# When to stop training

A major challenge in designing neural networks is understanding when to stop training them. If a network isn’t trained enough, it won’t fit either the training or test data sets provided to it, however, too much training will cause it to fit too well to the training data provided, causing errors in test data sets. The solution to this problem is to stop training the network when the training error increases1.

A training error is created by running the neural network on a set of data that is hasn’t seen and learnt from and checking how well it performs on this unseen data, typically using validation loss as a metric for performance.

# Probability distribution of outputs of Y

Probability distribution is the chance of a specific event occurring out of all the possible outcomes for an experiment2, and is calculated through the use of a SoftMax function.

After running the program for 100 epochs, I ended up with the following outputs:

Text

Description automatically generated

# Learning Curve

(for the first 100 epochs)

Chart

Description automatically generated

# Concise Table of Weights

First 3dp (not rounded)

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Step | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| W04 | 0.9 | 0.872 | 0.878 | 0.884 | 0.889 | 0.895 | 0.900 | 0.905 | 0.910 | 0.914 | 0.918 |
| W14 | 0.74 | 0.717 | 0.715 | 0.713 | 0.711 | 0.709 | 0.707 | 0.705 | 0.703 | 0.701 | 0.700 |
| W24 | 0.8 | 0.771 | 0.766 | 0.762 | 0.758 | 0.755 | 0.751 | 0.747 | 0.744 | 0.740 | 0.737 |
| W34 | 0.35 | 0.324 | 0.321 | 0.318 | 0.315 | 0.312 | 0.310 | 0.307 | 0.305 | 0.302 | 0.300 |
| W05 | 0.45 | 0.416 | 0.408 | 0.398 | 0.388 | 0.377 | 0.365 | 0.353 | 0.341 | 0.328 | 0.316 |
| W15 | 0.13 | 0.116 | 0.121 | 0.126 | 0.130 | 0.134 | 0.138 | 0.142 | 0.147 | 0.152 | 0.157 |
| W25 | 0.4 | 0.381 | 0.385 | 0.389 | 0.393 | 0.397 | 0.401 | 0.406 | 0.410 | 0.415 | 0.421 |
| W35 | 0.97 | 0.953 | 0.958 | 0.962 | 0.965 | 0.969 | 0.973 | 0.978 | 0.982 | 0.987 | 0.991 |
| W06 | 0.36 | 0.244 | 0.229 | 0.222 | 0.216 | 0.210 | 0.204 | 0.197 | 0.189 | 0.182 | 0.173 |
| W16 | 0.68 | 0.616 | 0.610 | 0.610 | 0.611 | 0.612 | 0.613 | 0.615 | 0.617 | 0.619 | 0.622 |
| W26 | 0.1 | 0.023 | 0.016 | 0.015 | 0.015 | 0.016 | 0.017 | 0.018 | 0.020 | 0.023 | 0.025 |
| W36 | 0.96 | 0.889 | 0.882 | 0.882 | 0.882 | 0.883 | 0.884 | 0.886 | 0.888 | 0.890 | 0.892 |
| W07 | 0.98 | 0.370 | 0.269 | 0.237 | 0.215 | 0.195 | 0.174 | 0.154 | 0.134 | 0.114 | 0.094 |
| W47 | 0.35 | -0.161 | -0.225 | -0.229 | -0.225 | -0.220 | -0.216 | -0.221 | -0.206 | -0.202 | -0.198 |
| W57 | 0.5 | 0.042 | -0.009 | -0.007 | 0.002 | 0.014 | 0.026 | 0.038 | 0.050 | 0.062 | 0.075 |
| W67 | 0.9 | 0.450 | 0.403 | 0.408 | 0.420 | 0.443 | 0.447 | 0.460 | 0.473 | 0.486 | 0.500 |
| W08 | 0.92 | 0.477 | 0.437 | 0.452 | 0.473 | 0.496 | 0.519 | 0.541 | 0.564 | 0.586 | 0.608 |
| W48 | 0.8 | 0.382 | 0.321 | 0.307 | 0.300 | 0.295 | 0.289 | 0.284 | 0.279 | 0.275 | 0.270 |
| W58 | 0.13 | -0.254 | -0.326 | -0.335 | -0.348 | -0.361 | -0.374 | -0.387 | -0.400 | -0.413 | -0.427 |
| W68 | 0.8 | 0.412 | 0.348 | 0.326 | 0.310 | 0.295 | 0.280 | 0.265 | 0.250 | 0.235 | 0.220 |

# References

1. Igareta, Angel (2021). The Million-Dollar Question: When to Stop Training your Deep Learning Model
2. Everitt, Brian (2006). The Cambridge dictionary of statistics (3rd ed.). Cambridge, UK: Cambridge University Press

# Appendix – Code

import math

import matplotlib.pyplot as plt

# Function to get net of input

def netx(inputNum, o):

return ( (inputNum[0] \* o[0]) + (inputNum[1] \* o[1]) + (inputNum[2] \* o[2] + (inputNum[3] \* o[3]) ))

# Function to get sigmoid of input

def sigmoidx(x):

return 1 / (1 + math.exp(-x))

# Function to get softmax of input

def softmaxx(net1, net2):

netAddition = math.exp(net1) + math.exp(net2)

return (math.exp(net1) / netAddition)

# Function to calculate a point for the learning curve

def trainingError(target1, target2, output1, output2):

return ((((target1-output1)\*\*2)+((target2-output2)\*\*2))/2)

# Define weights

#w04, w14, w24, w34

weightA4 = [0.9, 0.74, 0.8, 0.35]

#w05, w15, w25, w35

weightA5 = [0.45, 0.13, 0.4, 0.97]

#w06, w16, w26, w36

weightA6 = [0.36, 0.68, 0.1, 0.96]

#w07, w47, w57, w67

weightA7 = [0.98, 0.35, 0.5, 0.9]

#w08, w48, w58, w68

weightA8 = [0.92, 0.8, 0.13, 0.8]

# Define learning rates

learningRate = 0.1

# Create empty arrays to be filled later

testLine = []

fileData = []

trainingErrorPoints = []

# Create variable to hold the sum of training errors for each epoch

trainingErrorSum = 0

# Fill array with training data

# numLines to store how many lines of data are in the file

numLines = 0

# Open training data file

file = open("data-CMP2020M-item1-train.txt", "r")

# Read every line from training data into an array

while True:

#Split line into array

fileLine = (file.readline())

# If nothing on line, break loop

if fileLine == '':

break

# Split fileLine do

splitLine = fileLine.split()

# x0, x1, x2, x3, target1, target 2

line = [1, float(splitLine[0]), float(splitLine[1]), float(splitLine[2]), int(splitLine[3]), int(splitLine[4])]

fileData.append(line)

#fileData formatting:

#fileData[line][xX or targetX]

#e.g.

#fileData[3][2] = line 4 x2

#filedata[2][4] = line 3 target1

# Increment numLines

numLines += 1

# Close file after use

file.close()

# Run for given number of epochs

for epoch in range(1, 101):

# Runs through forward and backward step for each line in the training file

for i in range(0, numLines):

# Forward step

net4 = netx(weightA4, [fileData[i][0], fileData[i][1], fileData[i][2], fileData[i][3]])

net5 = netx(weightA5, [fileData[i][0], fileData[i][1], fileData[i][2], fileData[i][3]])

net6 = netx(weightA6, [fileData[i][0], fileData[i][1], fileData[i][2], fileData[i][3]])

o4 = sigmoidx (net4)

o5 = sigmoidx (net5)

o6 = sigmoidx (net6)

net7 = netx(weightA7, [fileData[i][0], o4, o5, o6])

net8 = netx(weightA8, [fileData[i][0], o4, o5, o6])

o7 = net7

o8 = net8

# Backward step

# Calculate errors

#Output errors

outputError7 = fileData[i][4] - o7

outputError8 = fileData[i][5] - o8

#Hidden errors

# a7[1] is w47, a8[1] is w48

hiddenError4 = o4 \* (1 - o4) \* ( (weightA7[1] \* outputError7) + (weightA8[1] \* outputError8))

# a7[1] is w57, a8[1] is w58

hiddenError5 = o5 \* (1 - o5) \* ( (weightA7[2] \* outputError7) + (weightA8[2] \* outputError8))

# a7[1] is w67, a8[1] is w68

hiddenError6 = o6 \* (1 - o6) \* ( (weightA7[3] \* outputError7) + (weightA8[3] \* outputError8))

# Display Errors for debug purposes

#print ("HiddenError4: " + str(hiddenError4) + "\nHiddenError5: " + str(hiddenError5) + "\nHiddenError6: " + str(hiddenError6))

#print ("OutputError1: " + str(outputError7) + "\nOutputError2: " + str(outputError8) )

# Recalculate Output Values

#weightUpdateX = learningRate \* hiddenErrorX \* weightX

#newWeight = oldWeight + weightUpdate

weightA4 = [ (weightA4[0] + (learningRate \* hiddenError4 \* fileData[i][0])), (weightA4[1] + (learningRate \* hiddenError4 \* fileData[i][1])), (weightA4[2] + (learningRate \* hiddenError4 \* fileData[i][2])), (weightA4[3] + (learningRate \* hiddenError4 \* fileData[i][3]))]

weightA5 = [ (weightA5[0] + (learningRate \* hiddenError5 \* fileData[i][0])), (weightA5[1] + (learningRate \* hiddenError5 \* fileData[i][1])), (weightA5[2] + (learningRate \* hiddenError5 \* fileData[i][2])), (weightA5[3] + (learningRate \* hiddenError5 \* fileData[i][3]))]

weightA6 = [ (weightA6[0] + (learningRate \* hiddenError6 \* fileData[i][0])), (weightA6[1] + (learningRate \* hiddenError6 \* fileData[i][1])), (weightA6[2] + (learningRate \* hiddenError6 \* fileData[i][2])), (weightA6[3] + (learningRate \* hiddenError6 \* fileData[i][3]))]

#weightUpdateX = learningRate \* outputErrorx \* (weightX or ox)

#newWeight = oldWeight + weightUpdate

weightA7 = [ (weightA7[0] + (learningRate \* outputError7 \* fileData[i][0])), (weightA7[1] + (learningRate \* outputError7 \* o4)), (weightA7[2] + (learningRate \* outputError7 \* o5)), (weightA7[3] + (learningRate \* outputError7 \* o6))]

weightA8 = [ (weightA8[0] + (learningRate \* outputError8 \* fileData[i][0])), (weightA8[1] + (learningRate \* outputError8 \* o4)), (weightA8[2] + (learningRate \* outputError8 \* o5)), (weightA8[3] + (learningRate \* outputError8 \* o6))]

# Add training error of current step to total training error of given epoch

trainingErrorSum += trainingError(fileData[i][4], fileData[i][5], o7, o8)

# Append total sum of training errors for epoch to array

trainingErrorPoints.append(trainingErrorSum)

# Reset sum of traing errors for next epoch

trainingErrorSum = 0

# Display output values for debug purposes

#print (str(epoch))

#print ("w04: " + str(weightA4[0]) + "\nw14: " + str(weightA4[1]) + "\nw24: " + str(weightA4[2]) + "\nw34: " + str(weightA4[3]))

#print ("w05: " + str(weightA5[0]) + "\nw15: " + str(weightA5[1]) + "\nw25: " + str(weightA5[2]) + "\nw35: " + str(weightA5[3]))

#print ("w06: " + str(weightA6[0]) + "\nw16: " + str(weightA6[1]) + "\nw26: " + str(weightA6[2]) + "\nw36: " + str(weightA6[3]))

#print ("w07: " + str(weightA7[0]) + "\nw47: " + str(weightA7[1]) + "\nw57: " + str(weightA7[2]) + "\nw67: " + str(weightA7[3]))

#print ("w08: " + str(weightA8[0]) + "\nw48: " + str(weightA8[1]) + "\nw58: " + str(weightA8[2]) + "\nw68: " + str(weightA8[3]) + "\n\n\n")

# Fill array with test data

# Open test data file

testFile = open("data-CMP2020M-item1-test.txt", "r")

#Split line into array

testFileLine = (testFile.readline())

# Split fileLine do

splitLine = testFileLine.split()

# x0, x1, x2, x3

testLine = [1, float(splitLine[0]), float(splitLine[1]), float(splitLine[2])]

# Close file after use

testFile.close()

# Calculate output from test values

net4 = netx(weightA4, [testLine[0], testLine[1], testLine[2], testLine[3]])

net5 = netx(weightA5, [testLine[0], testLine[1], testLine[2], testLine[3]])

net6 = netx(weightA6, [testLine[0], testLine[1], testLine[2], testLine[3]])

o4 = sigmoidx (net4)

o5 = sigmoidx (net5)

o6 = sigmoidx (net6)

net7 = netx(weightA7, [testLine[0], o4, o5, o6])

net8 = netx(weightA8, [testLine[0], o4, o5, o6])

o7 = net7

o8 = net8

# Display output of test values

print ("y1: " + str(net7))

print ("y2: " + str(net8))

print ("softmax y1/Probability distribution for y1: " + str(softmaxx(o7, o8)))

print ("softmax o8/Probability distribution for y1: " + str(softmaxx(o8, o7)))

# Take the training error points and plot it as a line graph.

plt.plot(trainingErrorPoints)

plt.show()