Practical No : 1

Performing matrix multiplication and finding eigen vectors and eigen values using TensorFlow

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 :

```
Matrix Multiplication Demo
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:
[[3.0372672 6.6474676]
 [5.6447554 4.8327074]]


Eigen Vectors:
[[-0.76061237 -0.6492064 ]
 [ 0.6492064  -0.76061237]]

Eigen Values:
[-1.7807075  9.650682 ]

>>> |
```

Practical No : 2

Solving XOR problem using deep feed forward network.

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 :

```
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([[-0.19081533, -0.5733911 ],
       [-0.22991914,  0.8169273 ]], dtype=float32), array([0., 0.], dtype=float32), array([[ 0.05397427],
       [-0.3534118 ]], dtype=float32), array([0.], dtype=float32)]
Epoch 1/1000
1/1 [==============================] - ETA: 0s - loss: 0.7213 - accuracy: 0.50001/1 [
==============================] - 3s 3s/step - loss: 0.7213 - accuracy: 0.5000
Epoch 2/1000
1/1 [==============================] - ETA: 0s - loss: 0.7211 - accuracy: 0.50001/1 [
==============================] - 0s 31ms/step - loss: 0.7211 - accuracy: 0.5000
Epoch 3/1000
1/1 [==============================] - ETA: 0s - loss: 0.7210 - accuracy: 0.50001/1 [
==============================] - 0s 15ms/step - loss: 0.7210 - accuracy: 0.5000
Epoch 4/1000
1/1 [==============================] - ETA: 0s - loss: 0.7208 - accuracy: 0.50001/1 [
==============================] - 0s 26ms/step - loss: 0.7208 - accuracy: 0.5000
Epoch 5/1000
1/1 [==============================] - ETA: 0s - loss: 0.7206 - accuracy: 0.50001/1 [
==============================] - 0s 54ms/step - loss: 0.7206 - accuracy: 0.5000
Epoch 6/1000
1/1 [==============================] - ETA: 0s - loss: 0.7205 - accuracy: 0.50001/1 [
==============================] - 0s 30ms/step - loss: 0.7205 - accuracy: 0.5000
Epoch 7/1000
1/1 [==============================] - ETA: 0s - loss: 0.7203 - accuracy: 0.50001/1 [
```

```
Epoch 990/1000
1/1 [==============================] - ETA: 0s - loss: 0.6332 - accuracy: 0.75001/1
==============================] - 0s 14ms/step - loss: 0.6332 - accuracy: 0.7500
Epoch 991/1000
1/1 [==============================] - ETA: 0s - loss: 0.6330 - accuracy: 0.75001/1 [
==============================] - 0s 14ms/step - loss: 0.6330 - accuracy: 0.7500
Epoch 992/1000
1/1 [==============================] - ETA: 0s - loss: 0.6328 - accuracy: 0.75001/1 [
==============================] - 0s 14ms/step - loss: 0.6328 - accuracy: 0.7500
Epoch 993/1000
1/1 [==============================] - ETA: 0s - loss: 0.6326 - accuracy: 0.75001/1 [
==============================] - 0s 14ms/step - loss: 0.6326 - accuracy: 0.7500
Epoch 994/1000
1/1 [==============================] - ETA: 0s - loss: 0.6324 - accuracy: 0.75001/1 [
==============================] - 0s 13ms/step - loss: 0.6324 - accuracy: 0.7500
Epoch 995/1000
1/1 [==============================] - ETA: 0s - loss: 0.6322 - accuracy: 0.75001/1 [
==============================] - 0s 15ms/step - loss: 0.6322 - accuracy: 0.7500
Epoch 996/1000
1/1 [==============================] - ETA: 0s - loss: 0.6320 - accuracy: 0.75001/1 [
==============================] - 0s 14ms/step - loss: 0.6320 - accuracy: 0.7500
Epoch 997/1000
1/1 [==============================] - ETA: 0s - loss: 0.6318 - accuracy: 0.75001/1 [
==============================] - 0s 15ms/step - loss: 0.6318 - accuracy: 0.7500
Epoch 998/1000
1/1 [==============================] - ETA: 0s - loss: 0.6316 - accuracy: 0.75001/1 [
==============================] - 0s 15ms/step - loss: 0.6316 - accuracy: 0.7500
Epoch 999/1000
1/1 [==============================] - ETA: 0s - loss: 0.6314 - accuracy: 0.75001/1 [
==============================] - 0s 15ms/step - loss: 0.6314 - accuracy: 0.7500
Epoch 1000/1000
1/1 [==============================] - ETA: 0s - loss: 0.6312 - accuracy: 0.75001/1 [
==============================] - 0s 15ms/step - loss: 0.6312 - accuracy: 0.7500
[array([[-0.19081533, -0.8942301 ],
       [-0.22991914,  0.89508426]], dtype=float32), array([ 0.        , -0.00045995], dtype=float32), array([[0.05397427],
       [0.62021667]], dtype=float32), array([-0.13156578], dtype=float32)]
[[0.46715593]
 [0.6042715 ]
 [0.46715593]
 [0.4672168 ]]
>>>
```

Practical No : 3

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.

```
>>> from numpy import loadtxt
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>>
>>>
```

```
>>> dataset= loadtxt('pima-indians-diabetes.csv', delimiter = ',')
>>> dataset
array([[  6.   , 148.   ,  72.   , ...,   0.627,  50.   ,   1.   ],
       [  1.   ,  85.   ,  66.   , ...,   0.351,  31.   ,   0.   ],
       [  8.   , 183.   ,  64.   , ...,   0.672,  32.   ,   1.   ],
       ...,
       [  5.   , 121.   ,  72.   , ...,   0.245,  30.   ,   0.   ],
       [  1.   , 126.   ,  60.   , ...,   0.349,  47.   ,   1.   ],
       [  1.   ,  93.   ,  70.   , ...,   0.315,  23.   ,   0.   ]])
>>> X=dataset[:,0:8]
>>> Y=dataset[:,8]
>>> X
array([[  6.   , 148.   ,  72.   , ...,  33.6  ,   0.627,  50.   ],
       [  1.   ,  85.   ,  66.   , ...,  26.6  ,   0.351,  31.   ],
       [  8.   , 183.   ,  64.   , ...,  23.3  ,   0.672,  32.   ],
       ...,
       [  5.   , 121.   ,  72.   , ...,  26.2  ,   0.245,  30.   ],
       [  1.   , 126.   ,  60.   , ...,  30.1  ,   0.349,  47.   ],
       [  1.   ,  93.   ,  70.   , ...,  30.4  ,   0.315,  23.   ]])
>>> Y
array([1., 0., 1., 0., 1., 0., 1., 0., 1., 1., 0., 1., 0., 1., 1., 1., 1.,
       1., 0., 1., 0., 0., 1., 1., 1., 1., 1., 0., 0., 0., 0., 1., 0., 0.,
       0., 0., 0., 1., 1., 1., 0., 0., 0., 1., 0., 1., 0., 0., 1., 0., 0.,
       0., 0., 1., 0., 0., 1., 0., 0., 0., 0., 1., 0., 0., 1., 0., 1., 0.,
       0., 0., 1., 0., 1., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 1.,
       0., 0., 0., 1., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 1., 1., 0.,
       0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 0., 0., 1., 1., 1., 0., 0.,
       0., 1., 0., 0., 0., 1., 1., 0., 0., 1., 1., 1., 1., 1., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
       0., 1., 1., 0., 0., 0., 1., 0., 0., 0., 0., 1., 1., 0., 0., 0., 0.,
       1., 1., 0., 0., 0., 1., 0., 1., 0., 1., 0., 0., 0., 0., 0., 1., 1.,
       1., 1., 1., 0., 0., 1., 1., 0., 1., 0., 1., 1., 1., 0., 0., 0., 0.,
       0., 0., 1., 1., 0., 1., 0., 0., 0., 1., 1., 1., 1., 0., 1., 1., 1.,
```

Creating model:

```
>>> model = Sequential()
```

```
>>> model.add(Dense(12, input_dim=8, activation='relu'))
>>> model.add(Dense(8, activation='relu'))
>>> model.add(Dense(1,activation='sigmoid'))
>>>
```

Compiling and fitting model:

```
>>> model.compile(loss ='binary_crossentropy',optimizer ='adam',metrics =['accuracy'])
>>> model.fit(X,Y,epochs=150,batch_size=10)

Epoch 1/150
 1/77 [..............................] - ETA: 58s - loss: 0.4628 - accuracy: 0.8000
26/77 [=========>....................] - ETA: 0s - loss: 0.6344 - accuracy: 0.7269 
49/77 [==================>...........] - ETA: 0s - loss: 0.5815 - accuracy: 0.742970
/77 [==========================>...] - ETA: 0s - loss: 0.5612 - accuracy: 0.744377/7
7 [==============================] - 1s 2ms/step - loss: 0.5528 - accuracy: 0.7487
Epoch 2/150
 1/77 [..............................] - ETA: 0s - loss: 0.2094 - accuracy: 0.90002
6/77 [=========>....................] - ETA: 0s - loss: 0.4719 - accuracy: 0.780850/
77 [==================>...........] - ETA: 0s - loss: 0.4759 - accuracy: 0.792073/77
[===========================>..] - ETA: 0s - loss: 0.4843 - accuracy: 0.793277/77 [=
=============================] - 0s 2ms/step - loss: 0.4873 - accuracy: 0.7917
Epoch 3/150
 1/77 [..............................] - ETA: 0s - loss: 0.3357 - accuracy: 1.00002
5/77 [========>.....................] - ETA: 0s - loss: 0.5162 - accuracy: 0.756048/
77 [=================>............] - ETA: 0s - loss: 0.5344 - accuracy: 0.760469/77
[=========================>....] - ETA: 0s - loss: 0.5128 - accuracy: 0.771077/77 [=
=============================] - 0s 2ms/step - loss: 0.5099 - accuracy: 0.7695
Epoch 4/150
 1/77 [..............................] - ETA: 0s - loss: 0.6268 - accuracy: 0.60001
3/77 [====>.........................] - ETA: 0s - loss: 0.4346 - accuracy: 0.784622/
77 [=======>......................] - ETA: 0s - loss: 0.4642 - accuracy: 0.777330/77
[==========>...................] - ETA: 0s - loss: 0.4646 - accuracy: 0.776735/77 [=
===========>.................] - ETA: 0s - loss: 0.4728 - accuracy: 0.771439/77 [===
===========>...............] - ETA: 0s - loss: 0.4721 - accuracy: 0.766741/77 [=====
=========>..............] - ETA: 0s - loss: 0.4675 - accuracy: 0.775642/77 [=======
========>.............] - ETA: 0s - loss: 0.4671 - accuracy: 0.778649/77 [=========
=========>..........] - ETA: 0s - loss: 0.4779 - accuracy: 0.771466/77 [===========
=============>.....] - ETA: 0s - loss: 0.4786 - accuracy: 0.778877/77 [=============
=================] - 1s 7ms/step - loss: 0.4945 - accuracy: 0.7656
Epoch 5/150
 1/77 [..............................] - ETA: 0s - loss: 0.4703 - accuracy: 0.80002
4/77 [========>.....................] - ETA: 0s - loss: 0.4994 - accuracy: 0.783346/
77 [================>............] - ETA: 0s - loss: 0.5047 - accuracy: 0.763068/77
[=========================>....] - ETA: 0s - loss: 0.5105 - accuracy: 0.755977/77 [=
=============================] - 0s 2ms/step - loss: 0.5239 - accuracy: 0.7513
```

Evaluating the accuracy:

```
>>> _,accuracy = model.evaluate(X,Y)

 1/24 [>.............................] - ETA: 9s - loss: 0.6747 - accuracy: 0.6250
8/24 [=========>....................] - ETA: 0s - loss: 0.4596 - accuracy: 0.789124/
24 [==============================] - 0s 4ms/step - loss: 0.4141 - accuracy: 0.8112
>>> 
```

```
>>> print('Accuracy of model is',(accuracy*100))

Accuracy of model is 81.11979365348816
>>> 
```

Using model for prediction class:

```
>>> prediction=model.predict(X)
>>> exec("for i in range(5) :print(X[i].tolist(),prediction[i],Y[i])")
[6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0] [0.5428344] 1.0
[1.0, 85.0, 66.0, 29.0, 0.0, 26.6, 0.351, 31.0] [0.08321831] 0.0
[8.0, 183.0, 64.0, 0.0, 0.0, 23.3, 0.672, 32.0] [0.8285245] 1.0
[1.0, 89.0, 66.0, 23.0, 94.0, 28.1, 0.167, 21.0] [0.01245207] 0.0
[0.0, 137.0, 40.0, 35.0, 168.0, 43.1, 2.288, 33.0] [0.8305522] 1.0
>>> 
```

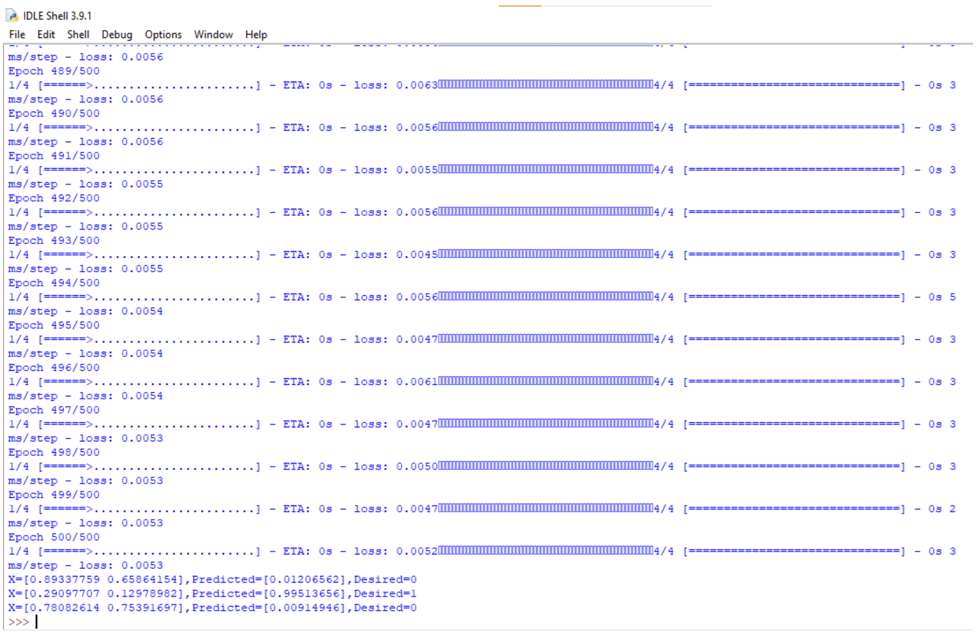
Practical No : 4

A] Using deep feed forward network with two hidden layers for performing classification and predicting the class.

Code :

```
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(Xnew)
for i in range(len(Xnew)):
 print("X=%s,Predicted=%s,Desired=%s"%(Xnew[i],Ynew[i],Yreal[i]))
```

Output :



```
IDLE Shell 3.9.1
File  Edit  Shell  Debug  Options  Window  Help
ms/step - loss: 0.0056
Epoch 489/500
1/4 [======>.......................] - ETA: 0s - loss: 0.00634/4 [==============================] - 0s 3
ms/step - loss: 0.0056
Epoch 490/500
1/4 [======>.......................] - ETA: 0s - loss: 0.00564/4 [==============================] - 0s 3
ms/step - loss: 0.0056
Epoch 491/500
1/4 [======>.......................] - ETA: 0s - loss: 0.00554/4 [==============================] - 0s 3
ms/step - loss: 0.0055
Epoch 492/500
1/4 [======>.......................] - ETA: 0s - loss: 0.00564/4 [==============================] - 0s 3
ms/step - loss: 0.0055
Epoch 493/500
1/4 [======>.......................] - ETA: 0s - loss: 0.00454/4 [==============================] - 0s 3
ms/step - loss: 0.0055
Epoch 494/500
1/4 [======>.......................] - ETA: 0s - loss: 0.00564/4 [==============================] - 0s 5
ms/step - loss: 0.0054
Epoch 495/500
1/4 [======>.......................] - ETA: 0s - loss: 0.00474/4 [==============================] - 0s 3
ms/step - loss: 0.0054
Epoch 496/500
1/4 [======>.......................] - ETA: 0s - loss: 0.00614/4 [==============================] - 0s 3
ms/step - loss: 0.0054
Epoch 497/500
1/4 [======>.......................] - ETA: 0s - loss: 0.00474/4 [==============================] - 0s 3
ms/step - loss: 0.0053
Epoch 498/500
1/4 [======>.......................] - ETA: 0s - loss: 0.00504/4 [==============================] - 0s 3
ms/step - loss: 0.0053
Epoch 499/500
1/4 [======>.......................] - ETA: 0s - loss: 0.00474/4 [==============================] - 0s 2
ms/step - loss: 0.0053
Epoch 500/500
1/4 [======>.......................] - ETA: 0s - loss: 0.00524/4 [==============================] - 0s 3
ms/step - loss: 0.0053
X=[0.89337759 0.65864154],Predicted=[0.01206562],Desired=0
X=[0.29097707 0.12978982],Predicted=[0.99513656],Desired=1
X=[0.78082614 0.75391697],Predicted=[0.00914946],Desired=0
>>>
```

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

Code :

```
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(Xnew)

Ynew=model.predict(Xnew)

for i in range(len(Xnew)):

 print("X=%s,Predicted_probability=%s,Predicted_class=%s"%(Xnew[i],Ynew[i],Yclass[i]))
```

Output :

```
ms/step - loss: 0.6932
Epoch 489/500
1/4 [======>.......................] - ETA: 0s - loss: 0.69474/4 [==============================] - 0s 3
ms/step - loss: 0.6931
Epoch 490/500
1/4 [======>.......................] - ETA: 0s - loss: 0.69184/4 [==============================] - 0s 3
ms/step - loss: 0.6932
Epoch 491/500
1/4 [======>.......................] - ETA: 0s - loss: 0.69354/4 [==============================] - 0s 2
ms/step - loss: 0.6932
Epoch 492/500
1/4 [======>.......................] - ETA: 0s - loss: 0.69154/4 [==============================]
ms/step - loss: 0.6932
Epoch 493/500
1/4 [======>.......................] - ETA: 0s - loss: 0.69274/4 [==============================] - 0s 2
ms/step - loss: 0.6932
Epoch 494/500
1/4 [======>.......................] - ETA: 0s - loss: 0.69514/4 [==============================] - 0s 3
ms/step - loss: 0.6932
Epoch 495/500
1/4 [======>.......................] - ETA: 0s - loss: 0.69224/4 [==============================] - 0s 2
ms/step - loss: 0.6932
Epoch 496/500
1/4 [======>.......................] - ETA: 0s - loss: 0.69324/4 [==============================] - 0s 4
ms/step - loss: 0.6932
Epoch 497/500
1/4 [======>.......................] - ETA: 0s - loss: 0.69424/4 [==============================] - 0s 2
ms/step - loss: 0.6932
Epoch 498/500
1/4 [======>.......................] - ETA: 0s - loss: 0.69224/4 [==============================] - 0s 3
ms/step - loss: 0.6932
Epoch 499/500
1/4 [======>.......................] - ETA: 0s - loss: 0.69324/4 [==============================] - 0s 2
ms/step - loss: 0.6932
Epoch 500/500
1/4 [======>.......................] - ETA: 0s - loss: 0.69214/4 [==============================] - 0s 3
ms/step - loss: 0.6932
X=[0.89337759 0.65864154],Predicted_probability=[0.49575293],Predicted_class=[0.49575293]
X=[0.29097707 0.12978982],Predicted_probability=[0.49575293],Predicted_class=[0.49575293]
X=[0.78082614 0.75391697],Predicted_probability=[0.49575293],Predicted_class=[0.49575293]
>>>
```

C] Using a deep field forward network with two hidden layers for performing linear regression and predicting values.

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 :

```
X=[0.29466096 0.30317302],Predicted=[0.16992235]
X=[0.39445118 0.79390858],Predicted=[0.7379006]
X=[0.02884127 0.6208843 ],Predicted=[0.4013802]
>>>
```

Practical No : 5

A] Evaluating feed forward deep network for regression using KFold cross validation.

Code :

```
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.pipeline import Pipeline
# load dataset
dataframe = pandas.read_csv("iris.txt", header=None)
dataset = dataframe.values
X = dataset[:,0:4].astype(float)
Y = dataset[:,4]
# 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(8, input_dim=4, activation='relu'))
	model.add(Dense(8, input_dim=4, activation='relu'))
	model.add(Dense(3, activation='linear'))
```

```
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
    return model

estimator = KerasClassifier(build_fn=baseline_model, epochs=10, 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 :

```
Baseline: 36.00% (13.40%)
>>>
```

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

Code :

```
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.pipeline import Pipeline
# load dataset
dataframe = pandas.read_csv("iris.txt", header=None)
dataset = dataframe.values
X = dataset[:,0:4].astype(float)
Y = dataset[:,4]
# 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(8, input_dim=4, activation='relu'))

	model.add(Dense(8, input_dim=4, activation='relu'))

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

	# Compile model

	model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

	return model


estimator = KerasClassifier(build_fn=baseline_model, epochs=10, 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 :

```
Baseline: 80.67% (12.45%)
>>>
```

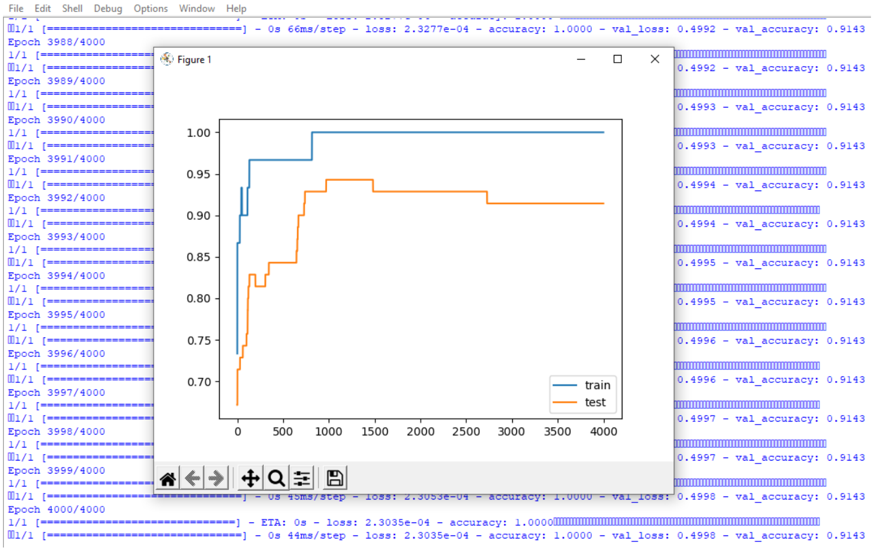
Practical No : 6

Implementing regularization to avoid overfitting in binary classification..

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:]
#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 :



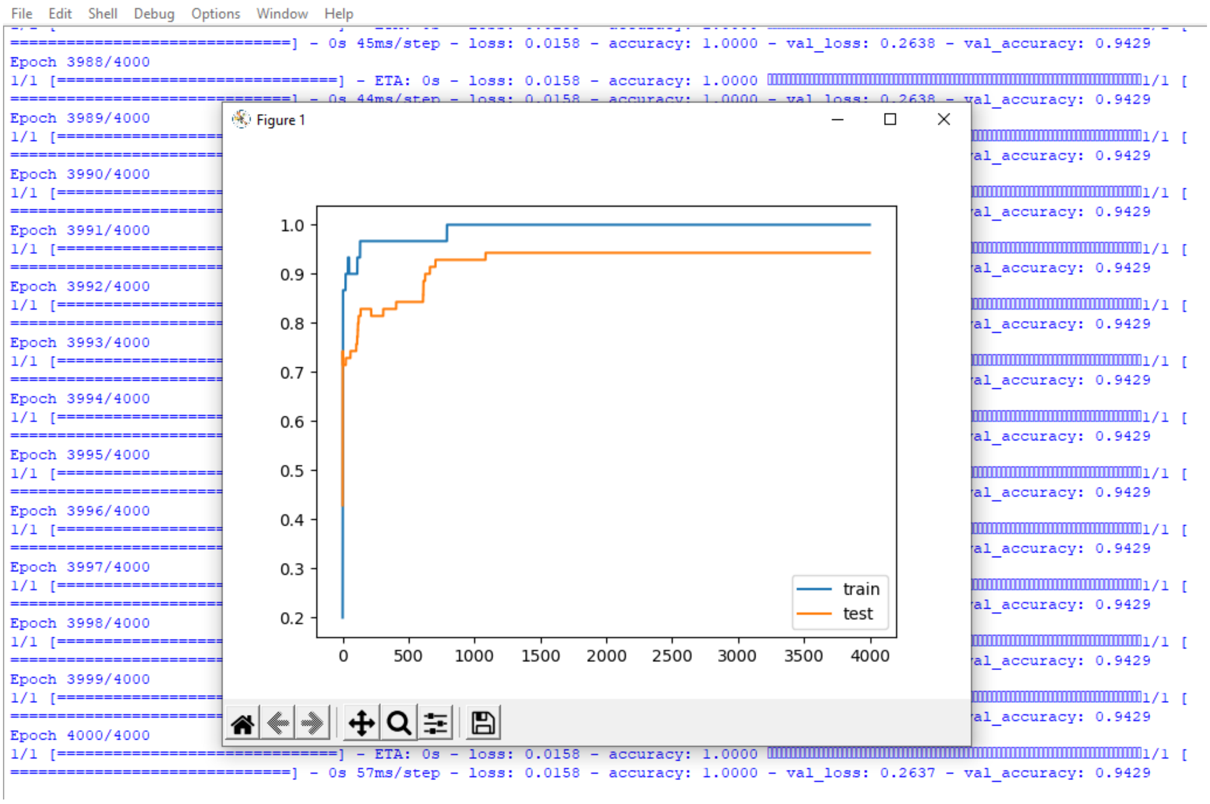
The above code and resultant graph demonstrate overfitting with accuracy of testing data less than accuracy of training data also the accuracy of testing data increases once and then start decreases gradually.to solve this problem we can use regularization.

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

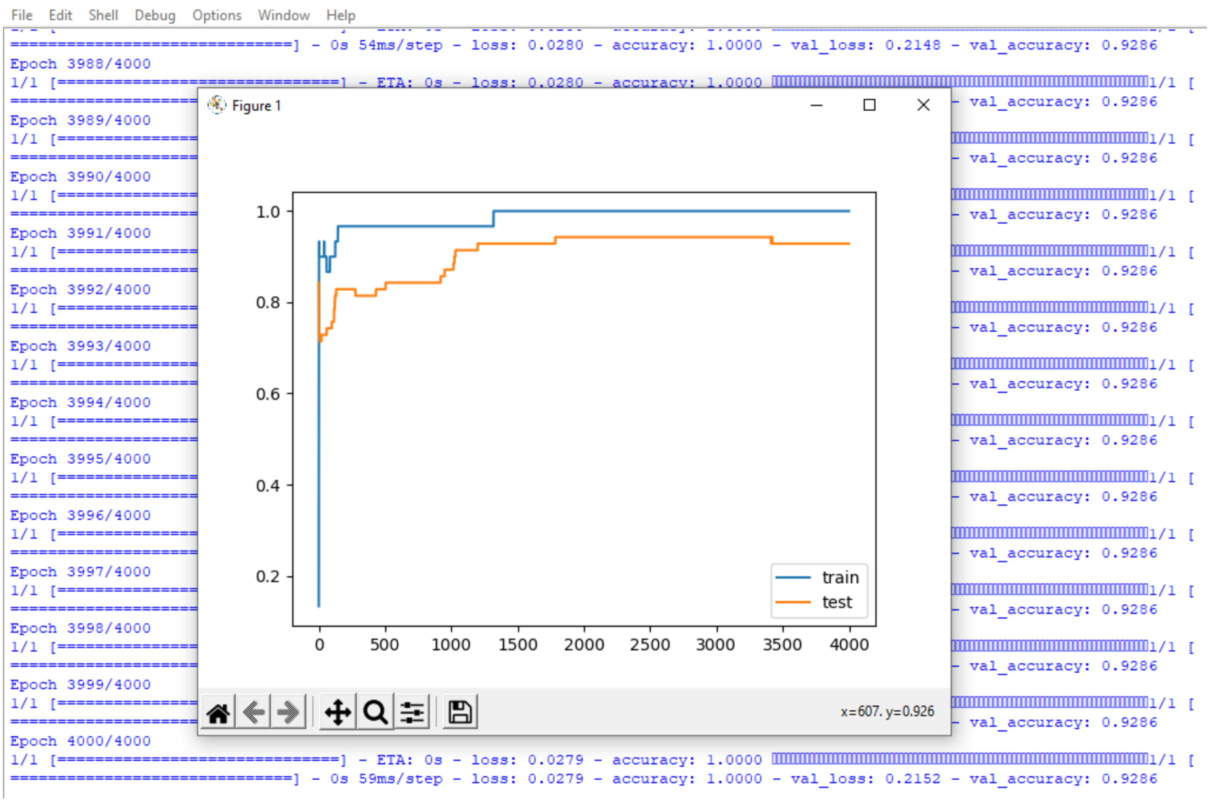
Code :

```
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:]

#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()
```

Output :



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.

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:]
#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 :