg.solve(x, y)  
print(scalars)

**Output**

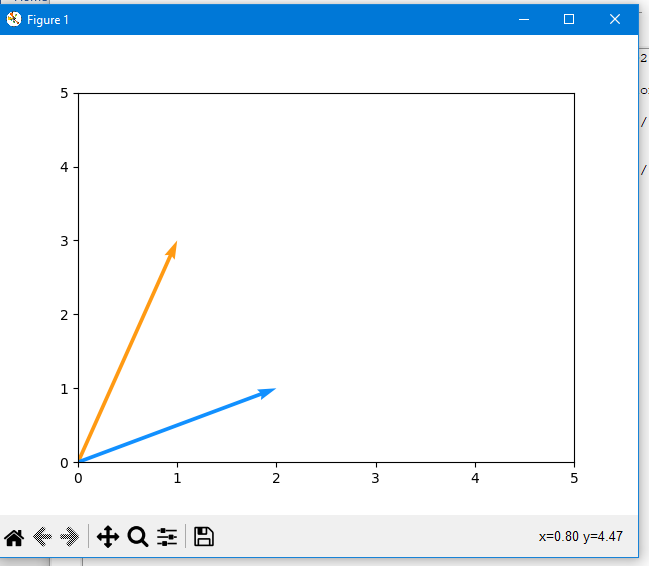


**[3b] Linear Combination**

**Source Code:**

**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
  
**def** plotVectors(vecs, cols, alpha =1 ):  
 plt.figure()  
 plt.axvline(x=0, color = **'#A9A9A9'**, zorder = 0)  
 **for** i **in** range(len(vecs)):  
 x = np.concatenate([[0,0],vecs[i]])  
 plt.quiver([x[0]],  
 [x[1]],  
 [x[2]],  
 [x[3]],  
 angles=**'xy'**, scale\_units=**'xy'**, scale=1,color=cols[i], alpha=alpha)  
orange= **'#FF9A13'**blue= **'#1190FF'**plotVectors([[1,3],[2,1]],[orange,blue])  
plt.xlim(0,5)  
plt.ylim(0,5)  
plt.show()

**Output**



**[4a] Linear Equation**

**Source Code:**

*# Suppose, a fruit-seller sold 20 mangoes and 10 oranges in one day for a total of $350. The next day he sold 17 mangoes and 22 oranges for $500. If the prices of the fruits remained unchanged on both the days, what was the price of one mango and one orange?  
#  
# This problem can be easily solved with a system of two linear equations.  
#  
# Let's say the price of one mango is x and the price of one orange is y. The above problem can be converted like this:  
#  
# 20x + 10y = 350  
# 17x + 22y = 500***import** numpy **as** np  
  
A = np.array([[20, 10], [17, 22]])  
B = np.array([350, 500])  
R = np.linalg.solve(A,B)  
x, y = np.linalg.solve(A,B)  
  
print(R)  
print(**"x ="**, x)  
print(**"y ="**, y)

**Output**

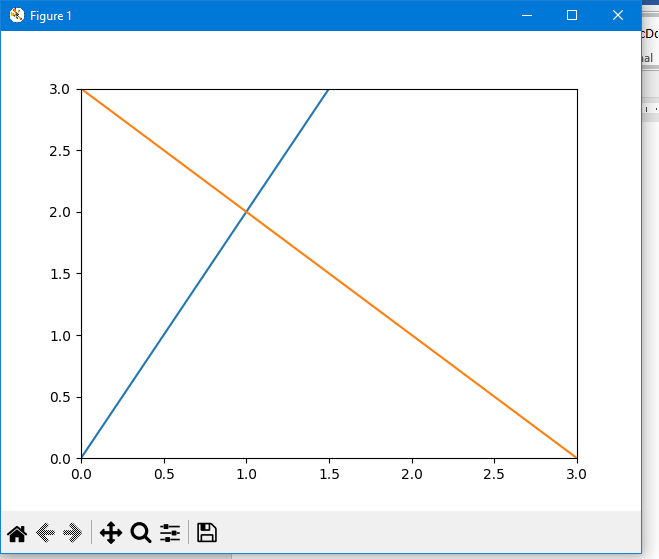


**[4b] Linear Equation**

**Source Code:**

**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
  
x = np.arange(-10,10)  
y = 2\*x  
y1 = -x + 3  
plt.figure()  
plt.plot(x,y)  
plt.plot(x,y1)  
plt.xlim(0,3)  
plt.ylim(0,3)  
plt.axvline(x=0, color=**'grey'**)  
plt.show()

**Output**



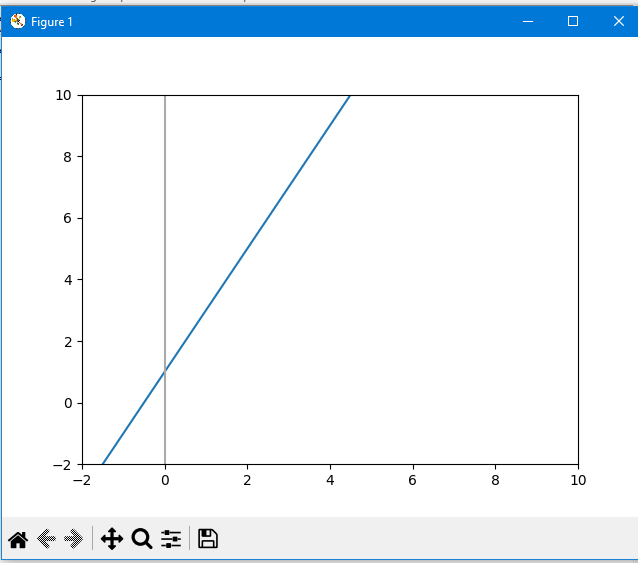
**[4c] Linear Equation**

**Source Code:**

**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt

x = np.arange(-10,10)  
y = 2\*x + 1  
  
plt.figure()  
plt.plot(x,y)  
plt.xlim(-2,10)  
plt.ylim(-2,10)  
  
plt.axvline(x=0, color = **'#A9A9A9'**)  
plt.show()

**Output**



**[5a] Norm one-dimensional**

**Source Code:**

*# import library***import** numpy **as** np  
  
*# initialize vector*oned = np.arange(10)  
  
*# compute norm of vector*manh\_norm = np.linalg.norm(oned)  
  
print(**"Manhattan norm:"**)  
print(manh\_norm)

**Output**

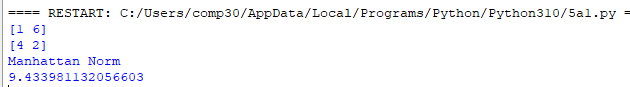


**[5a 1] Norm one-dimensional**

**Source Code:**

**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
  
u = np.array([1,6])  
print(u)  
v = np.array([4,2])  
print(v)  
  
manhatan\_norm = np.linalg.norm(u+v)  
print(**"Manhattan Norm"**)  
print(manhatan\_norm)

**Output**



**[5b] Norm two-dimensional**

**Source Code:**

*# import library***import** numpy **as** np  
  
*# initialize matrix*twod = np.array([[ 1, 2, 3],  
 [4, 5, 6]])  
  
*# compute norm of matrix*eucl\_norm = np.linalg.norm(twod)  
  
print(**"Euclidean norm:"**)  
print(eucl\_norm)

**Output**



**[5c] Norm three-dimensional**

**Source Code:**

*# import library***import** numpy **as** np  
  
*# initialize matrix*threed = np.array([[[ 1, 2, 3],  
 [4, 5, 6]],[[ 11, 12, 13],  
 [14, 15, 16]]])  
  
*# compute norm of matrix*mink\_norm = np.linalg.norm(threed)  
  
print(**"Minkowski norm:"**)  
print(mink\_norm)

**Output**

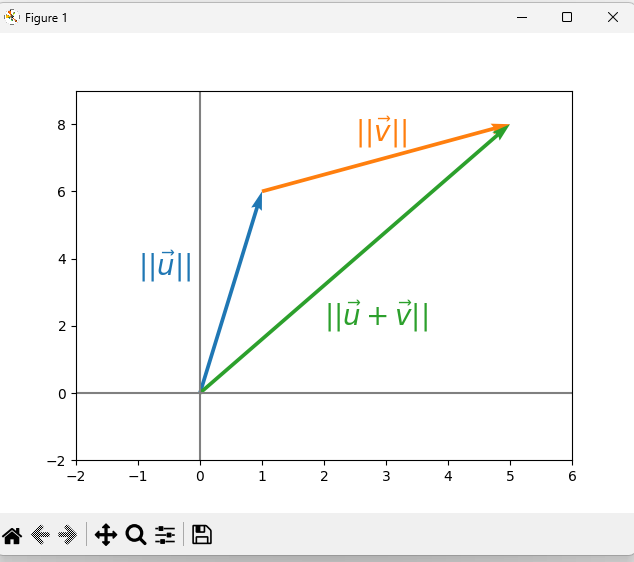


**[5d] Norm**

**Source Code:**

**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
**import** seaborn **as** sns  
  
u = [0,0,1,6]  
v = [0,0,4,2]  
u\_bis = [1,6,v[2],v[3]]  
w = [0,0,5,8]  
plt.quiver([u[0], u\_bis[0], w[0]],  
 [u[1], u\_bis[1], w[1]],  
 [u[2], u\_bis[2], w[2]],  
 [u[3], u\_bis[3], w[3]],  
 angles = **'xy'**, scale\_units = **'xy'**, scale = 1, color = sns.color\_palette())  
  
plt.xlim(-2,6)  
plt.ylim(-2,9)  
plt.axvline(x=0, color=**'grey'**)  
plt.axhline(y=0, color=**'grey'**)  
plt.text(-1, 3.5, **r'$||\vec{u}||$'**, color = sns.color\_palette()[0], size =20)  
plt.text(2.5, 7.5, **r'$||\vec{v}||$'**, color = sns.color\_palette()[1], size =20)  
plt.text(2, 2, **r'$||\vec{u}+\vec{v}||$'**, color = sns.color\_palette()[2], size =20)  
plt.show()

**Output**

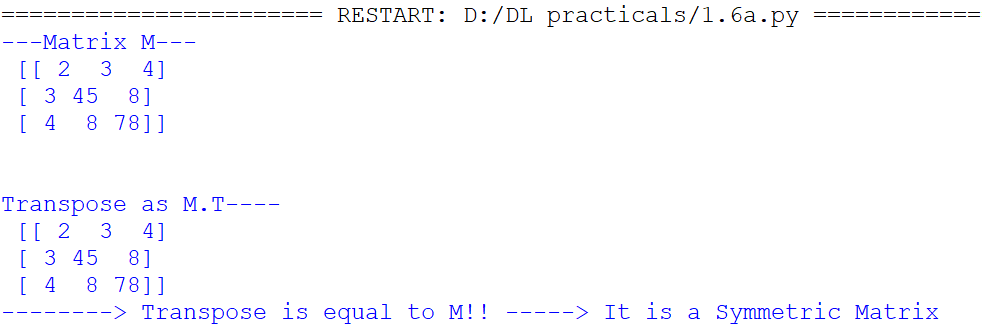
****

**[6a] Symmetric Matrix**

**Source Code:**

*# Linear Algebra Learning Sequence  
# Transpose using different Method***import** numpy **as** np  
  
M = np.array([[2,3,4], [3,45,8], [4,8,78]])  
print(**"---Matrix M---\n"**, M)  
  
*# Transposing the Matrix M*print(**'\n\nTranspose as M.T----\n'**, M.T)  
  
**if** M.T.all() == M.all():  
 print(**"--------> Transpose is equal to M!! -----> It is a Symmetric Matrix"**)  
**else**:  
 print(**"---------> Transpose is not equal o M!! ------> It is not a Symmetric Matrix"**)

**Output**

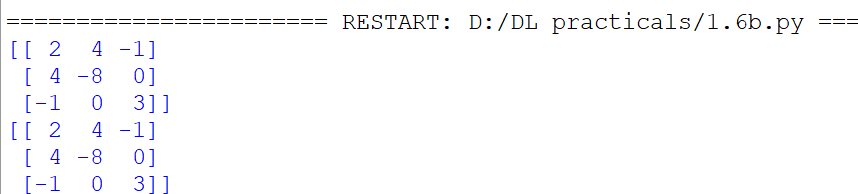


**[6b] Symmetric Matrix**

**Source Code:**

**import** numpy **as** np  
  
A = np.array([[2,4,-1],[4,-8,0],[-1,0,3]])  
print(A)  
print(A.T)

**Output**

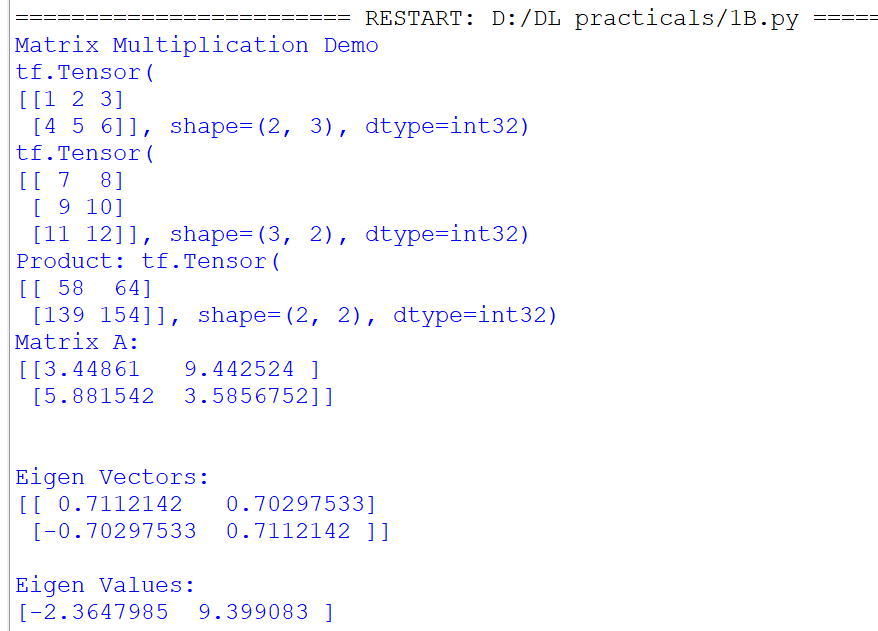


**[B] Aim: Performing matrix multiplication and finding eigen vectors and eigen values using TensorFlow.**

**Source Code:**

**import** tensorflow **as** tf  
print(**"Matrix Multiplication Demo"**)  
x=tf.constant([1,2,3,4,5,6],shape=[2,3])  
print(x)  
y=tf.constant([7,8,9,10,11,12],shape=[3,2])  
print(y)  
z=tf.matmul(x,y)  
print(**"Product:"**,z)  
e\_matrix\_A=tf.random.uniform([2,2],minval=3,maxval=10,dtype=tf.float32,name=**"matrixA"**)  
print(**"Matrix A:\n{}\n\n"**.format(e\_matrix\_A))  
eigen\_values\_A,eigen\_vectors\_A=tf.linalg.eigh(e\_matrix\_A)  
print(**"Eigen Vectors:\n{}\n\nEigen Values:\n{}\n"**.format(eigen\_vectors\_A,eigen\_values\_A))

**Output**



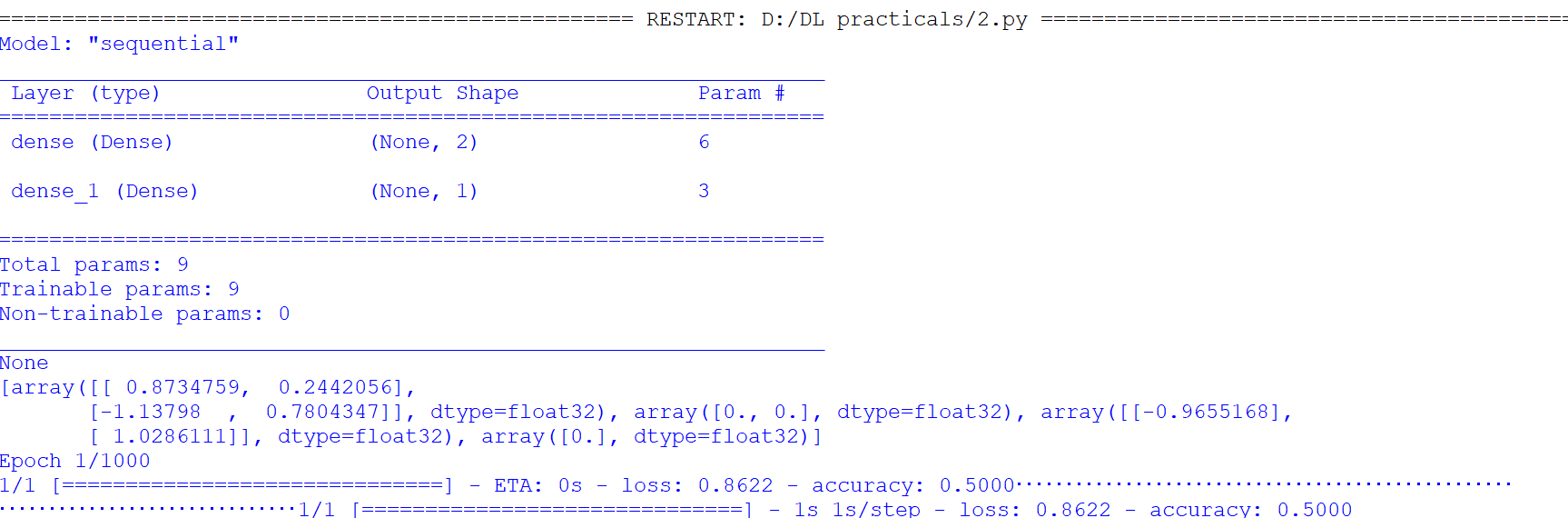
**PRACTICAL** **2**

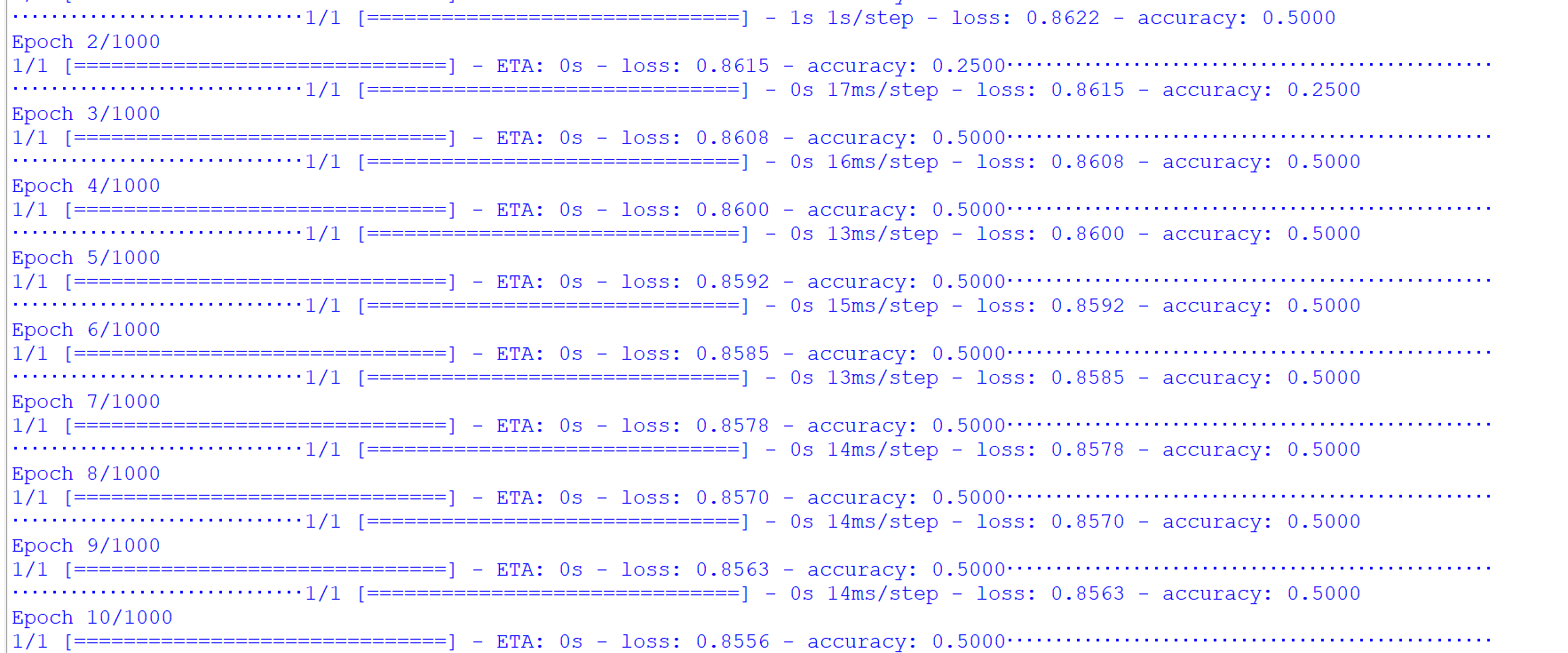
**Aim: Solving XOR problem using deep feed forward network.**

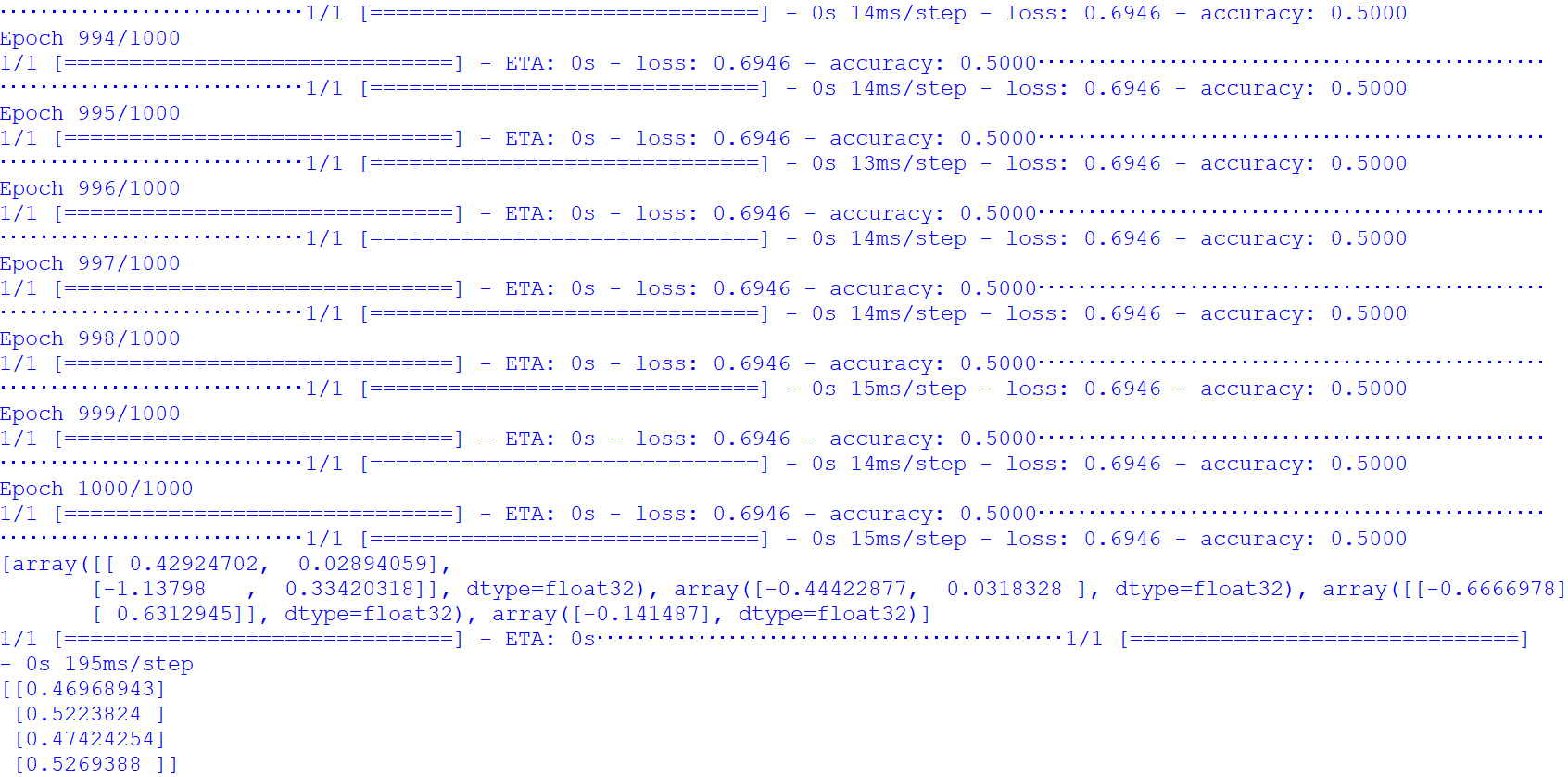
**Source Code**

**import** numpy **as** np  
**from** keras.layers **import** Dense  
**from** keras.models **import** Sequential  
  
  
model = Sequential()  
model.add(Dense(units=2, activation=**'relu'**, input\_dim=2))  
model.add(Dense(units=1, activation=**'sigmoid'**))  
model.compile(loss=**'binary\_crossentropy'**, optimizer=**'adam'**, metrics=[**'accuracy'**])  
print(model.summary())  
print(model.get\_weights())  
X = np.array([[0., 0.], [0., 1.], [1., 0.], [1., 1.]])  
Y = np.array([0., 1., 1., 0.])  
model.fit(X, Y, epochs=1000, batch\_size=4)  
print(model.get\_weights())  
print(model.predict(X, batch\_size=4))

**Output**

****



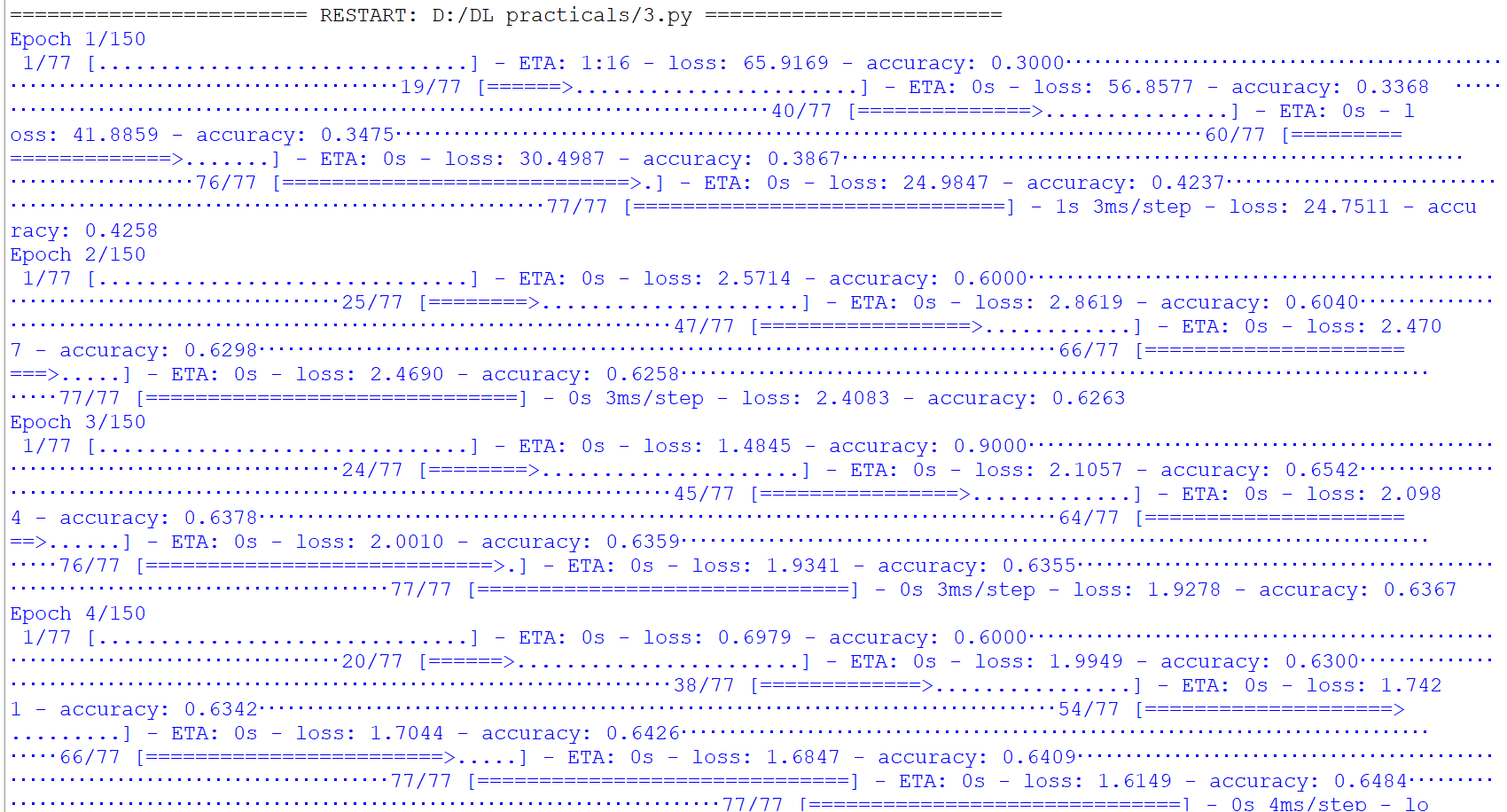


**PRACTICAL 3**

**Aim: Implementing deep neural network for performing binary classification task.**

**Source Code**

**from** numpy **import** loadtxt  
**from** keras.models **import** Sequential  
**from** keras.layers **import** Dense  
  
dataset = loadtxt(**"D:/DL practicals/pima-indians-diabetes.csv"** , delimiter = **','**)  
X=dataset[:,0:8]  
Y=dataset[:,8]  
model=Sequential()  
model.add(Dense(12,input\_dim=8,activation=**'relu'**))  
model.add(Dense(8,activation=**'relu'**))  
model.add(Dense(1,activation=**'sigmoid'**))  
model.compile(loss=**'binary\_crossentropy'**,optimizer=**'adam'**,metrics=[**'accuracy'**])  
model.fit(X,Y,epochs =150,batch\_size=10)  
accuracy=model.evaluate(X,Y)  
print(**'Accuracy of model is'**, (accuracy\*100))  
prediction=model.predict(X)  
exec(**"for i in range(5):print(X[i].tolist(),prediction[i],Y[i])"**)

**Output** ****

****

**PRACTICAL 4**

**[A] Aim: Using a deep field forward network with two hidden layers for**

**performing classification and predicting the probability of class.**

**Source Code**

**import** numpy **as** np

**from** sklearn.datasets **import** load\_iris

**from** keras.models **import** Sequential

**from** keras.layers **import** Dense

**from** keras.utils **import** to\_categorical

**from** sklearn.model\_selection **import** train\_test\_split

*# Load the iris dataset*

iris = load\_iris()

X = iris.data

y = iris.target

*# Convert target variable to one-hot encoded format*

y = to\_categorical(y)

*# Split the data into training and test sets*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

*# Define the model*

model = Sequential()

model.add(Dense(64, input\_dim=X\_train.shape[1], activation=**'relu'**))

model.add(Dense(32, activation=**'relu'**))

model.add(Dense(y\_train.shape[1], activation=**'softmax'**))

*# Compile the model*

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

*# Train the model*

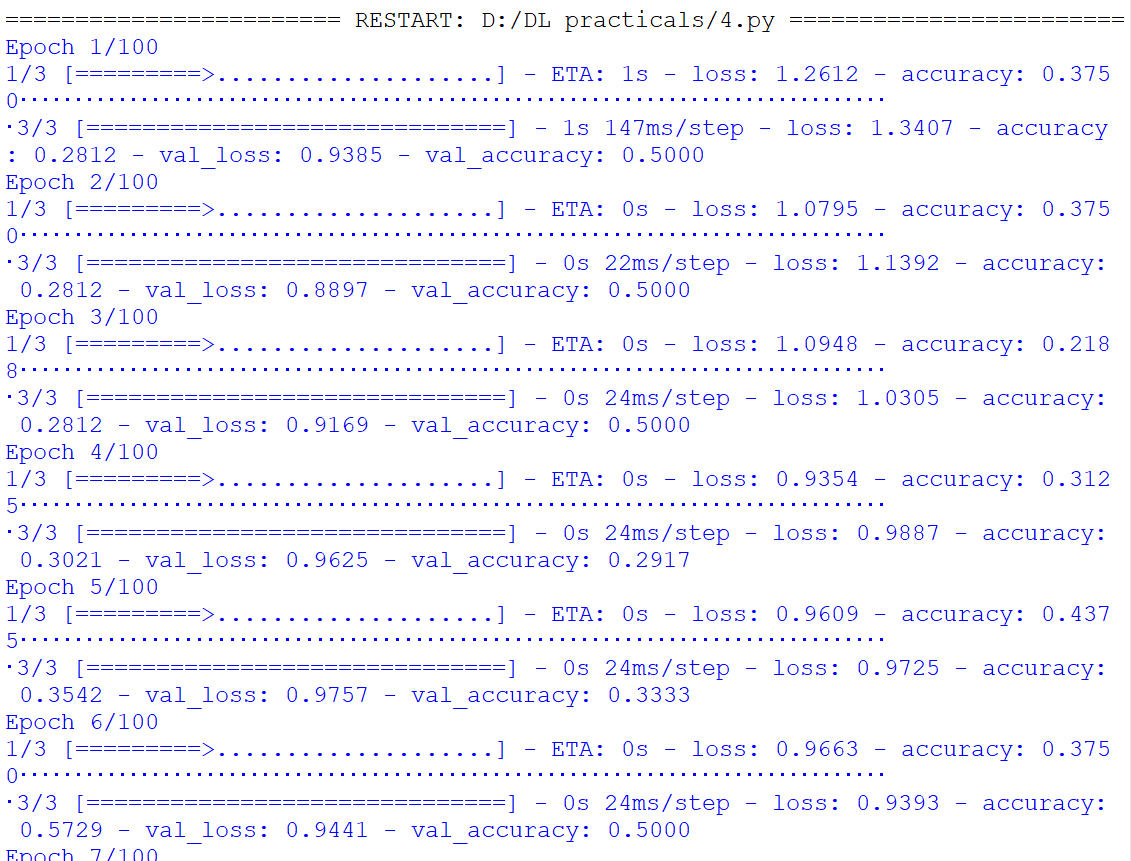
model.fit(X\_train, y\_train, epochs=100, batch\_size=32, validation\_split=0.2)

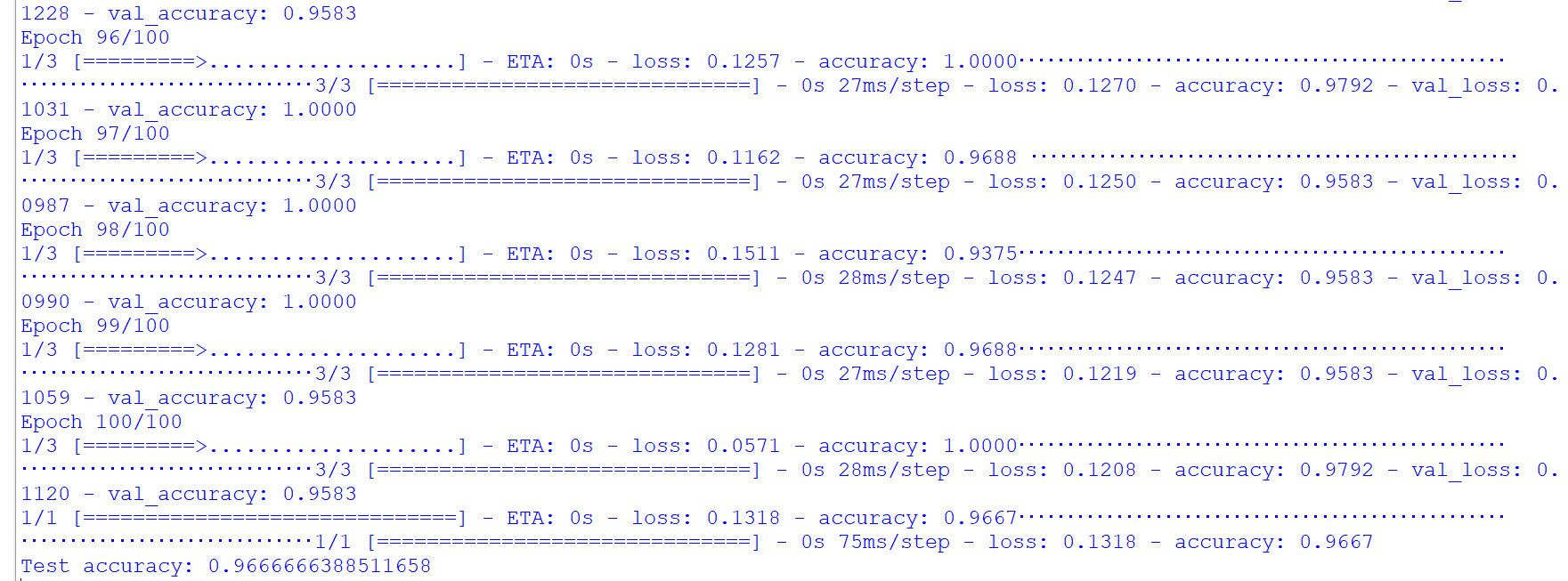
*# Evaluate the model on the test set*

loss, accuracy = model.evaluate(X\_test, y\_test)

print(**'Test accuracy:'**, accuracy)

**Output**

****

****

**[B] Aim: Using a deep field forward network with two hidden layers for performing classification and predicting the probability of class.**

**Source Code**

from tensorflow import keras

from keras.models import Sequential

from keras.layers import Dense

from sklearn.datasets import make\_blobs

from sklearn.preprocessing import MinMaxScaler

X,Y=make\_blobs(n\_samples=100,centers=2,n\_features=2,random\_state=1)

scalar=MinMaxScaler()

scalar.fit(X)

X=scalar.transform(X)

models=keras.Sequential()

models.add(Dense(4,input\_dim=2,activation='relu'))

models.add(Dense(4,activation='relu'))

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

models.compile(loss='binary\_crossentropy',optimizer='adam')

models.fit(X,Y,epochs=500)

import numpy as np

Xnew,Yreal=make\_blobs(n\_samples=3,centers=2,n\_features=2,random\_state=1)

Xnew=scalar.transform(Xnew)

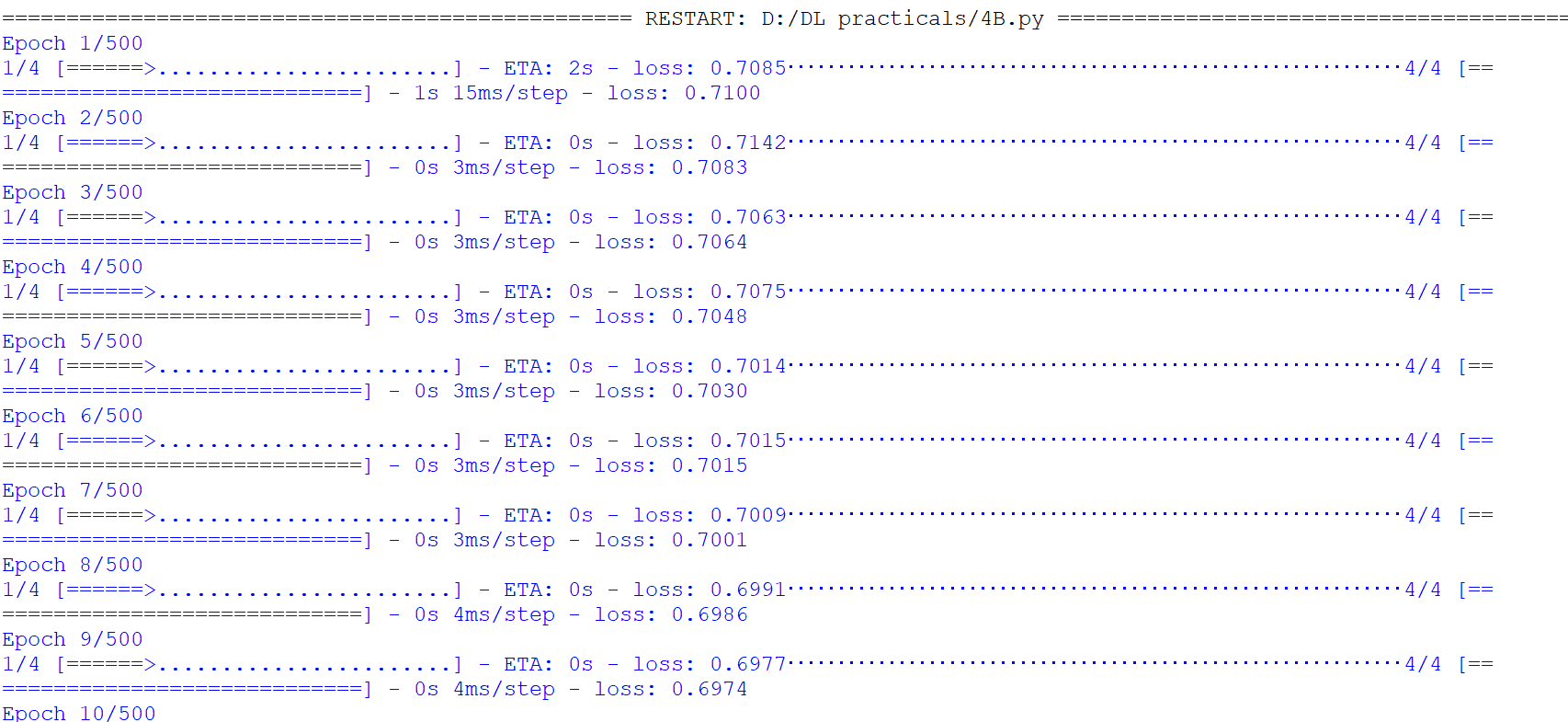
Yclass=np.argmax(models.predict(Xnew), axis=-1)

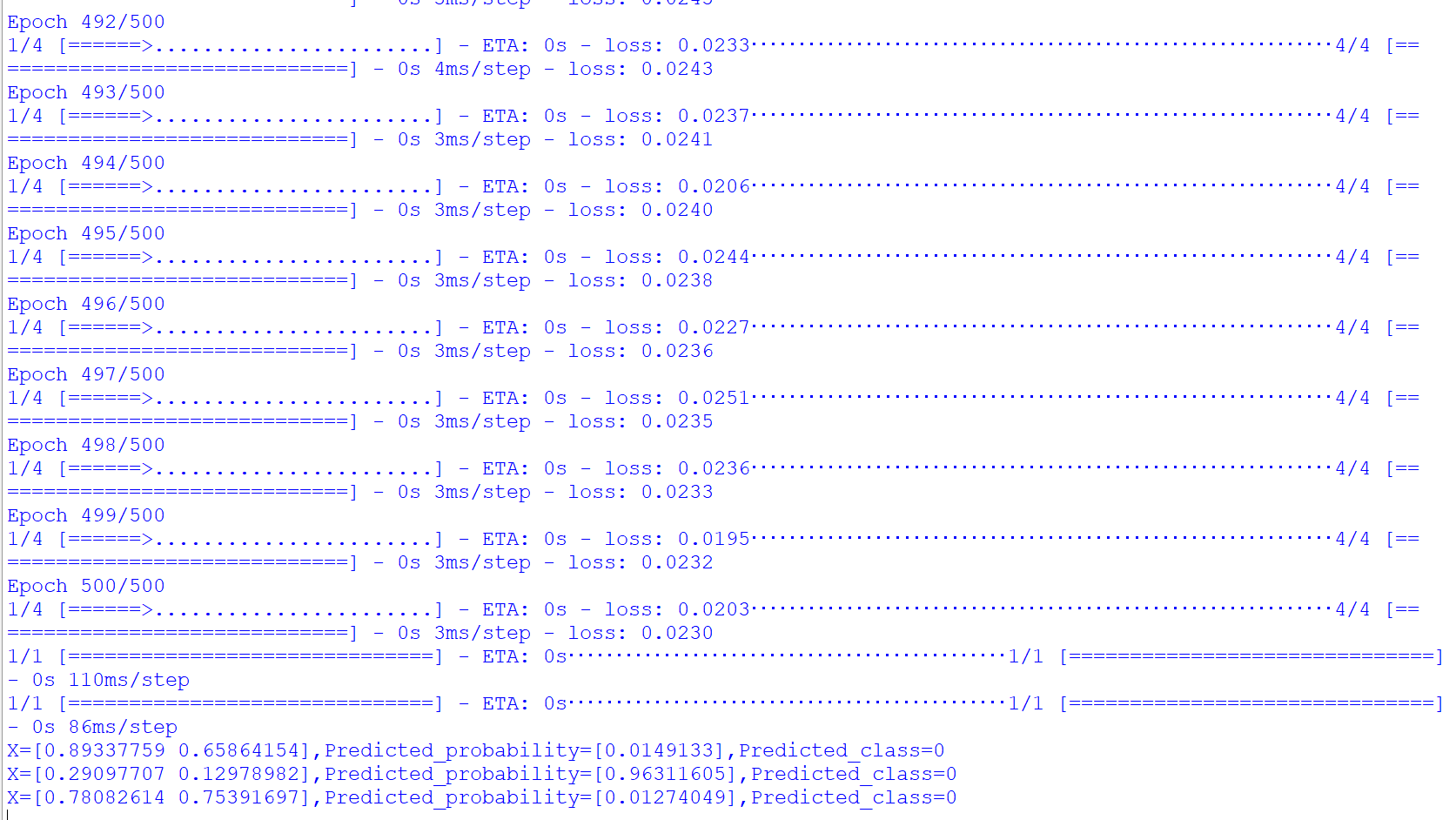
Ynew=models.predict(Xnew)

for i in range(len(Xnew)):

print("X=%s,Predicted\_probability=%s,Predicted\_class=%s"%(Xnew[i],Ynew[i],Yclass[i]))

**Output**

****

****

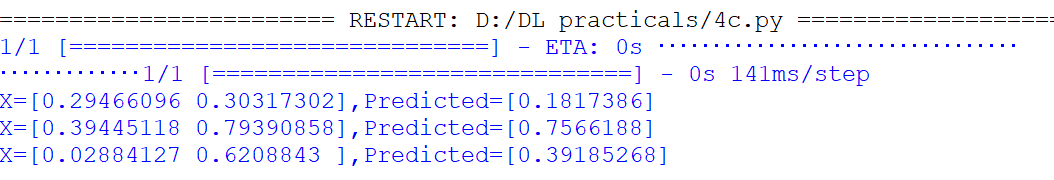
**[C] Aim: Using a deep field forward network with two hidden layers for**

**performing linear regression and predicting values.**

**Source Code**

**from** keras.models **import** Sequential  
**from** keras.layers **import** Dense  
**from** sklearn.datasets **import** make\_regression  
**from** sklearn.preprocessing **import** MinMaxScaler  
  
X,Y=make\_regression(n\_samples=100,n\_features=2,noise=0.1,random\_state=1)  
scalarX,scalarY=MinMaxScaler(),MinMaxScaler()  
  
scalarX.fit(X)  
scalarY.fit(Y.reshape(100,1))  
X=scalarX.transform(X)  
Y=scalarY.transform(Y.reshape(100,1))  
  
model=Sequential()  
model.add(Dense(4,input\_dim=2,activation=**'relu'**))  
model.add(Dense(4,activation=**'relu'**))  
model.add(Dense(1,activation=**'sigmoid'**))  
model.compile(loss=**'mse'**,optimizer=**'adam'**)  
model.fit(X,Y,epochs=1000,verbose=0)  
  
Xnew,a=make\_regression(n\_samples=3,n\_features=2,noise=0.1,random\_state=1)  
Xnew=scalarX.transform(Xnew)  
  
Ynew=model.predict(Xnew)  
  
**for** i **in** range(len(Xnew)):  
 print(**"X=%s,Predicted=%s"**%(Xnew[i],Ynew[i]))

**Output**

****

**PRACTICAL 5**

**[A] Aim: Evaluating feed forward deep network for regression using KFold cross**

**validation.**

**Source Code**

import numpy as np

from sklearn.model\_selection import KFold

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

*# Generate sample data*

X = np.random.rand(1000, 10)

y = np.sum(X, axis=1)

*# Define KFold cross-validation*

kfold = KFold(n\_splits=5, shuffle=True, random\_state=42)

*# Initialize list to store evaluation metrics*

eval\_metrics = []

*# Iterate through each fold*

for train\_index, test\_index in kfold.split(X):

*# Split data into training and testing sets*

X\_train, X\_test = X[train\_index], X[test\_index]

y\_train, y\_test = y[train\_index], y[test\_index]

*# Define and compile model*

model = Sequential()

model.add(Dense(64, activation='relu', input\_dim=10))

model.add(Dense(1))

model.compile(optimizer='adam', loss='mse', metrics=['mae'])

*# Fit model to training data*

model.fit(X\_train, y\_train, epochs=100, batch\_size=32, verbose=0)

*# Evaluate model on testing data*

eval\_metrics.append(model.evaluate(X\_test, y\_test))

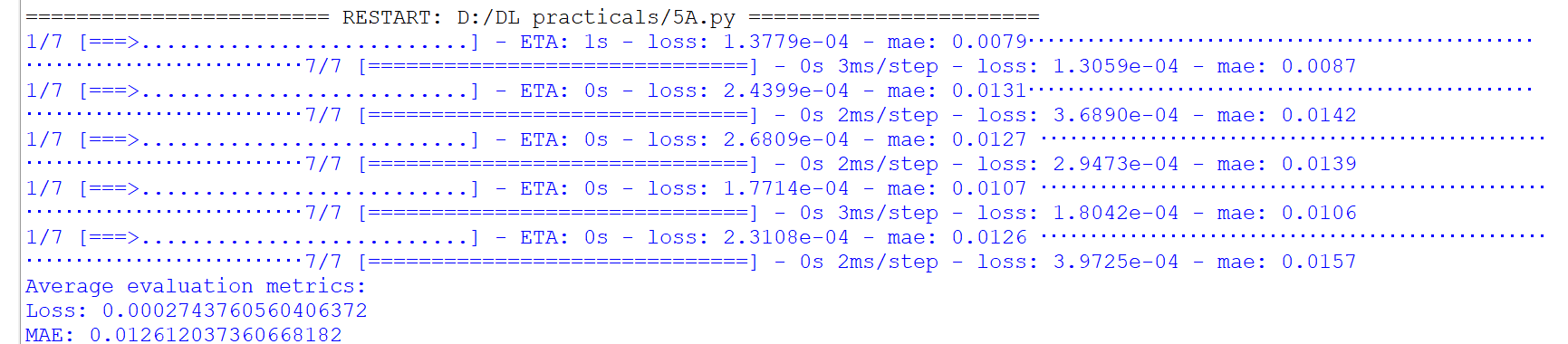
*# Print average evaluation metrics across all folds*

print("Average evaluation metrics:")

print("Loss:", np.mean([m[0] for m in eval\_metrics]))

print("MAE:", np.mean([m[1] for m in eval\_metrics]))

**Output**

****

**[B] Aim: Evaluating feed forward deep network for multiclass Classification using KFold cross-validation.**

**Source Code**

*# multi-class classification with Keras*

import pandas

from keras.models import Sequential

from keras.layers import Dense

from keras.wrappers.scikit\_learn import KerasClassifier

from keras.utils import np\_utils

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import KFold

from sklearn.preprocessing import LabelEncoder

from sklearn import datasets

from sklearn.pipeline import Pipeline

*# load dataset*

dataset = datasets.load\_iris()

X = dataset.data[:,0:4].astype(float)

Y = dataset.target

*# encode class values as integers*

encoder = LabelEncoder()

encoder.fit(Y)

encoded\_Y = encoder.transform(Y)

*# convert integers to dummy variables (i.e. one hot encoded)*

dummy\_y = np\_utils.to\_categorical(encoded\_Y)

*# define baseline model*

def baseline\_model():

*# create model*

model = Sequential()

   model.add(Dense(8, input\_dim=4, activation='relu'))

   model.add(Dense(3, activation='softmax'))

*# Compile model*

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

   return model

estimator = KerasClassifier(build\_fn=baseline\_model, epochs=200, batch\_size=5, verbose=0)

kfold = KFold(n\_splits=10, shuffle=True)

results = cross\_val\_score(estimator, X, dummy\_y, cv=kfold)

print("Baseline: %.2f%% (%.2f%%)" % (results.mean()\*100, results.std()\*100))

**Output**



**PRACTICAL 6**

**[A] Aim: Implementing regularization to avoid overfitting in binary classification.**

**Source Code**

from matplotlib import pyplot

from sklearn.datasets import make\_moons

from keras.models import Sequential

from keras.layers import Dense

X,Y=make\_moons(n\_samples=100,noise=0.2,random\_state=1)

n\_train=30

trainX,testX=X[:n\_train,:],X[n\_train:]

trainY,testY=Y[:n\_train],Y[n\_train:]

model= Sequential()

model.add(Dense(500,input\_dim=2,activation='relu'))

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

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

history=model.fit(trainX,trainY,validation\_data=(testX,testY),epochs=4000)

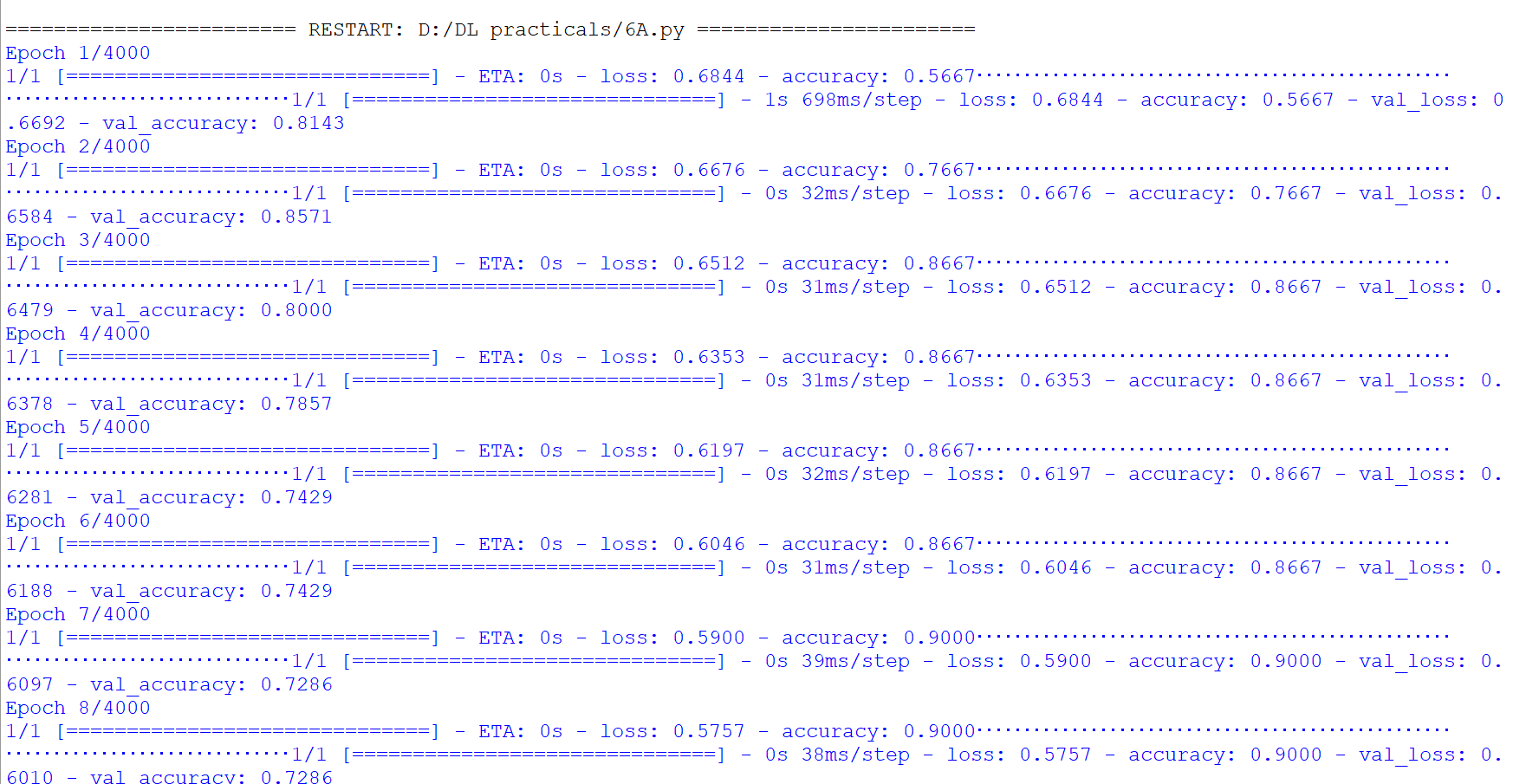
pyplot.plot(history.history['accuracy'],label='train')

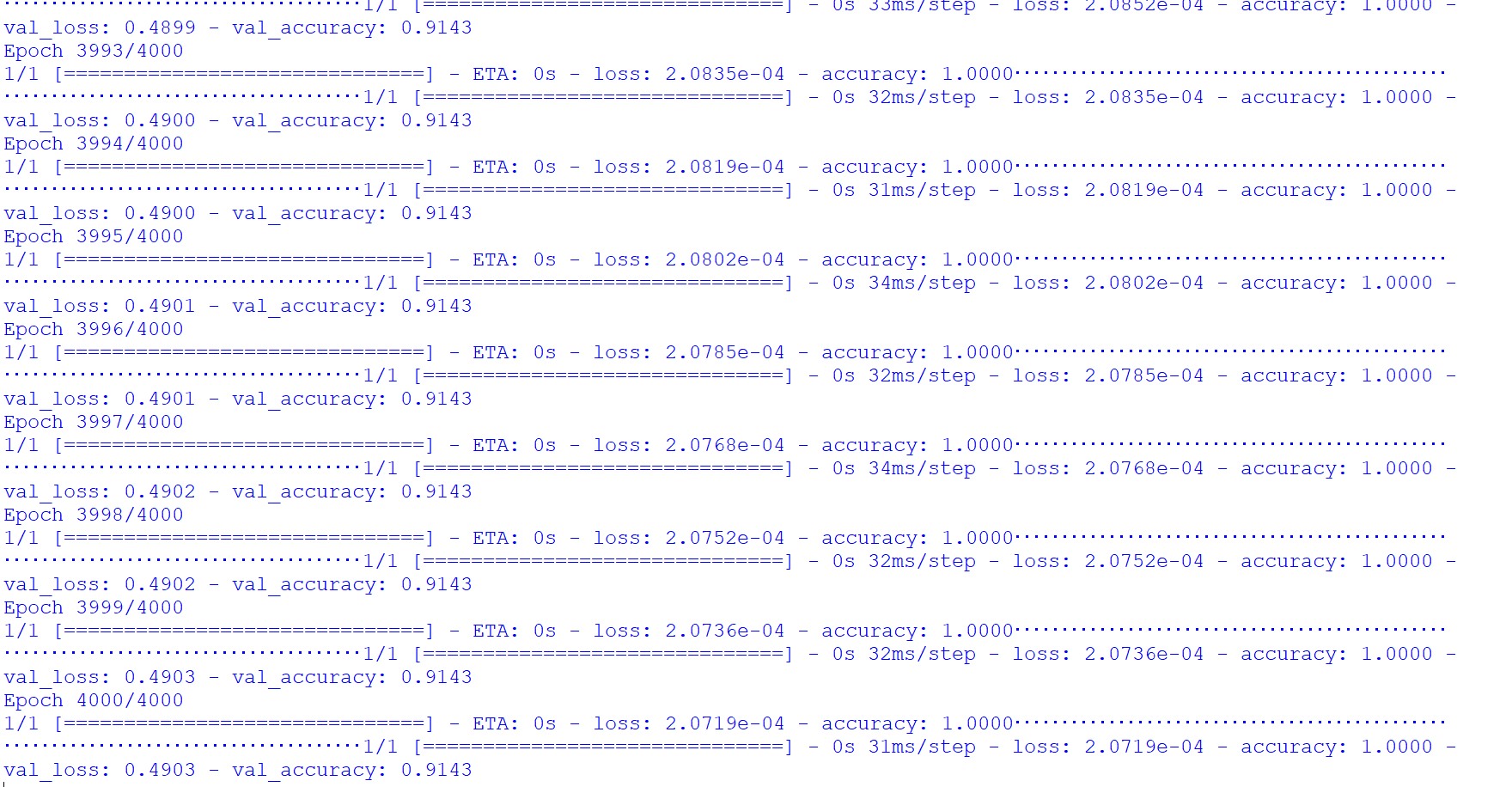
pyplot.plot(history.history['val\_accuracy'],label='test')

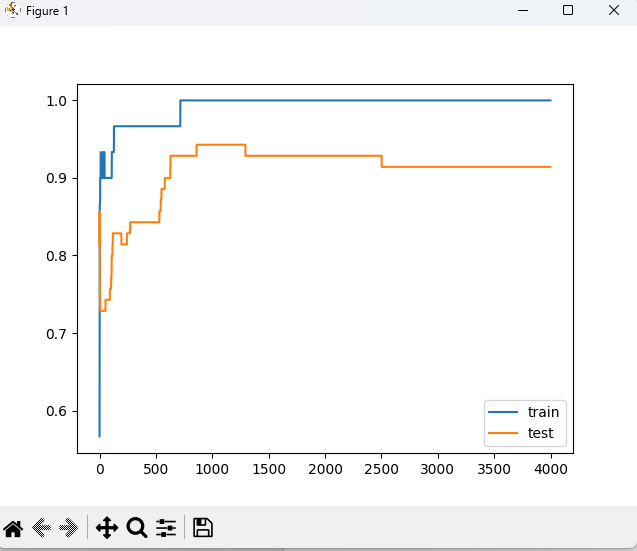
pyplot.legend()

pyplot.show()

**Output**

****

****

****

**[B] Aim: Implementing L2 regularization**

**Source Code**

from matplotlib import pyplot

from sklearn.datasets import make\_moons

from keras.models import Sequential

from keras.layers import Dense

from keras.regularizers import l2

X,Y=make\_moons(n\_samples=100,noise=0.2,random\_state=1)

n\_train=30

trainX,testX=X[:n\_train,:],X[n\_train:]

trainY,testY=Y[:n\_train],Y[n\_train:]

model= Sequential()

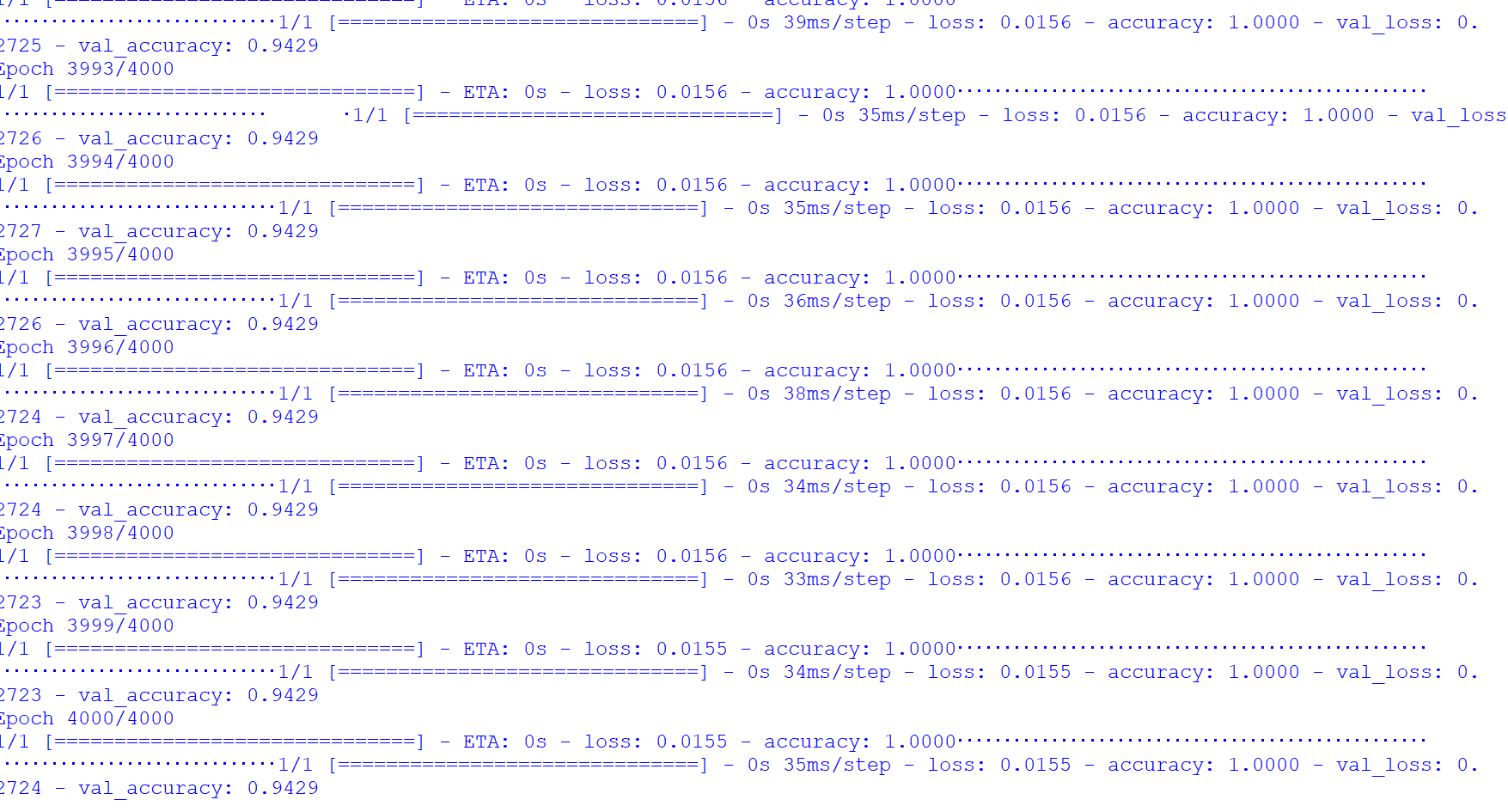
model.add(Dense(500,input\_dim=2,activation='relu',kernel\_regularizer=l2(0.001)))

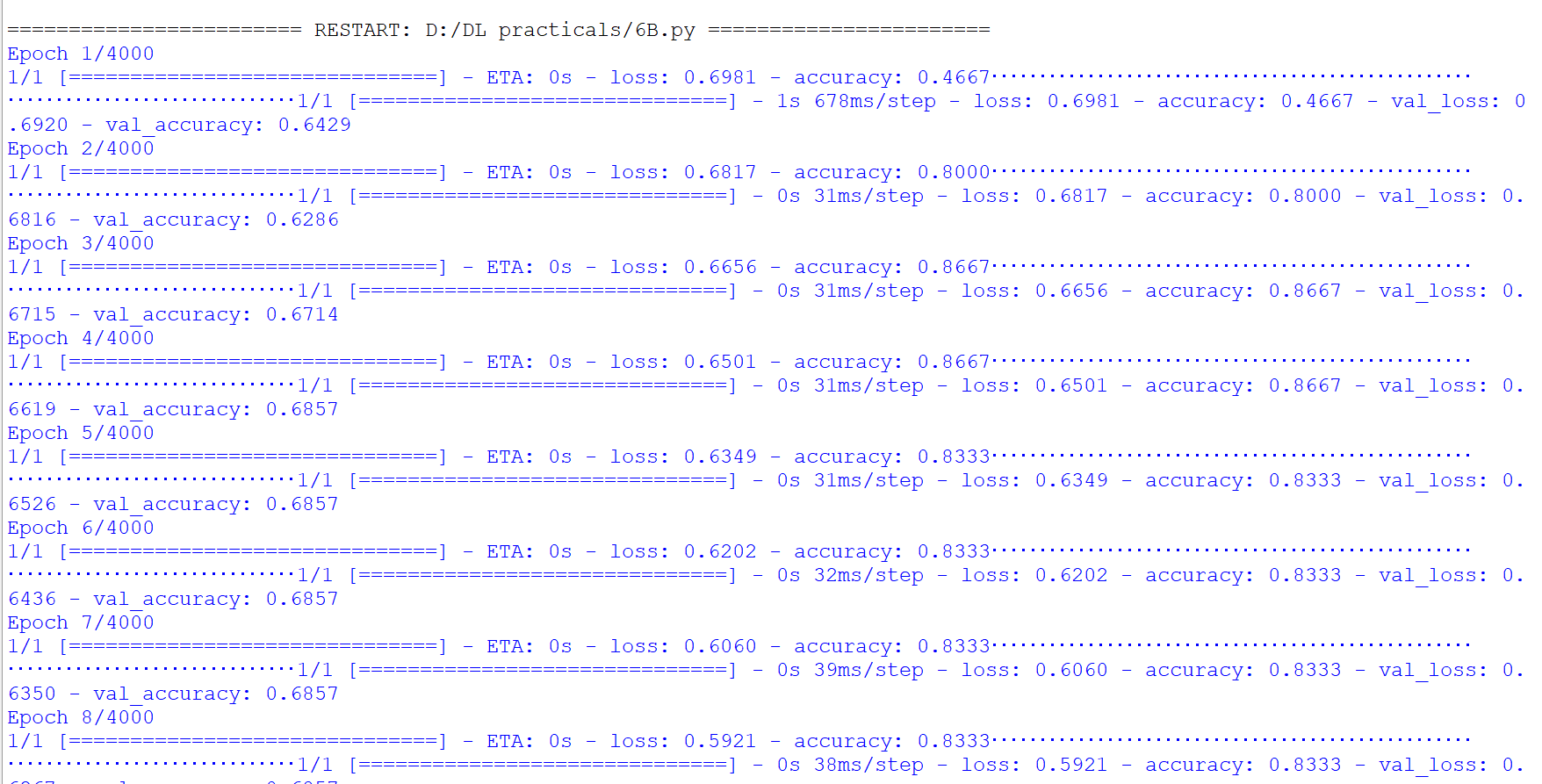
model.add(Dense(1,activation='sigmoid'))

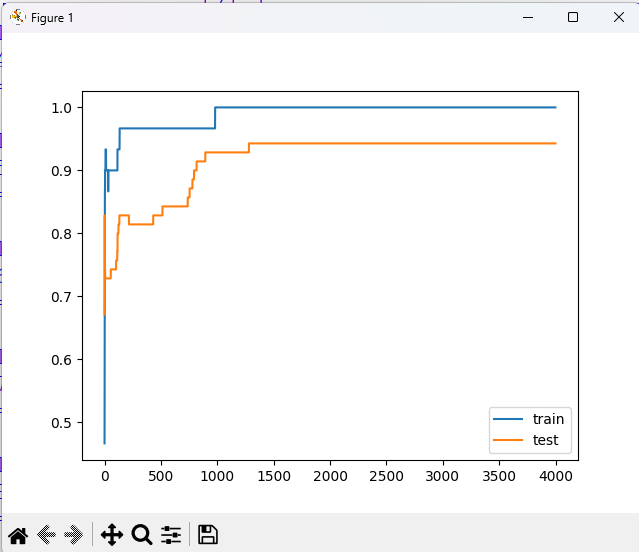
model.compile(loss='binary\_crossentropy',optimizer='adam',metrics=['accuracy'])

history=model.fit(trainX,trainY,validation\_data=(testX,testY),epochs=4000)

**Output**

****

****

****

**[C] Aim: Replacing L2 regularizer with L1 regularizer**

**Source Code**

from matplotlib import pyplot

from sklearn.datasets import make\_moons

from keras.models import Sequential

from keras.layers import Dense

from keras.regularizers import l1\_l2

X,Y=make\_moons(n\_samples=100,noise=0.2,random\_state=1)

n\_train=30

trainX,testX=X[:n\_train,:],X[n\_train:]

trainY,testY=Y[:n\_train],Y[n\_train:]

model= Sequential()

model.add(Dense(500,input\_dim=2,activation='relu',kernel\_regularizer=l1\_l2(l1=0.001,l2=0.001)))

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

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

history=model.fit(trainX,trainY,validation\_data=(testX,testY),epochs=4000)

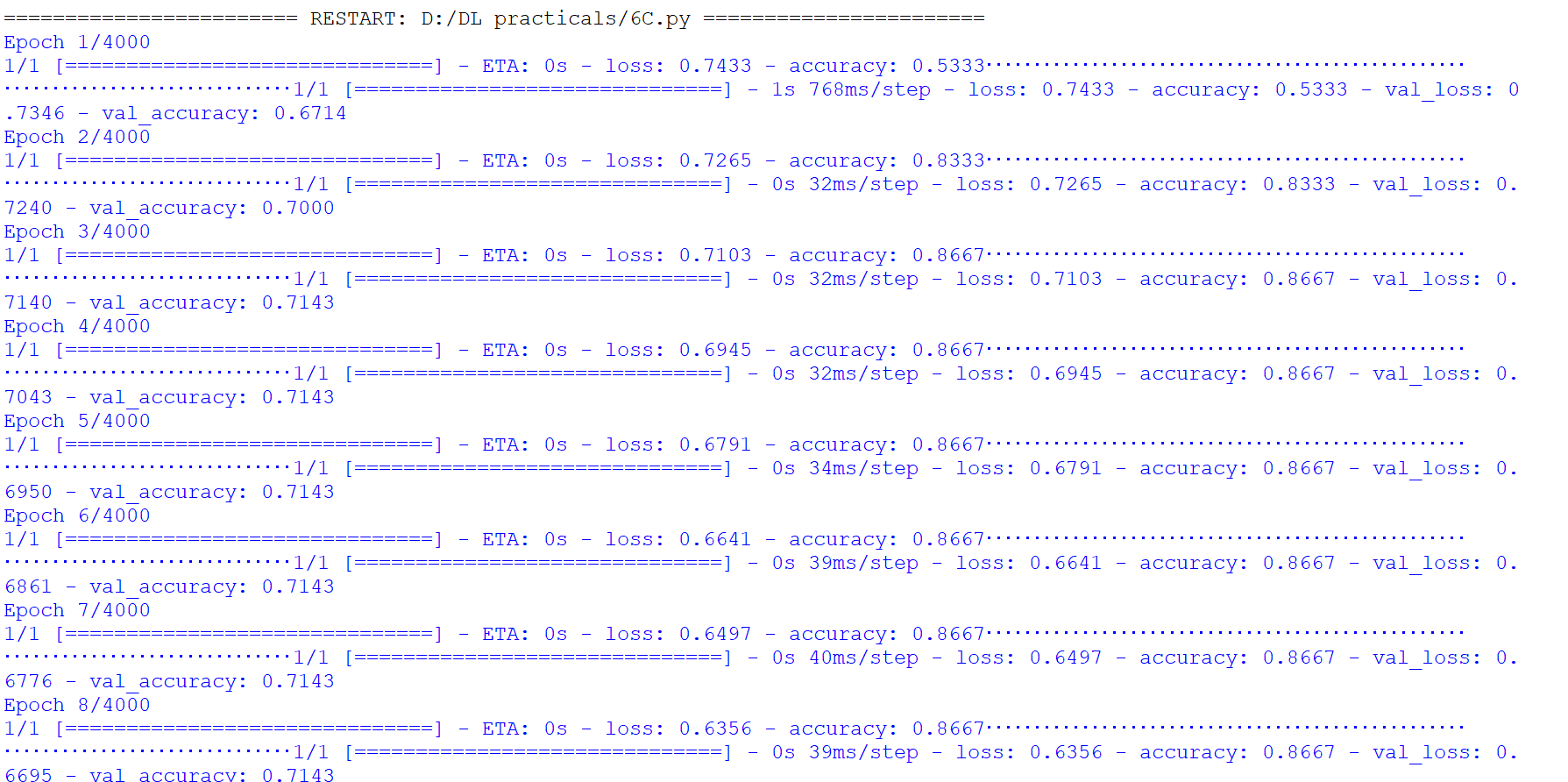
pyplot.plot(history.history['accuracy'],label='train')

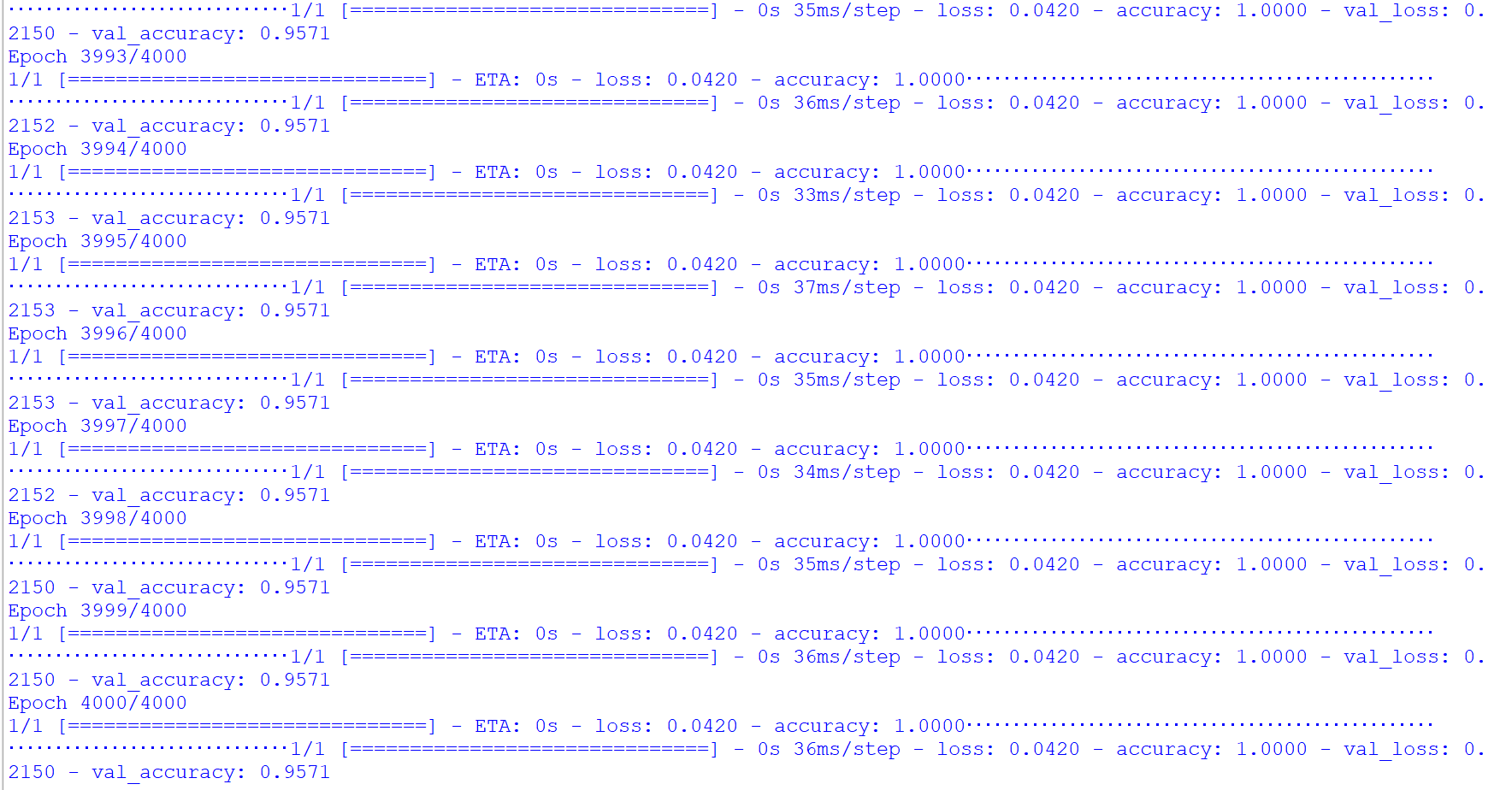
pyplot.plot(history.history['val\_accuracy'],label='test')

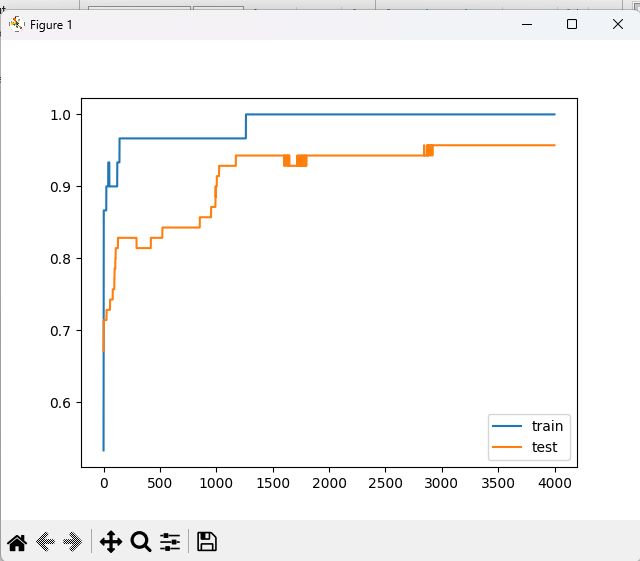
pyplot.legend()

pyplot.show()

**Output**

****

****



**PRACTICAL 7**

**[B] Aim:Demonstrate recurrent neural network that learns to perform sequence analysis for stock price.**

**Source Code**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import LSTM

from keras.layers import Dropout

from sklearn.preprocessing import MinMaxScaler

dataset\_train=pd.read\_csv("C:/Users/Admin/Downloads/Google\_Stock\_Price\_Train.csv")

*#print(dataset\_train)*

training\_set=dataset\_train.iloc[:,1:2].values

print(training\_set)

sc=MinMaxScaler(feature\_range=(0,1))

training\_set\_scaled=sc.fit\_transform(training\_set)

print(training\_set\_scaled)

X\_train=[]

Y\_train=[]

for i in range(60,1258):

X\_train.append(training\_set\_scaled[i-60:i,0])

Y\_train.append(training\_set\_scaled[i,0])

X\_train,Y\_train=np.array(X\_train),np.array(Y\_train)

print(X\_train)

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

print(Y\_train)

X\_train=np.reshape(X\_train,(X\_train.shape[0],X\_train.shape[1],1))

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

print(X\_train)

regressor=Sequential()

regressor.add(LSTM(units=50,return\_sequences=True,input\_shape=(X\_train.shape[1],1)))

regressor.add(Dropout(0.2))

regressor.add(LSTM(units=50,return\_sequences=True))

regressor.add(Dropout(0.2))

regressor.add(LSTM(units=50,return\_sequences=True))

regressor.add(Dropout(0.2))

regressor.add(LSTM(units=50))

regressor.add(Dropout(0.2))

regressor.add(Dense(units=1))

regressor.compile(optimizer='adam',loss='mean\_squared\_error')

regressor.fit(X\_train,Y\_train,epochs=100,batch\_size=32)

dataset\_test=pd.read\_csv("C:/Users/Admin/Downloads/Google\_Stock\_Price\_Test.csv")

real\_stock\_price=dataset\_test.iloc[:,1:2].values

dataset\_total=pd.concat((dataset\_train['Open'],dataset\_test['Open']),axis=0)

inputs=dataset\_total[len(dataset\_total)-len(dataset\_test)-60:].values

inputs=inputs.reshape(-1,1)

inputs=sc.transform(inputs)

X\_test=[]

for i in range(60,80):

X\_test.append(inputs[i-60:i,0])

X\_test=np.array(X\_test)

X\_test=np.reshape(X\_test,(X\_test.shape[0],X\_test.shape[1],1))

predicted\_stock\_price=regressor.predict(X\_test)

predicted\_stock\_price=sc.inverse\_transform(predicted\_stock\_price)

plt.plot(real\_stock\_price,color='red',label='real google stock price')

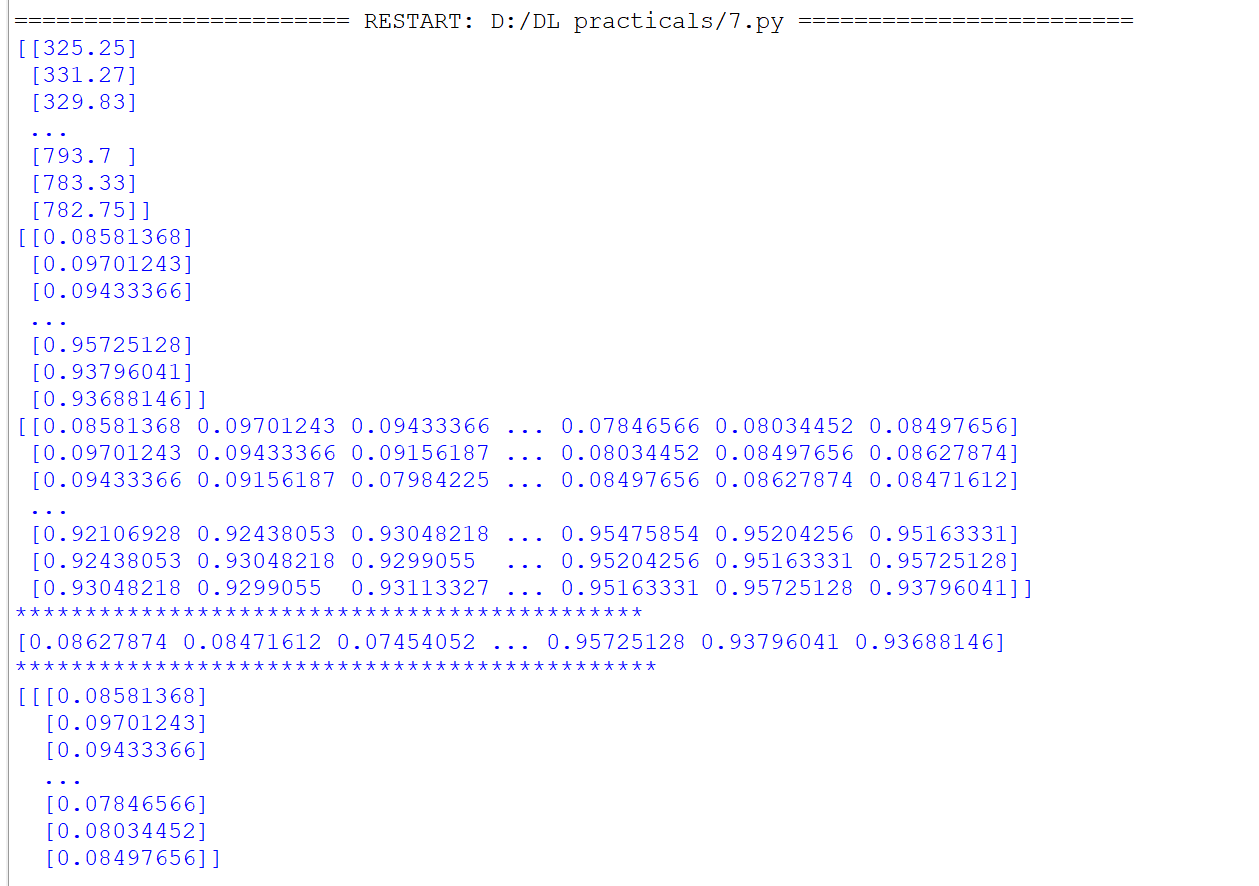
plt.plot(predicted\_stock\_price,color='blue',label='predicted stock price')

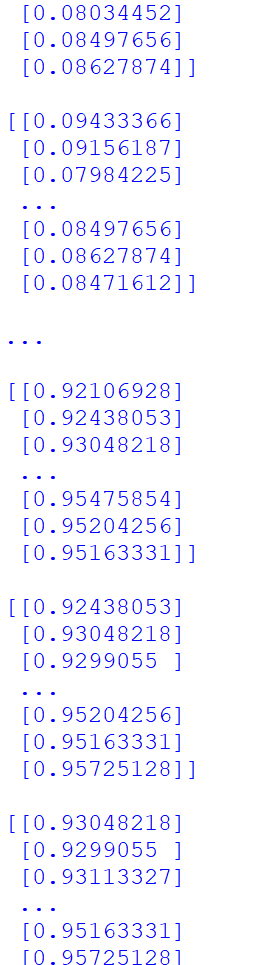
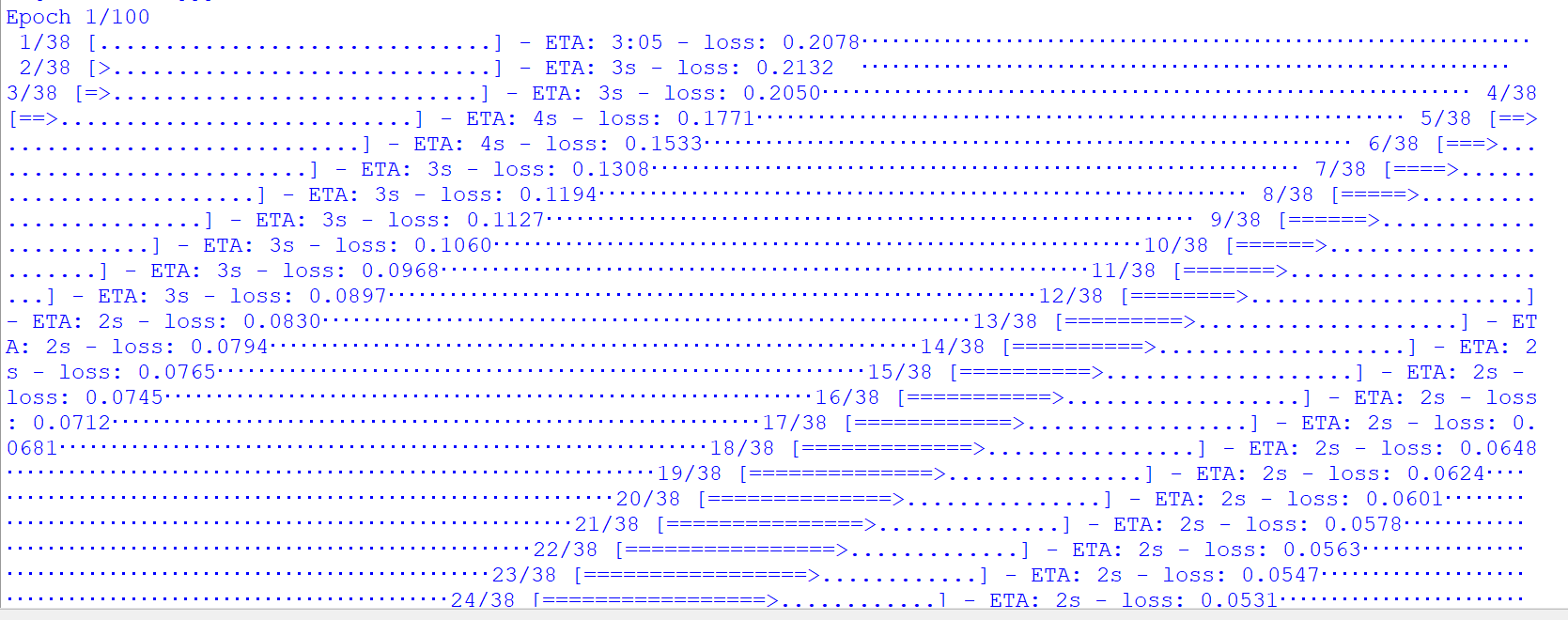
plt.xlabel('time')

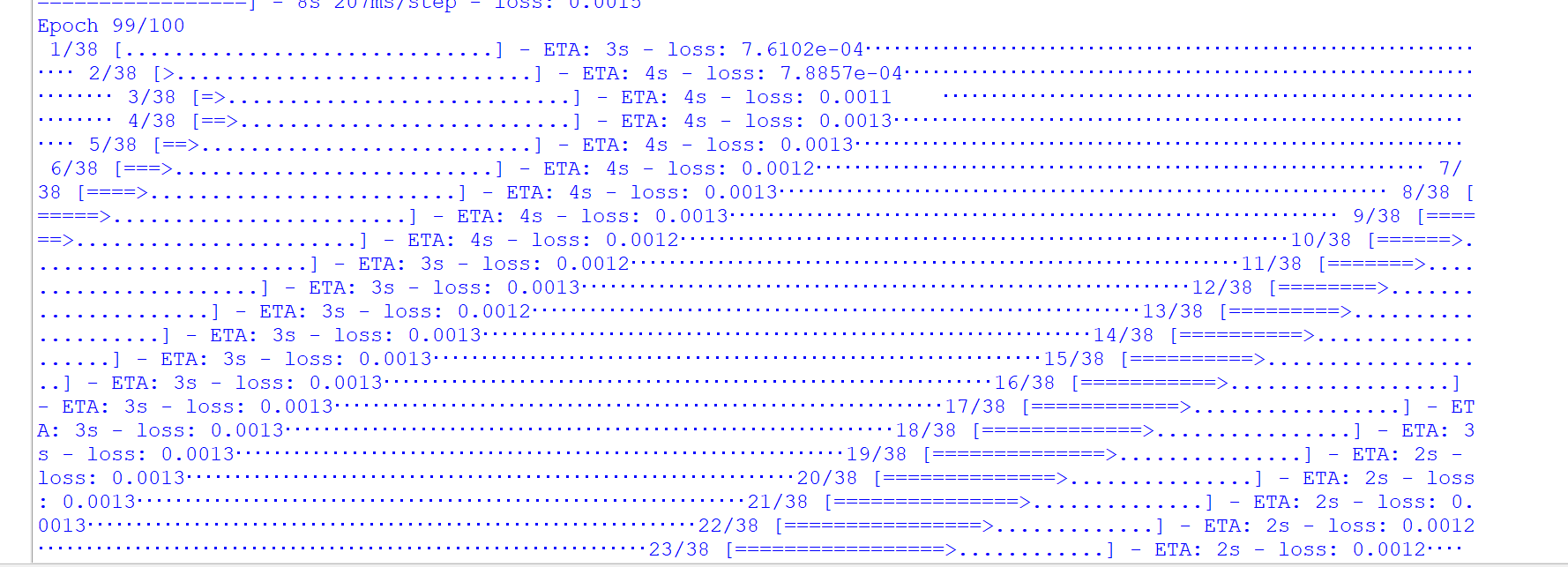
plt.ylabel('google stock price')

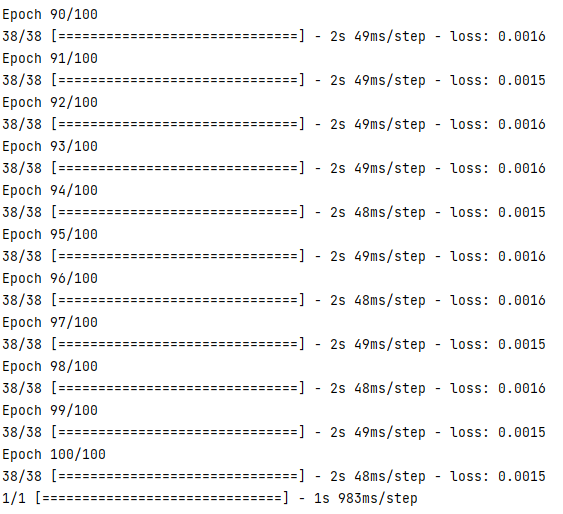
plt.legend()

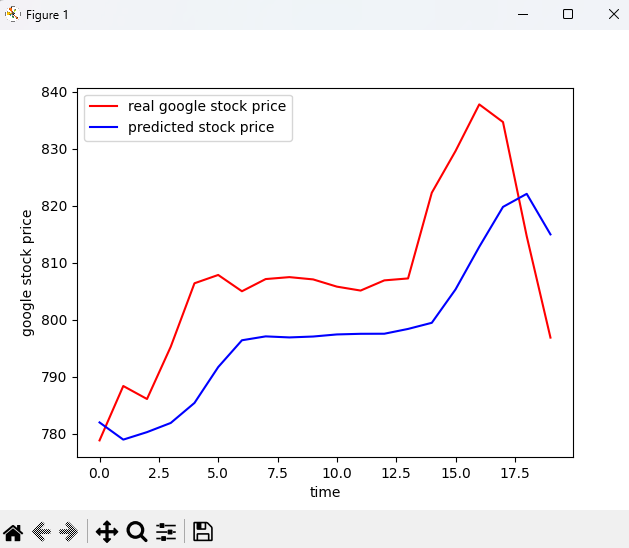
plt.show()

**Output** ****

**** 

****

****

****

**PRACTICAL 8**

**Aim: Performing encoding and decoding of images using deep autoencoder.**

**Source Code**

import keras

from keras import layers

from keras.datasets import mnist

import numpy as np

encoding\_dim=32

*#this is our input image*

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

*#"encoded" is the encoded representation of the input*

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

*#"decoded" is the lossy reconstruction of the input*

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

*#creating autoencoder model*

autoencoder=keras.Model(input\_img,decoded)

*#create the encoder model*

encoder=keras.Model(input\_img,encoded)

encoded\_input=keras.Input(shape=(encoding\_dim,))

*#Retrive the last layer of the autoencoder model*

decoder\_layer=autoencoder.layers[-1]

*#create the decoder model*

decoder=keras.Model(encoded\_input,decoder\_layer(encoded\_input))

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

*#scale and make train and test dataset*

(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)

*#train autoencoder with training dataset*

autoencoder.fit(X\_train,X\_train, epochs=50, batch\_size=256, shuffle=True, validation\_data=(X\_test,X\_test))

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

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

import matplotlib.pyplot as plt

n = 10 *# How many digits we will display*

plt.figure(figsize=(40, 4))

for i in range(10):

ax = plt.subplot(3, 20, i + 1)

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

plt.gray()

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

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

ax = plt.subplot(3, 20, i + 1 + 20)

plt.imshow(encoded\_imgs[i].reshape(8, 4))

plt.gray()

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

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

ax = plt.subplot(3, 20, 2 \* 20 + i + 1)

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

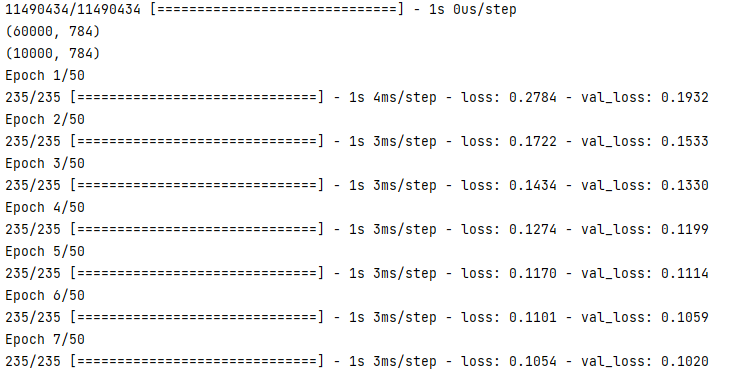
plt.gray()

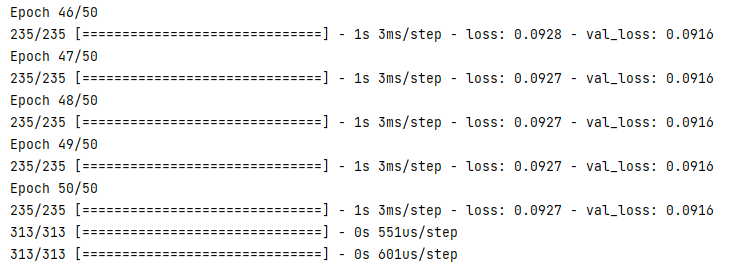
ax.get\_xaxis().set\_visible(False)

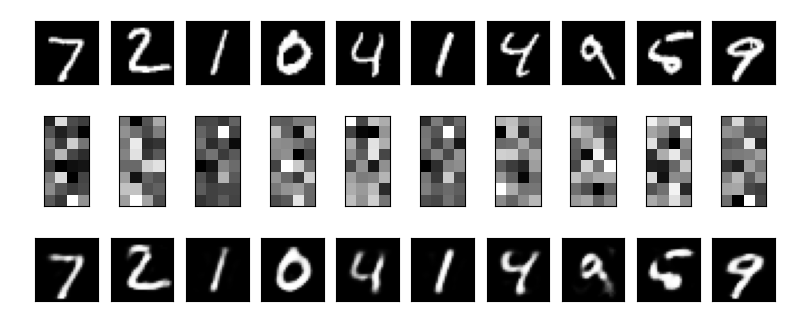
ax.get\_yaxis().set\_visible(False)

plt.show()

**Output**

****

****

****

**PRACTICAL 9**

**Aim: Implementation of convolutional neural network to predict numbers from number images**

**Source Code**

from keras.datasets import mnist

from keras.utils import to\_categorical

from keras.models import Sequential

from keras.layers import Dense,Conv2D,Flatten

import matplotlib.pyplot as plt

*#download mnist data and split into train and test sets*

(X\_train,Y\_train),(X\_test,Y\_test)=mnist.load\_data()

*#plot the first image in the dataset*

plt.imshow(X\_train[0])

plt.show()

print(X\_train[0].shape)

X\_train=X\_train.reshape(60000,28,28,1)

X\_test=X\_test.reshape(10000,28,28,1)

Y\_train=to\_categorical(Y\_train)

Y\_test=to\_categorical(Y\_test)

Y\_train[0]

print(Y\_train[0])

model=Sequential()

*#add model layers*

*#learn image features*

model.add(Conv2D(64,kernel\_size=3,activation='relu',input\_shape=(28,28,1)))

model.add(Conv2D(32,kernel\_size=3,activation='relu'))

model.add(Flatten())

model.add(Dense(10,activation='softmax'))

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

*#train*

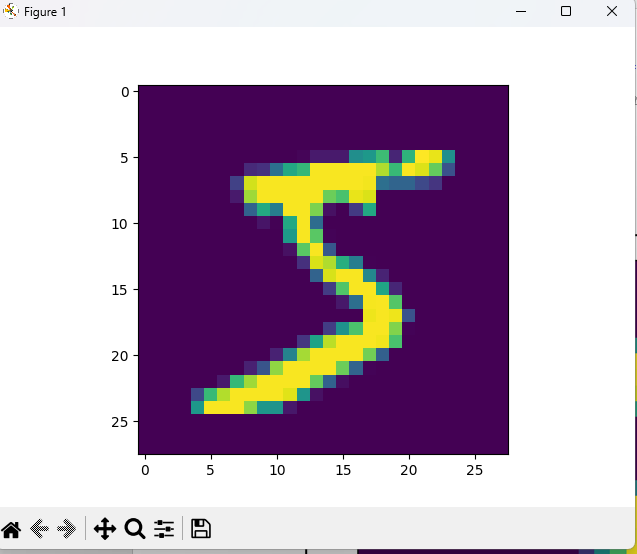
model.fit(X\_train,Y\_train,validation\_data=(X\_test,Y\_test),epochs=3)

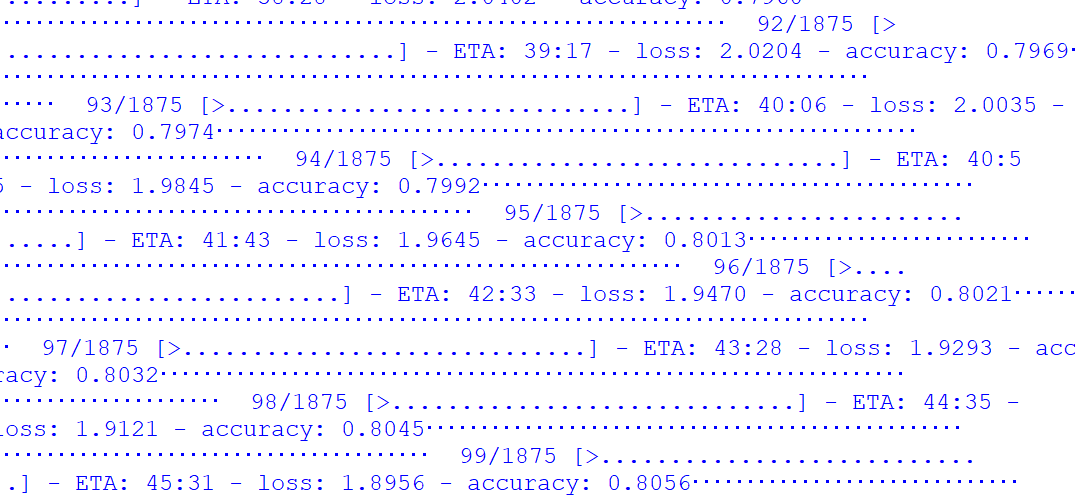
print(model.predict(X\_test[:4]))

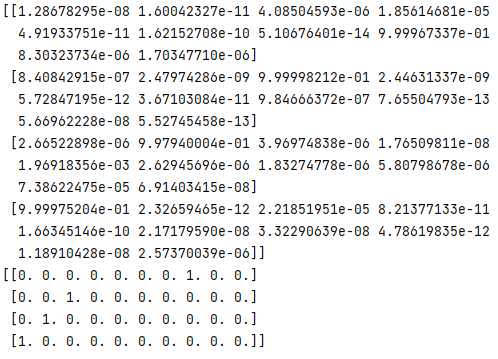
*#actual results for 1st 4 images in the test set*

print(Y\_test[:4])

**Output**

****

****

****

**PRACTICAL 10**

**Aim: Denoising of images using autoencoder.**

**Source Code**

import keras

from keras.datasets import mnist

from keras import layers

import numpy as np

from keras.callbacks import TensorBoard

import matplotlib.pyplot as plt

(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(1,n+1):

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

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()

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

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

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

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

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

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

x=layers.UpSampling2D((2,2))(x)

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

x=layers.UpSampling2D((2,2))(x)

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

autoencoder=keras.Model(input\_img,decoded)

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

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

epochs=3,

batch\_size=128,

shuffle=True,

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

callbacks=[TensorBoard(log\_dir='/tmo/tb',histogram\_freq=0,write\_graph=False)])

predictions=autoencoder.predict(X\_test\_noisy)

m=10

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

for i in range(1,m+1):

ax=plt.subplot(1,m,i)

plt.imshow(predictions[i].reshape(28,28))

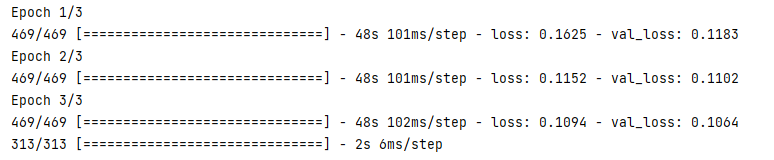
plt.gray()

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

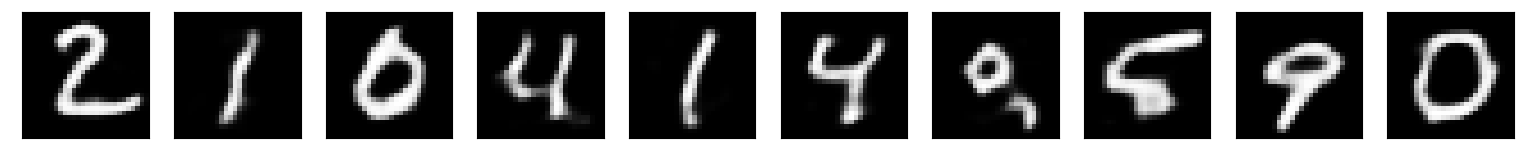
ax.get\_yaxis().set\_visible(False)

plt.show()

**Output**

****

****

****