**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)

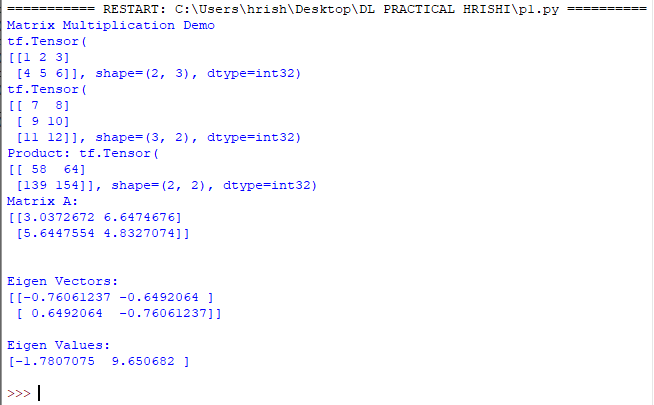
e\_matrix\_A=tf.random.uniform([2,2],minval=3,maxval=10,dtype=tf.float32,name="matrixA")

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 :**



**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.]])

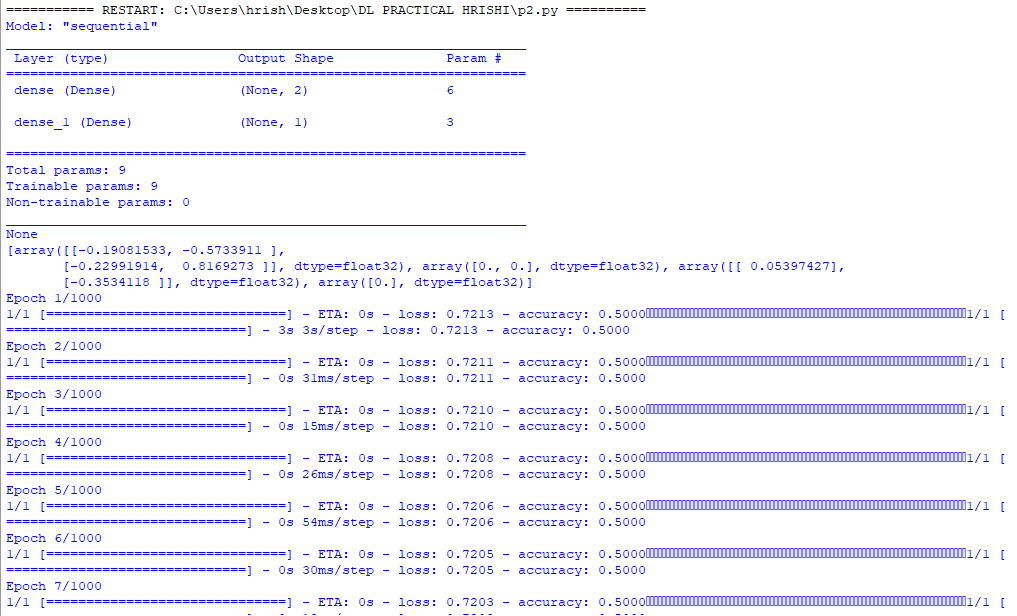
Y=np.array([0.,1.,1.,0.])

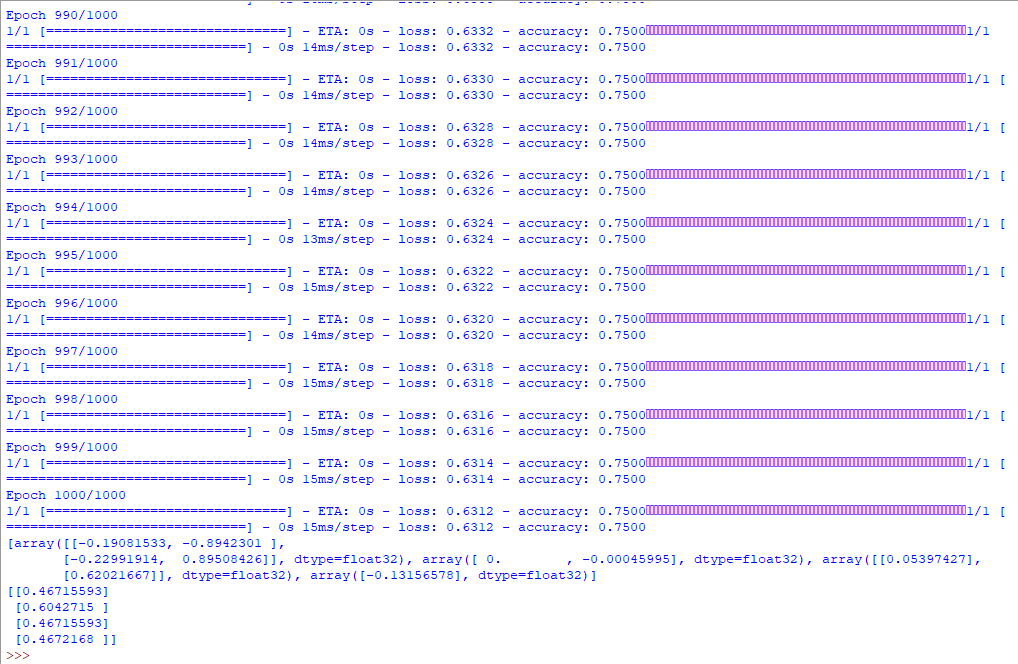
model.fit(X,Y,epochs=1000,batch\_size=4)

print(model.get\_weights())

print(model.predict(X,batch\_size=4))

**Output :**

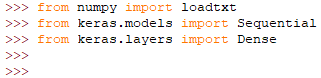


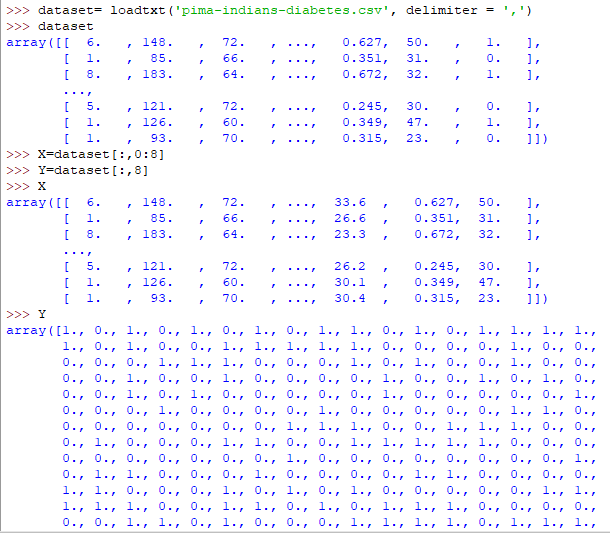


**Practical No : 3**

**Implementing deep neural network for performing classification task.**

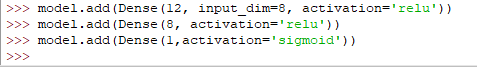
**Problem statement**: the given dataset comprises of health information about diabetic women patient. we need to create deep feed forward network that will classify women suffering from diabetes mellitus as 1.



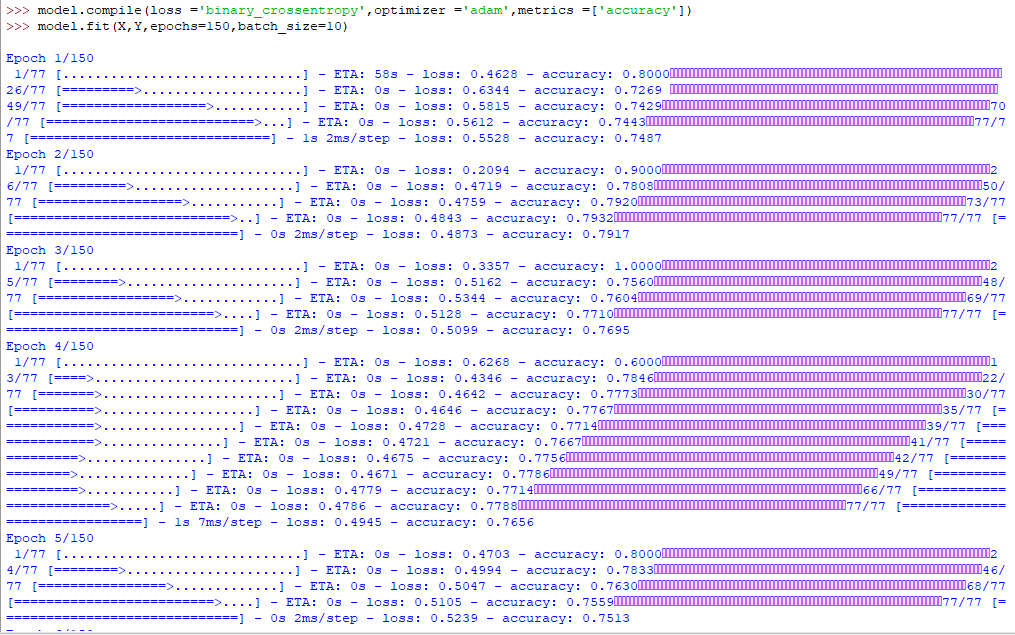


Creating model:

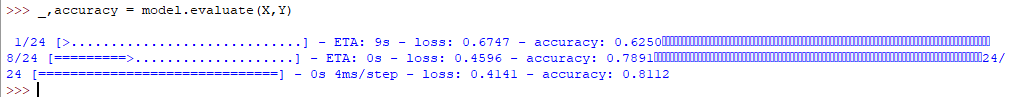




Compiling and fitting model:

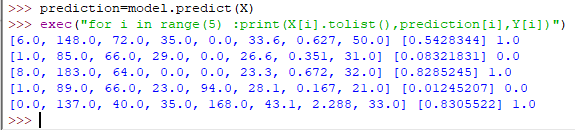


Evaluating the accuracy:



****

Using model for prediction class:



**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 :**

****

**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)

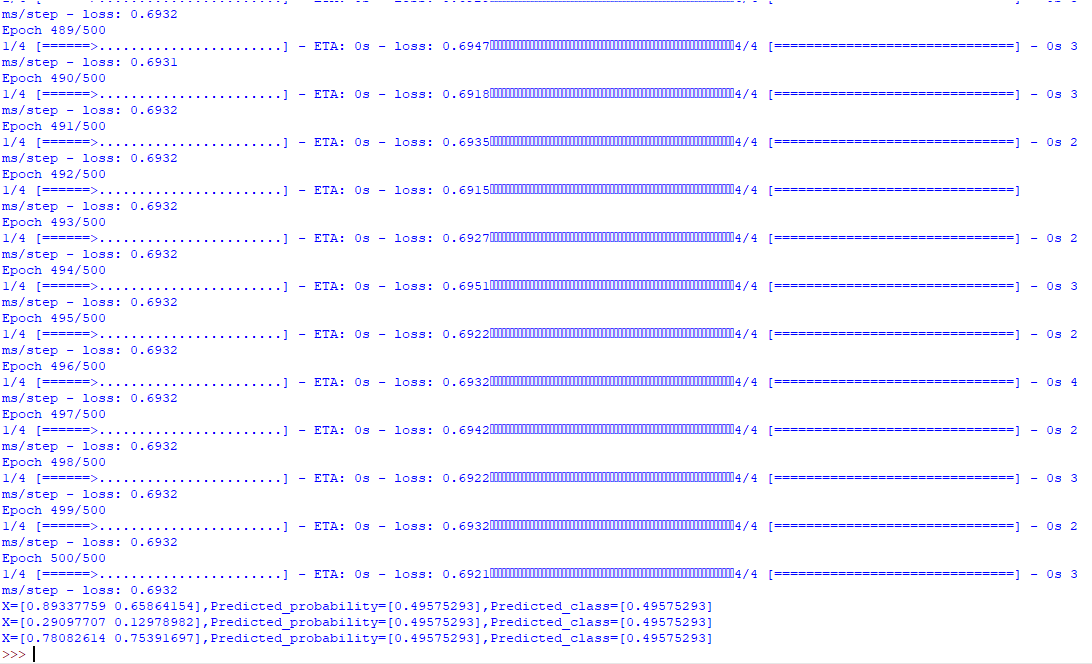
Yclass=model.predict(Xnew)

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 :**



**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)

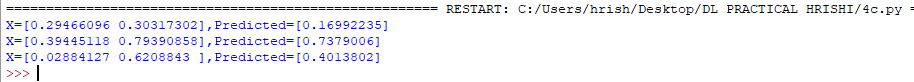
Xnew=scalarX.transform(Xnew)

Ynew=model.predict(Xnew)

for i in range(len(Xnew)):

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

**Output :**



**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 :**

****

**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 :**

****

**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)

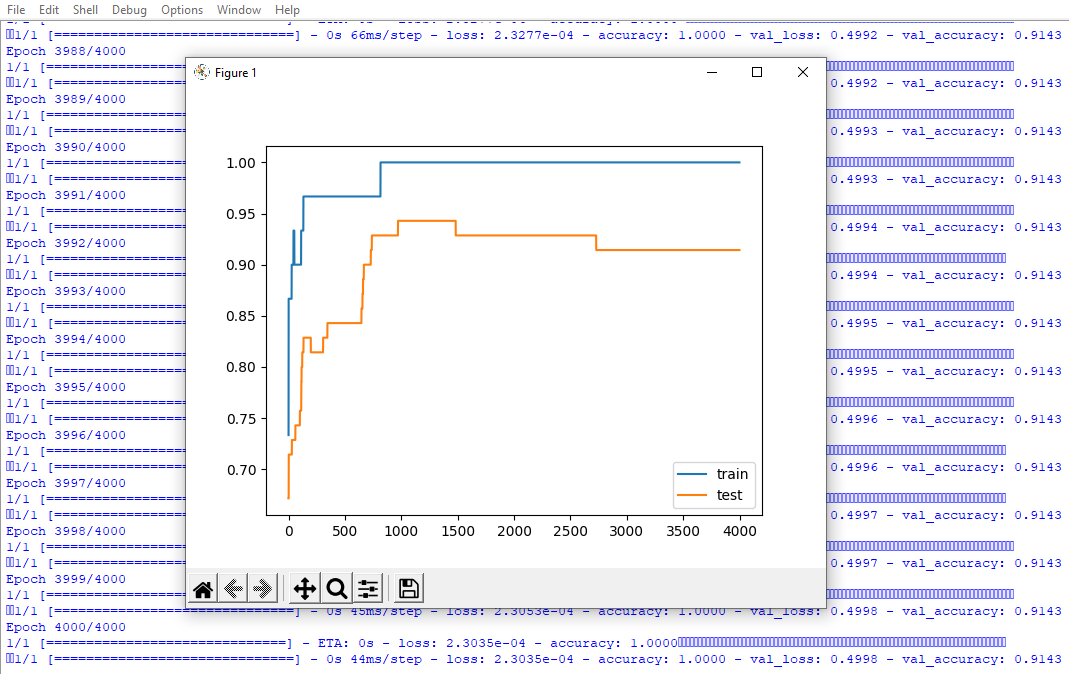
pyplot.plot(history.history['accuracy'],label='train')

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)

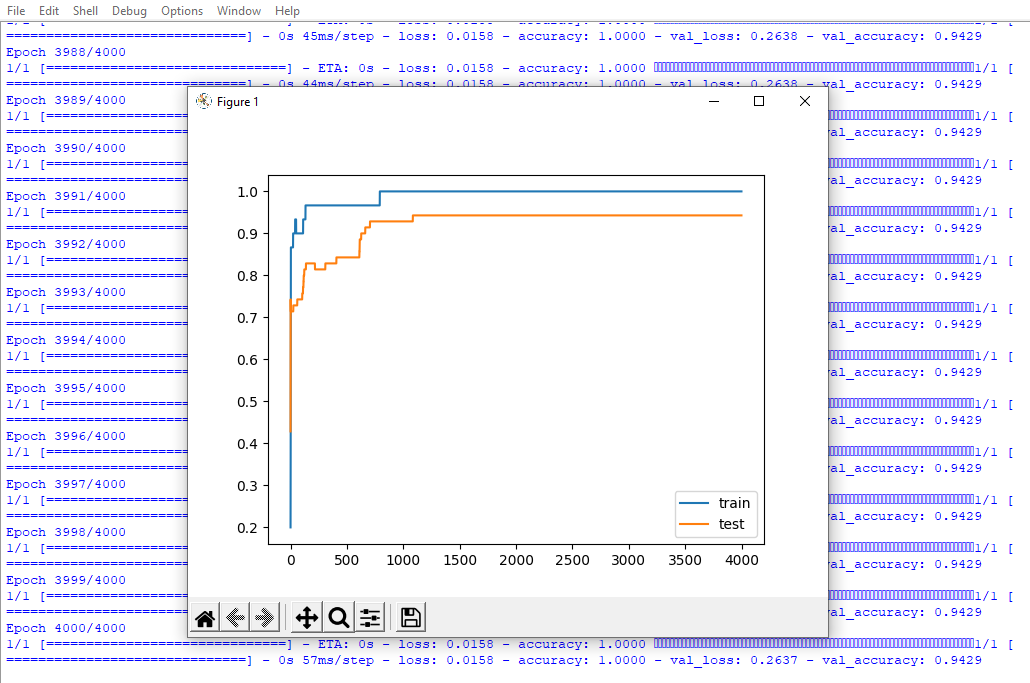
pyplot.plot(history.history['accuracy'],label='train')

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

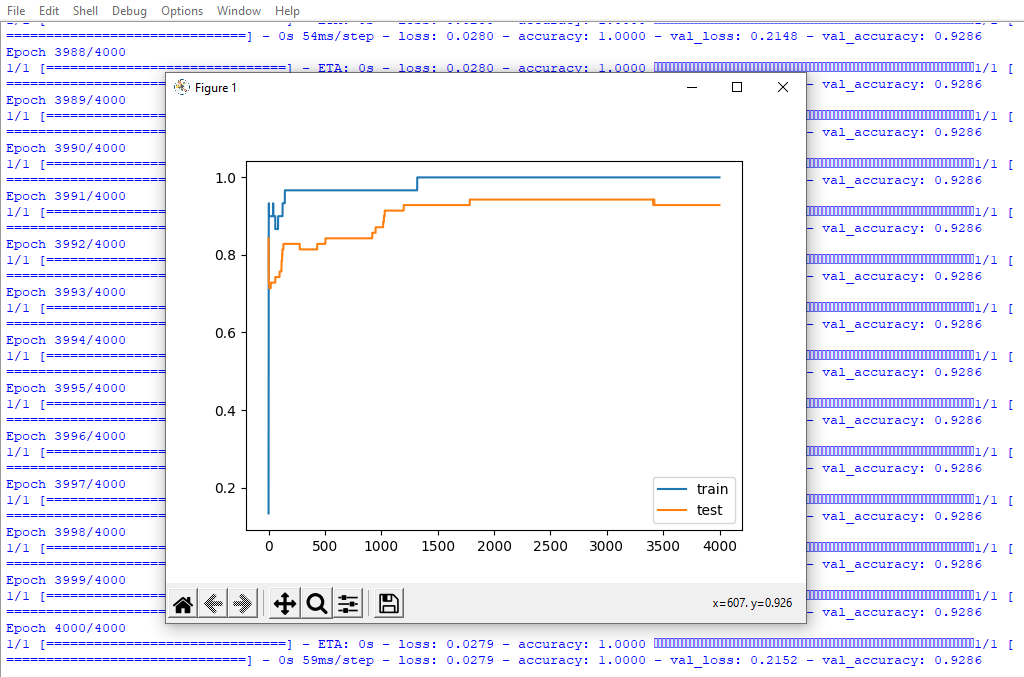
pyplot.legend()

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)

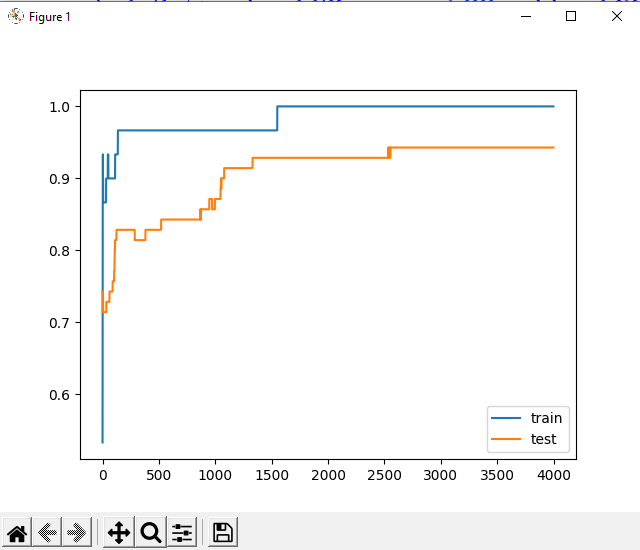
pyplot.plot(history.history['accuracy'],label='train')

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

pyplot.legend()

pyplot.show()

**Output :**

****