Artificial Neural Network
Home Work - 2

Trilokinath Modi

October 9, 2020

1 Problem 4

```
import numpy as np
import csv
import matplotlib.pyplot as plt

# Training data set
filePath = 'D:\\Masters␣Program␣Chalmers\\Projects␣and␣Labs\\ANN\\training_set.csv'
fileHandler = open(filePath, "r")
readFile = csv.reader(fileHandler)
storedPatterns = list(readFile)
fileHandler.close()
for iRow in range(len(storedPatterns)):
    for iList in range(len(storedPatterns[iRow])):
        storedPatterns[iRow][iList] = float(storedPatterns[iRow][iList])

targetNeurons = storedPatterns.copy()
storedPatterns = np.delete(storedPatterns, 2, 1)
targetNeurons = np.delete(targetNeurons, [0, 1], 1)
targetNeurons = np.transpose(targetNeurons)
targetNeurons = targetNeurons[0].copy()

# Test data set
filePath = 'D:\\Masters␣Program␣Chalmers\\Projects␣and␣Labs\\ANN\\training_set.csv'
fileHandler = open(filePath, "r")
readFile = csv.reader(fileHandler)
fedPatterns = list(readFile)
fileHandler.close()
for iRow in range(len(fedPatterns)):
    for iList in range(len(fedPatterns[iRow])):
        fedPatterns[iRow][iList] = float(fedPatterns[iRow][iList])

testNeurons = fedPatterns.copy()
fedPatterns = np.delete(fedPatterns, 2, 1)
testNeurons = np.delete(testNeurons, [0, 1], 1)
testNeurons = np.transpose(testNeurons)
testNeurons = testNeurons[0].copy()

numberOfPatterns = storedPatterns.shape[0]
patternDimension = storedPatterns.shape[1]
numberOfInputNeurons = patternDimension
numberOfOutputNeurons = 1

class1Indices = np.where(targetNeurons == 1)[0]
class0Indices = np.where(targetNeurons == -1)[0]
np.random.shuffle(class1Indices)
np.random.shuffle(class0Indices)
numberOfClass1 = len(class1Indices)
```

```python
numberOfClass0 = len(class0Indices)

nFolds = 8
foldSize = int(np.floor(numberOfPatterns / nFolds))
foldIndices = np.zeros([nFolds, foldSize])
class1FoldSize = int(np.floor(numberOfClass1 / nFolds))
class0FoldSize = int(np.ceil(numberOfClass0 / nFolds))

# Stratified sampling
for iFold in range(nFolds):
    try:
        foldIndices[iFold][0: class1FoldSize] = class1Indices[iFold * class1FoldSize:(iFold + 1) *
            ↪ class1FoldSize]
        foldIndices[iFold][class1FoldSize: len(foldIndices[iFold])] = \
            class0Indices[iFold * class0FoldSize:(iFold + 1) * class0FoldSize]
    except:
        foldIndices[iFold][0: (numberOfClass1 - iFold * class1FoldSize)] = \
            class1Indices[iFold * class1FoldSize: numberOfClass1]
        foldIndices[iFold][(numberOfClass1 - iFold * class1FoldSize): len(foldIndices[iFold])] = \
            class0Indices[iFold * class0FoldSize:numberOfClass0]
    np.random.shuffle(foldIndices[iFold])

numberOfHiddenLayers = 2
flagM1M2 = 0
learningRate = 0.02
classificationError = 0
iterations = 125
epochIterations = 300

# Outer loop is M2 because in general, the initial hidden layer has higher neuron, so the inner loop
    ↪ will can be easily
# incremented
iCombination = 0
M1 = 5
M2 = 5
flagError = 0
while flagM1M2 == 0 and flagError == 0:
    epoch = 0
    errorForPlot = np.zeros(epochIterations)
    numberOfNeuronsHiddenLayer = [M1, M2]

    weightsLayer01 = np.random.uniform(-2, 2, numberOfNeuronsHiddenLayer[0] * numberOfInputNeurons)
    weightsLayer01 = np.reshape(weightsLayer01, (numberOfNeuronsHiddenLayer[0], numberOfInputNeurons))
    weightsLayer12 = np.random.uniform(-2, 2, numberOfNeuronsHiddenLayer[1] *
        ↪ numberOfNeuronsHiddenLayer[0])
    weightsLayer12 = np.reshape(weightsLayer12, (numberOfNeuronsHiddenLayer[1],
        ↪ numberOfNeuronsHiddenLayer[0]))
    weightsLayer23 = np.random.uniform(-2, 2, numberOfOutputNeurons * numberOfNeuronsHiddenLayer[1])
    weightsLayer23 = np.reshape(weightsLayer23, (numberOfOutputNeurons, numberOfNeuronsHiddenLayer[1]))

    thresholdLayer01 = np.random.uniform(-1, 1, numberOfNeuronsHiddenLayer[0])
    thresholdLayer12 = np.random.uniform(-1, 1, numberOfNeuronsHiddenLayer[1])
    thresholdLayer23 = np.random.uniform(-1, 1, numberOfOutputNeurons)

    neuronValueM1 = np.zeros(M1)
    neuronValueM2 = np.zeros(M2)
    outputNeurons = np.zeros(numberOfPatterns)
    outputNeuronsTest = np.zeros(len(testNeurons))

    errorValueM1 = np.zeros(M1)
    errorValueM2 = np.zeros(M2)
```

```
errorOutput = 0 # Since there is only one output neuron

while epoch < epochIterations and flagError == 0:
    pickedArray = np.zeros(numberOfPatterns)
    epoch += 1
    for iFold in range(nFolds):
        outputNeurons = np.zeros(numberOfPatterns)
        iter = 0
        classificationError = 0
        while iter < iterations:
            randomPatternIndex = np.random.choice(foldIndices[iFold], 1)
            randomPatternIndex = int(randomPatternIndex.item(0))
            pickedArray[randomPatternIndex] += 1

            # Forward propagation
            for iM1 in range(M1): # Hidden layer 1
                neuronValueM1[iM1] = np.tanh(
                    -thresholdLayer01[iM1] + np.dot(weightsLayer01[iM1], storedPatterns[
                        ↪ randomPatternIndex]))
            for iM2 in range(M2): # Hidden layer 2
                neuronValueM2[iM2] = np.tanh(
                    -thresholdLayer12[iM2] + np.dot(weightsLayer12[iM2], neuronValueM1))
            # Since there is only one output neuron, no for loop here.
            outputNeurons[randomPatternIndex] = np.tanh(
                -thresholdLayer23 + np.dot(weightsLayer23, neuronValueM2))

            # Error computation for output layer, again one output neuron so no for loop
            errorOutput = np.dot(weightsLayer23, neuronValueM2) - thresholdLayer23
            errorOutput = (1 - (np.tanh(errorOutput)) ** 2) # g'(B_i)
            errorOutput = errorOutput * (targetNeurons[randomPatternIndex] - outputNeurons[
                ↪ randomPatternIndex])

            # Back propagation i.e. error for hidden layers
            for iM2 in range(M2): # Hidden layer 1
                errorValueM2[iM2] = np.dot(weightsLayer12[iM2], neuronValueM1) - thresholdLayer12[iM2
                    ↪ ]
                errorValueM2[iM2] = (1 - (np.tanh(errorValueM2[iM2])) ** 2) # g'(b_i)
                # We can multiply without dot as there is only one output neuron
                errorValueM2[iM2] = errorValueM2[iM2] * errorOutput * [row[iM2] for row in
                    ↪ weightsLayer23]
            for iM1 in range(M1): # Hidden layer 1
                errorValueM1[iM1] = np.dot(weightsLayer01[iM1], storedPatterns[randomPatternIndex]) -
                    ↪  \
                                    thresholdLayer01[iM1]
                errorValueM1[iM1] = (1 - (np.tanh(errorValueM1[iM1])) ** 2) # g'(b_i)
                # Note here we are summing over columns times error in previous computed hidden layer
                    ↪  error
                errorValueM1[iM1] = errorValueM1[iM1] * np.dot([row[iM1] for row in weightsLayer12],
                                                                errorValueM2)

            # Update weights and thresholds
            weightsLayer01 = weightsLayer01 + learningRate * \
                             np.matmul(np.transpose([errorValueM1]), [storedPatterns[
                                 ↪ randomPatternIndex]])
            weightsLayer12 = weightsLayer12 + learningRate * \
                             np.matmul(np.transpose([errorValueM2]), [neuronValueM1])
            weightsLayer23 = weightsLayer23 + learningRate * errorOutput * neuronValueM2
            thresholdLayer01 = thresholdLayer01 - learningRate * errorValueM1
            thresholdLayer12 = thresholdLayer12 - learningRate * errorValueM2
            thresholdLayer23 = thresholdLayer23 - learningRate * errorOutput
            iter += 1
```

```
    for iValidate in range(len(testNeurons)):
        for iM1 in range(M1): # Hidden layer 1
            neuronValueM1[iM1] = np.tanh(
                -thresholdLayer01[iM1] + np.dot(weightsLayer01[iM1], fedPatterns[iValidate]))
        for iM2 in range(M2): # Hidden layer 2
            neuronValueM2[iM2] = np.tanh(
                -thresholdLayer12[iM2] + np.dot(weightsLayer12[iM2], neuronValueM1))
        # Since there is only one output neuron, no for loop here.
        outputNeuronsTest[iValidate] = np.tanh(
            -thresholdLayer23 + np.dot(weightsLayer23[0], neuronValueM2))
        outputNeuronsTest[iValidate] = np.sign(outputNeuronsTest[iValidate])
        classificationError += np.abs(outputNeuronsTest[iValidate] - testNeurons[iValidate])
    classificationError = classificationError / (2 * len(testNeurons))
    errorForPlot[epoch - 1] = classificationError
    print("Classification␣Error␣=␣", classificationError)
    print("Epoch␣=␣", epoch)
    if classificationError < 0.12:
        flagError = 1
        print("Error␣came␣down␣to␣0.12")
        print("Layer␣1␣=␣", M1)
        print("Layer␣2␣=␣", M2)
        w1 = np.asarray(weightsLayer01)
        np.savetxt("D:\\Masters␣Program␣Chalmers\\Projects␣and␣Labs\\ANN\\w1.csv", w1, delimiter=","
            ↪ )
        w2 = np.asarray(weightsLayer12)
        np.savetxt("D:\\Masters␣Program␣Chalmers\\Projects␣and␣Labs\\ANN\\w2.csv", w2, delimiter=","
            ↪ )
        w3 = np.asarray(weightsLayer23)
        np.savetxt("D:\\Masters␣Program␣Chalmers\\Projects␣and␣Labs\\ANN\\w3.csv", w3, delimiter=","
            ↪ )
        t1 = np.asarray(thresholdLayer01)
        np.savetxt("D:\\Masters␣Program␣Chalmers\\Projects␣and␣Labs\\ANN\\t1.csv", t1, delimiter=","
            ↪ )
        t2 = np.asarray(thresholdLayer12)
        np.savetxt("D:\\Masters␣Program␣Chalmers\\Projects␣and␣Labs\\ANN\\t2.csv", t2, delimiter=","
            ↪ )
        t3 = np.asarray(thresholdLayer23)
        np.savetxt("D:\\Masters␣Program␣Chalmers\\Projects␣and␣Labs\\ANN\\t3.csv", t3, delimiter=","
            ↪ )

plt.figure(iCombination)
plt.plot(errorForPlot)
plt.xlabel("Epochs")
plt.ylabel("Classification␣Error")
plotTitle = "Classification␣Error␣for␣M1␣=␣" + str(M1) + ",␣M2␣=␣" + str(M2)
plt.title(plotTitle)
fileName = "classificationErrorPlot_" + str(M1) + "_" + str(M2)
plt.savefig("D:\\Masters␣Program␣Chalmers\\Projects␣and␣Labs\\ANN\\%s.png" % fileName)

if iCombination % 2 == 0:
    M1 += 2
    M2 += 2
elif iCombination % 3 == 0:
    M1 += 2
    M2 += 3
else:
    M1 += 3
    M2 += 2
iCombination += 1
if M1 > 15 or M2 > 15:
    flagM1M2 = 1
```