**“DEEP LEARNING”**

**PRACTICAL MANUAL**

Submitted in partial fulfillment of the

requirements for award of the Master degree of

**MASTERS OF SCIENCE** **(INFORMATION TECHNOLOGY)**

**submitted by**

**BHAVIKA MANOHAR MULCHANDANI**

**SEAT NO - 4136294**

**Under the esteemed guidance of**

**ASST. PROF. DIVYA PANDEY**



**DEPARTMENT OF SCIENCE (INFORMATION TECHNOLOGY)**

**S.S.T. COLLEGE OF ARTS & COMMERCE**

***(Affiliated to University of Mumbai)* ULHASNAGAR-421004**

**MAHARASHTRA 2022-2023**

**Index**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr.No** | **Topic** | **Page no.** | **Date** | **Remark** |
| 1 | Performing matrix multiplication and finding eigen vectors and eigen values using TensorFlow | 3-3 |  |  |
| 2 | Solving XOR problem using deep feed forward network. | 4-5 |  |  |
| 3 | Implementing deep neural network for performing binary classification task. | 6-8 |  |  |
| 4 | 1. Aim: Using deep feed forward network with two hidden layers for performing multiclass classification and predicting the class. 2. Aim: Using a deep feed forward network with two hidden layers for performing classification and predicting the probability of class. 3. Aim: Using a deep feed forward network with two hidden layers for performing linear regression and predicting values. | 9-10  11-11  12-12 |  |  |
| 5 | 1. Evaluating feed forward deep network for regression using KFold cross validation. 2. Evaluating feed forward deep network for multiclass Classification using KFold cross-validation. | 13-14  15-19 |  |  |
| 6 | Implementing regularization to avoid overfitting in binary classification. | 20-25 |  |  |
| 7 | Demonstrate recurrent neural network that learns to perform sequence analysis for stock price. | 26-28 |  |  |
| 8 | Performing encoding and decoding of images using deep autoencoder. | 29-31 |  |  |
| 9 | Implementation of convolutional neural network to predict numbers from number images | 32-34 |  |  |
| 10 | Denoising of images using autoencoder. | 35-36 |  |  |

## Practical No:1

### Aim: Performing matrix multiplication and finding eigen vectors and eigen values using TensorFlow.

import tensorflow as tf print("Matrix Multiplication Demo")

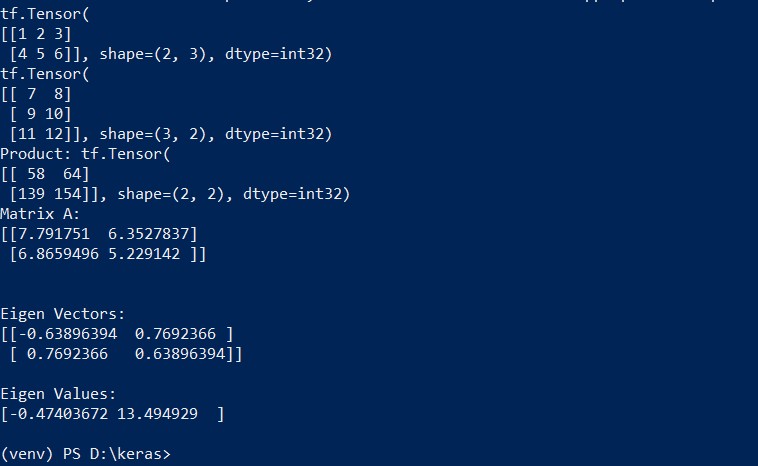
x=tf.constant([1,2,3,4,5,6],shape=[2,3]) print(x) y=tf.constant([7,8,9,10,11,12],shape=[3,2]) print(y)

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 No:2

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

import numpy as np

from keras.layers import Dense

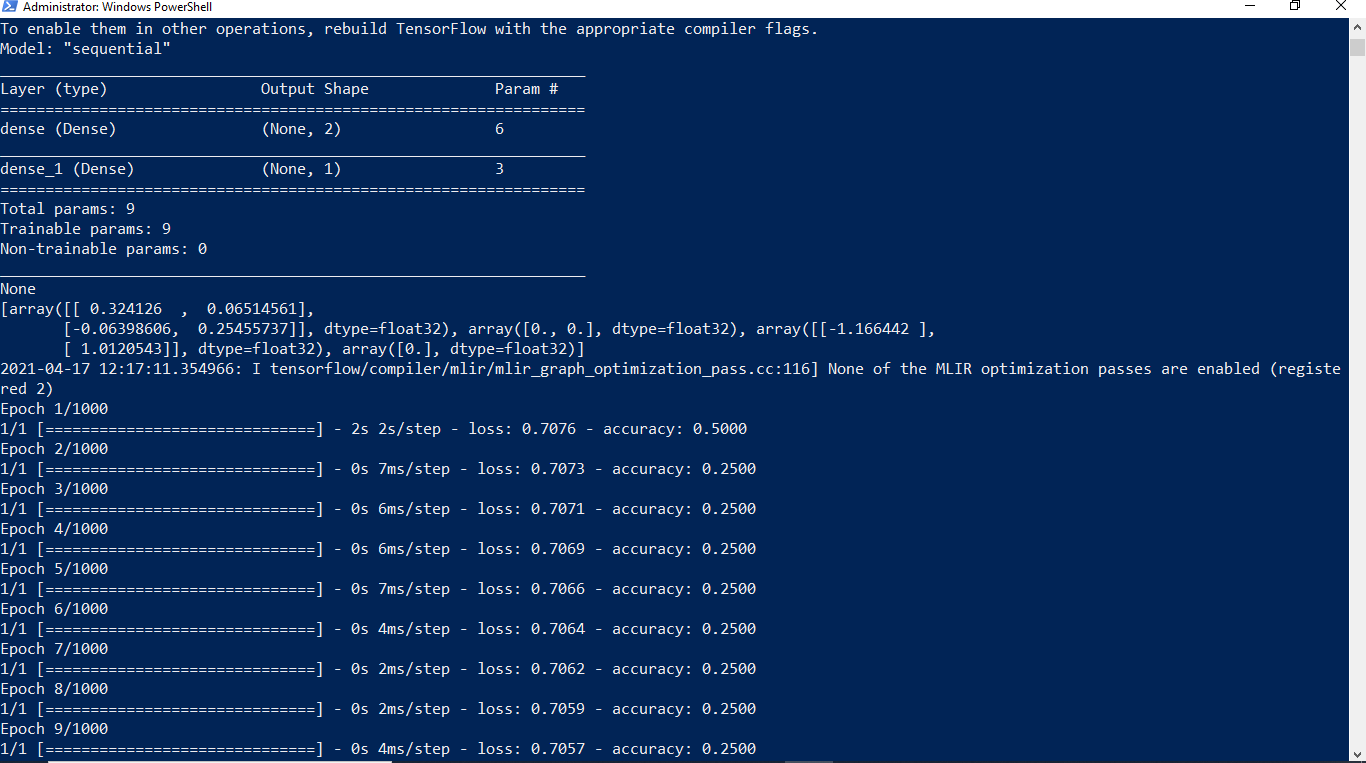
from keras.models import Sequential model=Sequential()

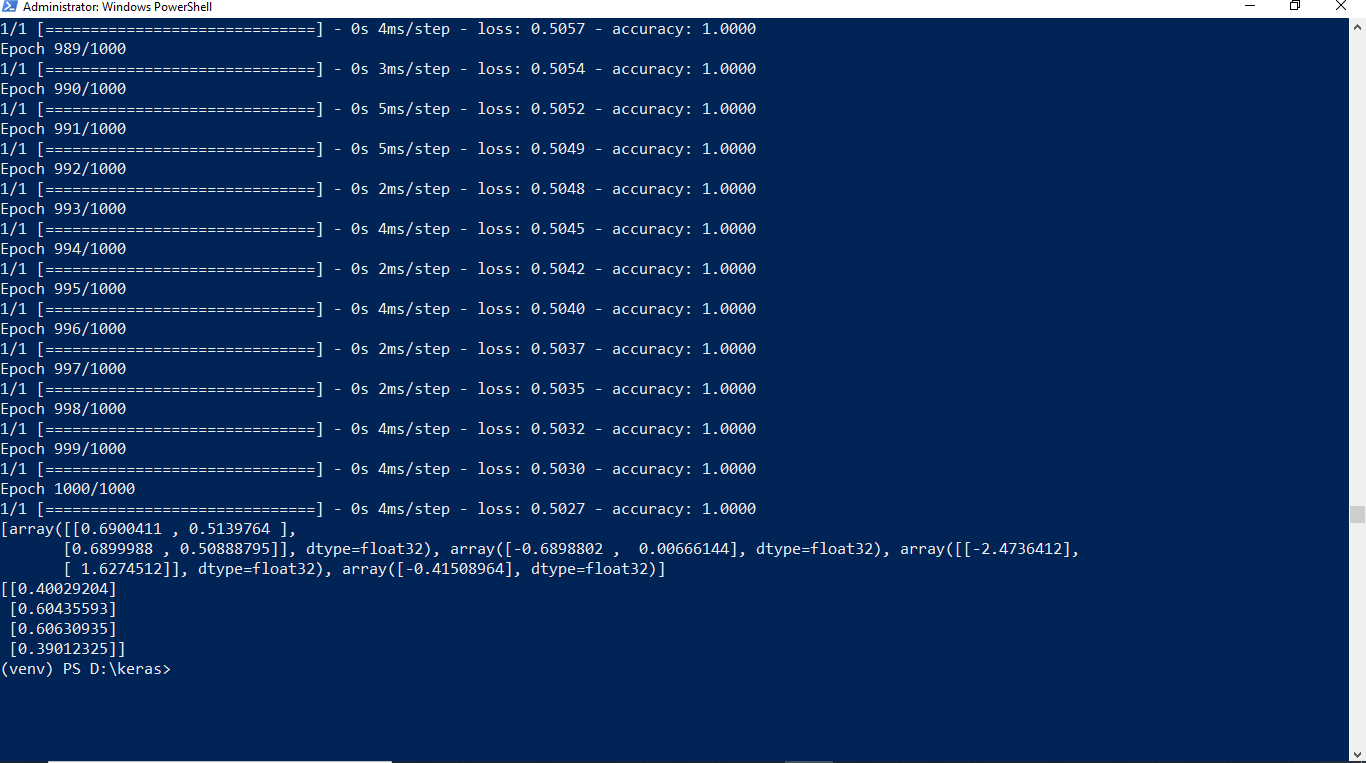
model.add(Dense(units=2,activation='relu',input\_dim=2)) model.add(Dense(units=1,activation='sigmoid')) model.compile(loss='binary\_crossentropy',optimizer='adam',metrics=['accuracy']) print(model.summary())

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:

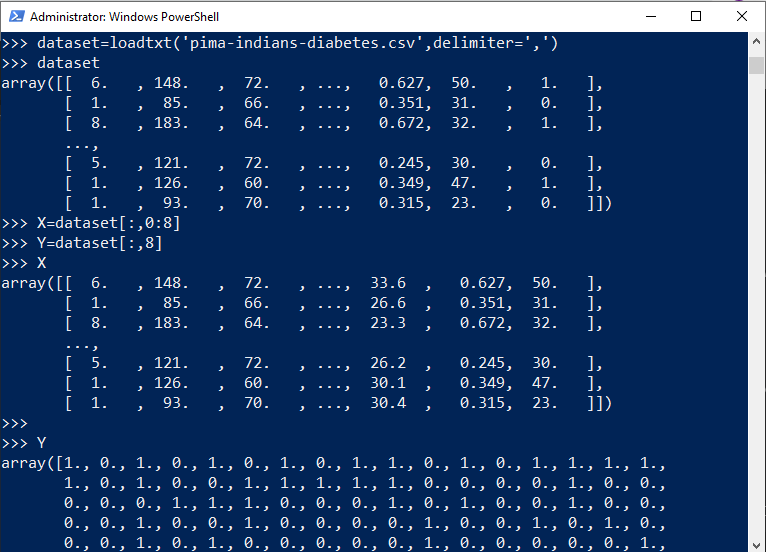
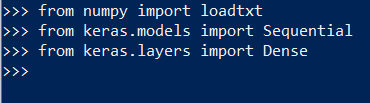




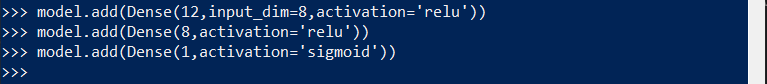
## Practical No:3

### Aim: Implementing deep neural network for performing classification task.

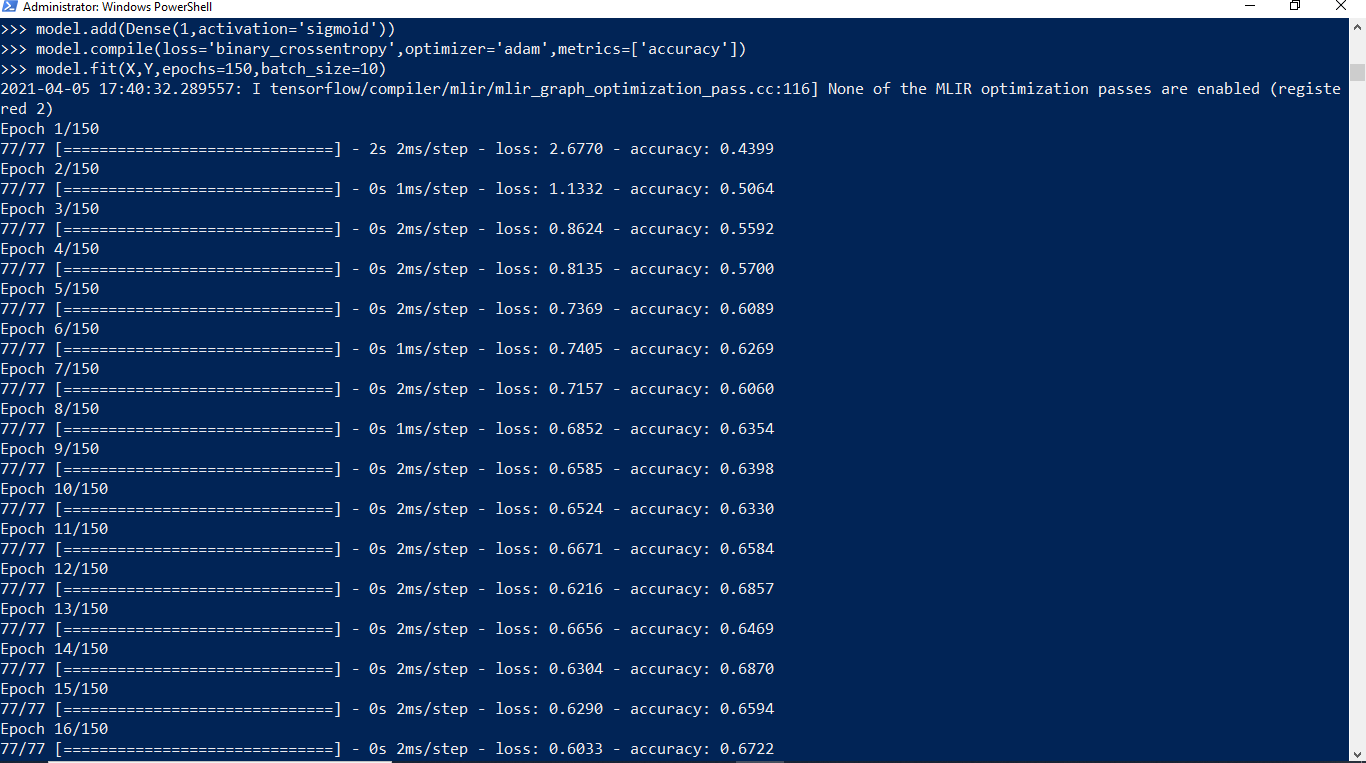
**Problem statement:** the given dataset comprises of health information about diabetic women patient. we need to create deep feed forward network that will classify women suffering from diabetes mellitus as 1.



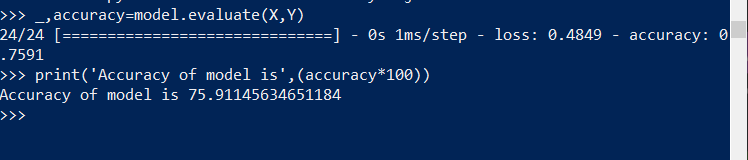
Creating model:



Compiling and fitting model:

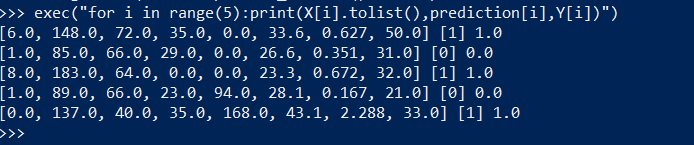


Evaluating the accuracy:



Using model for prediction class:





## Practical No:4

### Aim: Using deep feed forward network with two hidden layers for performing classification and predicting the class.

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)

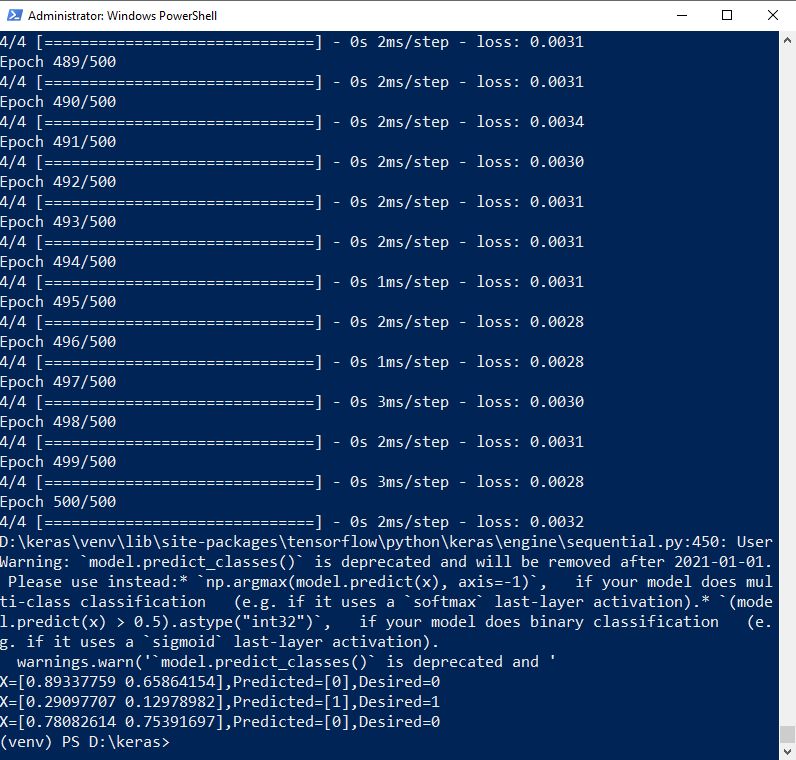
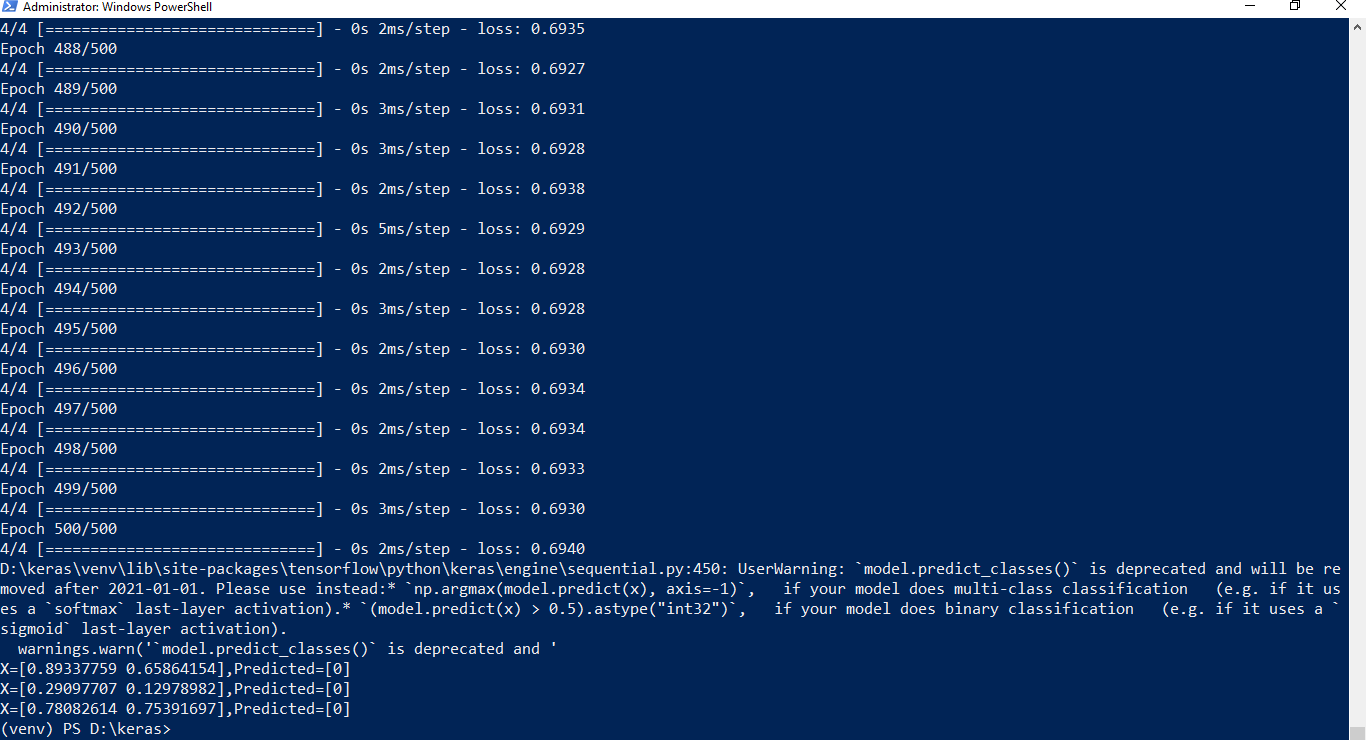
model=Sequential() model.add(Dense(4,input\_dim=2,activation='relu')) model.add(Dense(4,activation='relu')) model.add(Dense(1,activation='sigmoid')) model.compile(loss='binary\_crossentropy',optimizer='adam') model.fit(X,Y,epochs=500)

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

Ynew=model.predict\_classes(Xnew) for i in range(len(Xnew)):

print("X=%s,Predicted=%s,Desired=%s"%(Xnew[i],Ynew[i],Yreal[i]))

# OUTPUT:



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

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) model=Sequential()

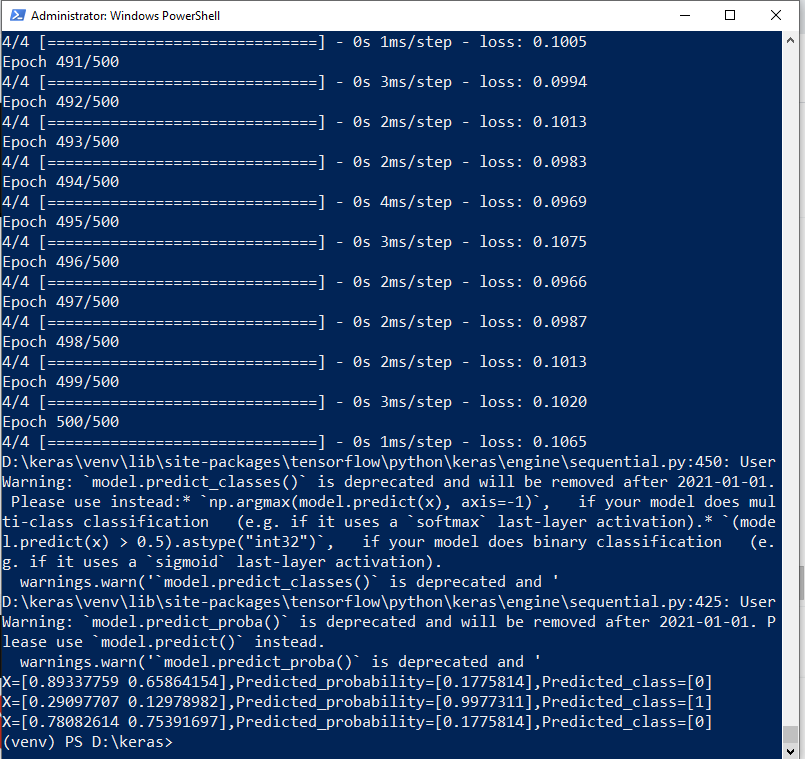
model.add(Dense(4,input\_dim=2,activation='relu')) model.add(Dense(4,activation='relu')) model.add(Dense(1,activation='sigmoid')) model.compile(loss='binary\_crossentropy',optimizer='adam') model.fit(X,Y,epochs=500)

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

Yclass=model.predict\_classes(Xnew) Ynew=model.predict\_proba(Xnew) for i in range(len(Xnew)):

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

# OUTPUT:



### Aim: Using a deep field forward network with two hidden layers for performing linear regression and predicting values.

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))

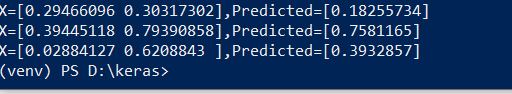
model=Sequential() model.add(Dense(4,input\_dim=2,activation='relu')) model.add(Dense(4,activation='relu')) model.add(Dense(1,activation='sigmoid')) model.compile(loss='mse',optimizer='adam') model.fit(X,Y,epochs=1000,verbose=0)

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 No:5(a)

### Aim: Evaluating feed forward deep network for regression using KFold cross validation.

import pandas as pd

from keras.models import Sequential from keras.layers import Dense

from keras.wrappers.scikit\_learn import KerasRegressor from sklearn.model\_selection import cross\_val\_score from sklearn.model\_selection import KFold

from sklearn.preprocessing import StandardScaler from sklearn.pipeline import Pipeline

dataframe=pd.read\_csv("housing.csv",delim\_whitespace=True,header=None) dataset=dataframe.values

X=dataset[:,0:13] Y=dataset[:,13] def wider\_model():

model=Sequential() model.add(Dense(15,input\_dim=13,kernel\_initializer='normal',activation='relu')) model.add(Dense(13,kernel\_initializer='normal',activation='relu')) model.add(Dense(1,kernel\_initializer='normal')) model.compile(loss='mean\_squared\_error',optimizer='adam')

return model estimators=[]

estimators.append(('standardize',StandardScaler())) estimators.append(('mlp',KerasRegressor(build\_fn=wider\_model,epochs=100,batch\_size=5))) pipeline=Pipeline(estimators)

kfold=KFold(n\_splits=10) results=cross\_val\_score(pipeline,X,Y,cv=kfold)

print("Wider: %.2f (%.2f) MSE" % (results.mean(), results.std()))

**OUTPUT:**



(After changing neuron)

model.add(Dense(20, input\_dim=13,kernel\_initializer='normal',activation='relu'))



## Practical No:5b

### Aim: Evaluating feed forward deep network for multiclass Classification using KFold cross-validation.

#loading libraries 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 #loading dataset df=pandas.read\_csv('Flower.csv',header=None) print(df)

#splitting dataset into input and output variables X = df.iloc[:,0:4].astype(float)

y=df.iloc[:,4] #print(X) #print(y)

#encoding string output into numeric output encoder=LabelEncoder()

encoder.fit(y) encoded\_y=encoder.transform(y) print(encoded\_y) dummy\_Y=np\_utils.to\_categorical(encoded\_y) print(dummy\_Y)

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

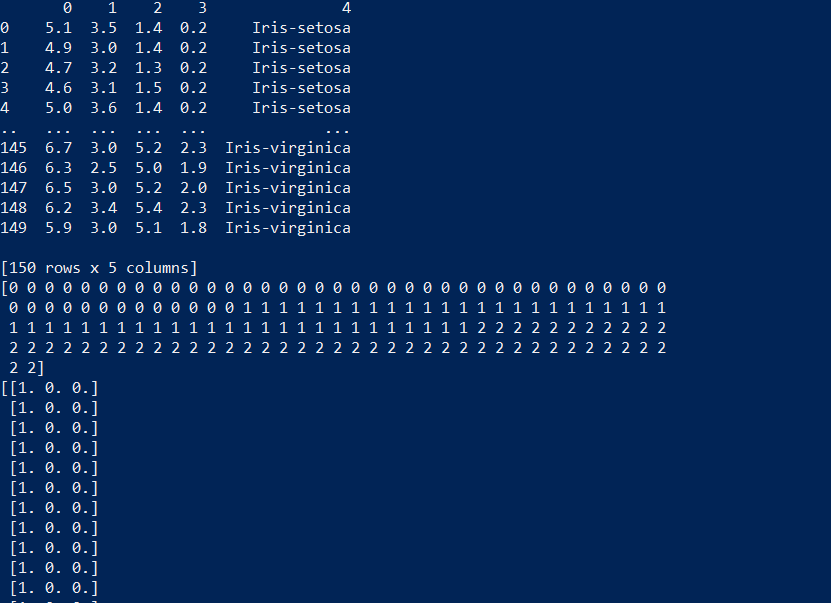
estimator=baseline\_model() estimator.fit(X,dummy\_Y,epochs=100,shuffle=True) action=estimator.predict(X)

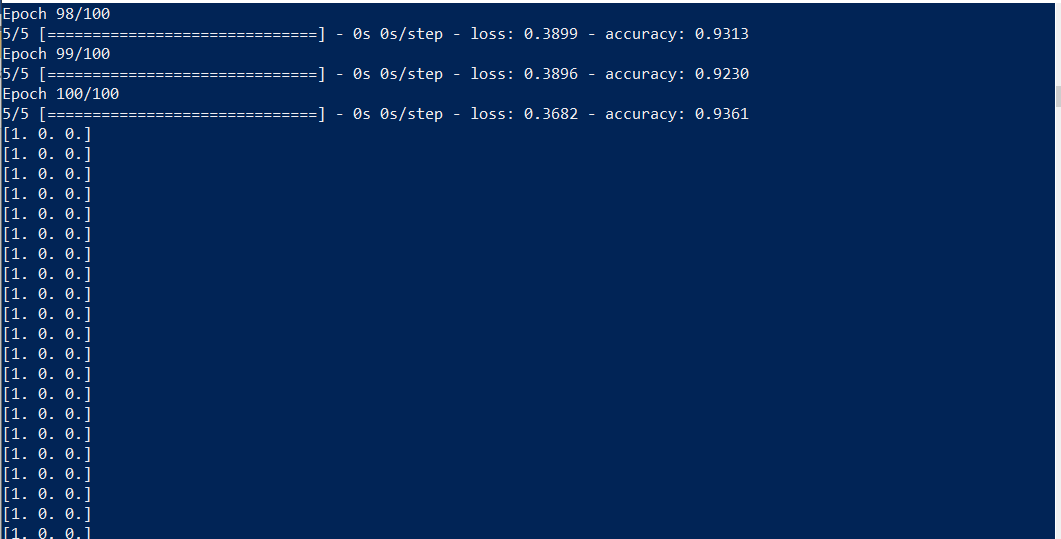
for i in range(25): print(dummy\_Y[i])

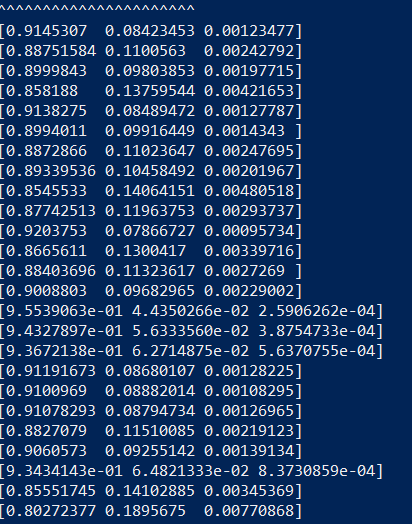
print('^^^^^^^^^^^^^^^^^^^^^^') for i in range(25):

print(action[i])

### OUTPUT:







**Code 2:**

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

dataset=pandas.read\_csv("Flower.csv",header=None)

dataset1=dataset.values X=dataset1[:,0:4].astype(float) Y=dataset1[:,4]

print(Y) encoder=LabelEncoder() encoder.fit(Y)

encoder\_Y=encoder.transform(Y) print(encoder\_Y) dummy\_Y=np\_utils.to\_categorical(encoder\_Y) print(dummy\_Y)

def baseline\_model():

model=Sequential() model.add(Dense(8,input\_dim=4,activation='relu')) model.add(Dense(3,activation='softmax'))

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

estimator=KerasClassifier(build\_fn=baseline\_model,epochs=100,batch\_size=5) 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))



(Changing neuron) model.add(Dense(10,input\_dim=4,activation='relu'))



## Practical No :6

### Aim: implementing regularization to avoid overfitting in binary classification.

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

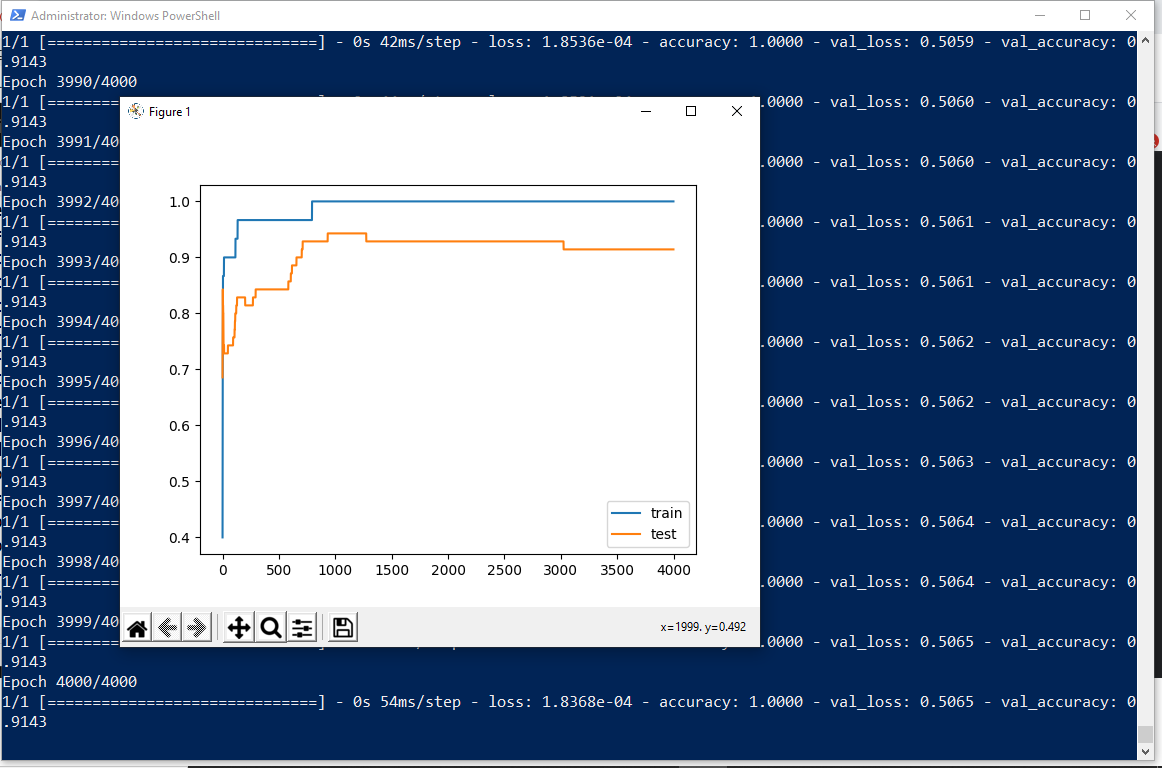
trainX,testX=X[:n\_train,:],X[n\_train:] trainY,testY=Y[:n\_train],Y[n\_train:] #print(trainX)

#print(trainY) #print(testX) #print(testY) 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:



The above code and resultant graph demonstrate overfitting with accuracy of testing data less than accuracy of training data also the accuracy of testing data increases once and then start decreases gradually.to solve this problem we can use regularization

Hence, we will add two lines in the above code as highlighted below to implement l2 regularization with alpha=0.001

from matplotlib import pyplot

from sklearn.datasets import make\_moons from keras.models import Sequential from keras.layers import Dense

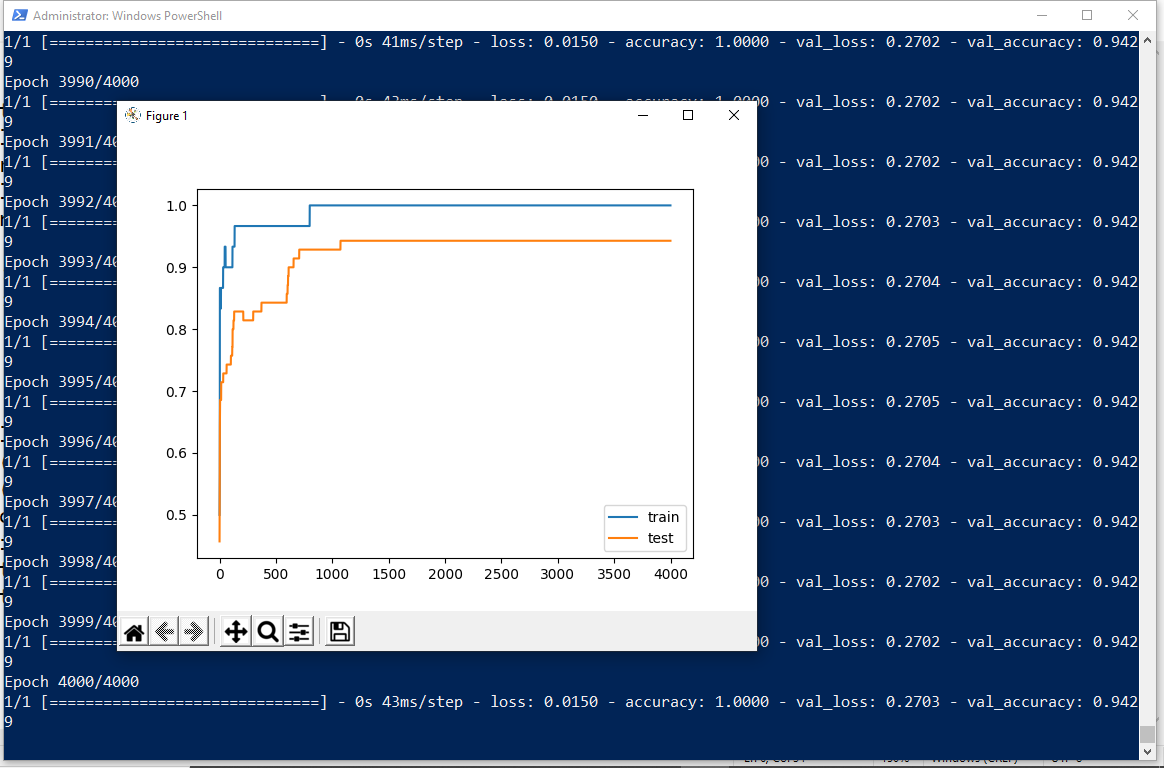
**from keras.regularizers import l2** X,Y=make\_moons(n\_samples=100,noise=0.2,random\_state=1) n\_train=30

trainX,testX=X[:n\_train,:],X[n\_train:] trainY,testY=Y[:n\_train],Y[n\_train:] #print(trainX)

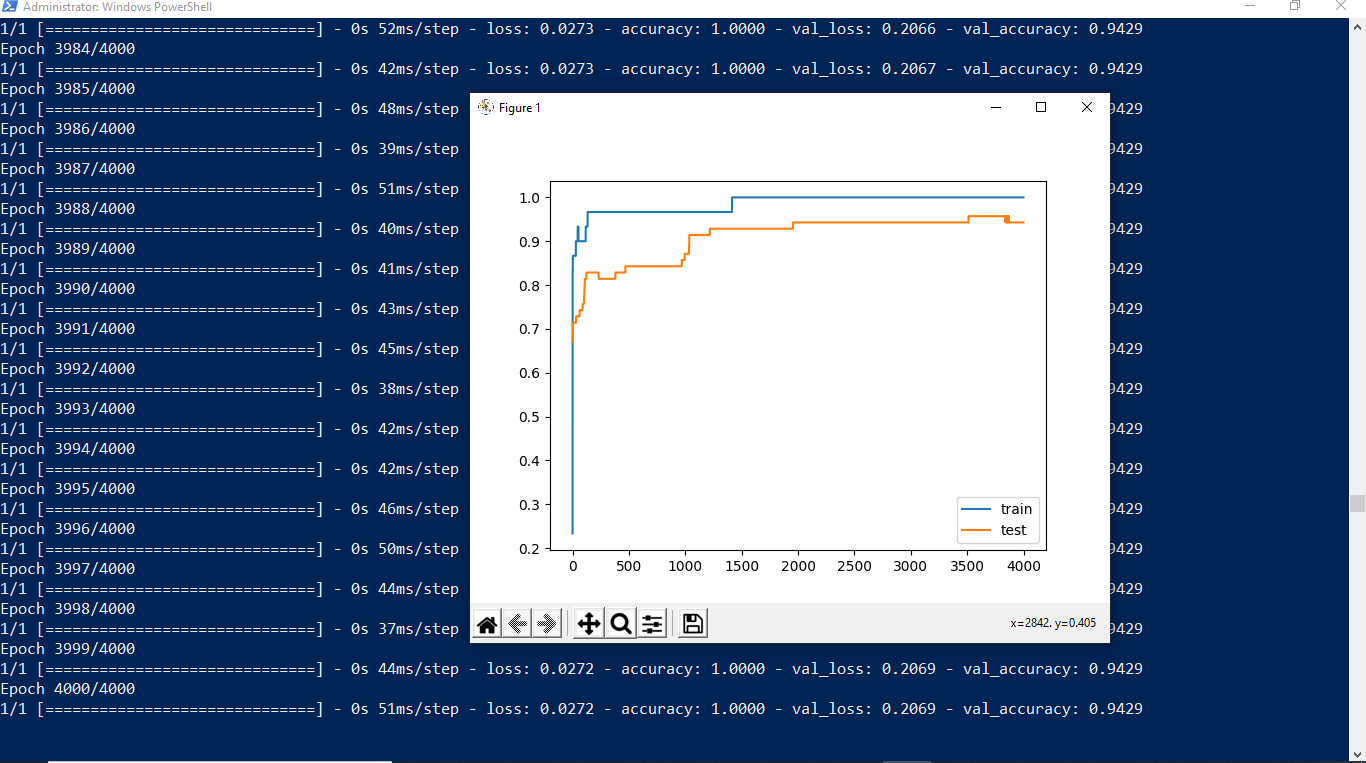
#print(trainY) #print(testX) #print(testY) model=Sequential()

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

pyplot.legend() pyplot.show()

By replacing l2 regularizer with l1 regularizer at the same learning rate 0.001 we get the following output.



By applying l1 and l2 regularizer we can observe the following changes in accuracy of both trainig and testing data. The changes in code are also highlighted.

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

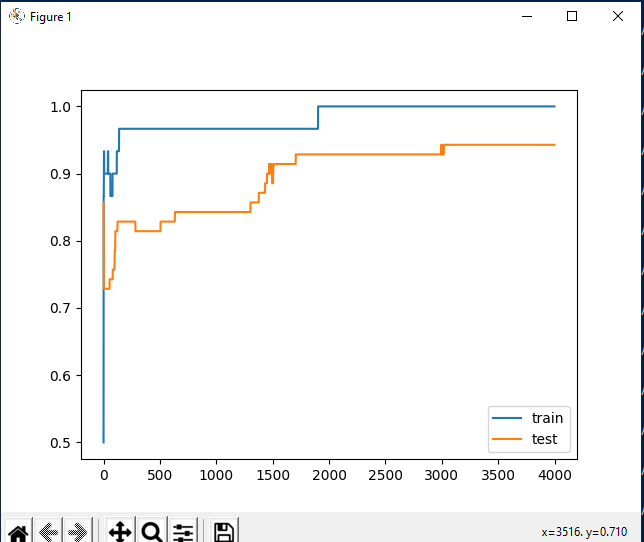
trainX,testX=X[:n\_train,:],X[n\_train:] trainY,testY=Y[:n\_train],Y[n\_train:] #print(trainX)

#print(trainY) #print(testX) #print(testY) 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 No:7

### Aim: Demonstrate recurrent neural network that learns to perform sequence analysis for stock price.

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('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('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)

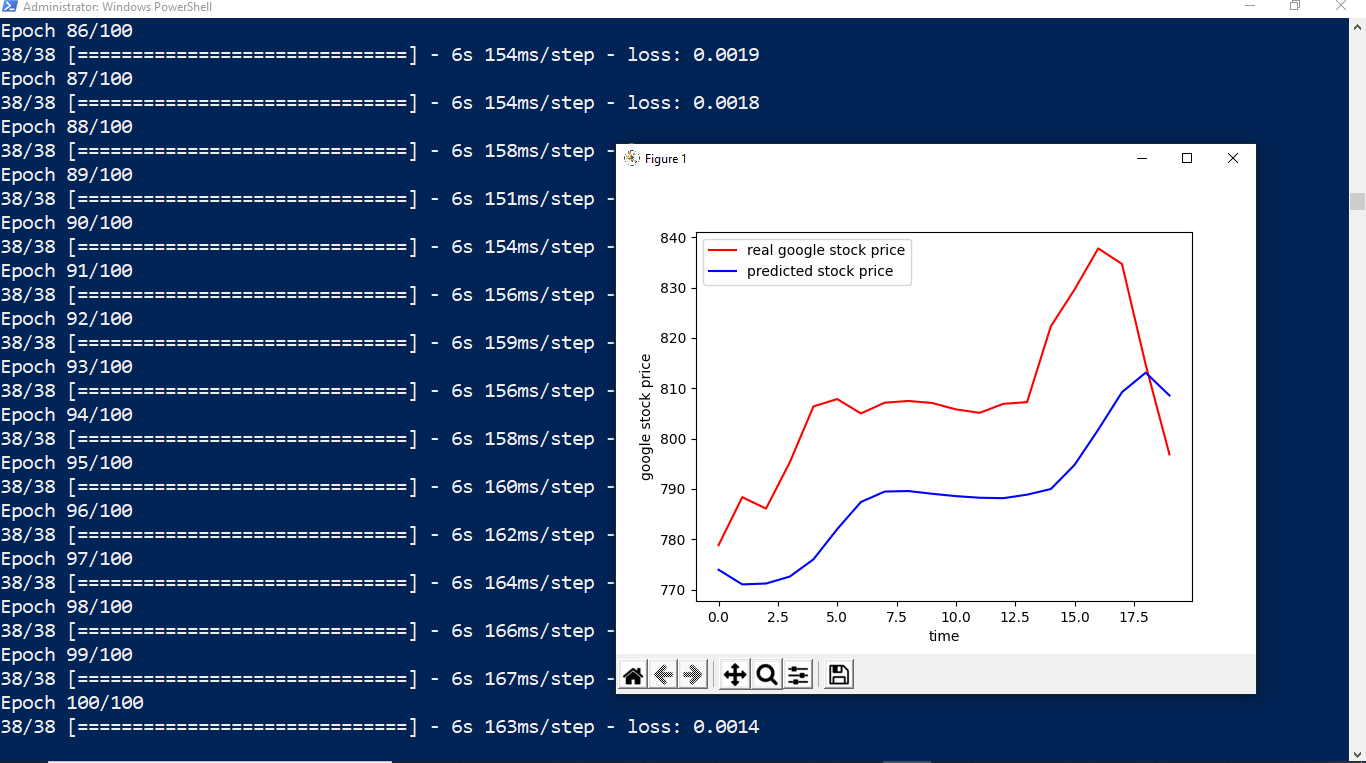
X\_test=np.reshape(X\_test,(X\_test.shape[0],X\_test.shape[1],1)) predicted\_stock\_price=regressor.predict(X\_test)

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 No:8

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

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):

# display original

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)

# display encoded image

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) # display reconstruction

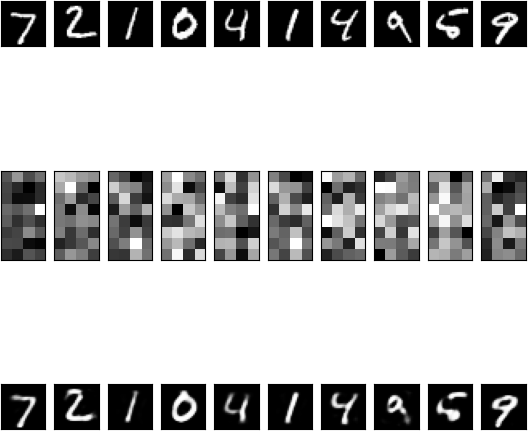
ax = plt.subplot(3, 20, 2\*20 +i+ 1) plt.imshow(decoded\_imgs[i].reshape(28, 28)) plt.gray()

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

plt.show()

# OUTPUT:





## Practical No:9

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

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'))

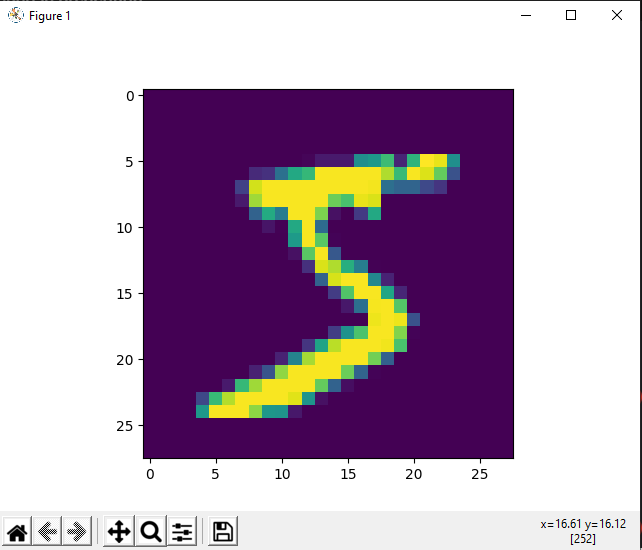
model.compile(optimizer='adam',loss='categorical\_crossentropy',metrics=['accuracy']) #train

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])

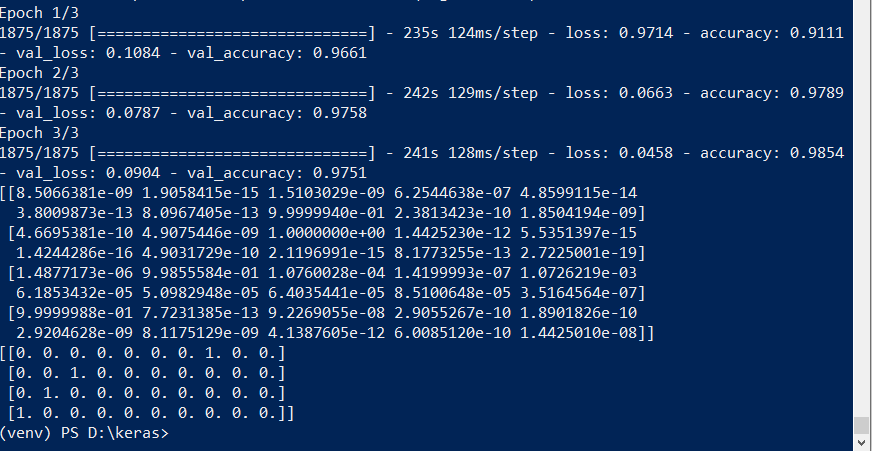
# OUTPUT:



(28, 28)

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





## Practical No:10

### Aim: Denoising of images using autoencoder.

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)])

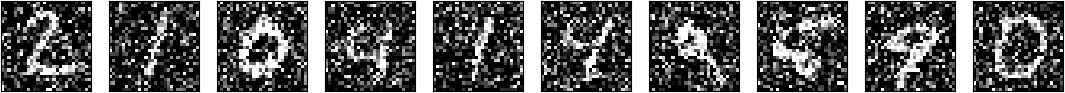
predictions=autoencoder.predict(X\_test\_noisy) m=10

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:



After 3 epochs:

