**Backpropagation:**

**Parameters:**

Learning Rate: 0.01

Number of Hidden Layers: 4

Number of neurons: 4

Bias: 0.1

**Parameter setting goals:**

Transfusion dataset contains 748 instances with four features and class label (i/e 0 and 1). As there are four features, number of neurons in each of 4 hidden layers is set to 4 in order to achieve maximum learning by each neurons to get maximum performance as more the number of weights present more the model will learn efficiently. Learning rate i set to 0.01 so that model should learn slowly and reach the global minimum to reduce the error. Bias of 0.1 is used so that the model fits the given data properly.

**Source Code:**

#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Thu Jan 30 13:50:57 2020

@author: rahul

"""

#-------------------------------------------------Importing libraries-------------------------------------------------------

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from itertools import chain

#--------------------------------------------------------------------------------------------------------------------------

#--------------------------------------------------Initializing Parameters--------------------------------------------------

bs = 0.1

learningRate = 0.01

noOfClasses = 2

noOfHiddenLayers = 4

#--------------------------------------------------------------------------------------------------------------------------

#--------------------------------------------------Defining Functions------------------------------------------------------

#--------------------------------------------------Sigmoid Activation------------------------------------------------------

def sig(x):

calculate = 1/(1+np.exp(-x))

return calculate

#--------------------------------------------------------------------------------------------------------------------------

#--------------------------------------------------Relu Activation---------------------------------------------------------

def relu(x):

return max(0,x)

#---------------------------------------------------------------------------------------------------------------------------

#--------------------------------------------------Softmax Activation-------------------------------------------------------

def softmax(x):

exp = np.exp(x)

calculate = exp / exp.sum()

return calculate

#--------------------------------------------------------------------------------------------------------------------------

#--------------------------------------------------Accuracy Calculation-----------------------------------------------------

def predAccuracy(originalLabel,predicted):

matched = 0

for i in range(len(originalLabel)):

if originalLabel[i] == predicted[i]:

matched += 1

accuracyVal = matched / float(len(originalLabel))

return accuracyVal

#--------------------------------------------------------------------------------------------------------------------------

#--------------------------------------------------Local Gradient Calculation----------------------------------------------

def localGradient(s):

calculate = s\*(1-s)

return calculate

#--------------------------------------------------------------------------------------------------------------------------

#---------------------------------------------------Hidden Layer Calculation-----------------------------------------------

def neuronsOperation(row,neurons,we):

finalOutputs = []

if len(we) == 0:

weights = []

else:

weights = np.array(we).reshape(neurons,len(row))

for noOfNode in range(neurons):

output = 0

if len(we) == 0:

weight = np.random.rand(len(row))

weights.append(weight)

else:

weight = weights[noOfNode]

for i in range(len(row)):

output += row[i]\*weight[i]

finalOutputs.append( sig(output) + bs)

return finalOutputs,list( chain.from\_iterable(weights) )

#--------------------------------------------------------------------------------------------------------------------------

#--------------------------------------------------Testing Function--------------------------------------------------------

def test(inputTest, outputTest, allFinalWeights):

predictedOutput = []

m = 0

for row in inputTest:

inputs = row

calculatedOutput = []

neurons = 4

calculatedOutput.append(row)

for k in range(noOfHiddenLayers+1):

if k == noOfHiddenLayers:

neurons = noOfClasses

inputs,w = neuronsOperation(inputs,neurons,allFinalWeights[m][k])

calculatedOutput.append(inputs)

predictedOutput.append( np.argmax( softmax( calculatedOutput[-1] ) ) )

m = m+1

print("Testing Accuracy: ",predAccuracy(outputTest,predictedOutput) \* 100 , "%" )

#--------------------------------------------------------------------------------------------------------------------------

#--------------------------------------------------Training Function--------------------------------------------------------

def train(inputTrain,outputTrain):

allFinalWeights =[]

for epoch in range(23):

allHiddenLayersOutput = []

predictedOutput = []

m = 0

for row in inputTrain:

inputs = row

calculatedOutput = []

finalWeights =[]

neurons = 4

calculatedOutput.append(row)

for k in range(noOfHiddenLayers+1):

if k == noOfHiddenLayers:

neurons = noOfClasses

if epoch == 0:

inputs,w = neuronsOperation(inputs,neurons,[])

else:

inputs,w = neuronsOperation(inputs,neurons,allFinalWeights[m][k])

finalWeights.append(w)

calculatedOutput.append(inputs)

predictedOutput.append( np.argmax( softmax( calculatedOutput[-1] ) ) )

allHiddenLayersOutput.append(calculatedOutput)

if epoch == 0:

allFinalWeights.append(finalWeights)

m = m+1

print("Training Epoch : ", epoch+1,"............." )

#------------------------------------------Backpropagation----------------------------------------------------------------

for q in range(len(allHiddenLayersOutput)-1,-1,-1):

for r in range(len(allFinalWeights[q])-1,-1,-1):

noOfPreviousLayerNeurons = len(allHiddenLayersOutput[q][r])

l = 0

e = 0

for t in range(len(allFinalWeights[q][r])-1,-1,-1):

if l > noOfPreviousLayerNeurons-1:

l = 0

e = e + 1

#-----------------------------------------Updating Weights--------------------------------------------

allFinalWeights[q][r][t] = allFinalWeights[q][r][t] - learningRate \* localGradient(allHiddenLayersOutput[q][r+1][e]) \* allHiddenLayersOutput[q][r][l]

#-----------------------------------------------------------------------------------------------------

l = l + 1

#--------------------------------------------------------------------------------------------------------------------------

return allFinalWeights

#--------------------------------------------------------------------------------------------------------------------------

#--------------------------------------------------------------------------------------------------------------------------

#------------------------------------------------Main Function-------------------------------------------------------------

if \_\_name\_\_ == "\_\_main\_\_":

data = pd.read\_csv("transfusion.csv")

wholeDataset = pd.DataFrame(data)

wholeDataset = wholeDataset.astype(float)

inputData = wholeDataset.drop(columns=[wholeDataset.columns[-1]]).to\_numpy()

outputLabel = wholeDataset[wholeDataset.columns[-1]].to\_numpy()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(inputData, outputLabel, test\_size=0.33,random\_state = 42)

y\_train = y\_train.reshape(len(y\_train),1)

y\_test = y\_test.reshape(len(y\_test),1)

#------------------------------------------------Training Start------------------------------------------------------------

wt = train(X\_train,y\_train)

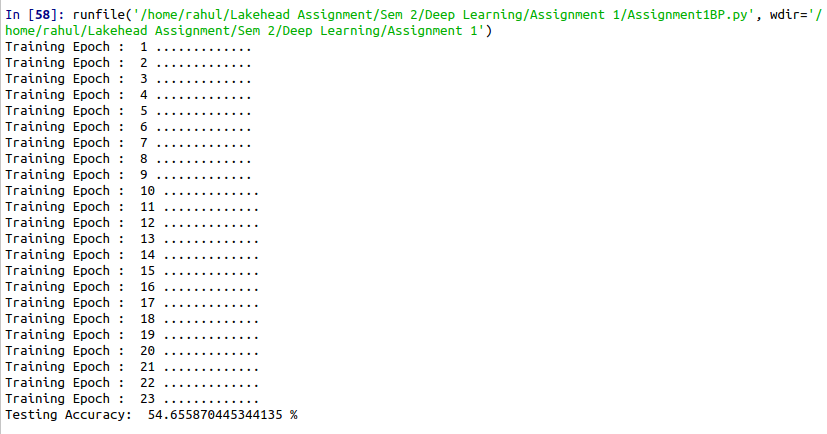
#------------------------------------------------Training End---------------------------------------------------------------

#------------------------------------------------Testing Start--------------------------------------------------------------

test(X\_test,y\_test,wt)

#--------------------------------------------Testing End------------------------------------------------------------------

**Result:**



**Extreme Learning Machine (ELM):**

**Parameters:**

Number of neurons: 6

Bias: 0.1

**Parameter setting goals:**

Transfusion dataset contains 748 instances with four features and class label (i/e 0 and 1). As there are four features, number of neurons in hidden layer is set to 6 inorder to achieve maximum learning by each neurons to get maximum performance. We have calculated beta which have same dimension as number of neurons, in order to predict output in testing phase to get maximum performance as compared to backpropagation. Bias of 0.1 is used so that the model fits the given data properly.

**Source Code:**

#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Wed Jan 29 13:21:41 2020

@author: rahul

"""

#--------------------------------------------------Importing Libraries----------------------------------------------------

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from itertools import chain

#-------------------------------------------------------------------------------------------------------------------------

#---------------------------------------------------Intializing Parameters------------------------------------------------

bs = 0.1

noOfNeurons = 6

#-------------------------------------------------------------------------------------------------------------------------

#---------------------------------------------------Defining Functions----------------------------------------------------

#---------------------------------------------------Calculation Output---------------------------------------------------

def predValue(inputs,w):

output = sig( np.dot(inputs, w) + bs )

return output

#-------------------------------------------------------------------------------------------------------------------------

#---------------------------------------------------Sigmoid Activation----------------------------------------------------

def sig(x):

calculate = 1/(1+np.exp(-x))

return calculate

#-------------------------------------------------------------------------------------------------------------------------

#--------------------------------------------------Softmax Activation-----------------------------------------------------

def softmax(x):

exp = np.exp(x)

calculate = exp / exp.sum()

return calculate

#-------------------------------------------------------------------------------------------------------------------------

#---------------------------------------------------Accuracy Prediction---------------------------------------------------

def predAccuracy(originalLabel,predicted):

matched = 0

for i in range(len(originalLabel)):

if originalLabel[i] == predicted[i]:

matched += 1

accuracyVal = matched / float(len(originalLabel))

return accuracyVal

#--------------------------------------------------------------------------------------------------------------------------

#----------------------------------------------------Training Function-----------------------------------------------------

def train(train\_input\_data,outputTrainLabel):

hMatrix = []

finalWeights = []

for row in train\_input\_data:

h =[]

weights = []

for i in range(noOfNeurons):

output = 0

weight = np.random.rand(len(row))

weights.append(weight)

for j in range(len(row)):

output += row[j]\*weight[j]

h.append(sig(output) + bs)

hMatrix.append(h)

finalWeights.append(weights)

beta = np.dot(np.linalg.pinv(hMatrix), outputTrainLabel)

return beta,finalWeights

#--------------------------------------------------------------------------------------------------------------------------

#-----------------------------------------------------Testing Function-----------------------------------------------------

def test(data, outputD,b,weights):

hMatrix = []

m = 0

for row in data:

weight = weights[m]

h =[]

for i in range(noOfNeurons):

output = 0

we = weight[i]

for j in range(len(row)):

output += row[j]\*we[j]

h.append(sig(output) + bs)

hMatrix.append(h)

m += 1

o = np.dot(hMatrix , b)

o[ o >= 0.5 ] = 1

o[ o < 0.5 ] = 0

acc = predAccuracy(outputD, o)

print("Testing Accuracy",acc\*100,"%")

#--------------------------------------------------------------------------------------------------------------------------

#--------------------------------------------------------------------------------------------------------------------------

#----------------------------------------------------Main Function-------------------------------------------------------

if \_\_name\_\_ == "\_\_main\_\_":

data = pd.read\_csv("transfusion.csv")

wholeDataset = pd.DataFrame(data)

wholeDataset = (wholeDataset).astype(float)

inputData = wholeDataset.drop(columns=[wholeDataset.columns[-1]]).to\_numpy()

outputLabel = wholeDataset[wholeDataset.columns[-1]].to\_numpy()

#---------------------------------------------------Train Test Splitting------------------------------------------------

X\_train, X\_test, y\_train, y\_test = train\_test\_split(inputData, outputLabel, test\_size=0.33,random\_state = 42)

#-----------------------------------------------------------------------------------------------------------------------

y\_train = y\_train.reshape(len(y\_train),1)

y\_test = y\_test.reshape(len(y\_test),1)

#---------------------------------------------------Training Start-----------------------------------------------------

beta,weights = train(X\_train,y\_train)

#---------------------------------------------------Training End-------------------------------------------------------

#---------------------------------------------------Testing Start------------------------------------------------------

test(X\_test,y\_test,beta,weights)

#---------------------------------------------------Testing End---------------------------------------------------------

#--------------------------------------------------------------------------------------------------------------------------

**Result:**

