## HOMEWORK 4

## *DEEP LEARNING CLASS*

## *PROF RICARDO*

## *APPLYING DNN , NN AND LOGISTIC REGRESSION ON DATASET.*

DATA SET:

**TV Commercial Detection**

Logistic Regression code:

## log reg, nn, deep neural net algorithm

## supervised learning classifier

#############################################################

import tensorflow as tf

import numpy as np

from numpy import genfromtxt

import sklearn

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score, f1\_score

import pandas as pd

import matplotlib.pyplot as plt

#############################################################

## remove if not using a mac

#import os

#os.environ['KMP\_DUPLICATE\_LIB\_OK']='True'

#############################################################

number\_epochs = 50

learning\_rate = 0.1

batch\_size = 1000

#############################################################

x\_train = genfromtxt('times\_train.csv', delimiter=',', usecols=(i for i in range(0,202)) , skip\_header=1)

y\_train = genfromtxt('times\_train.csv', delimiter=',', usecols=(0), skip\_header=1)

x\_test = genfromtxt('times\_test.csv', delimiter=',', usecols=(i for i in range(0,202)) ,skip\_header=1 )

y\_test = genfromtxt('times\_test.csv', delimiter=',', usecols=(0), skip\_header=1)

############################################################

## normalizing

sc = StandardScaler()

sc.fit(x\_train)

x\_train\_normalized = sc.transform(x\_train)

x\_test\_normalized = sc.transform(x\_test)

###########################################################

# Convert to one hot data

def convertOneHot\_data2(data):

y=np.array([int(i) for i in data])

#print y[:20]

rows = len(y)

columns = y.max() + 1

a = np.zeros(shape=(rows,columns))

#print a[:20,:]

print rows

print columns

#rr = raw\_input()

#y\_onehot=[0]\*len(y)

for i,j in enumerate(y):

#y\_onehot[i]=np.array([0]\*(y.max() + 1) )

#y\_onehot[i][j]=1

a[i][j]=1

return (a)

############################################################

# one-hot encoding

depth = 2

#y\_train\_onehot = tf.one\_hot(y\_train, depth)

#y\_test\_onehot = tf.one\_hot(y\_test, depth)

y\_train\_onehot = convertOneHot\_data2(y\_train)

y\_test\_onehot = convertOneHot\_data2(y\_test)

#############################################################

# features (A)

A = len(x\_train[0])

print A # number of features

#############################################################

# classes (B)

B = 2 #len(sess.run(y\_train\_onehot[0]))

print "number of classes ", B

############################################################

## print stats

precision\_scores\_list = []

accuracy\_scores\_list = []

def print\_stats\_metrics(y\_test, y\_pred):

print('Accuracy: %.2f' % accuracy\_score(y\_test, y\_pred) )

#Accuracy: 0.84

accuracy\_scores\_list.append(accuracy\_score(y\_test, y\_pred) )

confmat = confusion\_matrix(y\_true=y\_test, y\_pred=y\_pred)

print "confusion matrix"

print(confmat)

print pd.crosstab(y\_test, y\_pred, rownames=['True'], colnames=['Predicted'], margins=True)

precision\_scores\_list.append(precision\_score(y\_true=y\_test, y\_pred=y\_pred, average='weighted'))

print('Precision: %.3f' % precision\_score(y\_true=y\_test, y\_pred=y\_pred, average='weighted'))

print('Recall: %.3f' % recall\_score(y\_true=y\_test, y\_pred=y\_pred, average='weighted'))

print('F1-measure: %.3f' % f1\_score(y\_true=y\_test, y\_pred=y\_pred, average='weighted'))

##############################################################

def plot\_metric\_per\_epoch():

x\_epochs = []

y\_epochs = []

for i, val in enumerate(precision\_scores\_list):

x\_epochs.append(i)

y\_epochs.append(val)

plt.scatter(x\_epochs, y\_epochs,s=50,c='lightgreen', marker='s', label='score')

plt.xlabel('epoch')

plt.ylabel('score')

plt.title('Score per epoch')

plt.legend()

plt.grid()

plt.show()

############################################################

def layer(input, weight\_shape, bias\_shape):

bias\_init = tf.constant\_initializer(value=0)

weight\_stddev = (2.0/weight\_shape[0])\*\*0.5

w\_init = tf.random\_normal\_initializer(stddev=weight\_stddev)

W = tf.get\_variable("W", weight\_shape, initializer=w\_init)

b = tf.get\_variable("b", bias\_shape, initializer=bias\_init)

return tf.nn.relu( tf.matmul(input, W) + b )

############################################################

## a neural network

#def inference\_nn(x, A, B):

with tf.variable\_scope("hidden\_1"):

hidden\_1 = layer(x, [A, 128], [128] )

with tf.variable\_scope("output"):

output = layer(hidden\_1, [128,B], [B])

return output

############################################################

## deep neural network 3 hidden layers

#def inference\_deep\_3layers(x, A, B):

with tf.variable\_scope("hidden\_1"):

hidden\_1 = layer(x, [A,300], [300])

with tf.variable\_scope("hidden\_2"):

hidden\_2 = layer(hidden\_1, [300,200], [200])

with tf.variable\_scope("hidden\_3"):

hidden\_3 = layer(hidden\_2, [200,100], [100])

with tf.variable\_scope("output"):

output = layer(hidden\_3, [100, B], [B])

return output

############################################################

## logistic regression

def inference(x, A, B):

W = tf.Variable( tf.zeros([A, B]) )

b = tf.Variable(tf.zeros( [B]) )

output = tf.nn.softmax( tf.matmul(x, W) + b )

return output

############################################################

#def loss\_deep(output, y):

xentropy = tf.nn.softmax\_cross\_entropy\_with\_logits(logits=output, labels=y)

loss = tf.reduce\_mean(xentropy)

return loss

############################################################

def loss(output, y):

output = tf.clip\_by\_value(output, 1e-10, 1.0)

dot\_product = y \* tf.log(output)

xentropy = -tf.reduce\_sum( dot\_product )

loss = tf.reduce\_mean( xentropy )

return loss

#############################################################

def training(cost):

optimizer = tf.train.GradientDescentOptimizer(learning\_rate)

train\_op = optimizer.minimize(cost)

return train\_op

#############################################################

def evaluate(output, y):

correct\_prediction = tf.equal( tf.argmax(output,1) , tf.argmax(y,1) )

accuracy = tf.reduce\_mean( tf.cast(correct\_prediction, "float") )

return accuracy

#############################################################

x = tf.placeholder("float", [None, A])

y = tf.placeholder("float", [None, B])

#############################################################

## call the core functions

#output = inference\_nn(x, A, B) ## nn

#cost = loss\_deep(output, y) ## nn

#output = inference\_deep\_3layers(x, A, B) ## deep nn

#cost = loss\_deep(output, y) ## deep nn

output = inference(x, A, B) ###log reg

cost = loss(output, y) ###log reg

train\_op = training(cost)

eval\_op = evaluate(output, y)

############################################################

y\_pred\_metrics = tf.argmax(output,1)

############################################################

init = tf.global\_variables\_initializer()

sess = tf.Session()

sess.run(init)

###########################################################

## batch parameters

num\_samples\_train = len(y\_train)

print num\_samples\_train

num\_batches = int(num\_samples\_train/batch\_size)

############################################################

# MAIN\_LOOP()

#y\_test\_temp = sess.run(y\_test\_onehot)

#y\_train\_temp = sess.run(y\_train\_onehot)

y\_test\_temp = y\_test\_onehot

y\_train\_temp = y\_train\_onehot

print "running..."

for i in range(number\_epochs):

for batch\_n in range(num\_batches):

sta= batch\_n\*batch\_size

end= sta + batch\_size

sess.run( train\_op , feed\_dict={x: x\_train\_normalized[sta:end,:] , y: y\_train\_temp[sta:end, :]})

print "accuracy ..."

accuracy\_value, y\_pred = sess.run([eval\_op, y\_pred\_metrics], feed\_dict={x: x\_test\_normalized, y: y\_test\_temp})

print "run {}, {}".format(i, accuracy\_value)

y\_true = np.argmax(y\_test\_temp, 1)

print y\_true

print y\_pred

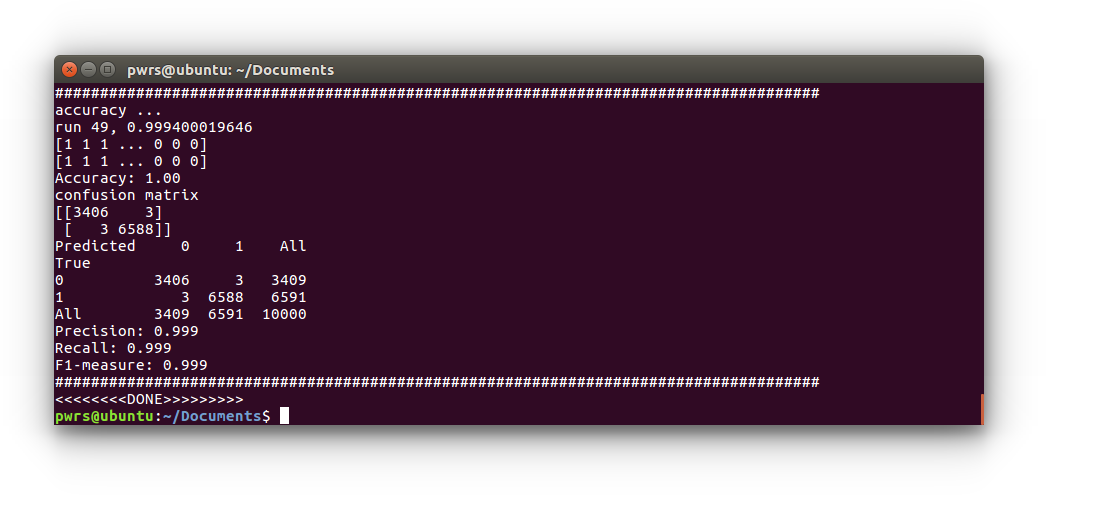
print\_stats\_metrics(y\_true, y\_pred)

print '#####################################################################################'

#############################################################

print "<<<<<<<<DONE>>>>>>>>>"

**Output:**



**Neural Network code:**

## log reg, nn, deep neural net algorithm

## supervised learning classifier

#############################################################

import tensorflow as tf

import numpy as np

from numpy import genfromtxt

import sklearn

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score, f1\_score

import pandas as pd

import matplotlib.pyplot as plt

#############################################################

## remove if not using a mac

#import os

#os.environ['KMP\_DUPLICATE\_LIB\_OK']='True'

#############################################################

number\_epochs = 50

learning\_rate = 0.1

batch\_size = 1000

#############################################################

x\_train = genfromtxt('times\_train.csv', delimiter=',', usecols=(i for i in range(0,202)) , skip\_header=1)

y\_train = genfromtxt('times\_train.csv', delimiter=',', usecols=(0), skip\_header=1)

x\_test = genfromtxt('times\_test.csv', delimiter=',', usecols=(i for i in range(0,202)) ,skip\_header=1 )

y\_test = genfromtxt('times\_test.csv', delimiter=',', usecols=(0), skip\_header=1)

############################################################

## normalizing

sc = StandardScaler()

sc.fit(x\_train)

x\_train\_normalized = sc.transform(x\_train)

x\_test\_normalized = sc.transform(x\_test)

###########################################################

# Convert to one hot data

def convertOneHot\_data2(data):

y=np.array([int(i) for i in data])

#print y[:20]

rows = len(y)

columns = y.max() + 1

a = np.zeros(shape=(rows,columns))

#print a[:20,:]

print rows

print columns

#rr = raw\_input()

#y\_onehot=[0]\*len(y)

for i,j in enumerate(y):

#y\_onehot[i]=np.array([0]\*(y.max() + 1) )

#y\_onehot[i][j]=1

a[i][j]=1

return (a)

############################################################

# one-hot encoding

depth = 2

#y\_train\_onehot = tf.one\_hot(y\_train, depth)

#y\_test\_onehot = tf.one\_hot(y\_test, depth)

y\_train\_onehot = convertOneHot\_data2(y\_train)

y\_test\_onehot = convertOneHot\_data2(y\_test)

#############################################################

# features (A)

A = len(x\_train[0])

print A # number of features

#############################################################

# classes (B)

B = 2 #len(sess.run(y\_train\_onehot[0]))

print "number of classes ", B

############################################################

## print stats

precision\_scores\_list = []

accuracy\_scores\_list = []

def print\_stats\_metrics(y\_test, y\_pred):

print('Accuracy: %.2f' % accuracy\_score(y\_test, y\_pred) )

#Accuracy: 0.84

accuracy\_scores\_list.append(accuracy\_score(y\_test, y\_pred) )

confmat = confusion\_matrix(y\_true=y\_test, y\_pred=y\_pred)

print "confusion matrix"

print(confmat)

print pd.crosstab(y\_test, y\_pred, rownames=['True'], colnames=['Predicted'], margins=True)

precision\_scores\_list.append(precision\_score(y\_true=y\_test, y\_pred=y\_pred, average='weighted'))

print('Precision: %.3f' % precision\_score(y\_true=y\_test, y\_pred=y\_pred, average='weighted'))

print('Recall: %.3f' % recall\_score(y\_true=y\_test, y\_pred=y\_pred, average='weighted'))

print('F1-measure: %.3f' % f1\_score(y\_true=y\_test, y\_pred=y\_pred, average='weighted'))

##############################################################

def plot\_metric\_per\_epoch():

x\_epochs = []

y\_epochs = []

for i, val in enumerate(precision\_scores\_list):

x\_epochs.append(i)

y\_epochs.append(val)

plt.scatter(x\_epochs, y\_epochs,s=50,c='lightgreen', marker='s', label='score')

plt.xlabel('epoch')

plt.ylabel('score')

plt.title('Score per epoch')

plt.legend()

plt.grid()

plt.show()

############################################################

def layer(input, weight\_shape, bias\_shape):

bias\_init = tf.constant\_initializer(value=0)

weight\_stddev = (2.0/weight\_shape[0])\*\*0.5

w\_init = tf.random\_normal\_initializer(stddev=weight\_stddev)

W = tf.get\_variable("W", weight\_shape, initializer=w\_init)

b = tf.get\_variable("b", bias\_shape, initializer=bias\_init)

return tf.nn.relu( tf.matmul(input, W) + b )

############################################################

## a neural network

def inference\_nn(x, A, B):

with tf.variable\_scope("hidden\_1"):

hidden\_1 = layer(x, [A, 10], [10] )

with tf.variable\_scope("output"):

output = layer(hidden\_1, [10,B], [B])

return output

############################################################

## deep neural network 3 hidden layers

#def inference\_deep\_3layers(x, A, B):

with tf.variable\_scope("hidden\_1"):

hidden\_1 = layer(x, [A,300], [300])

with tf.variable\_scope("hidden\_2"):

hidden\_2 = layer(hidden\_1, [300,200], [200])

with tf.variable\_scope("hidden\_3"):

hidden\_3 = layer(hidden\_2, [200,100], [100])

with tf.variable\_scope("output"):

output = layer(hidden\_3, [100, B], [B])

return output

############################################################

## logistic regression

#def inference(x, A, B):

W = tf.Variable( tf.zeros([A, B]) )

b = tf.Variable(tf.zeros( [B]) )

output = tf.nn.softmax( tf.matmul(x, W) + b )

return output

############################################################

def loss\_deep(output, y):

xentropy = tf.nn.softmax\_cross\_entropy\_with\_logits(logits=output, labels=y)

loss = tf.reduce\_mean(xentropy)

return loss

############################################################

#def loss(output, y):

output = tf.clip\_by\_value(output, 1e-10, 1.0)

dot\_product = y \* tf.log(output)

xentropy = -tf.reduce\_sum( dot\_product )

loss = tf.reduce\_mean( xentropy )

return loss

#############################################################

def training(cost):

optimizer = tf.train.GradientDescentOptimizer(learning\_rate)

train\_op = optimizer.minimize(cost)

return train\_op

#############################################################

def evaluate(output, y):

correct\_prediction = tf.equal( tf.argmax(output,1) , tf.argmax(y,1) )

accuracy = tf.reduce\_mean( tf.cast(correct\_prediction, "float") )

return accuracy

#############################################################

x = tf.placeholder("float", [None, A])

y = tf.placeholder("float", [None, B])

#############################################################

## call the core functions

#output = inference\_deep\_3layers(x, A, B) ## deep nn

#cost = loss\_deep(output, y) ## deep nn

#output = inference(x, A, B) ###log reg

#cost = loss(output, y) ###log reg

output = inference\_nn(x, A, B) ## nn

cost = loss\_deep(output, y) ## nn

train\_op = training(cost)

eval\_op = evaluate(output, y)

############################################################

y\_pred\_metrics = tf.argmax(output,1)

############################################################

init = tf.global\_variables\_initializer()

sess = tf.Session()

sess.run(init)

###########################################################

## batch parameters

num\_samples\_train = len(y\_train)

print num\_samples\_train

num\_batches = int(num\_samples\_train/batch\_size)

############################################################

# MAIN\_LOOP()

#y\_test\_temp = sess.run(y\_test\_onehot)

#y\_train\_temp = sess.run(y\_train\_onehot)

y\_test\_temp = y\_test\_onehot

y\_train\_temp = y\_train\_onehot

print "running..."

for i in range(number\_epochs):

for batch\_n in range(num\_batches):

sta= batch\_n\*batch\_size

end= sta + batch\_size

sess.run( train\_op , feed\_dict={x: x\_train\_normalized[sta:end,:] , y: y\_train\_temp[sta:end, :]})

print "accuracy ..."

accuracy\_value, y\_pred = sess.run([eval\_op, y\_pred\_metrics], feed\_dict={x: x\_test\_normalized, y: y\_test\_temp})

print "run {}, {}".format(i, accuracy\_value)

y\_true = np.argmax(y\_test\_temp, 1)

print y\_true

print y\_pred

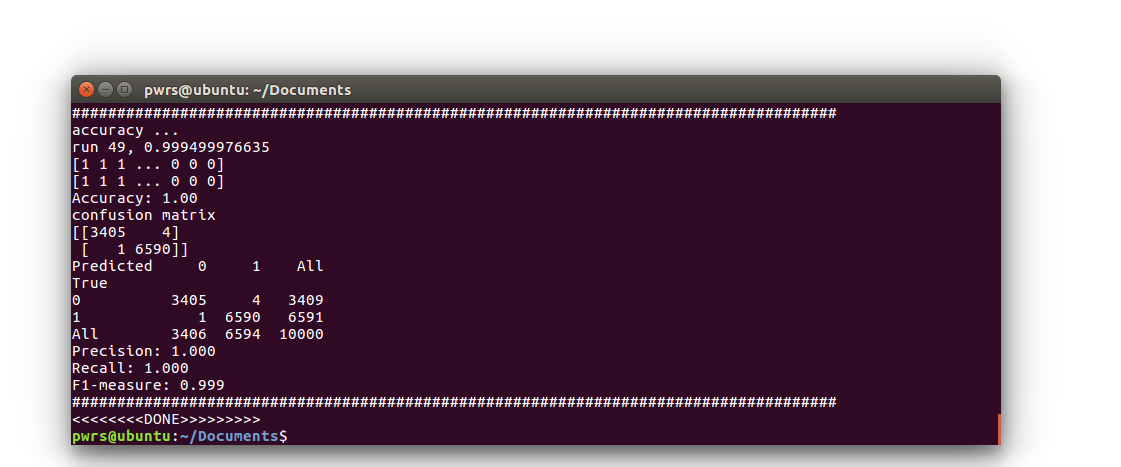
print\_stats\_metrics(y\_true, y\_pred)

print '#####################################################################################'

#############################################################

print "<<<<<<<<DONE>>>>>>>>>"

**Output:**



**2 Layer Deep Neural Network code:**

## log reg, nn, deep neural net algorithm

## supervised learning classifier

#############################################################

import tensorflow as tf

import numpy as np

from numpy import genfromtxt

import sklearn

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score, f1\_score

import pandas as pd

import matplotlib.pyplot as plt

#############################################################

## remove if not using a mac

#import os

#os.environ['KMP\_DUPLICATE\_LIB\_OK']='True'

#############################################################

number\_epochs = 50

learning\_rate = 0.1

batch\_size = 1000

#############################################################

x\_train = genfromtxt('times\_train.csv', delimiter=',', usecols=(i for i in range(0,202)) , skip\_header=1)

y\_train = genfromtxt('times\_train.csv', delimiter=',', usecols=(0), skip\_header=1)

x\_test = genfromtxt('times\_test.csv', delimiter=',', usecols=(i for i in range(0,202)) ,skip\_header=1 )

y\_test = genfromtxt('times\_test.csv', delimiter=',', usecols=(0), skip\_header=1)

############################################################

## normalizing

sc = StandardScaler()

sc.fit(x\_train)

x\_train\_normalized = sc.transform(x\_train)

x\_test\_normalized = sc.transform(x\_test)

###########################################################

# Convert to one hot data

def convertOneHot\_data2(data):

y=np.array([int(i) for i in data])

#print y[:20]

rows = len(y)

columns = y.max() + 1

a = np.zeros(shape=(rows,columns))

#print a[:20,:]

print rows

print columns

#rr = raw\_input()

#y\_onehot=[0]\*len(y)

for i,j in enumerate(y):

#y\_onehot[i]=np.array([0]\*(y.max() + 1) )

#y\_onehot[i][j]=1

a[i][j]=1

return (a)

############################################################

# one-hot encoding

depth = 2

#y\_train\_onehot = tf.one\_hot(y\_train, depth)

#y\_test\_onehot = tf.one\_hot(y\_test, depth)

y\_train\_onehot = convertOneHot\_data2(y\_train)

y\_test\_onehot = convertOneHot\_data2(y\_test)

#############################################################

# features (A)

A = len(x\_train[0])

print A # number of features

#############################################################

# classes (B)

B = 2 #len(sess.run(y\_train\_onehot[0]))

print "number of classes ", B

############################################################

## print stats

precision\_scores\_list = []

accuracy\_scores\_list = []

def print\_stats\_metrics(y\_test, y\_pred):

print('Accuracy: %.2f' % accuracy\_score(y\_test, y\_pred) )

#Accuracy: 0.84

accuracy\_scores\_list.append(accuracy\_score(y\_test, y\_pred) )

confmat = confusion\_matrix(y\_true=y\_test, y\_pred=y\_pred)

print "confusion matrix"

print(confmat)

print pd.crosstab(y\_test, y\_pred, rownames=['True'], colnames=['Predicted'], margins=True)

precision\_scores\_list.append(precision\_score(y\_true=y\_test, y\_pred=y\_pred, average='weighted'))

print('Precision: %.3f' % precision\_score(y\_true=y\_test, y\_pred=y\_pred, average='weighted'))

print('Recall: %.3f' % recall\_score(y\_true=y\_test, y\_pred=y\_pred, average='weighted'))

print('F1-measure: %.3f' % f1\_score(y\_true=y\_test, y\_pred=y\_pred, average='weighted'))

##############################################################

def plot\_metric\_per\_epoch():

x\_epochs = []

y\_epochs = []

for i, val in enumerate(precision\_scores\_list):

x\_epochs.append(i)

y\_epochs.append(val)

plt.scatter(x\_epochs, y\_epochs,s=50,c='lightgreen', marker='s', label='score')

plt.xlabel('epoch')

plt.ylabel('score')

plt.title('Score per epoch')

plt.legend()

plt.grid()

plt.show()

############################################################

def layer(input, weight\_shape, bias\_shape):

bias\_init = tf.constant\_initializer(value=0)

weight\_stddev = (2.0/weight\_shape[0])\*\*0.5

w\_init = tf.random\_normal\_initializer(stddev=weight\_stddev)

W = tf.get\_variable("W", weight\_shape, initializer=w\_init)

b = tf.get\_variable("b", bias\_shape, initializer=bias\_init)

return tf.nn.relu( tf.matmul(input, W) + b )

############################################################

## a neural network

#def inference\_nn(x, A, B):

with tf.variable\_scope("hidden\_1"):

hidden\_1 = layer(x, [A, 10], [10] )

with tf.variable\_scope("output"):

output = layer(hidden\_1, [10,B], [B])

return output

############################################################

## deep neural network 2 hidden layers

def inference\_deep\_2layers(x, A, B):

with tf.variable\_scope("hidden\_1"):

hidden\_1 = layer(x, [A,30], [30])

with tf.variable\_scope("hidden\_2"):

hidden\_2 = layer(hidden\_1, [30,20], [20])

with tf.variable\_scope("output"):

output = layer(hidden\_2, [20, B], [B])

return output

############################################################

## logistic regression

#def inference(x, A, B):

W = tf.Variable( tf.zeros([A, B]) )

b = tf.Variable(tf.zeros( [B]) )

output = tf.nn.softmax( tf.matmul(x, W) + b )

return output

############################################################

def loss\_deep(output, y):

xentropy = tf.nn.softmax\_cross\_entropy\_with\_logits(logits=output, labels=y)

loss = tf.reduce\_mean(xentropy)

return loss

############################################################

#def loss(output, y):

output = tf.clip\_by\_value(output, 1e-10, 1.0)

dot\_product = y \* tf.log(output)

xentropy = -tf.reduce\_sum( dot\_product )

loss = tf.reduce\_mean( xentropy )

return loss

#############################################################

def training(cost):

optimizer = tf.train.GradientDescentOptimizer(learning\_rate)

train\_op = optimizer.minimize(cost)

return train\_op

#############################################################

def evaluate(output, y):

correct\_prediction = tf.equal( tf.argmax(output,1) , tf.argmax(y,1) )

accuracy = tf.reduce\_mean( tf.cast(correct\_prediction, "float") )

return accuracy

#############################################################

x = tf.placeholder("float", [None, A])

y = tf.placeholder("float", [None, B])

#############################################################

## call the core functions

#output = inference(x, A, B) ###log reg

#cost = loss(output, y) ###log reg

#output = inference\_nn(x, A, B) ## nn

#cost = loss\_deep(output, y) ## nn

output = inference\_deep\_2layers(x, A, B) ## deep nn

cost = loss\_deep(output, y) ## deep nn

train\_op = training(cost)

eval\_op = evaluate(output, y)

############################################################

y\_pred\_metrics = tf.argmax(output,1)

############################################################

init = tf.global\_variables\_initializer()

sess = tf.Session()

sess.run(init)

###########################################################

## batch parameters

num\_samples\_train = len(y\_train)

print num\_samples\_train

num\_batches = int(num\_samples\_train/batch\_size)

############################################################

# MAIN\_LOOP()

#y\_test\_temp = sess.run(y\_test\_onehot)

#y\_train\_temp = sess.run(y\_train\_onehot)

y\_test\_temp = y\_test\_onehot

y\_train\_temp = y\_train\_onehot

print "running..."

for i in range(number\_epochs):

for batch\_n in range(num\_batches):

sta= batch\_n\*batch\_size

end= sta + batch\_size

sess.run( train\_op , feed\_dict={x: x\_train\_normalized[sta:end,:] , y: y\_train\_temp[sta:end, :]})

print "accuracy ..."

accuracy\_value, y\_pred = sess.run([eval\_op, y\_pred\_metrics], feed\_dict={x: x\_test\_normalized, y: y\_test\_temp})

print "run {}, {}".format(i, accuracy\_value)

y\_true = np.argmax(y\_test\_temp, 1)

print y\_true

print y\_pred

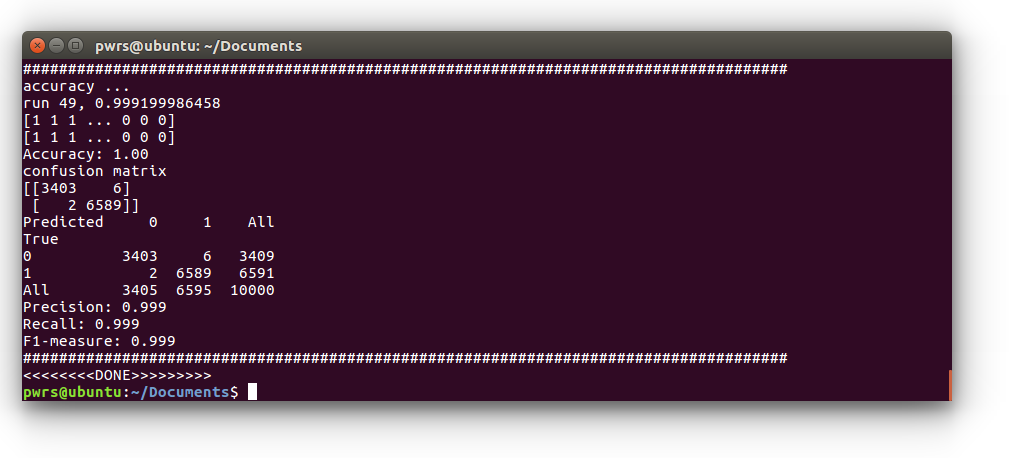
print\_stats\_metrics(y\_true, y\_pred)

print '#####################################################################################'

#############################################################

print "<<<<<<<<DONE>>>>>>>>>"

**Output:**



**3 Layer Deep Neural Network code:**

## log reg, nn, deep neural net algorithm

## supervised learning classifier

#############################################################

import tensorflow as tf

import numpy as np

from numpy import genfromtxt

import sklearn

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score, f1\_score

import pandas as pd

import matplotlib.pyplot as plt

#############################################################

## remove if not using a mac

#import os

#os.environ['KMP\_DUPLICATE\_LIB\_OK']='True'

#############################################################

number\_epochs = 50

learning\_rate = 0.1

batch\_size = 1000

#############################################################

x\_train = genfromtxt('times\_train.csv', delimiter=',', usecols=(i for i in range(0,202)) , skip\_header=1)

y\_train = genfromtxt('times\_train.csv', delimiter=',', usecols=(0), skip\_header=1)

x\_test = genfromtxt('times\_test.csv', delimiter=',', usecols=(i for i in range(0,202)) ,skip\_header=1 )

y\_test = genfromtxt('times\_test.csv', delimiter=',', usecols=(0), skip\_header=1)

############################################################

## normalizing

sc = StandardScaler()

sc.fit(x\_train)

x\_train\_normalized = sc.transform(x\_train)

x\_test\_normalized = sc.transform(x\_test)

###########################################################

# Convert to one hot data

def convertOneHot\_data2(data):

y=np.array([int(i) for i in data])

#print y[:20]

rows = len(y)

columns = y.max() + 1

a = np.zeros(shape=(rows,columns))

#print a[:20,:]

print rows

print columns

#rr = raw\_input()

#y\_onehot=[0]\*len(y)

for i,j in enumerate(y):

#y\_onehot[i]=np.array([0]\*(y.max() + 1) )

#y\_onehot[i][j]=1

a[i][j]=1

return (a)

############################################################

# one-hot encoding

depth = 2

#y\_train\_onehot = tf.one\_hot(y\_train, depth)

#y\_test\_onehot = tf.one\_hot(y\_test, depth)

y\_train\_onehot = convertOneHot\_data2(y\_train)

y\_test\_onehot = convertOneHot\_data2(y\_test)

#############################################################

# features (A)

A = len(x\_train[0])

print A # number of features

#############################################################

# classes (B)

B = 2 #len(sess.run(y\_train\_onehot[0]))

print "number of classes ", B

############################################################

## print stats

precision\_scores\_list = []

accuracy\_scores\_list = []

def print\_stats\_metrics(y\_test, y\_pred):

print('Accuracy: %.2f' % accuracy\_score(y\_test, y\_pred) )

#Accuracy: 0.84

accuracy\_scores\_list.append(accuracy\_score(y\_test, y\_pred) )

confmat = confusion\_matrix(y\_true=y\_test, y\_pred=y\_pred)

print "confusion matrix"

print(confmat)

print pd.crosstab(y\_test, y\_pred, rownames=['True'], colnames=['Predicted'], margins=True)

precision\_scores\_list.append(precision\_score(y\_true=y\_test, y\_pred=y\_pred, average='weighted'))

print('Precision: %.3f' % precision\_score(y\_true=y\_test, y\_pred=y\_pred, average='weighted'))

print('Recall: %.3f' % recall\_score(y\_true=y\_test, y\_pred=y\_pred, average='weighted'))

print('F1-measure: %.3f' % f1\_score(y\_true=y\_test, y\_pred=y\_pred, average='weighted'))

##############################################################

def plot\_metric\_per\_epoch():

x\_epochs = []

y\_epochs = []

for i, val in enumerate(precision\_scores\_list):

x\_epochs.append(i)

y\_epochs.append(val)

plt.scatter(x\_epochs, y\_epochs,s=50,c='lightgreen', marker='s', label='score')

plt.xlabel('epoch')

plt.ylabel('score')

plt.title('Score per epoch')

plt.legend()

plt.grid()

plt.show()

############################################################

def layer(input, weight\_shape, bias\_shape):

bias\_init = tf.constant\_initializer(value=0)

weight\_stddev = (2.0/weight\_shape[0])\*\*0.5

w\_init = tf.random\_normal\_initializer(stddev=weight\_stddev)

W = tf.get\_variable("W", weight\_shape, initializer=w\_init)

b = tf.get\_variable("b", bias\_shape, initializer=bias\_init)

return tf.nn.relu( tf.matmul(input, W) + b )

############################################################

## a neural network

#def inference\_nn(x, A, B):

with tf.variable\_scope("hidden\_1"):

hidden\_1 = layer(x, [A, 10], [10] )

with tf.variable\_scope("output"):

output = layer(hidden\_1, [10,B], [B])

return output

############################################################

## deep neural network 3 hidden layers

def inference\_deep\_3layers(x, A, B):

with tf.variable\_scope("hidden\_1"):

hidden\_1 = layer(x, [A,30], [30])

with tf.variable\_scope("hidden\_2"):

hidden\_2 = layer(hidden\_1, [30,20], [20])

with tf.variable\_scope("hidden\_3"):

hidden\_3 = layer(hidden\_2, [20,10], [10])

with tf.variable\_scope("output"):

output = layer(hidden\_3, [10, B], [B])

return output

############################################################

## logistic regression

#def inference(x, A, B):

W = tf.Variable( tf.zeros([A, B]) )

b = tf.Variable(tf.zeros( [B]) )

output = tf.nn.softmax( tf.matmul(x, W) + b )

return output

############################################################

def loss\_deep(output, y):

xentropy = tf.nn.softmax\_cross\_entropy\_with\_logits(logits=output, labels=y)

loss = tf.reduce\_mean(xentropy)

return loss

############################################################

#def loss(output, y):

output = tf.clip\_by\_value(output, 1e-10, 1.0)

dot\_product = y \* tf.log(output)

xentropy = -tf.reduce\_sum( dot\_product )

loss = tf.reduce\_mean( xentropy )

return loss

#############################################################

def training(cost):

optimizer = tf.train.GradientDescentOptimizer(learning\_rate)

train\_op = optimizer.minimize(cost)

return train\_op

#############################################################

def evaluate(output, y):

correct\_prediction = tf.equal( tf.argmax(output,1) , tf.argmax(y,1) )

accuracy = tf.reduce\_mean( tf.cast(correct\_prediction, "float") )

return accuracy

#############################################################

x = tf.placeholder("float", [None, A])

y = tf.placeholder("float", [None, B])

#############################################################

## call the core functions

#output = inference(x, A, B) ###log reg

#cost = loss(output, y) ###log reg

#output = inference\_nn(x, A, B) ## nn

#cost = loss\_deep(output, y) ## nn

output = inference\_deep\_3layers(x, A, B) ## deep nn

cost = loss\_deep(output, y) ## deep nn

train\_op = training(cost)

eval\_op = evaluate(output, y)

############################################################

y\_pred\_metrics = tf.argmax(output,1)

############################################################

init = tf.global\_variables\_initializer()

sess = tf.Session()

sess.run(init)

###########################################################

## batch parameters

num\_samples\_train = len(y\_train)

print num\_samples\_train

num\_batches = int(num\_samples\_train/batch\_size)

############################################################

# MAIN\_LOOP()

#y\_test\_temp = sess.run(y\_test\_onehot)

#y\_train\_temp = sess.run(y\_train\_onehot)

y\_test\_temp = y\_test\_onehot

y\_train\_temp = y\_train\_onehot

print "running..."

for i in range(number\_epochs):

for batch\_n in range(num\_batches):

sta= batch\_n\*batch\_size

end= sta + batch\_size

sess.run( train\_op , feed\_dict={x: x\_train\_normalized[sta:end,:] , y: y\_train\_temp[sta:end, :]})

print "accuracy ..."

accuracy\_value, y\_pred = sess.run([eval\_op, y\_pred\_metrics], feed\_dict={x: x\_test\_normalized, y: y\_test\_temp})

print "run {}, {}".format(i, accuracy\_value)

y\_true = np.argmax(y\_test\_temp, 1)

print y\_true

print y\_pred

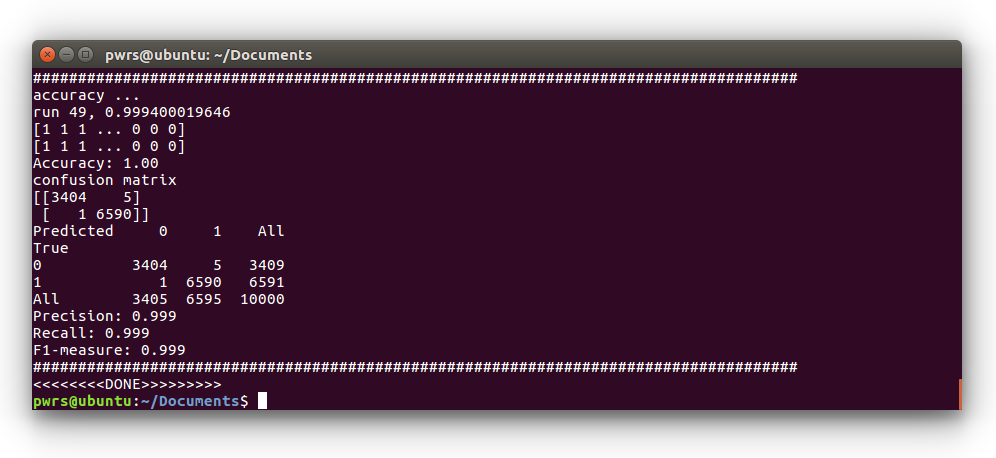
print\_stats\_metrics(y\_true, y\_pred)

print '#####################################################################################'

#############################################################

print "<<<<<<<<DONE>>>>>>>>>"

**Output:**



**4 Layer Deep Neural Network code:**

## log reg, nn, deep neural net algorithm

## supervised learning classifier

#############################################################

import tensorflow as tf

import numpy as np

from numpy import genfromtxt

import sklearn

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score, f1\_score

import pandas as pd

import matplotlib.pyplot as plt

#############################################################

## remove if not using a mac

#import os

#os.environ['KMP\_DUPLICATE\_LIB\_OK']='True'

#############################################################

number\_epochs = 50

learning\_rate = 0.1

batch\_size = 1000

#############################################################

x\_train = genfromtxt('times\_train.csv', delimiter=',', usecols=(i for i in range(0,202)) , skip\_header=1)

y\_train = genfromtxt('times\_train.csv', delimiter=',', usecols=(0), skip\_header=1)

x\_test = genfromtxt('times\_test.csv', delimiter=',', usecols=(i for i in range(0,202)) ,skip\_header=1 )

y\_test = genfromtxt('times\_test.csv', delimiter=',', usecols=(0), skip\_header=1)

############################################################

## normalizing

sc = StandardScaler()

sc.fit(x\_train)

x\_train\_normalized = sc.transform(x\_train)

x\_test\_normalized = sc.transform(x\_test)

###########################################################

# Convert to one hot data

def convertOneHot\_data2(data):

y=np.array([int(i) for i in data])

#print y[:20]

rows = len(y)

columns = y.max() + 1

a = np.zeros(shape=(rows,columns))

#print a[:20,:]

print rows

print columns

#rr = raw\_input()

#y\_onehot=[0]\*len(y)

for i,j in enumerate(y):

#y\_onehot[i]=np.array([0]\*(y.max() + 1) )

#y\_onehot[i][j]=1

a[i][j]=1

return (a)

############################################################

# one-hot encoding

depth = 2

#y\_train\_onehot = tf.one\_hot(y\_train, depth)

#y\_test\_onehot = tf.one\_hot(y\_test, depth)

y\_train\_onehot = convertOneHot\_data2(y\_train)

y\_test\_onehot = convertOneHot\_data2(y\_test)

#############################################################

# features (A)

A = len(x\_train[0])

print A # number of features

#############################################################

# classes (B)

B = 2 #len(sess.run(y\_train\_onehot[0]))

print "number of classes ", B

############################################################

## print stats

precision\_scores\_list = []

accuracy\_scores\_list = []

def print\_stats\_metrics(y\_test, y\_pred):

print('Accuracy: %.2f' % accuracy\_score(y\_test, y\_pred) )

#Accuracy: 0.84

accuracy\_scores\_list.append(accuracy\_score(y\_test, y\_pred) )

confmat = confusion\_matrix(y\_true=y\_test, y\_pred=y\_pred)

print "confusion matrix"

print(confmat)

print pd.crosstab(y\_test, y\_pred, rownames=['True'], colnames=['Predicted'], margins=True)

precision\_scores\_list.append(precision\_score(y\_true=y\_test, y\_pred=y\_pred, average='weighted'))

print('Precision: %.3f' % precision\_score(y\_true=y\_test, y\_pred=y\_pred, average='weighted'))

print('Recall: %.3f' % recall\_score(y\_true=y\_test, y\_pred=y\_pred, average='weighted'))

print('F1-measure: %.3f' % f1\_score(y\_true=y\_test, y\_pred=y\_pred, average='weighted'))

##############################################################

def plot\_metric\_per\_epoch():

x\_epochs = []

y\_epochs = []

for i, val in enumerate(precision\_scores\_list):

x\_epochs.append(i)

y\_epochs.append(val)

plt.scatter(x\_epochs, y\_epochs,s=50,c='lightgreen', marker='s', label='score')

plt.xlabel('epoch')

plt.ylabel('score')

plt.title('Score per epoch')

plt.legend()

plt.grid()

plt.show()

############################################################

def layer(input, weight\_shape, bias\_shape):

bias\_init = tf.constant\_initializer(value=0)

weight\_stddev = (2.0/weight\_shape[0])\*\*0.5

w\_init = tf.random\_normal\_initializer(stddev=weight\_stddev)

W = tf.get\_variable("W", weight\_shape, initializer=w\_init)

b = tf.get\_variable("b", bias\_shape, initializer=bias\_init)

return tf.nn.relu( tf.matmul(input, W) + b )

############################################################

## a neural network

#def inference\_nn(x, A, B):

with tf.variable\_scope("hidden\_1"):

hidden\_1 = layer(x, [A, 10], [10] )

with tf.variable\_scope("output"):

output = layer(hidden\_1, [10,B], [B])

return output

############################################################

## deep neural network 4 hidden layers

def inference\_deep\_4layers(x, A, B):

with tf.variable\_scope("hidden\_1"):

hidden\_1 = layer(x, [A,30], [30])

with tf.variable\_scope("hidden\_2"):

hidden\_2 = layer(hidden\_1, [30,20], [20])

with tf.variable\_scope("hidden\_3"):

hidden\_3 = layer(hidden\_2, [20,10], [10])

with tf.variable\_scope("hidden\_4"):

hidden\_4 = layer(hidden\_3, [10,5], [5])

with tf.variable\_scope("output"):

output = layer(hidden\_4, [5, B], [B])

return output

############################################################

## logistic regression

#def inference(x, A, B):

W = tf.Variable( tf.zeros([A, B]) )

b = tf.Variable(tf.zeros( [B]) )

output = tf.nn.softmax( tf.matmul(x, W) + b )

return output

############################################################

def loss\_deep(output, y):

xentropy = tf.nn.softmax\_cross\_entropy\_with\_logits(logits=output, labels=y)

loss = tf.reduce\_mean(xentropy)

return loss

############################################################

#def loss(output, y):

output = tf.clip\_by\_value(output, 1e-10, 1.0)

dot\_product = y \* tf.log(output)

xentropy = -tf.reduce\_sum( dot\_product )

loss = tf.reduce\_mean( xentropy )

return loss

#############################################################

def training(cost):

optimizer = tf.train.GradientDescentOptimizer(learning\_rate)

train\_op = optimizer.minimize(cost)

return train\_op

#############################################################

def evaluate(output, y):

correct\_prediction = tf.equal( tf.argmax(output,1) , tf.argmax(y,1) )

accuracy = tf.reduce\_mean( tf.cast(correct\_prediction, "float") )

return accuracy

#############################################################

x = tf.placeholder("float", [None, A])

y = tf.placeholder("float", [None, B])

#############################################################

## call the core functions

#output = inference(x, A, B) ###log reg

#cost = loss(output, y) ###log reg

#output = inference\_nn(x, A, B) ## nn

#cost = loss\_deep(output, y) ## nn

output = inference\_deep\_4layers(x, A, B) ## deep nn

cost = loss\_deep(output, y) ## deep nn

train\_op = training(cost)

eval\_op = evaluate(output, y)

############################################################

y\_pred\_metrics = tf.argmax(output,1)

############################################################

init = tf.global\_variables\_initializer()

sess = tf.Session()

sess.run(init)

###########################################################

## batch parameters

num\_samples\_train = len(y\_train)

print num\_samples\_train

num\_batches = int(num\_samples\_train/batch\_size)

############################################################

# MAIN\_LOOP()

#y\_test\_temp = sess.run(y\_test\_onehot)

#y\_train\_temp = sess.run(y\_train\_onehot)

y\_test\_temp = y\_test\_onehot

y\_train\_temp = y\_train\_onehot

print "running..."

for i in range(number\_epochs):

for batch\_n in range(num\_batches):

sta= batch\_n\*batch\_size

end= sta + batch\_size

sess.run( train\_op , feed\_dict={x: x\_train\_normalized[sta:end,:] , y: y\_train\_temp[sta:end, :]})

print "accuracy ..."

accuracy\_value, y\_pred = sess.run([eval\_op, y\_pred\_metrics], feed\_dict={x: x\_test\_normalized, y: y\_test\_temp})

print "run {}, {}".format(i, accuracy\_value)

y\_true = np.argmax(y\_test\_temp, 1)

print y\_true

print y\_pred

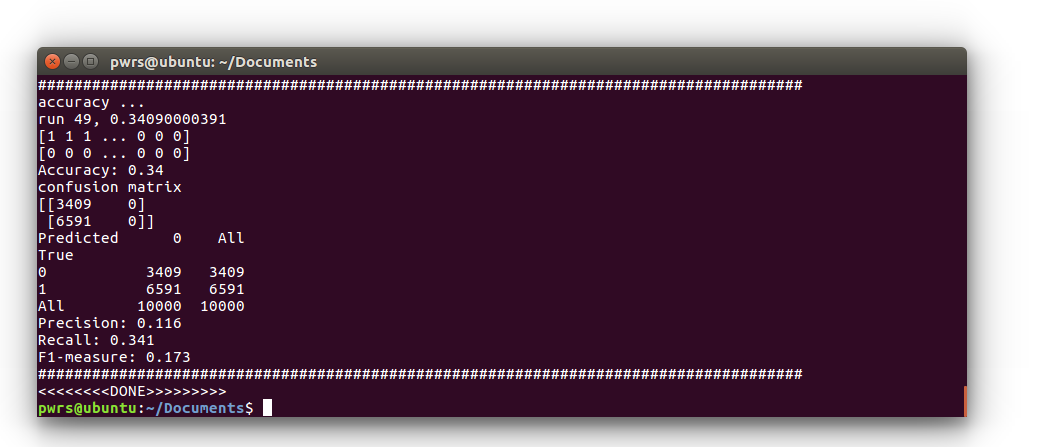
print\_stats\_metrics(y\_true, y\_pred)

print '#####################################################################################'

#############################################################

print "<<<<<<<<DONE>>>>>>>>>"

**Output:**



RESULTS:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **PRECISION** | **RECALL** | **ACCURACY** | **F- MEASURE** |
| **LOG\_ REGRESSION** | 0.999 | 0.999 | 1.00 | 0.999 |
| **NEURAL NETWORK** | 1.00 | 1.00 | 1.00 | 0.999 |
| **2 LAYER DEEP NEURAL NETWORK** | 0.999 | 0.999 | 0.999 | 0.999 |
| **3 LAYER DEEP NEURAL NETWORK** | 0.999 | 0.999 | 0.999 | 0.999 |
| **4 LAYER DEEP NEURAL NETWORK** | 0.116 | 0.341 | 0.34 | 0.173 |