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

```
        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)
```

[3] ✓ 17m 9.1s

```
accuracy =
0.9116022099447514
```