ASSIGNMENT - 8

ANIKEIT SETHI (190001003)

import random

import csv

import math

def Read(filename):

    file = open(filename)

    csvreader = csv.reader(file)

    header = next(csvreader)

    rows = []

    for row in csvreader:

        rows.append(row)

    file.close()

    return rows

def activation\_function(x):

    if x <= -100:

        return 0

    if x >= 100:

        return 1

    return 1/(1 + math.exp(-x))

def derivative\_activation\_function(x):

    return activation\_function(x) \* (1 - activation\_function(x))

#class SingleHiddenLayerNeuralNetwork

class SingleHiddenLayerNeuralNetwork:

    #here N denotes size of input vector, M denotes neurons in hidden layer

    def \_\_init\_\_(self,N,M):

        self.N = N

        self.M = M

        self.hidden\_weights = []

        self.hidden\_baises = []

        for \_ in range(M):

            self.hidden\_baises.append(random.uniform(-0.1,0.1))

            w\_ = []

            for i in range(N):

                w\_.append(random.uniform(-0.1,0.1))

            self.hidden\_weights.append(w\_)

        self.output\_bais = random.uniform(-0.1,0.1)

        w\_ = []

        for i in range(M):

            w\_.append(random.uniform(-0.1,0.1))

        self.output\_weights = w\_

    def train(self,trainData,trainLabels,learning\_rate,iterations):

        for \_ in range(iterations):

            for ii in range(len(trainData)):

                Data = trainData[ii]

                Label = trainLabels[ii]

                sum\_hidden\_layer = []

                activated\_sum\_hidden\_layer = []

                for i in range(self.M):

                    val = self.hidden\_baises[i]

                    for j in range(self.N):

                        val += self.hidden\_weights[i][j] \* Data[j]

                    sum\_hidden\_layer.append(val)

                    activated\_sum\_hidden\_layer.append(activation\_function(val))

                sum\_output\_layer = self.output\_bais

                activated\_sum\_output\_layer = 0

                for i in range(self.M):

                    sum\_output\_layer += activated\_sum\_hidden\_layer[i] \* self.output\_weights[i]

                activated\_sum\_output\_layer = activation\_function(sum\_output\_layer)

                if activated\_sum\_output\_layer > 0.5:

                    activated\_sum\_output\_layer = 1

                else:

                    activated\_sum\_output\_layer = 0

                #update output layer weights

                for i in range(self.M):

                    self.output\_weights[i] += -1 \* learning\_rate \* (activated\_sum\_output\_layer - Label) \* derivative\_activation\_function(sum\_output\_layer) \* activated\_sum\_hidden\_layer[i]

                #update output layer baises

                self.output\_bais += -1 \* learning\_rate \* (activated\_sum\_output\_layer - Label) \* derivative\_activation\_function(sum\_output\_layer)

                #update hidden layer weights

                for i in range(self.M):

                    for j in range(self.N):

                        self.hidden\_weights[i][j] += -1 \* learning\_rate \* (activated\_sum\_output\_layer - Label) \* derivative\_activation\_function(sum\_output\_layer) \* self.output\_weights[i] \* derivative\_activation\_function(activated\_sum\_hidden\_layer[i]) \* Data[j]

                #update hidden layer baises

                for i in range(self.M):

                    self.hidden\_baises[i] += -1 \* learning\_rate \* (activated\_sum\_output\_layer - Label) \* derivative\_activation\_function(sum\_output\_layer) \* self.output\_weights[i] \* derivative\_activation\_function(activated\_sum\_hidden\_layer[i])

    def test(self,testData,testLabels):

        correct = 0

        wrong = 0

        for ii in range(len(testData)):

            Data = testData[ii]

            Label = testLabels[ii]

            sum\_hidden\_layer = []

            activated\_sum\_hidden\_layer = []

            for i in range(self.M):

                val = self.hidden\_baises[i]

                for j in range(self.N):

                    val += self.hidden\_weights[i][j] \* Data[j]

                sum\_hidden\_layer.append(val)

                activated\_sum\_hidden\_layer.append(activation\_function(val))

            sum\_output\_layer = self.output\_bais

            activated\_sum\_output\_layer = 0

            for i in range(self.M):

                sum\_output\_layer += activated\_sum\_hidden\_layer[i] \* self.output\_weights[i]

            activated\_sum\_output\_layer = activation\_function(sum\_output\_layer)

            if activated\_sum\_output\_layer > 0.5:

                if Label == 1:

                    correct += 1

                else:

                    wrong += 1

            else:

                if Label == 0:

                    correct += 1

                else:

                    wrong += 1

        print("accuracy = ")

        print(float(correct)/float(correct+wrong))

testData\_ = Read('testData.csv')

testLabels\_ = Read('testLabels.csv')

trainData\_ = Read('trainData.csv')

trainLabels\_ = Read('trainLabels.csv')

testData = []

for row in testData\_:

    a = []

    for x in row:

        a.append(float(x))

    testData.append(a)

trainData = []

for row in trainData\_:

    a = []

    for x in row:

        a.append(float(x))

    trainData.append(a)

testLabels = []

for row in testLabels\_:

    testLabels.append(int(row[0]) - int(5))

trainLabels = []

for row in trainLabels\_:

    trainLabels.append(float(row[0]) - 5.0)

NN = SingleHiddenLayerNeuralNetwork(len(testData[0]),10)

NN.train(trainData,trainLabels,0.001,1000)

NN.test(testData,testLabels)

***Output: -***

accuracy = 0.9116022099447514

