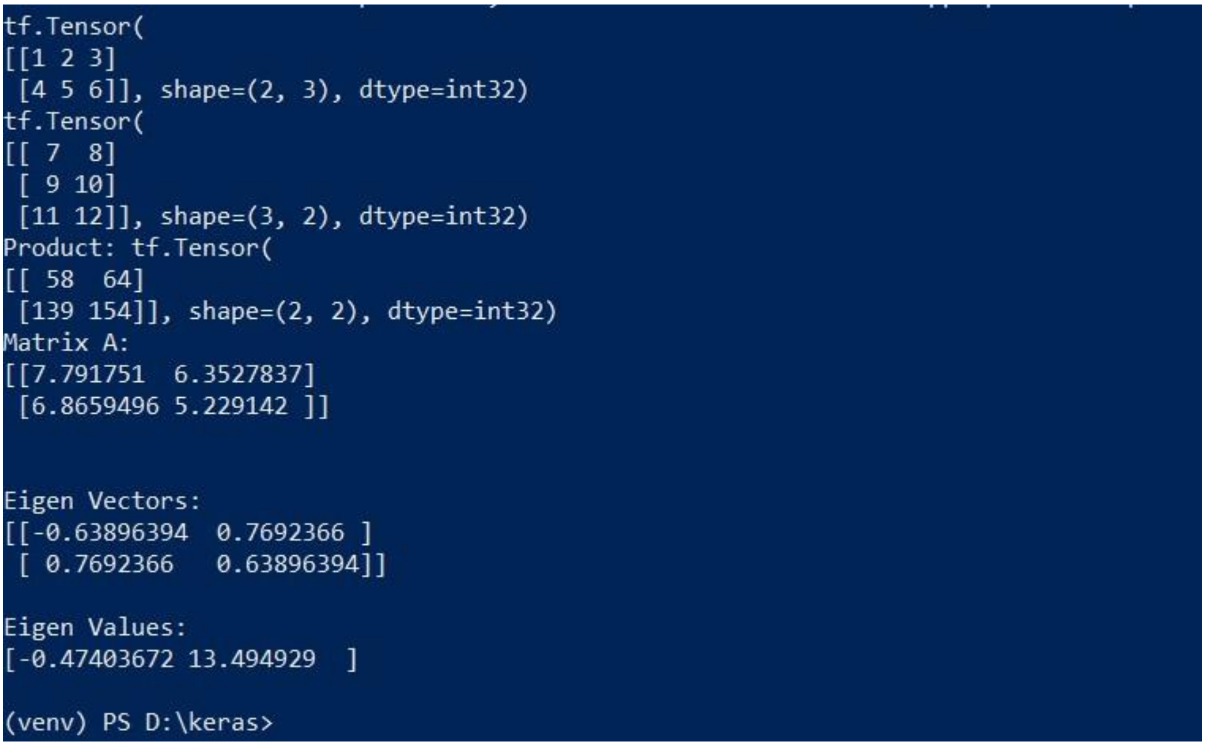
Practical No	Title
1	Performing matrix multiplication and finding eigen vectors and eigen values using TensorFlow
2	Solving XOR problem using deep feed forward network.
3	Implementing deep neural network for performing binary classification task.
4	Using deep feed forward network with two hidden layers for performing multiclass classification and predictingthe class.
5	Using a deep feed forward network with two hiddenlayers for performing classification and predicting the probability of class.
6	Using a deep feed forward network with two hidden layers for performing linear regression and predicting values.
7	Demonstrate recurrent neural network that learns to perform sequence analysis for stock price.

Practical No:1

Aim: Performing matrix multiplication and finding eigen vectors and eigenvalues 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:



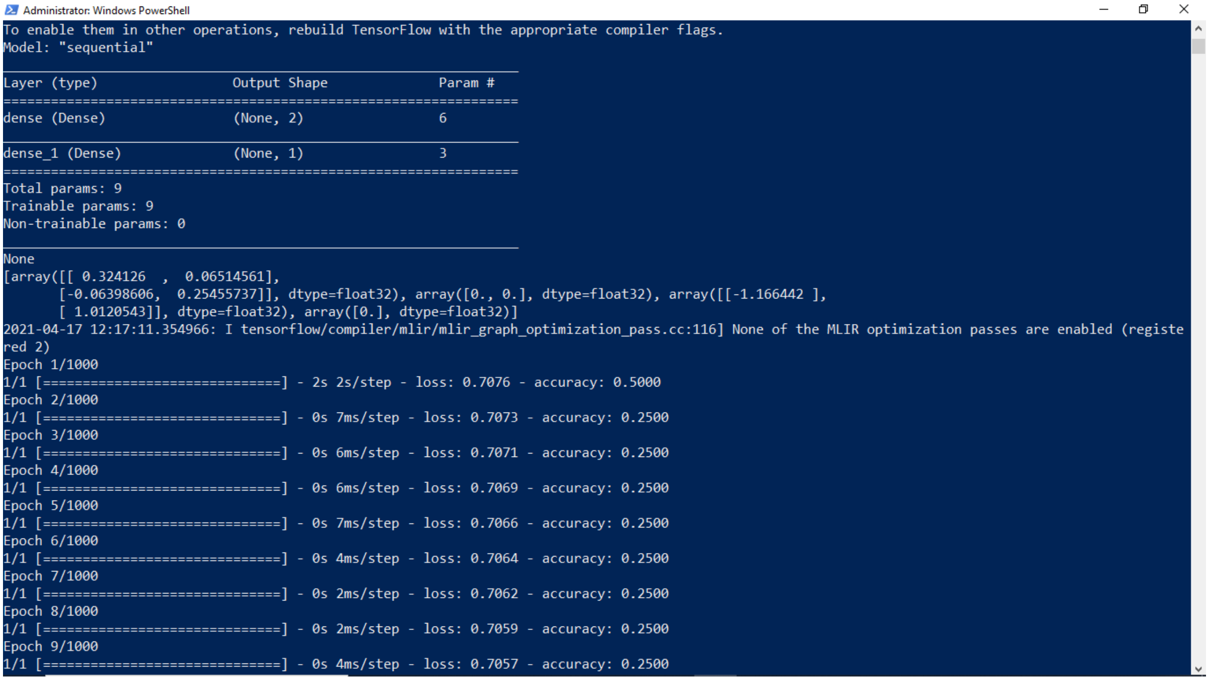
```
tf.Tensor(
[[1 2 3]
 [4 5 6]], shape=(2, 3), dtype=int32)
tf.Tensor(
[[ 7  8]
 [ 9 10]
 [11 12]], shape=(3, 2), dtype=int32)
Product: tf.Tensor(
[[ 58  64]
 [139 154]], shape=(2, 2), dtype=int32)
Matrix A:
[[7.791751  6.3527837]
 [6.8659496 5.229142 ]]


Eigen Vectors:
[[-0.63896394  0.7692366 ]
 [ 0.7692366   0.63896394]]

Eigen Values:
[-0.47403672 13.494929  ]

(venv) PS D:\keras>
```

Practical No:2

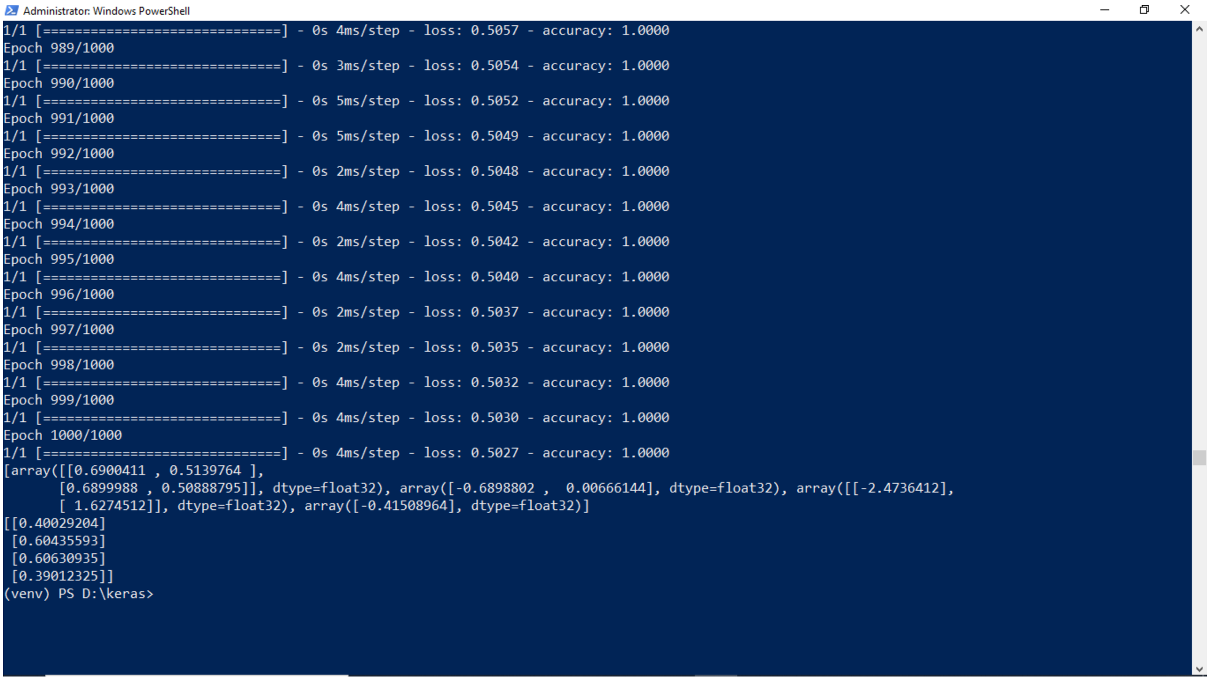
Aim: Solving XOR problem using deep feed forward network.

```
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:



```
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 2)                 6
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 3
=================================================================
Total params: 9
Trainable params: 9
Non-trainable params: 0
_________________________________________________________________
None
[array([[ 0.324126  ,  0.06514561],
       [-0.06398606,  0.25455737]], dtype=float32), array([0., 0.], dtype=float32), array([[-1.166442 ],
       [ 1.0120543]], dtype=float32), array([0.], dtype=float32)]
2021-04-17 12:17:11.354966: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:116] None of the MLIR optimization passes are enabled (registe
red 2)
Epoch 1/1000
1/1 [==============================] - 2s 2s/step - loss: 0.7076 - accuracy: 0.5000
Epoch 2/1000
1/1 [==============================] - 0s 7ms/step - loss: 0.7073 - accuracy: 0.2500
Epoch 3/1000
1/1 [==============================] - 0s 6ms/step - loss: 0.7071 - accuracy: 0.2500
Epoch 4/1000
1/1 [==============================] - 0s 6ms/step - loss: 0.7069 - accuracy: 0.2500
Epoch 5/1000
1/1 [==============================] - 0s 7ms/step - loss: 0.7066 - accuracy: 0.2500
Epoch 6/1000
1/1 [==============================] - 0s 4ms/step - loss: 0.7064 - accuracy: 0.2500
Epoch 7/1000
1/1 [==============================] - 0s 2ms/step - loss: 0.7062 - accuracy: 0.2500
Epoch 8/1000
1/1 [==============================] - 0s 2ms/step - loss: 0.7059 - accuracy: 0.2500
Epoch 9/1000
1/1 [==============================] - 0s 4ms/step - loss: 0.7057 - accuracy: 0.2500
```



```
1/1 [==============================] - 0s 4ms/step - loss: 0.5057 - accuracy: 1.0000
Epoch 989/1000
1/1 [==============================] - 0s 3ms/step - loss: 0.5054 - accuracy: 1.0000
Epoch 990/1000
1/1 [==============================] - 0s 5ms/step - loss: 0.5052 - accuracy: 1.0000
Epoch 991/1000
1/1 [==============================] - 0s 5ms/step - loss: 0.5049 - accuracy: 1.0000
Epoch 992/1000
1/1 [==============================] - 0s 2ms/step - loss: 0.5048 - accuracy: 1.0000
Epoch 993/1000
1/1 [==============================] - 0s 4ms/step - loss: 0.5045 - accuracy: 1.0000
Epoch 994/1000
1/1 [==============================] - 0s 2ms/step - loss: 0.5042 - accuracy: 1.0000
Epoch 995/1000
1/1 [==============================] - 0s 4ms/step - loss: 0.5040 - accuracy: 1.0000
Epoch 996/1000
1/1 [==============================] - 0s 2ms/step - loss: 0.5037 - accuracy: 1.0000
Epoch 997/1000
1/1 [==============================] - 0s 2ms/step - loss: 0.5035 - accuracy: 1.0000
Epoch 998/1000
1/1 [==============================] - 0s 4ms/step - loss: 0.5032 - accuracy: 1.0000
Epoch 999/1000
1/1 [==============================] - 0s 4ms/step - loss: 0.5030 - accuracy: 1.0000
Epoch 1000/1000
1/1 [==============================] - 0s 4ms/step - loss: 0.5027 - accuracy: 1.0000
[array([[0.6900411 , 0.5139764 ],
       [0.6899988 , 0.50888795]], dtype=float32), array([-0.6898802 ,  0.00666144], dtype=float32), array([[-2.4736412],
       [ 1.6274512]], dtype=float32), array([-0.41508964], dtype=float32)]
[[0.40029204]
 [0.60435593]
 [0.60630935]
 [0.39012325]]
(venv) PS D:\keras>
```

Practical No:3

Aim: Implementing deep neural network for performing classification task.

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

```
# Step 1: Import required libraries
from numpy import loadtxt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Step 2: Load the dataset (For simplicity, let's use a small dataset like Pima Indians Diabetes dataset)
# Download the dataset from: https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv!wget -nc https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv

# Load dataset (8 input features and 1 target label)
dataset = loadtxt('pima-indians-diabetes.data.csv', delimiter=',')
X = dataset[:, 0:8]  # Input features (first 8 columns)
y = dataset[:, 8]    # Output labels (last column)

# Step 3: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 4: Define the Deep Neural Network model
model = Sequential([
    Dense(12, input_dim=8, activation='relu'),  # Hidden Layer 1 with 12 neurons and ReLU activation
    Dense(8, activation='relu'),                # Hidden Layer 2 with 8 neurons and ReLU activation
    Dense(1, activation='sigmoid')              # Output Layer with 1 neuron and Sigmoid activation for binary classification
])
```

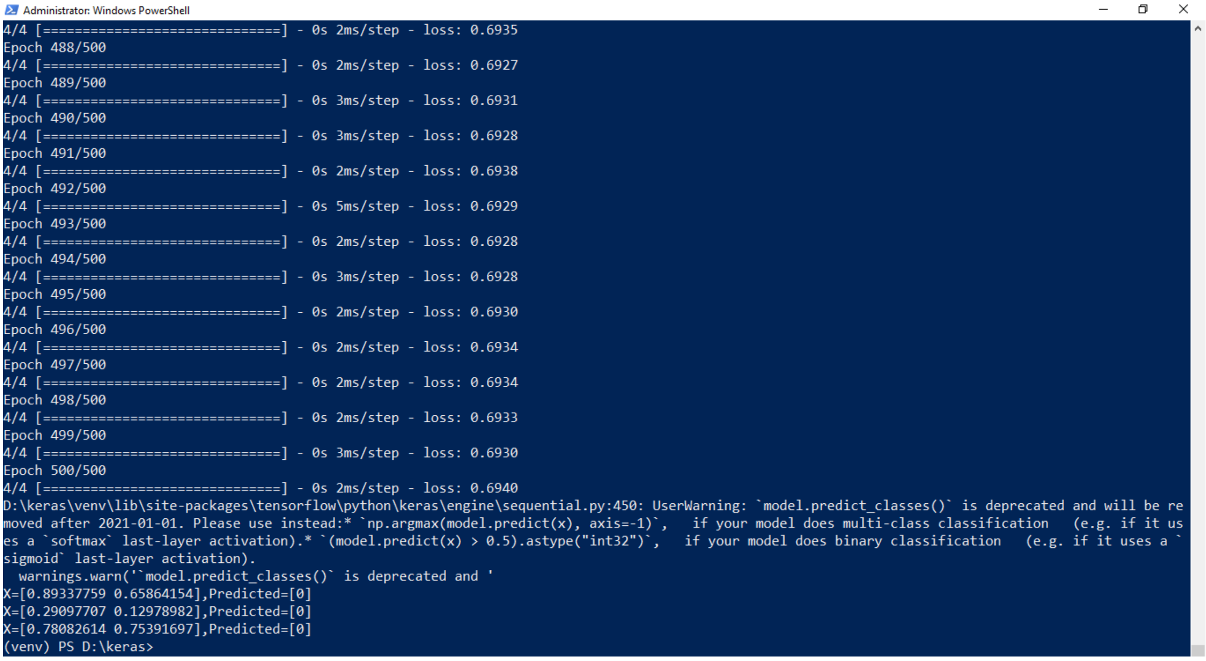
```
# Step 5: Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Step 6: Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=10, validation_split=0.1)

# Step 7: Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test Accuracy: {accuracy * 100:.2f}%')

# Step 8: Make predictions
y_pred = (model.predict(X_test) > 0.5).astype("int32")

# Step 9: Calculate and display accuracy score
accuracy_score_value = accuracy_score(y_test, y_pred)
print(f'Accuracy Score: {accuracy_score_value * 100:.2f}%')
```

Practical No:4

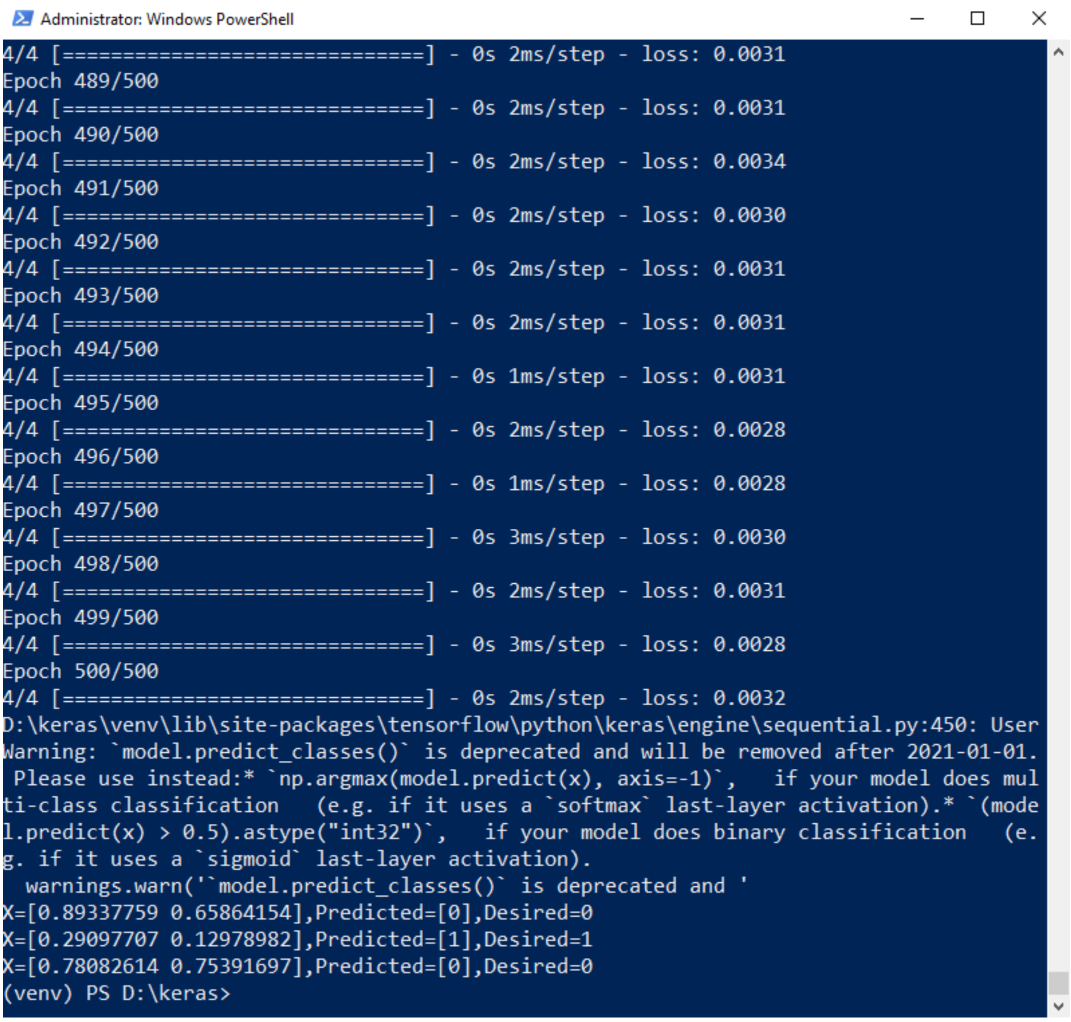
Aim: Using deep feed forward network with two hidden layers forperforming classification and predicting the class.

```
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:



```
4/4 [==============================] - 0s 2ms/step - loss: 0.6935
Epoch 488/500
4/4 [==============================] - 0s 2ms/step - loss: 0.6927
Epoch 489/500
4/4 [==============================] - 0s 3ms/step - loss: 0.6931
Epoch 490/500
4/4 [==============================] - 0s 3ms/step - loss: 0.6928
Epoch 491/500
4/4 [==============================] - 0s 2ms/step - loss: 0.6938
Epoch 492/500
4/4 [==============================] - 0s 5ms/step - loss: 0.6929
Epoch 493/500
4/4 [==============================] - 0s 2ms/step - loss: 0.6928
Epoch 494/500
4/4 [==============================] - 0s 3ms/step - loss: 0.6928
Epoch 495/500
4/4 [==============================] - 0s 2ms/step - loss: 0.6930
Epoch 496/500
4/4 [==============================] - 0s 2ms/step - loss: 0.6934
Epoch 497/500
4/4 [==============================] - 0s 2ms/step - loss: 0.6934
Epoch 498/500
4/4 [==============================] - 0s 2ms/step - loss: 0.6933
Epoch 499/500
4/4 [==============================] - 0s 3ms/step - loss: 0.6930
Epoch 500/500
4/4 [==============================] - 0s 2ms/step - loss: 0.6940
D:\keras\venv\lib\site-packages\tensorflow\python\keras\engine\sequential.py:450: UserWarning: `model.predict_classes()` is deprecated and will be re
moved after 2021-01-01. Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model does multi-class classification   (e.g. if it us
es a `softmax` last-layer activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model does binary classification   (e.g. if it uses a `
sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
X=[0.89337759 0.65864154],Predicted=[0]
X=[0.29097707 0.12978982],Predicted=[0]
X=[0.78082614 0.75391697],Predicted=[0]
(venv) PS D:\keras>
```



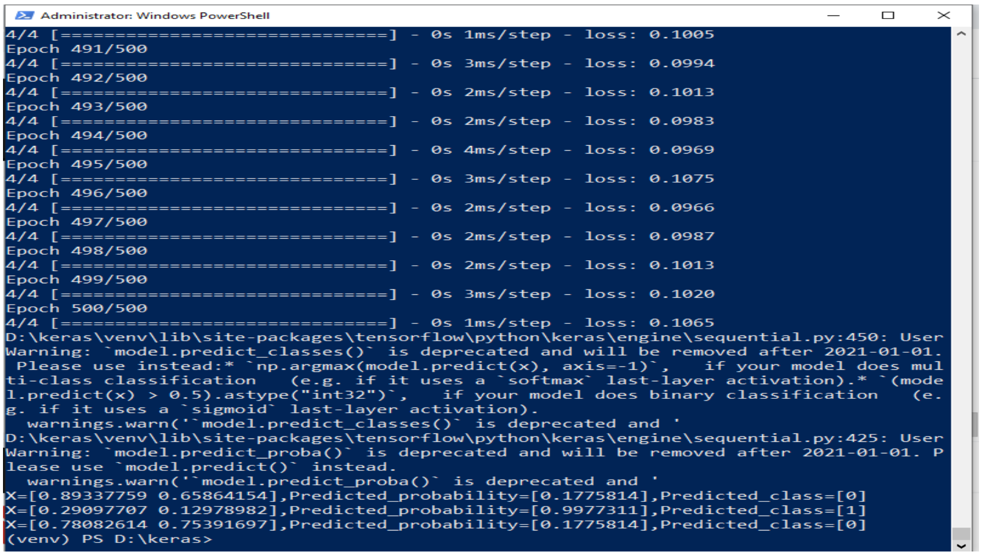
```
4/4 [==============================] - 0s 2ms/step - loss: 0.0031
Epoch 489/500
4/4 [==============================] - 0s 2ms/step - loss: 0.0031
Epoch 490/500
4/4 [==============================] - 0s 2ms/step - loss: 0.0034
Epoch 491/500
4/4 [==============================] - 0s 2ms/step - loss: 0.0030
Epoch 492/500
4/4 [==============================] - 0s 2ms/step - loss: 0.0031
Epoch 493/500
4/4 [==============================] - 0s 2ms/step - loss: 0.0031
Epoch 494/500
4/4 [==============================] - 0s 1ms/step - loss: 0.0031
Epoch 495/500
4/4 [==============================] - 0s 2ms/step - loss: 0.0028
Epoch 496/500
4/4 [==============================] - 0s 1ms/step - loss: 0.0028
Epoch 497/500
4/4 [==============================] - 0s 3ms/step - loss: 0.0030
Epoch 498/500
4/4 [==============================] - 0s 2ms/step - loss: 0.0031
Epoch 499/500
4/4 [==============================] - 0s 3ms/step - loss: 0.0028
Epoch 500/500
4/4 [==============================] - 0s 2ms/step - loss: 0.0032
D:\keras\venv\lib\site-packages\tensorflow\python\keras\engine\sequential.py:450: User
Warning: `model.predict_classes()` is deprecated and will be removed after 2021-01-01.
 Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model does mul
ti-class classification   (e.g. if it uses a `softmax` last-layer activation).* `(mode
l.predict(x) > 0.5).astype("int32")`,   if your model does binary classification   (e.
g. if it uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
X=[0.89337759 0.65864154],Predicted=[0],Desired=0
X=[0.29097707 0.12978982],Predicted=[1],Desired=1
X=[0.78082614 0.75391697],Predicted=[0],Desired=0
(venv) PS D:\keras>
```

Practical No:5

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:



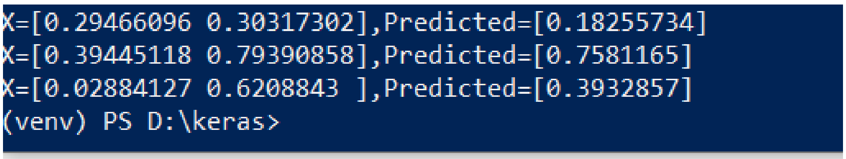
```
4/4 [==============================] - 0s 1ms/step - loss: 0.1005
Epoch 491/500
4/4 [==============================] - 0s 3ms/step - loss: 0.0994
Epoch 492/500
4/4 [==============================] - 0s 2ms/step - loss: 0.1013
Epoch 493/500
4/4 [==============================] - 0s 2ms/step - loss: 0.0983
Epoch 494/500
4/4 [==============================] - 0s 4ms/step - loss: 0.0969
Epoch 495/500
4/4 [==============================] - 0s 3ms/step - loss: 0.1075
Epoch 496/500
4/4 [==============================] - 0s 2ms/step - loss: 0.0966
Epoch 497/500
4/4 [==============================] - 0s 2ms/step - loss: 0.0987
Epoch 498/500
4/4 [==============================] - 0s 2ms/step - loss: 0.1013
Epoch 499/500
4/4 [==============================] - 0s 3ms/step - loss: 0.1020
Epoch 500/500
4/4 [==============================] - 0s 1ms/step - loss: 0.1065
D:\keras\venv\lib\site-packages\tensorflow\python\keras\engine\sequential.py:450: User
Warning: `model.predict_classes()` is deprecated and will be removed after 2021-01-01.
 Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model does mul
ti-class classification   (e.g. if it uses a `softmax` last-layer activation).* `(mode
l.predict(x) > 0.5).astype("int32")`,   if your model does binary classification   (e.
g. if it uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
D:\keras\venv\lib\site-packages\tensorflow\python\keras\engine\sequential.py:425: User
Warning: `model.predict_proba()` is deprecated and will be removed after 2021-01-01. P
lease use `model.predict()` instead.
  warnings.warn('`model.predict_proba()` is deprecated and '
X=[0.89337759 0.65864154],Predicted_probability=[0.1775814],Predicted_class=[0]
X=[0.29097707 0.12978982],Predicted_probability=[0.9977311],Predicted_class=[1]
X=[0.78082614 0.75391697],Predicted_probability=[0.1775814],Predicted_class=[0]
(venv) PS D:\keras>
```

Practical No:6

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:



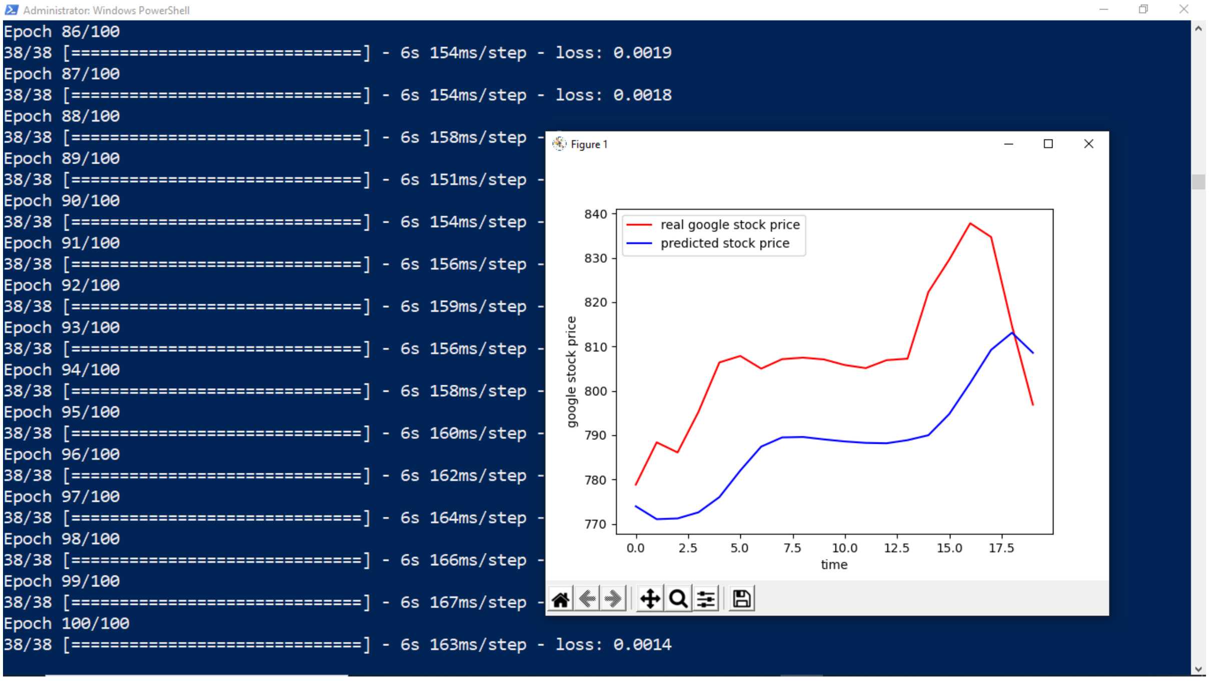
```
X=[0.29466096 0.30317302],Predicted=[0.18255734]
X=[0.39445118 0.79390858],Predicted=[0.7581165]
X=[0.02884127 0.6208843 ],Predicted=[0.3932857]
(venv) PS D:\keras>
```

Practical No:7

Aim: Demonstrate recurrent neural network that learns to performsequence analysis for stock price.

```
import numpy as np

import matplotlib.pyplot as pltimport

pandas as pd

from keras.models import Sequentialfrom

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:



```
Epoch 86/100
38/38 [==============================] - 6s 154ms/step - loss: 0.0019
Epoch 87/100
38/38 [==============================] - 6s 154ms/step - loss: 0.0018
Epoch 88/100
38/38 [==============================] - 6s 158ms/step -
Epoch 89/100
38/38 [==============================] - 6s 151ms/step -
Epoch 90/100
38/38 [==============================] - 6s 154ms/step -
Epoch 91/100
38/38 [==============================] - 6s 156ms/step -
Epoch 92/100
38/38 [==============================] - 6s 159ms/step -
Epoch 93/100
38/38 [==============================] - 6s 156ms/step -
Epoch 94/100
38/38 [==============================] - 6s 158ms/step -
Epoch 95/100
38/38 [==============================] - 6s 160ms/step -
Epoch 96/100
38/38 [==============================] - 6s 162ms/step -
Epoch 97/100
38/38 [==============================] - 6s 164ms/step -
Epoch 98/100
38/38 [==============================] - 6s 166ms/step -
Epoch 99/100
38/38 [==============================] - 6s 167ms/step -
Epoch 100/100
38/38 [==============================] - 6s 163ms/step - loss: 0.0014
```