Hassaan Naveed

N0898071

SOFT10101 Computer Science programming coursework 2019/20

# Specification

## Introduction

The project I have decided to create is a simple neural network designed to deal with the XOR logic problem. The network will take two inputs, each of these inputs being a ‘1’ or a ‘0’, and a label for what the expected output should be. These inputs are passed through various algorithms that train the network in order to produce an output as close to the label as possible.

The program should initialise a three-layer neural network of a given number of input, hidden, and output neurons. For the purposes of this test the input layer should have two neurons and the output layer should have one neuron. The hidden layer can have any number of neurons, but in this case, I will be using two. The network should be ‘fully connected’, meaning all neurons in one layer should connect to all neurons in the next layer. An abstraction of what the network will look like can be found below:

**Input Layer**

**Hidden Layer**

**Output Layer**

## Key Features

* There should be two datasets: one for testing and one for training
* The program should load these datasets from an external file
* The program should update the user on the progress of the network
* The program should ask the user how many times to train the network
* The program should ask the user for the learning rate of the network
* The program should test the network and output the results
* The user can press 1 to start the program
* The user can press 0 to exit the program

# Design and Implementation

## Program Flow

## Classes

**Menu**

menu()

**Network**

trainingData : <int>

testingData : <int>

weights : [Matrix]

biases : [Matrix]

netStructure : [int]

constructor()

trainNetwork()

testNetwork()

initMatrix()

vecToMat()

feedforward()

SDG()

backpropogation()

calculateGradient()

calculateDelta()

sigmoid()

sigmoidPrime()

**DataLoader**

trainingData : <int>

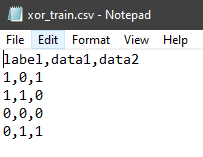
testingData : <int>

readData()

loadData()

## File Structure

The data sets for training and testing the program will be stored as .csv files. Each set of data will be stored in a separate line, for example:

The label is what the expected output should be, and data1 and data2 are the input data for the network. The program will use two .csv files, one to store training data and one to store testing data.

# Test Plan

|  |  |  |
| --- | --- | --- |
| Action | Test Method | Success Criteria |
| Begin the program | Press 1 at the menu | The program will create the network class and the user will be told that the network has been initialised and be given a prompt to begin training |
| Exit the program | Press 2 at the menu | The program will successfully exit |
| Begin training the program | Press enter during the prompt | The program will call the train function which calls SDG. |
| SDG / Backpropagation | Console outputs | The program will pass the data trough backpropagation. It will output the progress as each cycle is complete. The user will be given a prompt once the training is complete |
| Begin testing the program | Press enter during the prompt | The program will call the test function |
| Testing | Console outputs | The program will output the expected output versus the calculated output, and a percentage success rate at the end. |

# Evaluation

Overall, I think the program went well and achieved the goals it was set out to achieve. The program is able to read data from an external csv file and pack the data into vectors. It then uses the stochastic gradient descent algorithm to run each set of data through backpropagation. This slowly adjusts the randomly initialised weights and biases through a series of passes to arrive at an output close to the desired output.

Upon testing with the XOR data set, passing it through 50,000 times with a learning rate of 3, arrives at an output which is close to what is desired. I would consider it a success if the calculated output is within 0.5 of the expected output. This appeared to be the case with the parameters set to those values. The program is a working framework for a simple neural network which could be expanded on in the future and used as a library for future projects, with some changes.

Multiple parts of the program could be improved, for example the stochastic gradient descent algorithm is very simple, as it just runs the entire data set through backpropagation, and a different algorithm could be used in the future, such as mini batch gradient descent, which partitions the data set into small batches and calculates the average error throughout the batch. This could improve the accuracy and efficiency of the program. Furthermore, it could be made to be compatible with custom datasets, which would allow for more flexibility.