CIS3187- Assignment Documentation
Chakotay Incorvaia 358199(M)

Business Intelligence

Semester 1

2019/2020

B.Sc. IT(Hons) Software Development

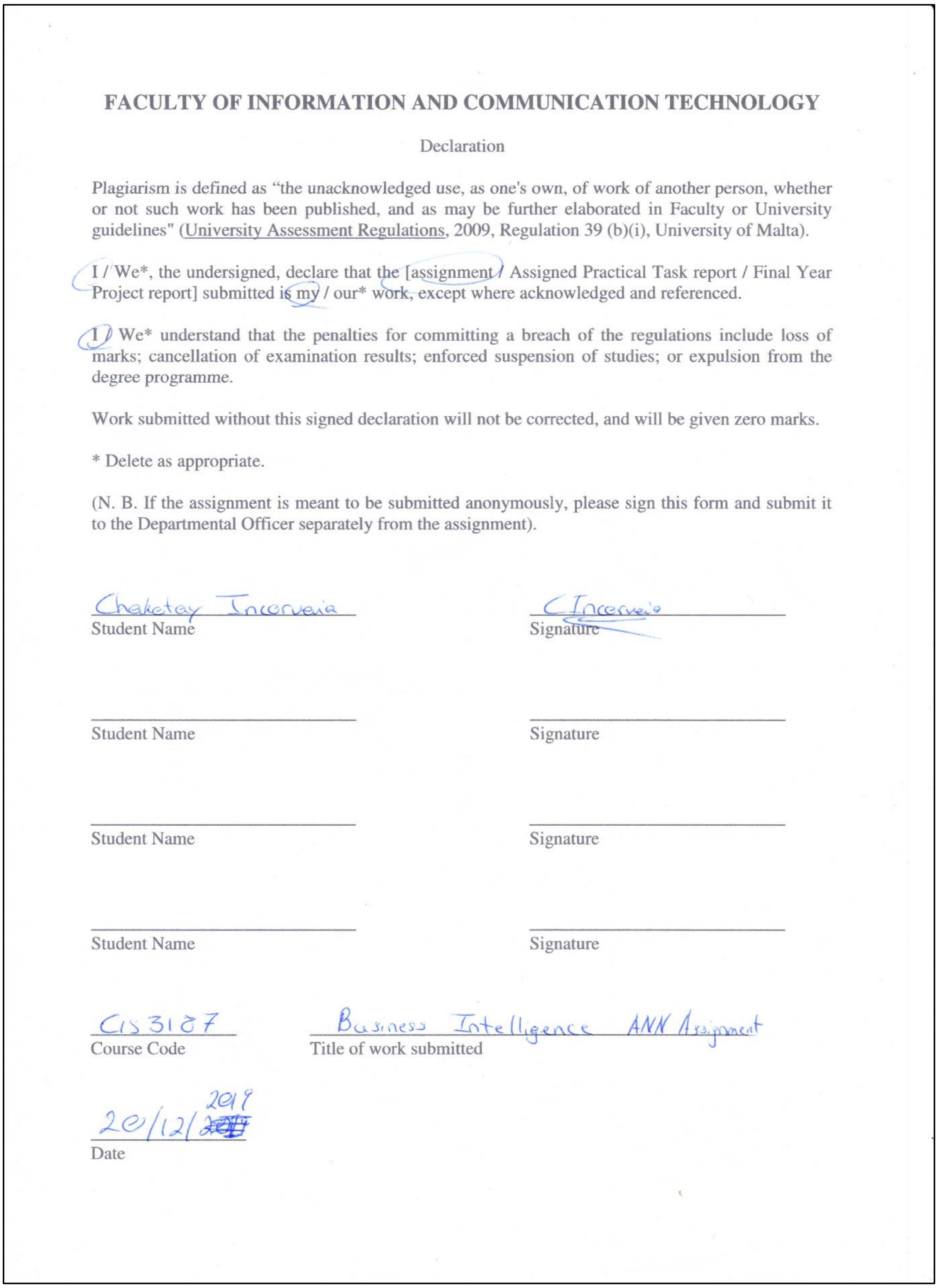
Table of Contents

Statement of Completion

Objective	Status
Training Set, Test Set and Boolean Function	Complete
Data Structures for all components of ANN	Complete
Feed forward	Complete
Error calculation and checking	Complete
Backpropagation with learning rate	Complete
Weight Updates	Complete
Plot graph of training set	Complete
Plot graph of testing set	Complete

All requirements have been met in order to declare this assignment as 100% completed.

Plagiarism Form



FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Declaration

Plagiarism is defined as "the unacknowledged use, as one's own, of work of another person, whether or not such work has been published, and as may be further elaborated in Faculty or University guidelines" (University Assessment Regulations, 2009, Regulation 39 (b)(i), University of Malta).

[✓ I] / We*, the undersigned, declare that the [[✓ assignment] / Assigned Practical Task report / Final Year Project report] submitted is [✓ my] / our* work, except where acknowledged and referenced.

[✓ I] / We* understand that the penalties for committing a breach of the regulations include loss of marks; cancellation of examination results; enforced suspension of studies; or expulsion from the degree programme.

Work submitted without this signed declaration will not be corrected, and will be given zero marks.

* Delete as appropriate.

(N. B. If the assignment is meant to be submitted anonymously, please sign this form and submit it to the Departmental Officer separately from the assignment).

Chakotay Incorvaia	C Incorvaia
Student Name	Signature
Student Name	Signature
Student Name	Signature
Student Name	Signature

CIS3187	Business Intelligence ANN Assignment
Course Code	Title of work submitted

20/12/~~[illegible]~~ 2019

Date

Development Stages

Planning Stage

Before any programming was done, a few hours were spent in understand the structure and operation of a neural network. Such as understanding the matrix multiplication required for feedforward as well as the equations required to do error back propagation.

The language chosen for this project was Python 3.8 given the wide availability of online resources as well as the prevalence of mathematical and statistical based libraries one may use.

It is important to note that no prior development was experienced with Python, and so most issues experienced in the project was due to lack of knowledge of syntax and other characteristics of the language.

Given that Python is a scripting language, the project is simply broken down into smaller bite-sized functions which perform a specific task, which in turn are called in order of operation. The entire program is run off one class called "Main.py" and was not split into numerous classes.

Finally, the project was uploaded to GitHub so that it was easier to work on the code whilst switching workstations.

Additional Note: The python version was later downgraded to Python 3.7.4 due to issues with finding an easy to use plotting and graphing library.

Data Set Used

Input	Target
00000	100
00001	100
00010	101
00011	101
00100	110
00101	110
00110	111
00111	111
01000	100
01001	100
01010	101
01011	101
01100	110
01101	110
01110	111
01111	111
10000	000
10001	000
10010	001

10011	001
10100	010
10101	010
10110	011
10111	011
11000	000
11001	000
11010	001
11011	001
11100	010
11101	010
11110	011
11111	011

The data set was placed inside the program automatically and not generated from a csv file as required. Due to only having only **5 inputs** the possible numbers are only up to 32. Given that the output can only be **3**, randomly generating the Boolean function is not worth the effort.

The Boolean function used in the program is of ABCDE = ¬ACD.

Early Development

The project first started with creating the basic data structures that are to be used throughout the program. A basic data set was also created as well as the resulting target output which is passed through a Boolean function. The Boolean function is quite simply ¬ACD from the numbers 0 to 31.

Initial development was not done with a randomized data set or testing set in order to hasten production. Only one fact was used of the data set so that the feedforward can be tested.

During the early stages of development, it was noticed that calling each method individually was not good practice and so a main method was created which in turn would call the required methods. This function's task is to encapsulate the entire program from running all the facts to calling back propagation at the appropriate time. The running of the test set and functions tasked with graphing the result would be instead run separated from the main functionality of the program.

At this stage of development, the feed forward functionality of the neural network was completed using basic matrix multiplication with the python library NumPy. Surprisingly passing an array to the sigmoid function returns a same sized array with each value within it have had sigmoid applied to it. These features greatly helped in reducing development time which would have had to be spent handling matrix multiplication.

No major issues were experienced so far early in development with regards to the functionality of the project. The biggest issues were learning the syntax of Python and issues experienced with PyCharm and git.

Main Development Stage

During the main stage of development, the back-propagation function was being developed. This proved to be the hardest task of the entire project as the understanding of the equations in the **Planning Stage** was **inaccurate** and needed thorough revision.

Indeed, the development for the hidden layer delta was most time consuming and once developed the neural network was not correctly fixing the graph.

Whilst the equation for the outer layer deltas were correct, the weights were not being adjusted properly. The same issue was occurring for the hidden layer delta calculation. This issue was later found to be caused by the indices of the arrays being set poorly. The cause was because the weight adjustment equation was not set properly to use the correct layer. Adjustments to the hidden layer weights had to be set to use the inputs whilst the outer layer weights needed to make use of the hidden layer.

Another issue was found to be with the error checking, which did not work in certain cases because the function was not disregarding whether the input was negative or positive when being compared to be > 0.2. This caused the neural network to not call the backpropagation algorithm when the value was negative such as -0.3.

Once these issues were fixed, the data set was separated as instructed by shuffling the data set into a training set and reserving 20% into a separate array for the testing set.

Final Development Stage

Once the program had its feedforward and backpropagation working successfully, it was edited in order to be able to run the entire training set and for the required number of epochs or until the condition is met.

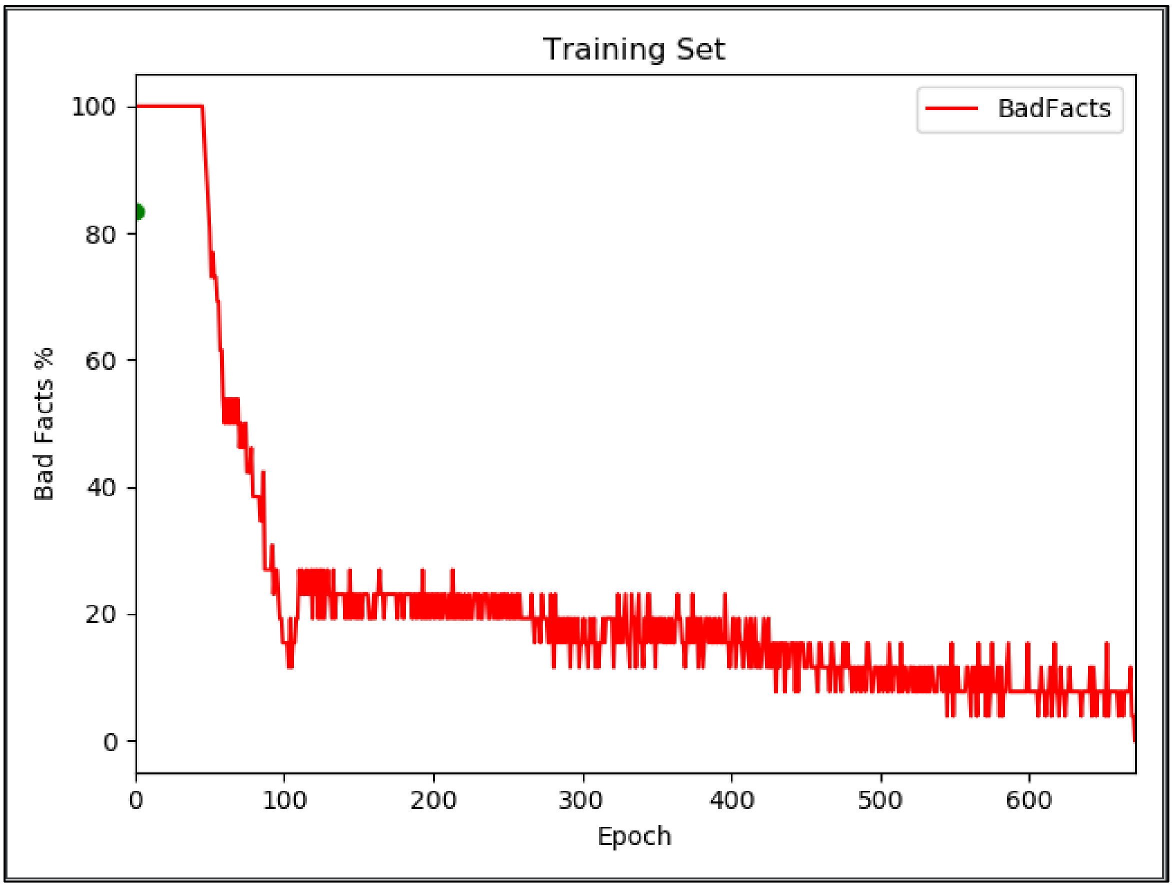
The test set will only run once, meaning that is represents only 1 epoch which will run at the end of training.

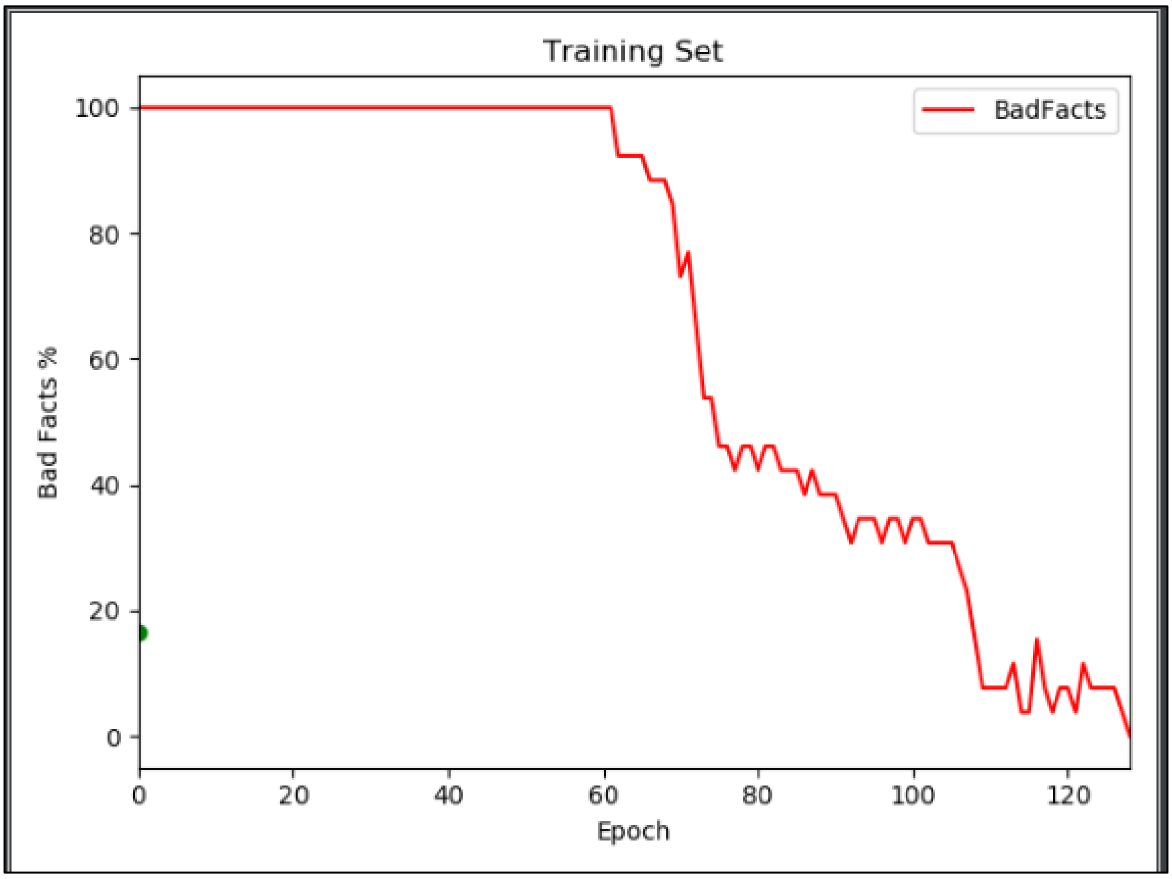
The last stage of development required the program to plot the results of the training set and testing set on a graph. This feature was time-consuming because it was difficult in finding an appropriate library which provided enough documentation for it to be used. Numerous attempts were done with **'plotly'** but alas they were all failures due to unclear documentation. It was then decided to downgrade python to 3.7.4 so that **'matplotlib'** could be used instead. **'Pandas'** was initially used for plotly but was kept for use in matplotlib by structuring a data frame from which the graph could be plotted neatly.

When the program is run, the chosen IDE should display the graph. The training set of the neural network is displayed in the line graph as a red line, whilst the test epoch is displayed as a green dot.

The y-axis represents the total percentage of bad facts in the epoch whilst the x-axis represents the number of epochs. **An Observation: Whilst the training set is fully trained within the 1000 Epoch limit, the test set does not always run without a bad epoch. There are also cases in which the training set does not train itself fully within the 1000 epoch limit.**

Please see below the **training set results and testing results**;





Improvements Required

The following are improvements that would have been made should time have permitted.

- Separation of main class into separate classes
- Neater and nicer graphs
- Latest python version
- Proper git dependency checking by using **'pipenv'** or another package managing tool to allow for easier setup for exam evaluation
- Documentation included in the README of the project with images
- Randomly generated Boolean function

Appendix 1 – Code

GitHub Access:

This project will be published and archived on GitHub for documentation purposes within 24 hours of assignment deadline. It will also provide insight on the duration of development and issues tackled during its development using the commit history.

https://github.com/CHAKOTAY99/BI_Assignment

Code:

```
import numpy as np
import random
import sys
import pandas as pd
import matplotlib.pyplot as plt
np.set_printoptions(threshold=sys.maxsize)

# Training Set 2^5 = 32 ABCDE = ¬ACD # : 80% of 32 = 26, 20% = 6
dataSet = np.array([["00000", "100"],
            ["00001", "100"],
            ["00010", "101"],
            ["00011", "101"],
            ["00100", "110"],
            ["00101", "110"],
            ["00110", "111"],
            ["00111", "111"],
            ["01000", "100"],
            ["01001", "100"],
            ["01010", "101"],
            ["01011", "101"],
            ["01100", "110"],
            ["01101", "110"],
            ["01110", "111"],
            ["01111", "111"],
            ["10000", "000"],
            ["10001", "000"],
            ["10010", "001"],
            ["10011", "001"],
            ["10100", "010"],
            ["10101", "010"],
            ["10110", "011"],
            ["10111", "011"],
            ["11000", "000"],
            ["11001", "000"],
            ["11010", "001"],
            ["11011", "001"],
            ["11100", "010"],
            ["11101", "010"],
            ["11110", "011"],
            ["11111", "011"],
            ])


def randomTrainingSet(dataSet):
  array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
  random.shuffle(dataSet)
  trainingSet = dataSet[:26]
  testSet = dataSet[26:]
  return trainingSet, testSet


def setInputAndTarget(fact):
  inputString = fact[0]
  targetString = fact[1]
  inputMatrix = np.array([0, 0, 0, 0, 0], dtype=np.float64)
  targetMatrix = np.array([0, 0, 0], dtype=np.float64)
  i = 0
  x = 0
  for aNumber in inputString:
    inputMatrix[i] = aNumber
    i = i + 1
  for aNumber in targetString:
    targetMatrix[x] = aNumber
    x = x + 1
  return inputMatrix, targetMatrix


def feedforward(input, target, wIn, wOut):
  netH = np.dot(input, wIn).astype(np.float64)
  outH = sigmoid(netH)
  netO = np.dot(outH, wOut).astype(np.float64)
  outO = sigmoid(netO)
  errorList = np.array([0, 0, 0], dtype=np.float64)
  i = 0
  for eachNumber in outO:
    errorList[i] = np.subtract(target[i], eachNumber)
    i = i + 1

  return outO, outH, errorList


def checkError(errorList):
    mu = 0.2
    for error in errorList:
        if abs(error) > mu:
            return False
    return True


def sigmoid(x):
    return 1 / (1 + np.exp(-x))


def mainFunction():
    # Weights random assignment between -1 and 1
    wIn = np.random.uniform(low=-1, high=1, size=(5, 4))
    wOut = np.random.uniform(low=-1, high=1, size=(4, 3))
    resultSet = randomTrainingSet(dataSet)
    totEpoch = np.zeros((1000,3), dtype=np.float64)
    i = 0
    while True:
        ## epochStorage good - bad fact %tage
        totEpoch[i, 0] = i
        # EPOCH START
        for fact in resultSet[0]:
            ## setInputAndTarget OUTPUT: inputMarix and targetMatrix in that order INPUT: fact
            factResult = setInputAndTarget(fact)
            ## feedforward OUTPUT: outO, outH and errorList in that order INPUT: input matrix, target
matrix, wIn, wOut
            ffResult = feedforward(factResult[0], factResult[1], wIn, wOut)
            passFail = checkError(ffResult[2])
            if passFail == False:
                # fact failed and call EBP
                totEpoch[i, 2] += 1
                ## backPropogation INPUT: outO, outH, targetList, wIn, wOut
                backPropogation(ffResult[0], ffResult[1], factResult[1], factResult[0], wIn, wOut)
            elif passFail == True:
                # fact passed and do nothing
                totEpoch[i, 1] += 1
            # End of Fact
        totEpoch[i, 1] = (totEpoch[i, 1] / 26) * 100
        totEpoch[i, 2] = (totEpoch[i, 2] / 26) * 100
        # End of Epoch
        if i == 999 or totEpoch[i, 2] == 0:
            ## If we reached the 1000 epoch limit just stop it or if no more bad epochs occur
            totEpoch = np.delete(totEpoch, np.s_[i+1:], axis=0)
            return totEpoch, resultSet[1], wIn, wOut
        # Add new line to epoch storage
        i = i + 1


def deltaOFormula(outO, targetList):
    i = 0
    deltaO = np.array([0, 0, 0], dtype=np.float64)
    for entry in outO:
        deltaO[i] = entry * (1 - entry) * (targetList[i] - entry)
        i = i + 1
    return deltaO


def sigmaDelta(deltaO, index, wOut):
    sum = 0
    x = 0
    for delta in deltaO:
        sum += (delta * wOut[index, x])
        x = x + 1
    return sum


def deltaHFormula(deltaO, outH, wOut):
    i = 0
    deltaH = np.array([0, 0, 0, 0], dtype=np.float64)
    for entry in outH:
        deltaH[i] = (entry * (1 - entry) * (sigmaDelta(deltaO, i, wOut)))
        i = i + 1
    return deltaH


def wOCalc(deltaList, outH, wOut):
    eta = 0.2
    i = 0
    for output in outH:
        x = 0
        for delta in deltaList:
            wOut[i][x] += eta * delta * output
            x = x + 1
        i = i + 1


def wInCalc(deltaList, inputList, wIn):
    eta = 0.2
    i = 0
    for input in inputList:
        x = 0
        for delta in deltaList:
            wIn[i][x] += eta * delta * input
            x = x + 1
        i = i + 1


def backPropogation(outO, outH, targetList, inputList, wIn, wOut):
    # Change weights of output - WOut
    deltaO = deltaOFormula(outO, targetList)
    wOCalc(deltaO, outH, wOut)
    # Change weights of hidden - wIn
    deltaH = deltaHFormula(deltaO, outH, wOut)
    wInCalc(deltaH, inputList, wIn)

def plotGraph(trainingData, testData):
    training = pd.DataFrame(data=trainingData[0:, 1:],
                            index=trainingData[0:, 0],
                            columns=trainingData[0, 1:])
    training.columns = ['GoodFacts', 'BadFacts']

    training.reset_index().plot(kind='line', x='index', y='BadFacts', color='red')

    plt.scatter(x=testData[0], y=testData[2], c='green')
    plt.xlabel('Epoch')
    plt.ylabel('Bad Facts %')
    plt.title('Training Set')
    plt.show()


def runTestSet(testSet, wIn, wOut):
    totEpoch = np.array([0, 0, 0], dtype=np.float64)
    # EPOCH START
    for fact in testSet:
        ## setInputAndTarget OUTPUT: inputMarix and targetMatrix in that order INPUT: fact
        factResult = setInputAndTarget(fact)
        ## feedforward OUTPUT: outO, outH and errorList in that order INPUT: input matrix, target
matrix, wIn, wOut
        ffResult = feedforward(factResult[0], factResult[1], wIn, wOut)
        passFail = checkError(ffResult[2])
        if passFail == False:
            # fact failed
            totEpoch[2] += 1
        elif passFail == True:
            # fact passed
            totEpoch[1] += 1
        # End of Fact
    totEpoch[1] = (totEpoch[1] / 6) * 100
    totEpoch[2] = (totEpoch[2] / 6) * 100
    return totEpoch

epochStorage = mainFunction()
testStorage = runTestSet(epochStorage[1], epochStorage[2], epochStorage[3])
plotGraph(epochStorage[0], testStorage)
```