Iris I could identify this flower

Machine Learning Algorithm Learning tool that demonstrates how a Neural Network identifies iris species.

Josh Lennon

Contents

Glossary 3
Analysis 4
Introduction 4
Preface 4
Software development method 4
Example 4
Background research 5
Problem 5
Inspiration 5
Existing alternatives 6
Core Research 8
Additional Research 11
Project plan 12
Identification of end user(s) 14
Target Audience 14
Potential Users 15
Required Skills 16
User needs and limitations 17
System requirements 17
Response to prototype 18
Objectives 19
Primary objectives 19
Secondary objectives 19
Chosen solution 20
Language / medium solutions 20
Activation function solutions 20
User solutions 20
Flowcharts 21
Initial variables – Data flow diagram 21
Design 22
High-level overview 22
Input/process/storage/output (IPSO) table 23
Prototype 25
Matrices prototype program 25
Early drafts / drafts I used whilst making the software 27

Initial modelling 28
Initial variables & pseudocode 28
Algorithms 29
Sigmoid squishification function 29
Network function 30
Matrices 31
Back propagation 32
Sorting algorithm 33
Proposed structure 34
Context diagram 35
Hierarchy charts and flow diagrams 36
Hierarchy chart 36
Flow diagrams 37
User interfaces 40
Data dictionary 42
Procedure dictionary 44
Technical Solution 45
Technical Overview 45
Iterative testing 45
Variables list 47
Functions and sub procedures 48
Generate network 49
Batches created 51
Train network 52
Forward propagation 54
Test network 56
Chronologically annotated development of code 59
Testing 64
Introduction 64
Testing strategy 64
Test descriptions 65
Input Validation 65
Output Verification 66
Functionality 67
Efficiency 68
Test evidence 69

Input Validation....69
Output Verification....75
Functionality....80
Efficiency....84
Actions required (as a result of testing)....85
Evaluation....86
High-level evaluation....86
Useability....86
Functionality....86
Efficiency....87
Reflection....88
Objectives met?....88
Primary objectives....88
Secondary objectives....90
Feedback....91
Extended....92
Backpropagation....92
Learning rate....92
User manual....92

Glossary

AI – Acronym for Artificial Intelligence.

Neural network - A model capable of processing input data and adjusting its internal mechanics to learn how to produce a desired result.

HTML – HyperText Markup Language.

IDE – Integrated Development Environment.

Analysis

Introduction

Preface

It is important to start by declaring the difference between machine learning and artificial intelligence. Artificial intelligence is a method in which computers / machines can complete sequential tasks in a way that is intelligent or would have previously required human intelligence e.g., speech recognition. Machine learning is a method of achieving artificial intelligence. I will attempt to create an algorithm that utilises a machine learning process in order to identify the species of an iris flower from set input parameters (petal length, petal width, sepal length and sepal width). The programs main goal is to act as a demonstration tool of how a neural network can be used to simulate machine learning and secondary objective to be a tool in order to identify a new data set to a significant degree of accuracy.

The program will:

- Operate in a windows form interface
- Allow the user to input a dataset for training
- Allow the user to input parameters and return a species once network is trained

For more detailed objectives turn to the problem section of pages 20.

Software development method

The most important part of this development process is proof of concept. As a result, I will approach the algorithm's development using the **Iterative development model.** This model will allow me a working version early in the process and make it less expensive to implement changes. One disadvantage is that resources can be quickly depleted however this is negligible to such a small project.

Example

In order to facilitate the use of my algorithm as a demonstrative tool, each property of the neural network must be displayed in a way that the significance of each parameter is clear to the user. In order to achieve this, the sequence in which each parameter (weight, bias, layer, and node) is declared will be displayed along with the final node value making tracking of the training algorithm status intuitive. It will include clear annotations of the ongoing corresponding inputs and outputs.

```
Start Training!...Here's the data:
[ { input: [ 0, 1 ], output: [ 1, 0 ] } ]
======== TRAINING ITERATION 1 ========
--------- Run input set 0: 0,1 ----------
-> Layer 2 has 3 nodes
   START NODE: 0
   -> bias for node 0: -0.08587803691625595
   -> weights for node 0:
     -> input value: 0, weight: -0.13223229348659515
     -> input value: 1, weight: -0.09341631829738617
   -> node value for layer: 2, node: 0 = -0.17929435521364212 ... value = bias + the sum of (dimension * weight) for every dimension in t
he layer}
   -> sigmoid the node value:  0.4552960991859436
   -> layer:2, node:0 is done
```

Figure 1

Figure 1 is an early example that aided my understanding in the research process. For this reason, I aim to replicate a similar style of notation when the program is in operation.

Background research

Problem

Machine learning is a promising field with new research papers being published daily and is widely referenced in science fiction and modern dystopian literature. AI machine learning is an important and relatively new branch of computer science being this exact algorithm was considered pioneering as recently as the mid-1980s. The notorious Iris data set was collected in 1936 and is frequently utilised and widely regarded as the 'hello world' of machine learning and neural networks. This context makes this algorithm a good tool for demonstrating a self-learning AI in practise, a useful tool in our evermore growing technological world.

The theory behind neural networks is more accessible but equally important to implementation of the concept. For that reason, I've identified demonstrating how implementation is possible with a theoretical background.

The program will aim to demonstrate how categorising Iris samples by species can be possible and once implemented will be a proof of concept that allows, in theory, any plant breed to be categorised in this way being of wider use. This will be done using the 1936 iris dataset to train the program in order for it to identify species from the same parameters accurately. Categorising plants in this way is useful as the AI can be trained on a user specified data set allowing for better custom application.

Inspiration

Most people's introduction to artificial intelligence is a powerful and complex tool that's a far-off reality, whether it be through, Hollywood dystopias, Elon Musk's infamous fear of AI or the gaming community. This was a similar experience to mine until I began to utilise online resources such as YouTube in order to get a better idea of and observe machine learning AI in practice.

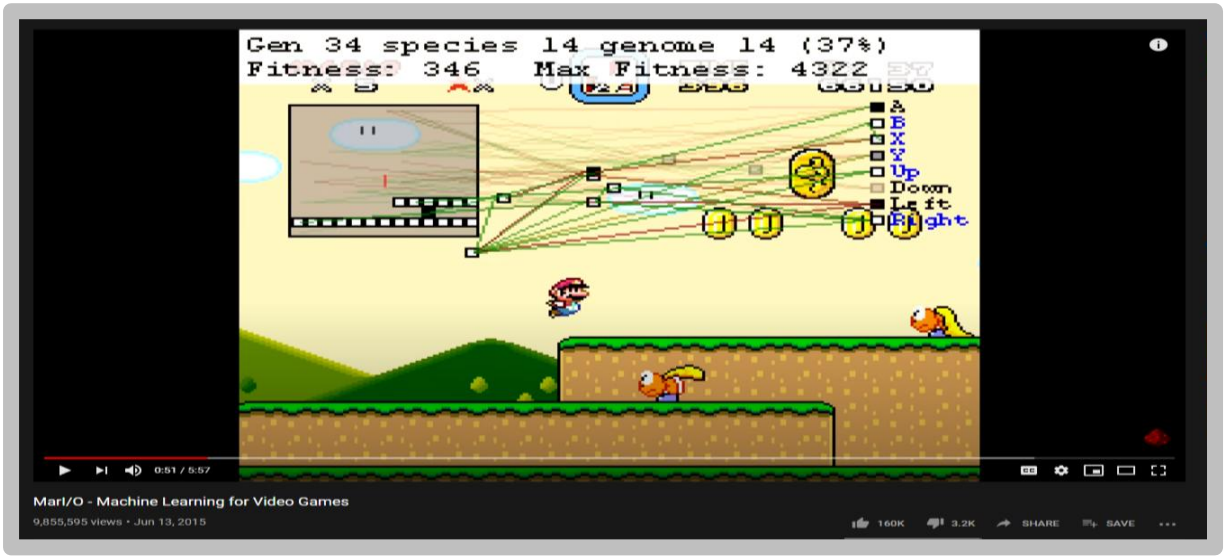


Figure 2 *Source:* https://www.youtube.com/watch?v=qv6UVOQ0F44&t=1s

Machine Learning AI has a large appeal (evident from the millions of views the results are rewarded with) and this particular video demonstration draws parallels to the functions of a human brain. The

way machine learning is presented through an evolving neural network similar to the human brain which conveys how powerful an AI of this manor can become making the concept an attractive idea. This program was inspired by the gap I found while researching between a working demonstration and the theory behind it. I want to have provided a tool that can be used to bridge your understanding of how implementation between point A, the theory, and point B, the application is possible.

After my interest peaked in terms of the power and capability of machine learning AI, I began to research the processes behind neural networks that allow for such powerful algorithms. I watched various YouTube tutorials in order to make this project possible (See *Figure 3*).

Existing alternatives

A Huge portion of the research and design phase is allocated to getting an understanding of the theory behind how a neural network operates. The time will be spread amongst articles, computing communities like GitHub and YouTube, a platform that offers many different resources and utilities for learning. Whilst researching neural networks and primarily forward propagation for this project I utilised many online resources including multiple YouTube videos. As tools for understanding neural networks, these often-had individual strengths and weaknesses that made covering the full topics more difficult. In order to get a good idea of the market for a learning tool, I analysed popular machine learning YouTube videos and articles of differing lesson intensity:

Figure 3

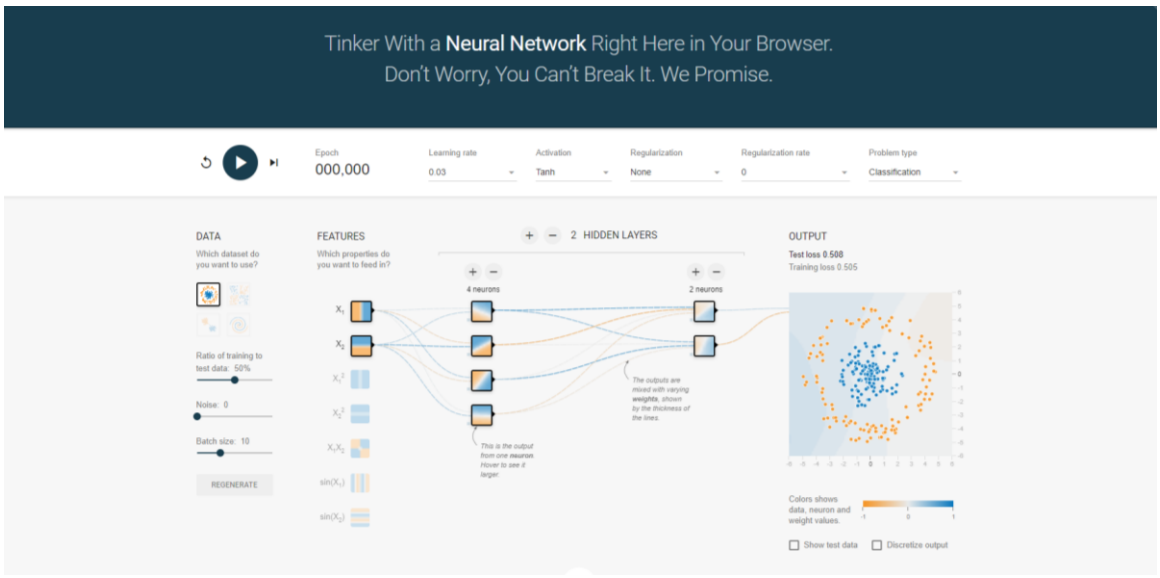
	Pros	Cons
https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/ A step by step back propagation example article by Matt Mazur	• Good example of how back propagation could be used • In depth analysis behind each function and how it can be optimised	• Challenging lexical choices for beginners • Use of the chain rule, matrices and general notation requires a background in mathematics
https://www.youtube.com/watch?v=aircAruvnKk&t=987s Neural Network Theory tutorial	• Good explanation of the theory behind how neural networks work • Introduction to complex terms such as nodes, weights and biases. • Good use of visuals to aid explanations • Long runtime	• Only covers general theory • Doesn't cover how to implement a neural network in practise
https://www.youtube.com/watch?v=9ZsyQZouOQ8&t=20s Figure 1 reference tutorial	• Covers how to implement a neural network in practise • explanation of source code • Source code available	• No in-depth explanation of backpropagation process • A lot of bias as understanding is assumed

My tool will build on these individual strengths and weakness in a way that allows for a useful tool that fills this gap in the market.

Resources for diagnosing the processes of a neural network are not short, the gap in the market I've identified and referred to prior is a shortage of resources bridging theoretical concepts to practical implementation for those concepts. For example, there are many available techniques that outline the principles behind how a neural network would operate, but not many intuitive ways I've found to apply the math when actually constructing one's own algorithm.

Current systems

Tensor flow offers a website in which anybody can manipulate a neural network in advanced depth. This allows users to gain an improved understanding in how to optimise a neural network and what sort of requirements and outputs you would see. This tool is a useful and effective learning device however includes very little to aid the user. The operation behind the network beyond a structure diagram is not depicted and is intended for use by those in experience with working neural networks previously. This solution only allows the user to see how output accuracy and confidence are affected by different parameters and techniques; it will not be useful for beginners



My solution will feature a feed detailing the value of each weight and bias and if possible, a refresh button so that the user can observe exactly how the weights and biases change over time or how the network 'learns'.

Core Research

Once I had identified that I would construct a machine learning algorithm, I entered the research phase of my project plan beginning with neural networks. Neural networks are visually represented using sometimes complex diagrams such as *Figure 4*. It consists of many inputs, weights and nodes. In order to be an affective learning resource, a smaller batch normalised network will be utilised in my program.

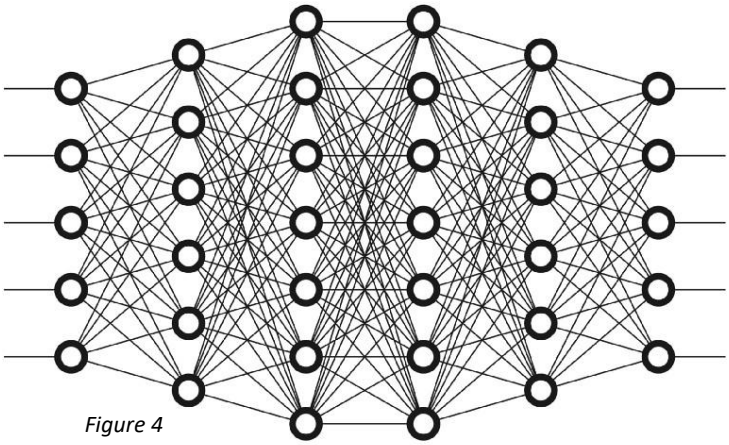
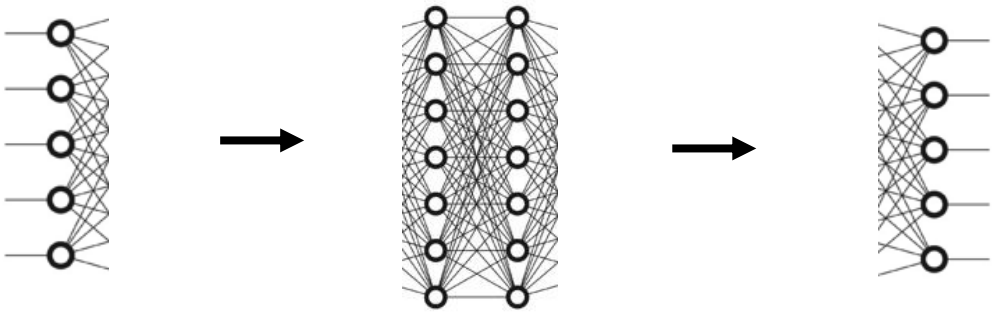


Figure 4

Each line in the diagram represents a stored value. The lines represent 'weights' and each node (circle) has a 'bias'. We take inputs and use these weights and biases along with an activation function to produce an output. *Figure 4* can be decomposed…

This is layer 1, the input layer These are the hidden layers This is layer 6, the output layer



Mathematical Processes

The diagram consists of vertical layers each consisting of a number of nodes, each node will provide an output from a neural network function. This neural network function requires established bias and weight values, so to begin with we randomise these values. One weight connecting each node to every node on the next layer, and 1 bias value per node. Each node on the first layer represents an input variable from a single data entry in the dataset. For example, two inputs of height and width would require two nodes on the first layer, and the output of each node for that layer is the inputs from the data set. In the forward propagation process, we are aiming to get an output for each node by taking the weighted sum of the previous nodes' connection to that particular node, then add a bias and run that value through an activation function.

Layer 1 was the inputs; we move to layer 2. For each node in layer 2 we perform the forward propagation neural network function where the output for each node is found by multiplying the output of each node from layer 1 by the connecting weight, adding these values together with the bias, and then running it through the activation function. The basic process carried out by a node can be represented in 4 simple steps:

1. Take the input and multiply it by the node's weight.
2. Add the node's bias.
3. Feed the result, x, into the activation function, f(x).
4. Take the output and transmit it to then next layer of nodes

Or Node Output = σ ((Weight1) (Input1) + (Weight2) (Input2) +… Bias)

These steps are the fundamentals of the forward propagation process and can be decomposed into a more efficient theoretical mathematical algorithm using matrices. By configuring the weights as a 2-dimensional array we can have x and y correspond to a previous input and neuron in the current layer it connects to respectively. This way when we split the input data into batches, we can transpose the batches array, so both the weights and batches array have an alternate axis corresponding to the neuron. In doing this, matrix multiplication will be possible between the two where, due to how matrix multiplication works, a new array of different parameters will be declared where the values are equal to the weighted sum of previous inputs for that neuron. This new array will have x corresponding to the batch number (out of 20) and y corresponding to the neuron number (depending on layer 4 or 3). This is a short overview of the operations of the program, later in the design process I'll explain the choice to implement batches, matrix multiplication and how object-oriented programming enabled this.

Potential Activation function

My program will utilise a non-linear activation function as they address the problems of a linear activation function. This will allow the use of backpropagation in order to correct weights and biases as well as 'stacking' of hidden node layers allowing for improved accuracy.

There are many common nonlinear activation functions to choose when applied in machine learning, each with individual advantages and disadvantages. In order to ensure my program is efficient I should choose a function that has appropriate advantages to what my program will achieve.

Sigmoid / Logistic

Advantages will include a smooth gradient and output values being bound for any value of x, 1 > f(x) > 0. This is due to the horizontal asymptotes of the function. Due to these asymptotes, predictions are clearer. Sigmoid became a standard despite not being the objectively greatest and is now the most widely used activation function, meaning there are more online resources for its application as opposed to its successors.

The main disadvantage is that for extreme values of x, there is no change to the prediction which can result in the network refusing to learn further. This is not an issue considering the scaled range of my inputs.

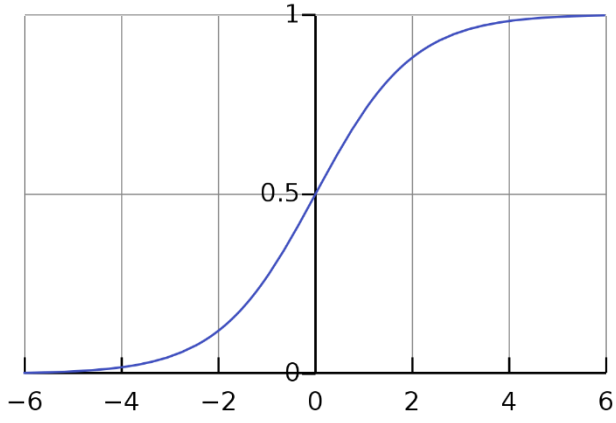


Figure 5 Source: https://en.wikipedia.org/wiki/Sigmoid_function

TanH / Hyperbolic Tangent

TanH is shackled by the same downsides and benefits as sigmoid, however offering a function centred on zero, making it easier to model extreme inputs. There is less support for TanH than its predecessor the sigmoid.

ReLU (Rectified Linear Unit)

ReLU is individual to itself when contrasted with the previous 2 functions, in that it appears to be linear but offers a derivative function which allows for backpropagation (which is the decided process I will research in order to correct weights and biases). It is also more computationally efficient than the two previous functions. ReLU however has a problem whereas inputs approach zero, the network cannot perform backpropagation.

My program will not be very intensive in order to cater to the simplistic aesthetic a learning tool requires / deserves, thus the performance benefit of both TanH and ReLU are negligible to my decision. Furthermore, ReLU's risk of the gradient reaching zero effectively nullifies it as an option. The additional support available for the use of sigmoid makes development more attractive compared to the use of TanH which's performance is provably better.

Back Propagation

Back propagation is the mathematical process that allows the program to 'learn' by altering the weights and biases used in the network function which eventually determines the outputs of the network. After each batch is forward propagated to run the inputs through the network, you can then implement backpropagation which corrects each weight and bias depending on these outputs. As a learning tool, it will not be completely necessary for the network to fully backpropagate in order to provide accurate outputs, only so much that the user can see how the outputs would change over time. In order to determine how each of the weights and biases need to be altered after each iteration, we calculate a value called 'error'. In order for Backpropagation to be successful, we iterate steps backwards in order to determine how much each weight needs to be altered.

Matrices may be important to the optimisation of this process as well as the core network function with forward propagation as iterating in the manor described later on page until you reach layer 1 would only alter 3 weights. As a result, many calculations will need to be performed; a problem with configuring AI is reducing suffering performance. Storing values as vectors e.g. a 2D array in which number of columns represent layer 3 nodes, and number of rows represent layer 4 nodes.

Additional Research

There are a variety of solutions I can use to meet the objectives of my AI project.

- **A program written in Visual Basic**

Advantages:

- Visual basic is a tool useful for creating clear user interfaces useful for a learning tool. The interface will be created visually allowing for a more intuitive user experience ideal for meeting the primary objectives of the end user.
- Multiple input variables for the user to manipulate in order to observe how the training process is affected in real time will be made more efficient via scaled use of features such as text boxes and buttons

Disadvantages:

- The programming language is not suited for the ideas and concepts for a typical AI solution due to its high-level nature
- The solution will be designed offline meaning it'll be compiled as a distributable program

- **A program written in Python**

Advantages:

- Python is an intuitive language with a low entry barrier due to its simple syntax allowing comfortable work with complex systems i.e. matrices calculations.
- The programming language is suited for the ideas and concepts for a typical AI solution, habitually used for AI projects

Disadvantages:

- Python is a slower programming language than C or C++
- Deep neural networks require a lot of memory. Python is not a choice for memory intensive tasks, this is not ideal as I am limited to 6 GB of RAM. Due to the flexibility of Python's data types, Python's memory consumption is also high.

- **A web interface using HTML**

Advantages:

- The solution can be opened and viewed on every internet browser allowing for ease of access to educative resources
- HTML is an easy to learn and use language
- The solution can be accessed online and remotely for wider access of educative resources

Disadvantages:

- Coding in HTML is large and produces complexity. This is not ideal as an AI solution is best optimised when there is the most reduced complexity as possible to achieve an efficient solution
- The program would not include a clear diagram of the neural network structure making it a less intuitive learning tool for the user

Coding techniques

Object Oriented Programming - Classes and Objects

Abstraction is one of the fundamental principles of object-oriented programming; abstraction means to simplify reality. Through the use of class-oriented programming, I can take a real-world object and abstract certain information in order to simplify a problem and create an object. This can be applied to the iris flower since the neural network only requires 4 specific inputs: petal length, petal width, sepal length, sepal width.

```
Public Class Iris_Flower
    Public Petal_Length As String
    Public Petal_Width As String
    Public Sepal_Length As String
    Public Sepal_Width As String

    Public Sub Neural_Network()
        ' Carry out process
    End Sub
End Class
```

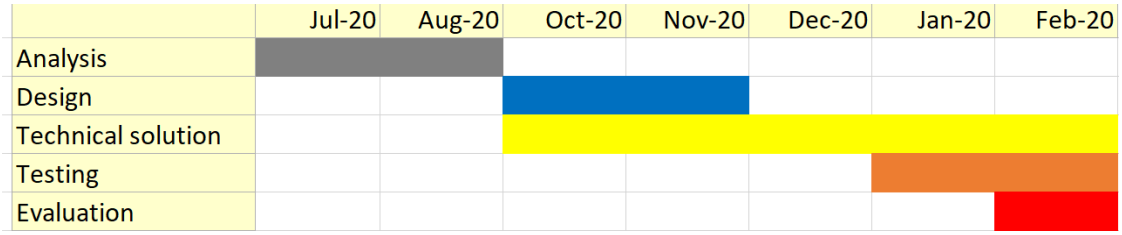
This is an example of how the Iris Flower can be created as a class and then later referenced as individual objects. With this class declared, I can use Iris flower as a variable type in order to create different instances of objects.

Matrices calculations

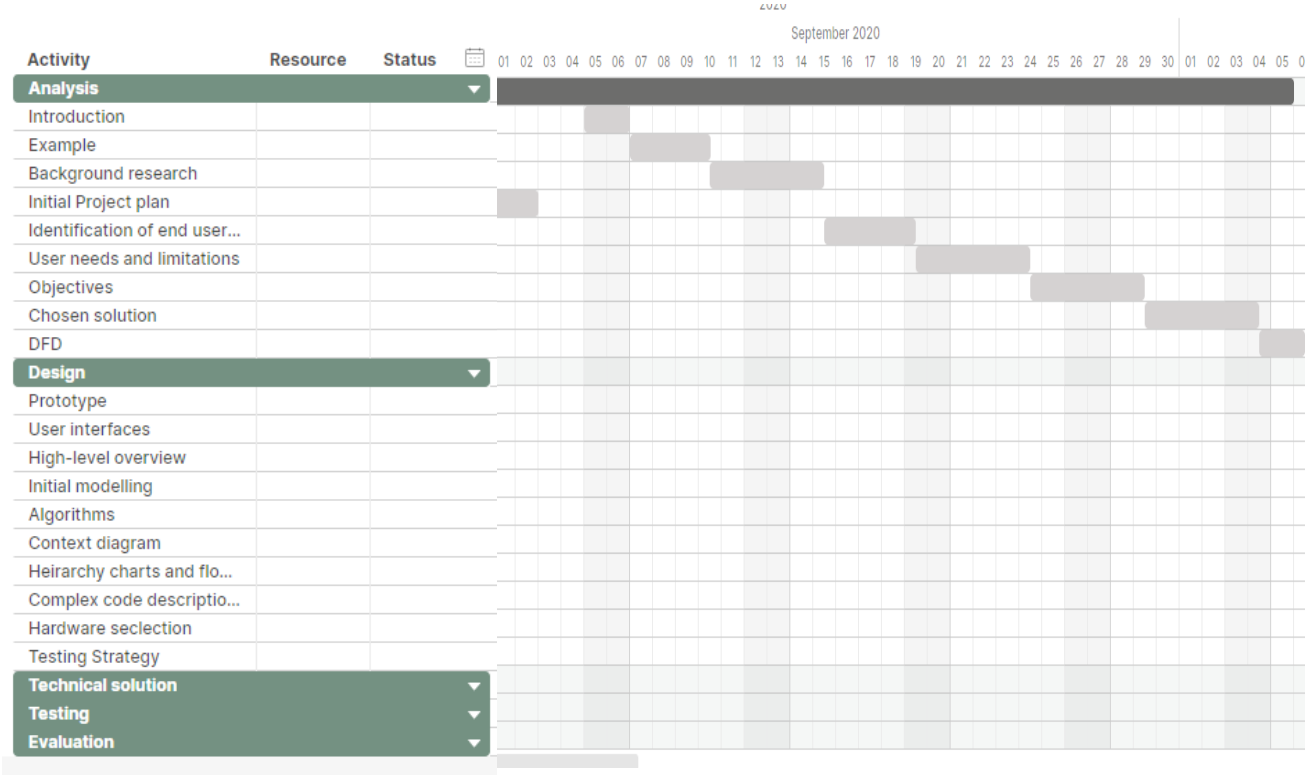
The idea of Matrices is a complex mathematical theory, however it can be applied to the calculations being carried out in my solution in order to improve performance. It will involve using a Multidimensional array, in which columns and rows will represent different variables in the calculation. E.g. during forward propagation training many variables need to be co-ordinated so that the correct values are applied with the correct processes. These calculations have to be iterated meaning for efficient use of loops to carry out the large equations efficiently, matrices will be a useful tool. In principle, Matrices which are well optimised will improve performance.

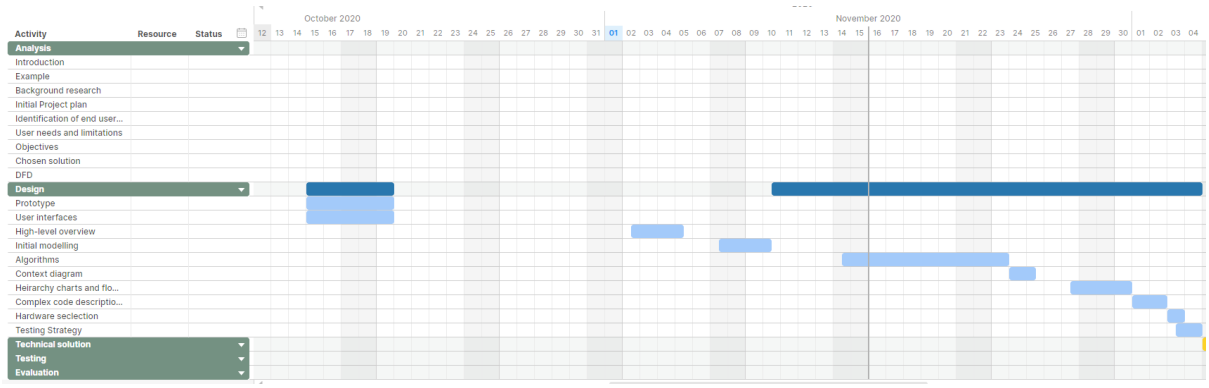
Project plan

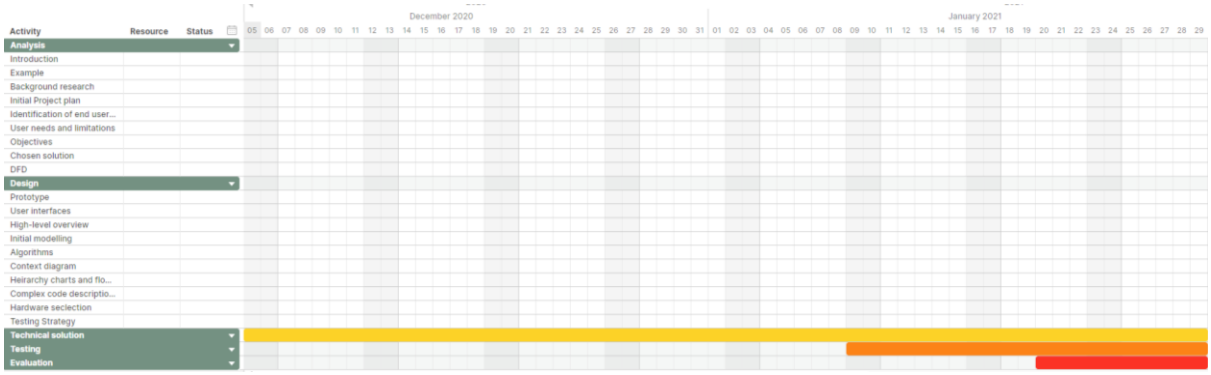
The initial project plan can be seen below and was constructed before the analysis section was started, later in the development process, an agreement was met on a progressed version of this project plan.



This initial project plan was very simple, and after agreeing with a suitable end user it was developed much further and can be seen below across multiple images. In order to gain an understanding between myself and the end user I developed an initial user interface prototype which can be seen in the design section, it allowed for an understanding of the way the solution will be mapped. This contact involved delaying production due to world circumstances and an understanding that lead to an agreement where production may be revisited during the technical solution stage naturally as part of an iterative design process. As a result this plan is simply a guideline and not strict parameters for when the project may be developed.







The schedule factors in a large research period as neural networks are a complex topic and require a deep understanding of the field of AI which I will need to begin with very little prefacing knowledge. Due to this reason a greater amount of time is allocated to the analysis section and will inevitably include early implementation of a programming structure

Identification of end user(s)

Target Audience

I conducted an electronic survey on a popular artificial intelligence forum (*https://www.reddit.com/r/artificial/*) in order to get an idea for the demographic of people interested in artificial intelligence. This allows me to not only pursue an accurate interview with a potential user but also get an idea of the background the user will have in computer science and AI. The survey concluded that people interested in AI are majority males with varying ages from university to occupation. A second conclusion my survey drew was that most of these potential users already had a background in computer science. This means my tool will be best optimised with a bias towards a pre assumed understanding of the more basic principles behind the operation of a neural network.

With this information I was able to follow up on these conclusions by performing a further interview in order to gain a deep understanding for the requirements of the problem. I contacted a student aiming to study AI at university, who is currently researching AI and neural networks (*Figure 6*):

Q1: What software and resources have you been using to learn about neural networks?	I have been using many YouTube videos from different creators, along with Pycharm which is an IDE for python.
Q2: What are some good and bad aspects of the resources?	When you're watching online resources it is difficult to find a video tailor to your specific needs, this makes application of theory more difficult. A good aspect however is the quantity of resources available for free. Pycharm is not really suited to AI, which further heightens the difficulty of tailoring a program to your specific needs.
Q3: How would you improve those resources?	Due to the amount of content on the platform, it is not always an efficient learning process. I would prefer if the implementation of the concepts that are being taught were more available. I would've used a different IDE, one that is suitable to AI.
Q4: Is there any specific part of learning neural networks you find you struggle with?	Working with matrices when manipulating the forward-propagation process is something that is difficult to branch the theory to implementing it into a programmable tool.

Figure 5

Results

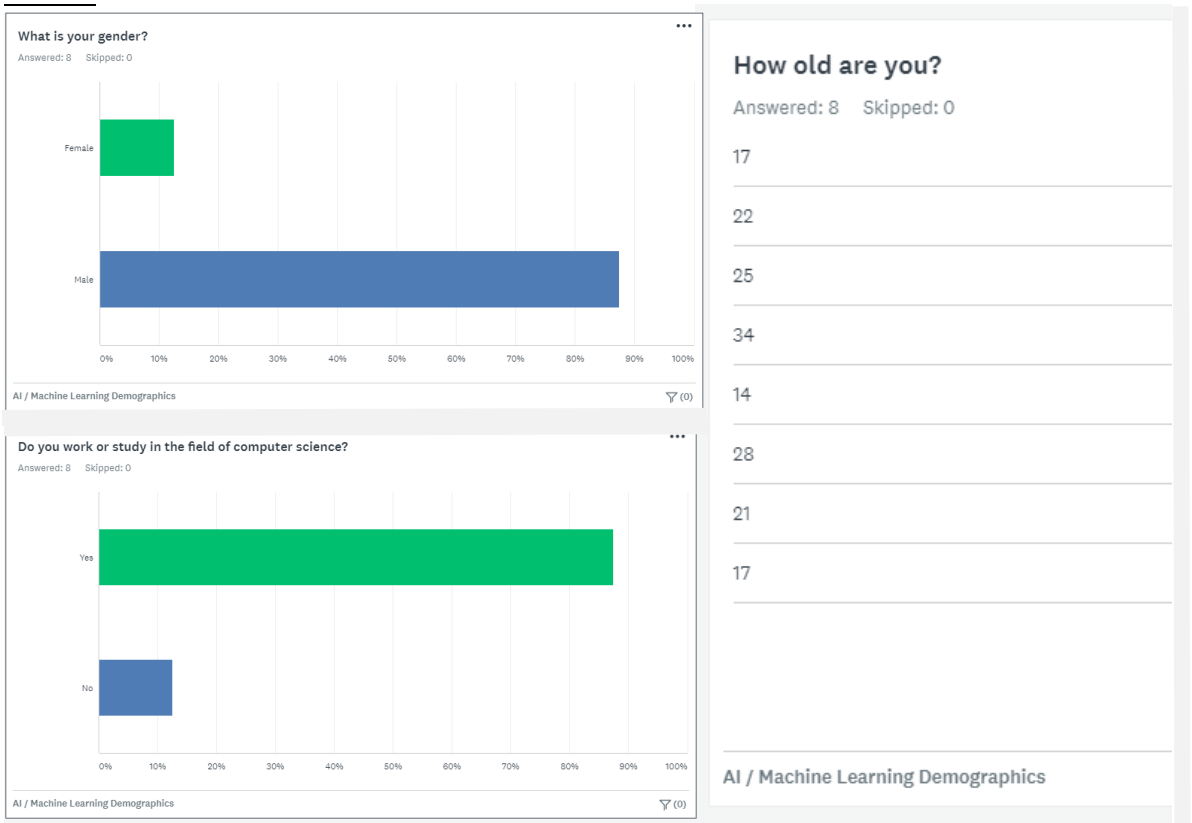


Figure 7

The target audience for this tool is not limited by typical demographic such as age and gender as it should be accessible to anyone interested in learning the basics of machine learning and neural networks. A crucial piece of data collected from the survey however, identified that those who will be using my tool will have an extensive background in computer science and an anterior understanding of neural network theory.

The interview in *Figure 6* was conducted in order to allow for a greater idea of the user's priority needs as opposed to unnecessary features.

Potential Users

The target audience for the user group of this software will be both individuals in AI education and those who wish to utilise the program as a tool for identifying the species of an iris flower. Due to this duality of functionality, potential users of the software will be:

- **Students** – Anyone involved in the learning process behind applying neural network theory to a functioning program. This is the main user as independent study is common amongst this demographic and the project primarily lends itself as a tool for study and reference.
- **Employees / individuals at informal Educational institutions –** Tutors, staff and parents alike would be able oversee a student's progress in learning more effectively by utilising this application behind neural network theory.
- **Gardeners / Iris breeders –** Whilst the project serves primarily as a learning tool, it's secondarily intended to be a functioning neural network that will be able to identify the species of an iris flower given the correct parameters.

Required Skills

The project is somewhat complex and in turn fairly demanding in terms of pre assumed knowledge of the user. The main differential in my target demographic to the general population is skill level.

Basic computer skills will be required in order to initialise a neural network. The user should be familiar with the windows operating system interface (i.e. managing files and appropriate file types) in order to effectively utilise the upload feature allowing users to author their own dataset when initialising the training process.

On top of this, the survey prior in *Figure 7* identified a crucial piece of data. It collected from the survey, that those who will be using my tool will have an extensive background in computer science and an anterior understanding of neural network theory. The program would be best optimised with a bias towards a pre assumed understanding of the more basic principles behind the operation of a neural network. This means an understanding of:

- Coding and programming theory
- Basic Neural network theory
- Forward backpropagation theory
- Extensive background in mathematics *Not required but useful

User needs and limitations

From all this collected data and supplemental analysis, a series of user requirements and limitation can be culminated. Firstly, an important declaration is that the user will be one studying the application of neural network and backpropagation theory. With this information we can identify key areas of development, the program must be:

- **Easy to use** – Intuitive user interfaces in order to simplify the operation of the tool as a neural network tool. Simple, clear notation with minimal buttons on screen required to iterate forward and train the network. Ideally the program will only require a single windows form.
- **Efficient –** In order to allow multiple iterations, the training process must be efficient. This allows the user for a greater manipulation of the program itself and in turn can aid the development of a better understanding. This can be done via the use of different mathematical techniques outlined previously (i.e., matrices)
- **Small –** School computers have limited training resources for the network and must be appropriate in order to allow for efficiency. This is done by using a small, normalised architecture which also further simplifies the users learning experience when applying certain concepts.
- **Reset –** Once the network has been trained it must be reset each time it is run. This allows the user to not only revisit the software as a way to solidify understanding but allows the user to observe the training process multiple times, which is the important part of applying neural network theory to an actual program.
- **Specialised –** The software must be a tool suited only to AI; application of backpropagation is simplified when the program is tailored to your needs.

It is not necessary for the program to include:

- **A Large Architecture** – The principles of backpropagation do not change as you increase the number of hidden layers and nodes. For this reason a small architecture is ideal for simplicity of calculations.
- **Network customisation –** Users will not have administration over network architecture i.e. number of nodes in each hidden layer and the number of hidden layers. The program serves as a demonstrative tool of what a neural network could look like.

System requirements

Machine learning is a promising field with new research papers being published daily, where today's AI can require a lot of resources in order to both train a network and produce accurate results. A small industrial grade project don't need a big system and won't require a GPU. A good CPU for this scale project could be an i7-7500U which would train an average of ~115 samples/second. My project utilises the Iris data set which contains 150 samples meaning a powerful CPU is not necessary. Any CPU that can process ~20 samples/second should be able to run the program with minimal optimisation problems. See the evaluation section on page _ for specific hardware requirements as a reflection of the final solution's use of computer resources.

Response to prototype

Before outlining objectives, it's important to have a good understanding of what the user wants. I was able to get feedback from a potential end user on an early prototype of the UI.

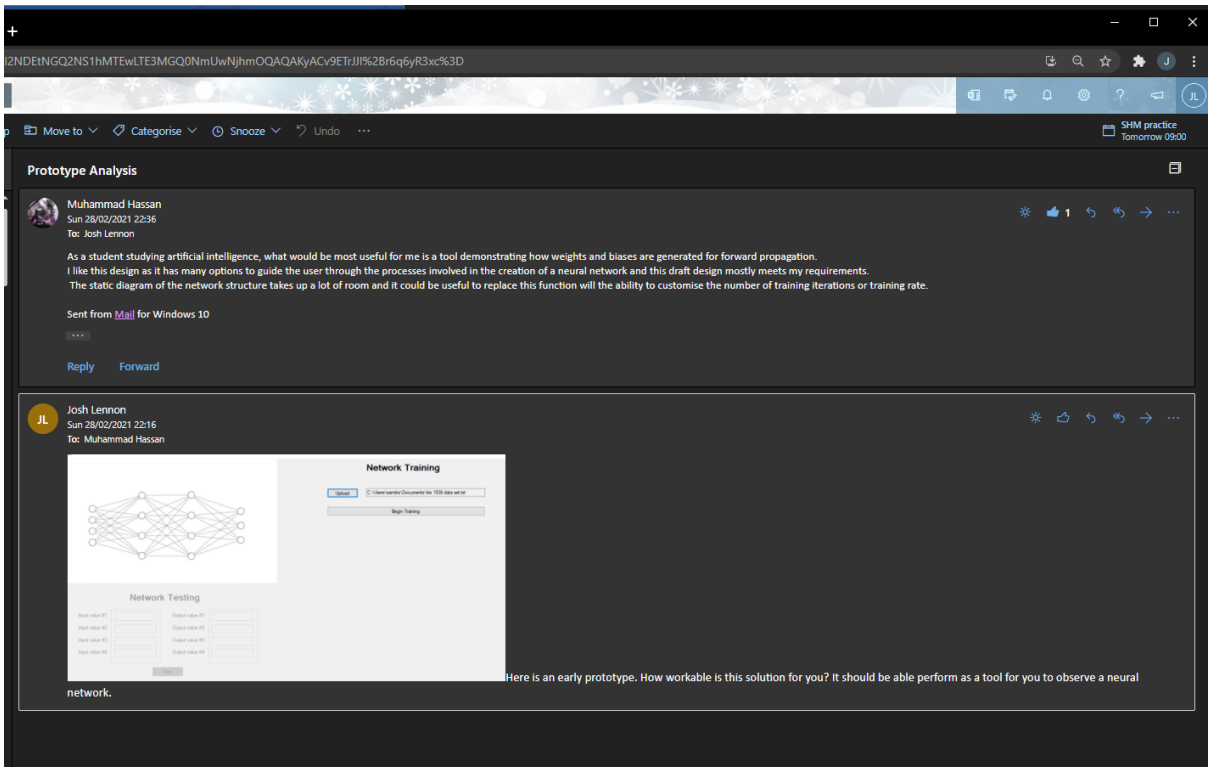
'*As a student studying artificial intelligence, what would be most useful for me is a tool demonstrating how weights and biases are generated for forward propagation.*

I like this design as it has many options to guide the user through the processes involved in the creation of a neural network and this draft design mostly meets my requirements.

The static diagram of the network structure takes up a lot of room and it could be useful to replace this function will the ability to customise the number of training iterations or training rate.'

This feedback allows straightforward defining of objectives: The solution allows users to follow the processes involved in the creation of a neural network where there is clear notation outlining the order of actions the program is executing. There will be a feed printing weights and biases for each network generated (only one can be generated at a time).

Importantly I also changed my approach to the solution after receiving this feedback. The main way the objectives of this solution were altered is the decision to relegate the use of backpropagation to a secondary objective as this is not useful for enhancing understanding. Beyond this change in plan, I decided to remove the static diagram of the network's structure and implement a textbox that allows the user to specify a custom number of times the network should iterate when training.



Objectives

This project has two overarching objectives; Primary objectives represent goals of designing the project as a learning tool, Secondary objectives represent goals of designing the project as a machine learning process that 'learns' to identify the species of an iris flower from set input parameters.

Primary objectives

1.1. The solution allows users to follow the processes involved in the creation of a neural network as a learning tool where there is clear notation outlining the order of actions the program is executing. There will be a feed printing weights and biases for each network generated (only one can be generated at a time).

1.2. The solution will be available for testing both before and after the network has completed training. The user can test the network with 4 individual inputs that the network has not seen previously, responding with 3 outputs via 3 individual textboxes. This includes a message box that outputs the species the network predicts with the highest confidence.

1.3. There will be an upload feature where the user will be able to select a text file before training of the network begins. The text file will be read from and utilised as the dataset for network training. A useful visualisation of the network architecture will be displayed to aid the user in selecting an appropriate dataset. While the network is training, the progress will be displayed via corresponding inputs and outputs being displayed in the textboxes on screen.

1.4. When a dataset is uploaded by the user, the 'batches' training method will be used (see) where 20 random lines will be read from the dataset text file in order to be used in forward propagation and saved as an array where each input from a line is split into 1 of 4 cells.

1.5. Forward propagation must take place in order to generate outputs (see) where this process can be iterated a user specified number of times

1.6. The user should be able to figure out how to use the program in under a minute and interpret the program with intuition given a background in both computer science and the theory behind how neural networks operate.

Secondary objectives

The secondary objectives for my program outline optional features that go beyond the demands of A-Level but are required for a fully functioning AI as opposed to a learning tool.

2.1. The program will initialise a network with random weights and biases and conclude with a network that gives accurate outputs. The inputs will be fed through the network and return several values for each node. The values produced in forward propagation will then be utilised in the backpropagation process to successfully alter the weight and bias of each node.

2.2. The program will be able to identify the species of iris flower accurately from 4 inputs when tested. In theory, upon initialisation the program should be able to train a network with any dataset that matches the chosen architecture. This means it can be applied in other contexts to identifying species of iris flower.

2.3. To ensure good performance, the feed outputting each weight and bias for each node will be updated via a 'refresh' button available to the user. The solution will also utilise matrices multiplication in frequent when propagating forward. This means as weights and bias are progressively altered in the training process, the change in each will only be displayed at the user's request.

Chosen solution

Language / medium solutions

The solution I will use for determining a medium / language is a tool useful for creating clear user interfaces useful for a learning tool. The project will be a program written as a visual basic windows form. This is because I would be able to write the program and interface according to my individually determined primary objectives (the user should be able to use and interpret the program with intuition **[Objective 1.1]**). The interface will be created visually allowing for a more intuitive user experience ideal for meeting the primary objectives of the end user **[Objective 1.6]**. Multiple input variables for the user to manipulate in order to observe how the training process is affected in real time will be made more efficient via scaled use of features such as text boxes and buttons. Both the programming language and its IDE provide the opportunity for notation concepts to be displayed intuitively.

Activation function solutions

I identified prior that my program will utilise a non-linear activation function due to their addressing of the problems that a linear activation function produces. It allows the use of backpropagation in order to correct weights and biases as well as 'stacking' of hidden node layers allowing for improved accuracy. This mean that when selecting an activation function, one that consistently supports use of the backpropagation process is ideal **[Objectives 2.1 and 2.2].** There are many common nonlinear activation functions to choose when applied in machine learning, each with individual advantages and disadvantages. After analysing these details, ReLU, despite being the most efficient and effective activation function was ruled out as an option for the solution. As inputs approach zero the network cannot perform backpropagation which is crucial. TanH and Sigmoid are virtually identical functions, however the availability of supporting resources for developing AI with sigmoid are overwhelming in comparison to the use of TanH. For these reasons my AI solution will utilise the sigmoid activation function.

User solutions

This project is a specialised AI tool, however, it is somewhat versatile in its application. After evaluation of potential users in different settings, two primary categories can be established. Use of the program as a teaching device in an educational setting of how an AI tool may operate **[Primary Objectives]** and use of the project as the tool itself **[Secondary Objectives]**. Both are appropriate users and in order to reach a solution for the end user, each category was separated between primary objectives and secondary objectives. This ensures the project meets its objective as an educational tool whilst simultaneously meeting the needs of a wider audience.

Flowcharts

Initial variables – Data flow diagram

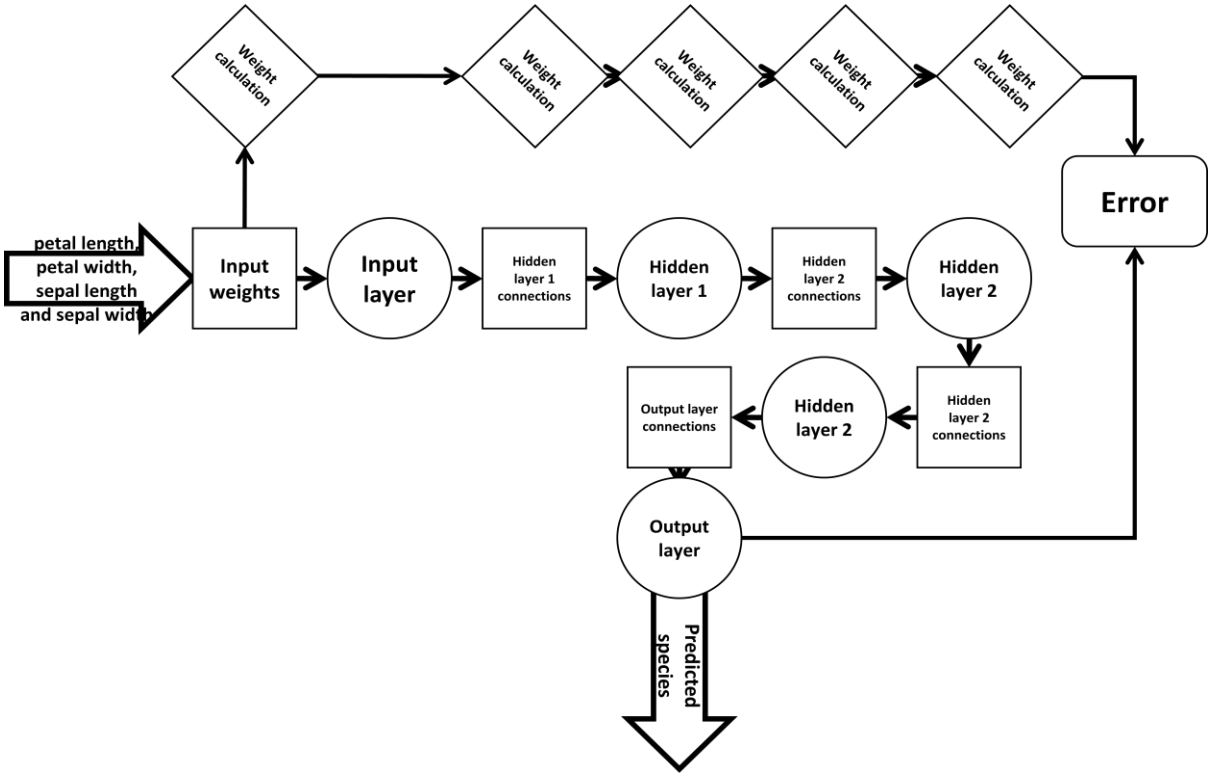


Figure 8

Design

High-level overview

This program will be a machine learning AI that operates through a neural network structure of 2 hidden layers containing 4 neurons each with 4 inputs and 3 outputs. It will utilise weights and biases across the network to create a network function that can be iterated forward through different inputs and provide outputs the network will 'learn' by propagating backwards and comparing the outputs to projected outputs and calculating an error. This error is used to alter the entirety of the weights and biases of the network **[Objective 2.1]**, a process which is repeated until the network is fully trained. At this point the network function should give an accurate output based on the training data.

The tool will provide an interface for students to interact with a functional neural network in a way that effectively exposes the variance across weights and biases as the program operates. It will demonstrate how an AI is able to learn through training using the iris dataset. Once the network is trained it will allow the user to test by providing 4 inputs and returning 3 float outputs. Each output is representative in the computer's confidence that the input parameters correspond to that species of iris flower.

Input/process/storage/output (IPSO) table

Input	**Explanation**
User selected data set	An option will be provided for the user to allow them to upload a text file in order to be used in the network training process. **[Objective 1.3]**
Menu buttons	The Buttons on the main form will be used by players to control the processes of the network. When a button is clicked using the mouse, the program will perform an action based on the button clicked.
Testing data	An option will be provided for the user to type custom values for each parameter in order to be used in the network testing process. This will operate together with the menu buttons in order to iterate the network. **[Objective 1.2]**.
Process	**Explanation**
Generate network	When the user clicks a button denoting 'GENERATE NETWORK', 3 object instances will be created representing a layer in the network. For each layer, a weights and biases array will be created and stored with randomised float values.
Network function (forward propagation)	The matrix containing weights connecting that layer to the previous is multiplied via the matrices multiplication process with the matrix containing outputs from the layer before. This will produce an array where each cell contains the weighted sum of the previous outputs called 'values. These values can then be optionally procedurally used in backpropagation if I decide it is necessary. Before storing this array however, each cell has the bias of that neuron added to its current value. This process is repeated for each layer object instigated. **[Objective 1.5]**.
Network test function	After or before training the network, it will iterate forward using only 1 set of values when testing as opposed to batches of 20 in the training process. A different process is used to training for this reason involving using the values stored in the weights and biases array at that current point in the program's operation. It should return 3 confidence outputs from 4 inputs. **[Objective 1.2]**.
Interpreting text file	When the user inputs the iris dataset, it will be 150 lines in length. This means that in order to properly optimise the training process, the input data will be split into randomised batches of 20. This is done by reading random lines in the text file 20 times and using the mid function in order to store the values and output as the correct variable from each line. **[Objective 1.4]**.

Transposing Matrices	Sometimes in order to allow two matrices to be multiplied you will need to transpose one of them. This means to swap the columns and rows. This can be done with a simple nested for loop where (x, y) becomes (y, x).
Multiply Matrices	I will make a separate prototype in order to construct the matrices multiplication process since it is core to the solution. It will be done by taking each cell and applying a mathematical process to that of the other matrix.
Sigmoid	The sigmoid squishification activation function is exactly what it's described as: a function that maps inputs to a number between -1 and 1. It will be applied to the values after biases produced in forward propagation.
Bubbles sort algorithm	A bubble sort algorithm will be implemented to return which species of iris flower the program predicts based on which has the highest confidence return output value for each time it's tested. **[Objective 1.2]**.
Storage	**Explanation**
Weights	Will store the weights connecting each neuron for each layer as an object where the rows represent each neuron on the previous layer and columns represent each neuron on the current layer.
Biases	Will store the biases for each neuron for each layer as an object in a single dimensional array.
Batches	Used to store the 4 values for each line of the text file in a 2d array where the rows represent the corresponding set of inputs and the columns represent. **[Objective 1.4]**.
Values	Used to store the result from the forward propagation process from multiplying the weights and batches matrices.
Testing Outputs	The outputs from the network test process are stored in a 2-dimensional array for each layer as an object.
Training Outputs	The outputs from the network training process are stored in a 2-dimensional array for each layer as an object.

Prototype

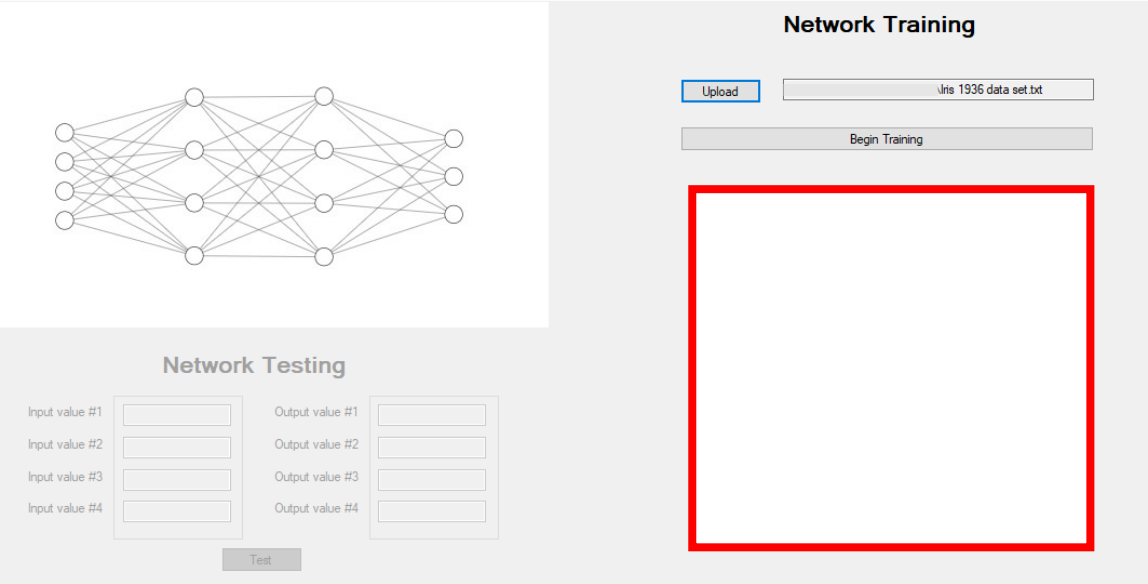


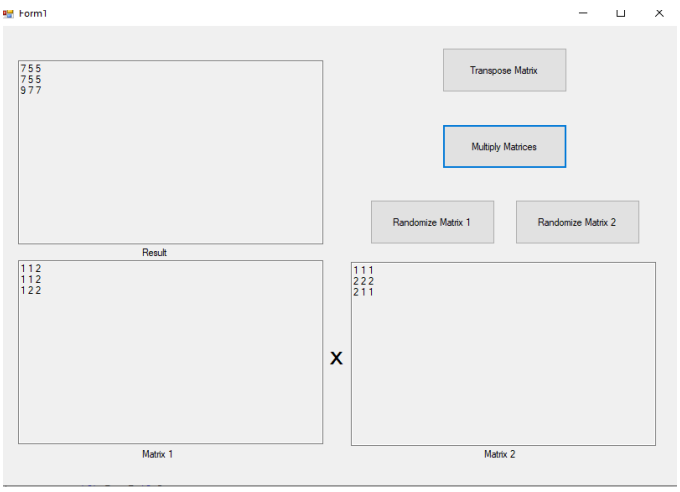
Figure 9

This prototype provided good feedback from a potential end user and was found to be a good draft based on my initial research. The prototype interface features a button that allows the user to upload a text file from their local storage in order to utilise as the dataset for training the network. Once the network is trained there is a testing section which will become enabled. It allows the user to provide inputs and based on the programs training will attempt to provide an accurate output. The inputs and outputs must obviously be of the same parameters.

Matrices prototype program

As part of the iterative design process, I started by making a solution that will both multiply and transpose two matrices. This is a technique that will be useful later in the final solution so can be seen as an initial prototype in the design phase.

This is the user interface. There are 4 buttons; 2 that randomise and display the matrices to be multiplied, 1 to multiply the two matrices and 1 to transpose the result.




```
Sub MultiplyMatrices(ByVal input1
As Array, ByVal input2 As Array)
        Dim sum As Integer
        For v = 1 To 3
            For z = 1 To 3
                sum = 0
                For j = 1 To 3
                    sum = sum +
mat1(v, j) * mat2(j, z)
                Next
                mat3(v, z) = sum

            Next
        Next
 End Sub
```

Figure 10

The code behind the multiplication mathematical process is contained within the subroutine 'MultiplyMatrices'. This subroutine follows an iterative procedure and instead of returning a value, makes the corresponding value in a third matrix equal to the sun.

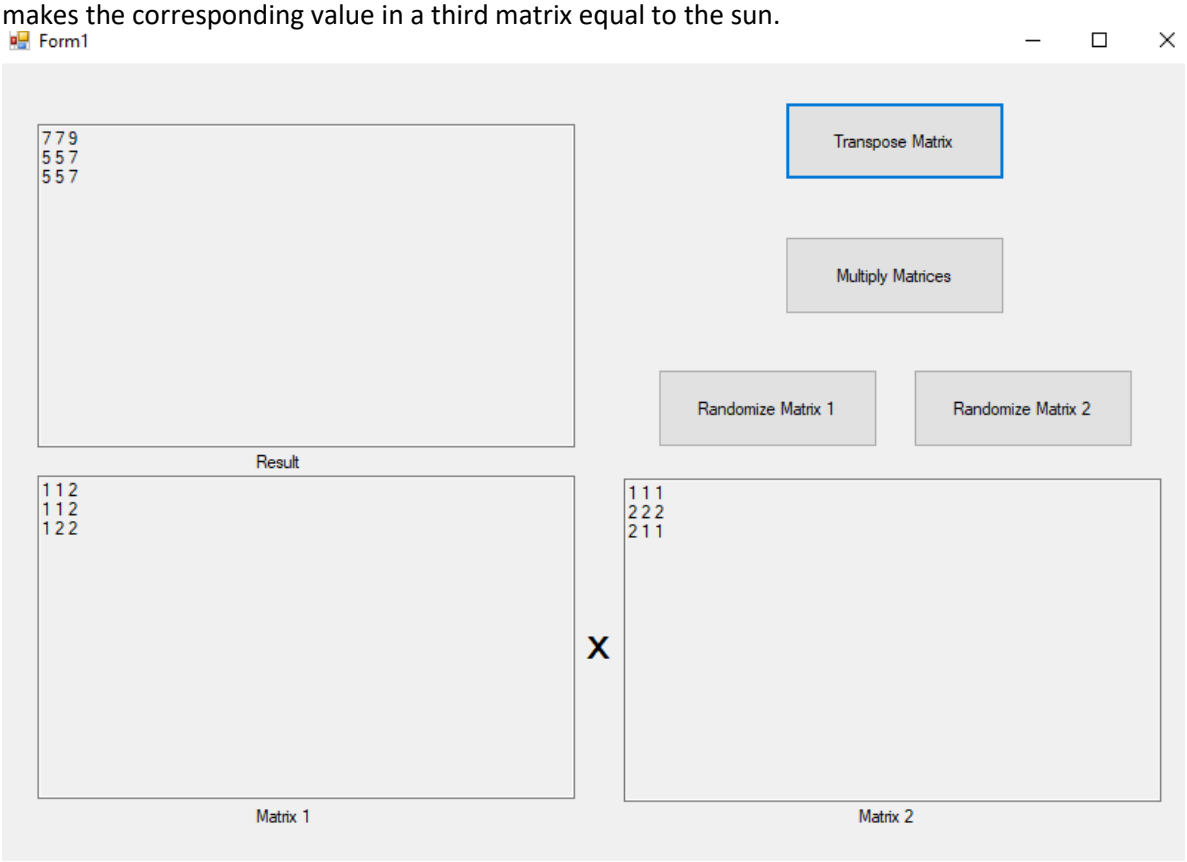


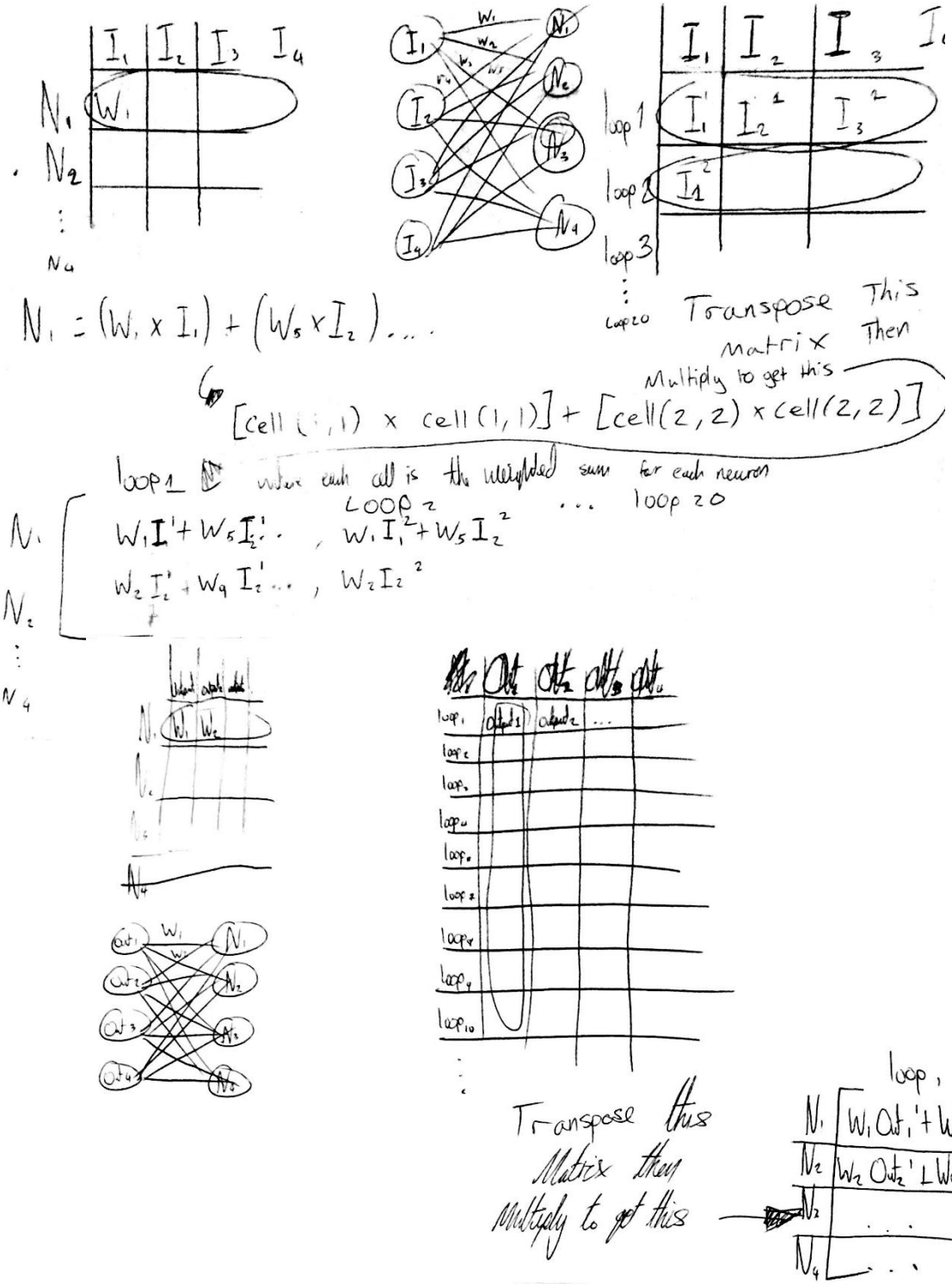
Figure 11

```
Private Sub Button1_Click(sender As System.Object, e As System.EventArgs) Handles
Button1.Click
        Matrix3.Text = Nothing
        If transposed = False Then
            For i = 1 To 3
                For j = 1 To 3
                    mat4(i, j) = mat3(j, i)
                Next
            Next
            PrintMatrix(Matrix3, mat4)
            transposed = True
        ElseIf transposed = True Then
            For i = 1 To 3
                For j = 1 To 3
                    mat3(i, j) = mat4(j, i)
                Next
            Next
            PrintMatrix(Matrix3, mat3)
            transposed = False
        End If
    End Sub
End Class
```

The code behind the transposing process is called upon immediately when the 'Transpose Matrix' button is pressed. This subroutine follows an iterative procedure and works differently depending on the current state the result is in. A boolean value denoting the state the result is in determines which direction the code is to be executed.

Early drafts / drafts I used whilst making the software

Below are my rough theory sketches that I used whilst designing / making the software. They include diagrams of the matrices I will need to multiply and conceptual designs of the desired output. This allowed me to determine whether or not I needed to transpose a matrix and if so when. I used a network structure diagram to determine the array boundaries and ensure the correct weights are being multiplied / summed with the correct values. This was the main area where I gained understanding and meant that I decided exactly how I wanted to develop this solution.



Initial modelling

A forefront issue with neural networks is the large quantities of data management. To tackle this my solution will have two classes declared in order to allow the program to utilise object-oriented programming to store variables. 3 objects will be declared as members of this class each representing a layer of the neural network. This way each of these layers can be assigned different stores of the same variables.

Initial variables & pseudocode

Records & Arrays

Storing values as vectors can reduce the number of mathematical operations that are needed to be carried out and eventually better optimise the programs performance when these functions are iterated. A 2D array in which number of columns represent layer 3 nodes, and number of rows represent layer 4 nodes is the best application of this concept in order to support back propagation. It allows for a highly structured system of arrays that can be used to effectively store values needed for different functions. The implementation and breakdown of this array dependant coding structure is broken down in the pseudocode section, however a diagram representation of how this concept can be used to store the weights of different layers is shown below.

	Previous Input 1	Previous Input 2	Previous Input 3	Previous Input 4
Neuron 1	Weight 1 , 1	Weight 2 , 1	Weight 3 , 1	Weight 4 , 1
Neuron 2	Weight 1 , 2	Weight 2 , 2	Weight 3 , 2	Weight 4 , 2
Neuron 3	Weight 1 , 3	Weight 2 , 3	Weight 3 , 3	Weight 4 , 3

The location of values stored in each cell correspond to individual neurons and inputs from previous neurons across each axis. These cells would then be able to contain the weights connecting those nodes allowing for ease of calculation and faster performance.

Pseudocode

Within the declared neural network class, the layers will need their own individual weights, biases, outputs and values. These each need to be their own array and so the program should look like this:

```
Public Class NeuralNetwork
Private Values As Array
Private Outputs As Array
Private Biases As Array
Private Weights As Array
```

Each layer will have a corresponding array for each of these variables that can be referenced in the main class via a public function that returns the specified value:

```
Public Function returnweights(PreviousInputs, neurons)
Return Weights(PreviousInputs, neurons)
End Function
```

I know that an imperative part of the solution's operation will involve a 'network function'. In order to simplify the code production process, I can map out the way this will work in advance by considering the way I've stored the variables aforementioned. Each layer has its own array of each variable and can be called upon individually by referencing the layer and the return sub e.g.

Layer1.returnweights. Developing this logic further allows me to create a complex pseudocode algorithm for how a final network function solution might operate:

```
Subroutine networkfunction

Loop for N = 1 To NumberOfNeurons
Loop for I = 1 To PreviousOutputs

temp1 = temp1 + (layer1.returnoutputs(I) * layer2.returnweights(I, N))

Next
temp1 = temp1 + layer2.returnbiases(N)
TempArray(N) = sigmoid(temp1)
layer2.Generateoutputs(TempArray)
Next

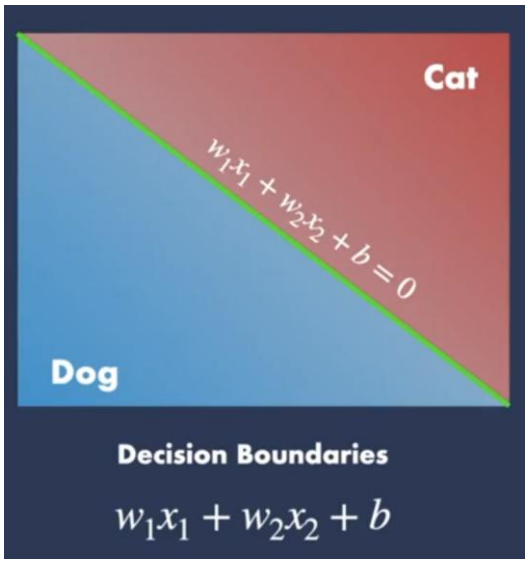

End Subroutine
```

Figure 12

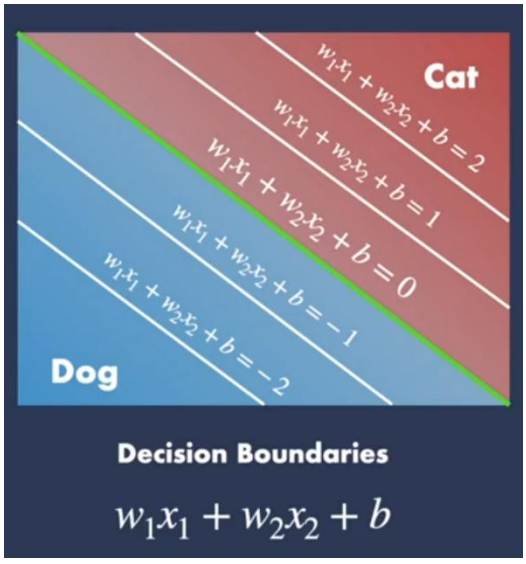
Algorithms

Sigmoid squishification function

The sigmoid function plays an important role in the context of logistic regression. Logistic regression is a technique used to predict the outcome of binary classification problems i.e. playing the role of an activation function in my solution. It is an S shaped curve that can be observed in figure 5 where any value of x in the sigmoid function will output a value between 0 and 1. To show how the sigmoid function is used to make predictions, we can consider the case of a binary classification between a dog and a cat image. In this case, the decision boundary is shown as a green line with the formula that represents the line shown above.



Applying sigmoid to this formula where the formula = 0 will give a prediction of 0.5 as the value represents a 50/50 chance between a cat and a dog picture.



The formula can have many output values representing different boundaries. If the formula were to output -2, the prediction would equal 0.1 as there is a 1/10 chance of the image being a dog.

The network function requires the value to be run through an activation function in order to produce a neuron output from 0 to 1. A non-linear activation function produces more useful results than a linear activation function produces. It allows the use of backpropagation in order to correct weights and biases as well as 'stacking' of hidden node layers allowing for improved accuracy. The function I used is non-linear and is denoted by the symbol σ (sigma) it's laid out mathematically:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad \ldots(1)$$

In order to implement this algorithm, I used an external function:

```
Function sigmoid(x As Double) As Double
    Return 1 / (1 + Math.Exp(-x))
End Function
```

Citation: https://stackoverflow.com/questions/38932637/sigmoid-function-in-vb

This function is only 3 lines of code and allows me to take a value from the network function and return the output of that neuron. This allowed me to implement an iterative testing process where I could insert breakpoints into the code, analyse network output values before the sigmoid function, then compare the programs outputs to a third party's outputs. The third-party version of the function I used can be found at: *https://keisan.casio.com/exec/system/15157249643325*

Network function

So how do we create a neural network? If we have a predefined network structure (see figure 9 or the proposed structure section for more information) we know what parameters should exist. Let's refer to another classic 'hello world' neural network analysis where the inputs are grayscale values of a bitmap image with a hand drawn number. The network's job is to return one of 10 outputs where each output identifies the numeric value of the handwritten digit i.e., the network can identify handwritten digits between 1 and 10. Let's say that we want a single neuron to identify something simple about the problem, for example whether there is an edge in a certain region of the image. We can start constructing network parameters by assigning a weight for every connection to our neuron and every neuron on the layer before.

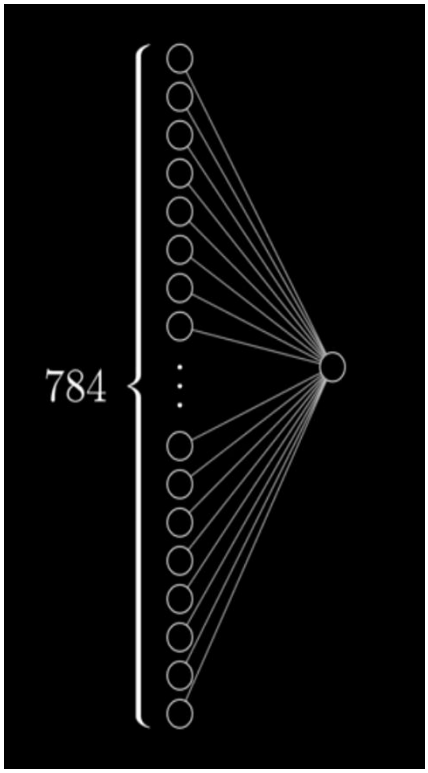


Figure 13 shows the 784 inputs to the network, and our selected neruon from layer1 that we hope may identify an edge in a specific region of the image. Each input would be a greyscale pixel value from the 28 x 28 image. The network structure diagram can be disected in this way: Each circle is a neuron and each line connecting the circles represent weights. Each line will have a value. Initially we randomise these weight values as random numbers. Then we take the inputs or values from the previous neurons and calculate their weighted sum. To that value we add the bias of our neuron (also a randomised value at this point) and run the result through an activation function. This can be represented mathematically as:

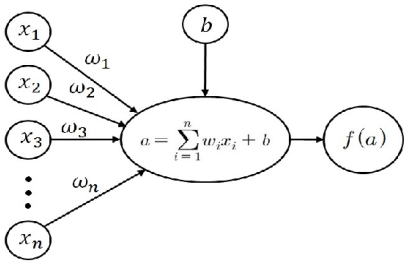


Figure 13

This process is an example of a perceptron which was the first iteration of a neural network created in 1958 Frank Rosenblatt. If we zoom in further to this algorithm we can identify what I have labelled as the 'network function' which is not always how the algorithm is universally referenced in educational settings, however seems intuitive to my explanation of the neural network's operation.

$$f\left(b+\sum_{i=1}^{n} x_i w_i\right)$$

It shows the exact mathematical notation behind my pseudocode algorithm in figure 12 where x is the output of a single neuron in the previous layer and w is the weight connection from that neuron to our own. The symbol capital sigma represents 'Sum of' the weighted values 'x_1w_1' from i=1 to n where n is the number of neurons. So then when we run the result of this network function through the sigmoid function, we can think of the output in this way; how positive the weighted sum or network function is will dictate how meaningfully the neuron is activated in the network. Developing this logic, we can think of the bias as a threshold i.e. only activate this neuron meaningfully when the weighted sum is greater than 10.

This example accounts for the process and function of just 1 neuron in the network. This process is repeated for each neuron in this layer, where every neuron is connected to each of 784 inputs, or in the case of my solution each of 4 inputs. This allows for a simplified proof of concept, where increasing the number of neurons wouldn't change how the network operates. When we look back to our own network it isn't easy to identify what the role of each neuron would be responsible for, however the functionality is no different. Once we have a result for every neuron in the layer, we can move forward to the next layer where the values we calculated from the previous layer are used to calculate the values of the next layer in the same way the inputs were used before. These values become the previous outputs and are used in our network function once more, using them in this way means the activation of previous neurons will affect the activation of neurons in the next layer which is what we would want to observe.

Matrices

I created a matrices prototype program that can multiply and transpose 2 matrix. A matrix is a relatively simple idea: a table of numbers. An array data structure when stored as a table can be used as a matrix. I will be using a 4 by 3 matrix and multiple 4 by 4 matrices. To add matrices, you add the corresponding value from each cell and the result is what is stored in that same cell, the same process is true for subtraction. However, the process is different for matrices multiplication where the process of multiplying the two matrices is noncommutative as opposed to standard multiplication which is commutative.

Matrices multiplication is a more complex mathematical process where in order to find the first term, you multiply the first-row vector by the first column vector of the second matrix then add this value to the multiplication of the second-row vector by the second column vector. Then to get the second value you repeat this but using the second column. If our example is a 2 by 2 matrix then we'll descend to the second row and continue to multiply out by both the first and second column of the 2nd matrix.

Where A is the notation for a matrix, A^T is the transpose of that matrix. So, what does it mean to take a transpose of a matrix? What this means is to manipulate the matrix in a way that the rows become columns and the columns become rows. If a 2 by 3 matrix were transposed it would become a 3 by 2 matrix; you simply write the 3 values in the column horizontally instead of how they were vertically in the original matrix. This process is fairly intuitive to the human brain however in the same way identifying a number from a handwritten bitmap image is intuitive for a human, it is not so simple to

translate into a way that a computer would be able to execute the process. Code evidence from the prototype section above can be referenced for how this process would look in practise.

Back propagation

Intuitive explanation

In order to determine how each of the weights and biases need to be altered after each iteration, we calculate error. In order for Backpropagation to be successful, we iterate steps backwards in order to determine how much each weight needs to be altered.

Target - Output

Layer 4

1. Layer 4 Delta = σ'(Input before sigmoid to node) x (error of node)
2. Calculate error for layer 3 to be used later = layer 4 delta x weight (connecting 1 - 2)
3. We can now change this weight = layer 4 delta * layer 3 output
4. Add this value on to the weight (connecting 1 - 2)

Layer 3

1. Layer 3 Delta = σ'(Input before sigmoid to node) x (error of node) — Layer 3 error calculated earlier
2. Calculate error for layer 2 to be used later = layer 3 delta x weight (connecting 2 - 3)
3. We can now change this weight = Layer 3 Delta x layer 2 output
4. Add this value on to the weight (connecting 2 – 3)

Iterate this process backwards through layers until each weight is altered…

Summed up if you consider how you would correct the output of a final neuron, you'd compare it to the target value and then based on the difference you would make changes to the network. Working backwards from this neurons value, we can see that the components that produce this value are consistent of previous values, the weights connecting those values and the neurons own bias. This is an intuitive formula for implementing back propagation into code, however the maths is very complex and in this section, I will explain the calculus and gradient descent behind how a network learns through back propagation through a network with layers consisting of a single neuron for simplicity.

Calculus

This section looks at the maths behind the cost of single training example when in reality backpropagation takes the cost of every training example into account.

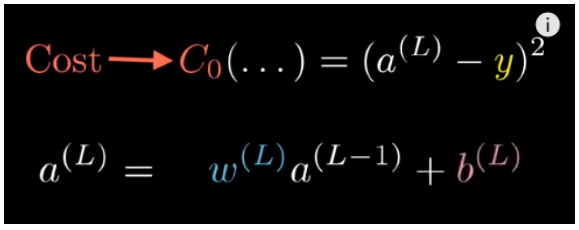


Figure 14

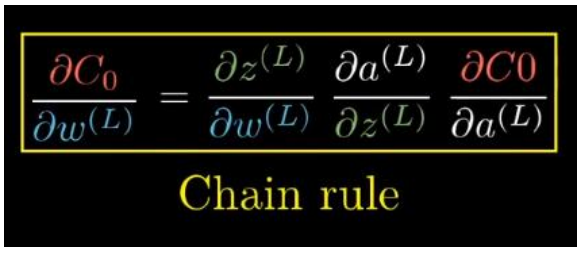


Figure 15

Where a^L is the activation of a neuron on the last layer and y is the desired output, cost can be found and denoted in the following way. In this notation a^L would thus be equal to the weighted sum of the values from the previous layer plus b^L the bias value from the neuron in that layer before finally being processed through the non-linear function sigmoid σ. In order to reduce notation clutter, the network function before sigmoid is denoted by z^L. Therefore $a^L = \sigma(z^L)$. Following this logic, the same values for the previous layer include the same base notation with superscripts L-1. With this notation we can jump to the chain

rule which states that the ratio of the derivative of cost with respect to w^L = the following derivate of z^L with respect to w^L and so on as shown in figure 15. Multiplying the three ratios together in the chain rule gives us the sensitivity of cost to changes in that weight.

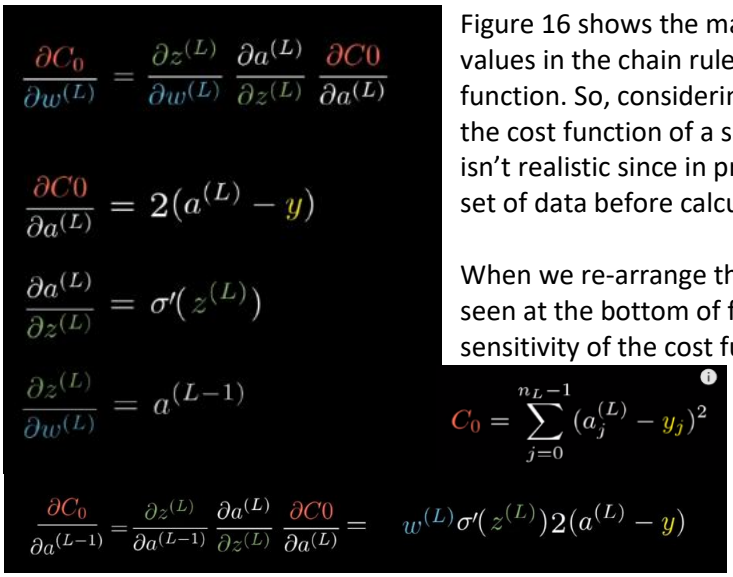
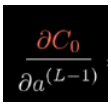


Figure 16

Figure 16 shows the mathematical definitions of each of the values in the chain rule. σ '(z^L) is the derivative of the sigmoid function. So, considering these values allows us to calculate the cost function of a specific single training example, which isn't realistic since in practice the program analyses the entire set of data before calculating a cost function for that neuron.

When we re-arrange this formula, we can get the formula seen at the bottom of figure 16 where we can then find the sensitivity of the cost function to the activation of the previous layer. This is the final formula that we can now use when introducing multiple neurons on each layer. When we give the layer multiple neurons, we can think of the process in the same way however with subscripts for the neuron number in that layer e.g., a_0^L and a_1^L. The difference is now you the sum of the errors or square of the errors instead of a single error (for reference see the right of figure 16).

We apply the described process to each neuron of each layer in order to find the sensitivity of the cost for each neuron to a specific weight. With multiple neurons you will also need to add the total individual cost functions relating to the different neurons it affects to get the equivalent of  for a neuron affecting multiple other neurons in a layer.

Sorting algorithm

A sorting algorithm will be used within my program. This will be used to compare output values and return the largest in order to determine which species of iris flower the computer has determined from the input data. This means I need a sorting algorithm best optimised to fewer data values since I'll only be sorting 3 values and eventually returning the largest. In order to implement this process, I compared the uses and advantages of different sorting algorithms:

- **Bubble sort**
- Works well with large datasets where the items are almost sorted because it takes only one iteration to detect whether the list is sorted or not however this not entirely useful with small datasets which is what the solution will work with.
- This algorithm is fastest on an extremely small or nearly sorted set of data which is ideal for the scenario it'll be applied in since it's only necessary to sort 3 values.
- **Insertion sort**
- This algorithm is stable, and it has the fastest running case when the list is nearly sorted.
- The memory usage is a constraint however is somewhat irrelevant to the solution since it's so small.
- **Merge sort**
- Merge sort is used when the data structure doesn't support random access since it works with pure sequential access iterators meaning it's optimised for an irrelevant situation.

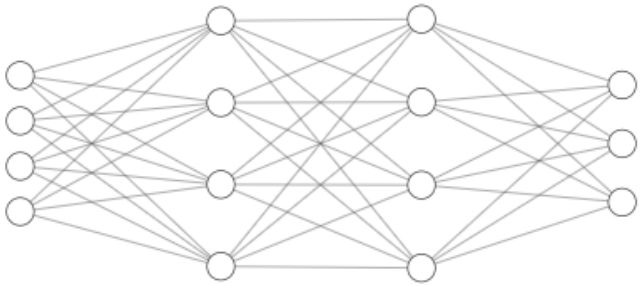
- The main advantage of the merge sort is its stability, the elements compared equally retain their original order.

Ideally out of the 3 algorithms I will be able to implement the bubble sort as it is most suitable to the parameters of my program. Since there are only 3 values the bubble sort is best optimised as there is both a higher likelihood that the list is already somewhat sorted and fewer items in the list means the bubble sort is best optimised.

A bubble sort works by allowing the values to 'bubble' to the left by comparing each value in sequence to the value to its right, then swapping the two if the one to the right is larger. This is iterated for every value in the list until the sort reaches the end. This process is then repeated until the list is fully sorted, or in the case for my project 3 iterations of this sort should be enough to fully sort the list from reverse state.

Proposed structure

The structure can be broken down and explained in 4 individual layers. The first layer provides one node for each input from that entry in the dataset. There are then 2 hidden layers containing 10 nodes each. 2 Layers is optimal for this demonstration as it is neither overly complex nor demanding, where a larger number of layers would require more operations to be carried out per iteration in turn increasing how computationally expensive the program is. Deciding the number of neurons in the hidden layers is a very important part of deciding the overall neural network architecture. Though these layers do not directly interact with the external environment, they have a tremendous influence on the final output. Too few neurons will result in something called under fitting whilst too many neurons in the hidden layers can result in several problems (overfitting and time it takes to train the network mostly). For these reasons I have applied the following principles when determining the structure of the hidden layers:



- The number of hidden neurons should be between the size of the input layer and the size of the output layer.
- The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.
- The number of hidden neurons should be less than twice the size of the input layer.

The fourth layer is the output layer, one node designated to each species where the target output for the node representative of the correct species would be 1, whilst the others would be 0. This allows for a confidence system where there is a greater room for error, whilst still giving the correct output.

Context diagram

Figure 17 shows a context diagram for how external factors will interact with the network. Rectangular red shaded boxes represent external factors, the yellow circle is the network, and the arrows are the data being passed between the entities.

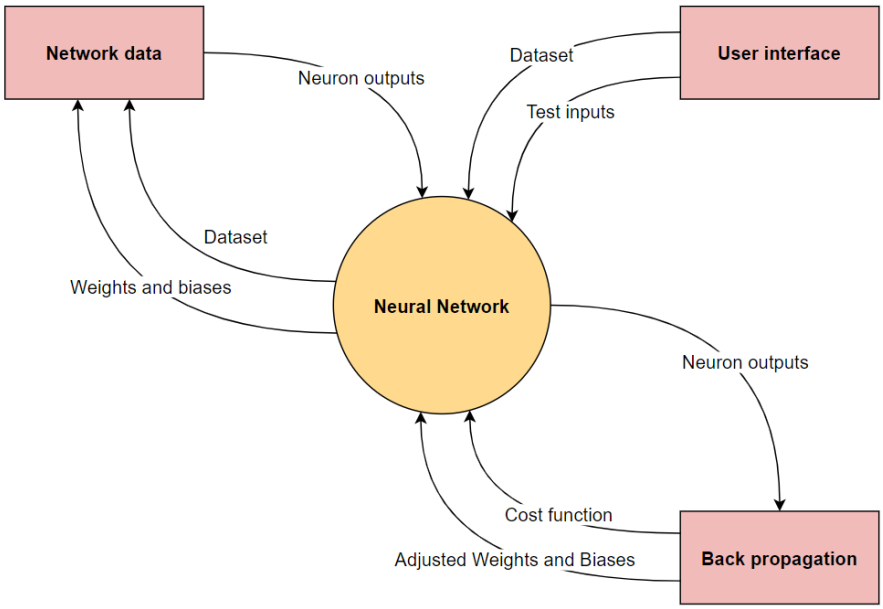


Figure 17

To the right is figure 18 which is the context Diagram for the iris identifier program itself interacting with the network and user interface factors.

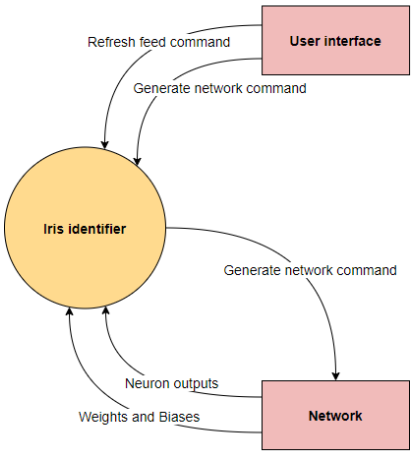
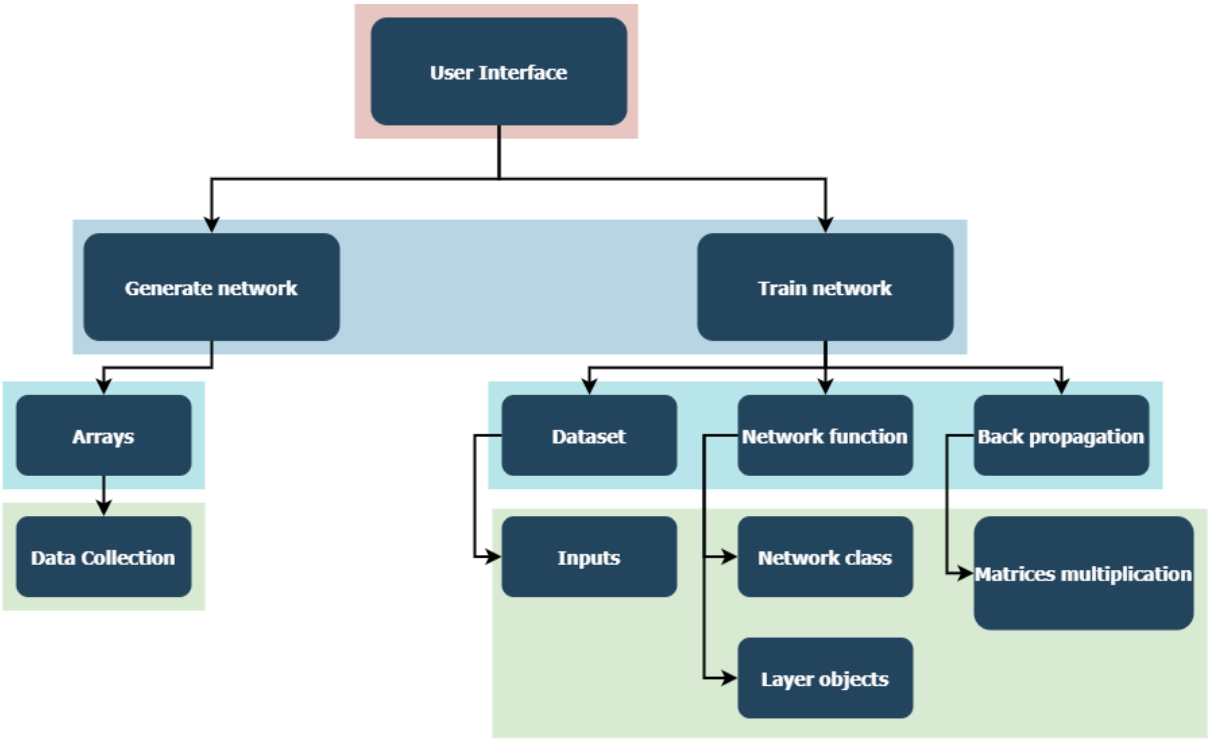


Figure 18

Hierarchy charts and flow diagrams

Hierarchy chart



Flow diagrams

Initial model

There are many different types of neural network that operate in different orders and sequences, for example another type of neural network could be laid out as such where figure 19 was an early concept design for how the processes behind my neural network may operate:

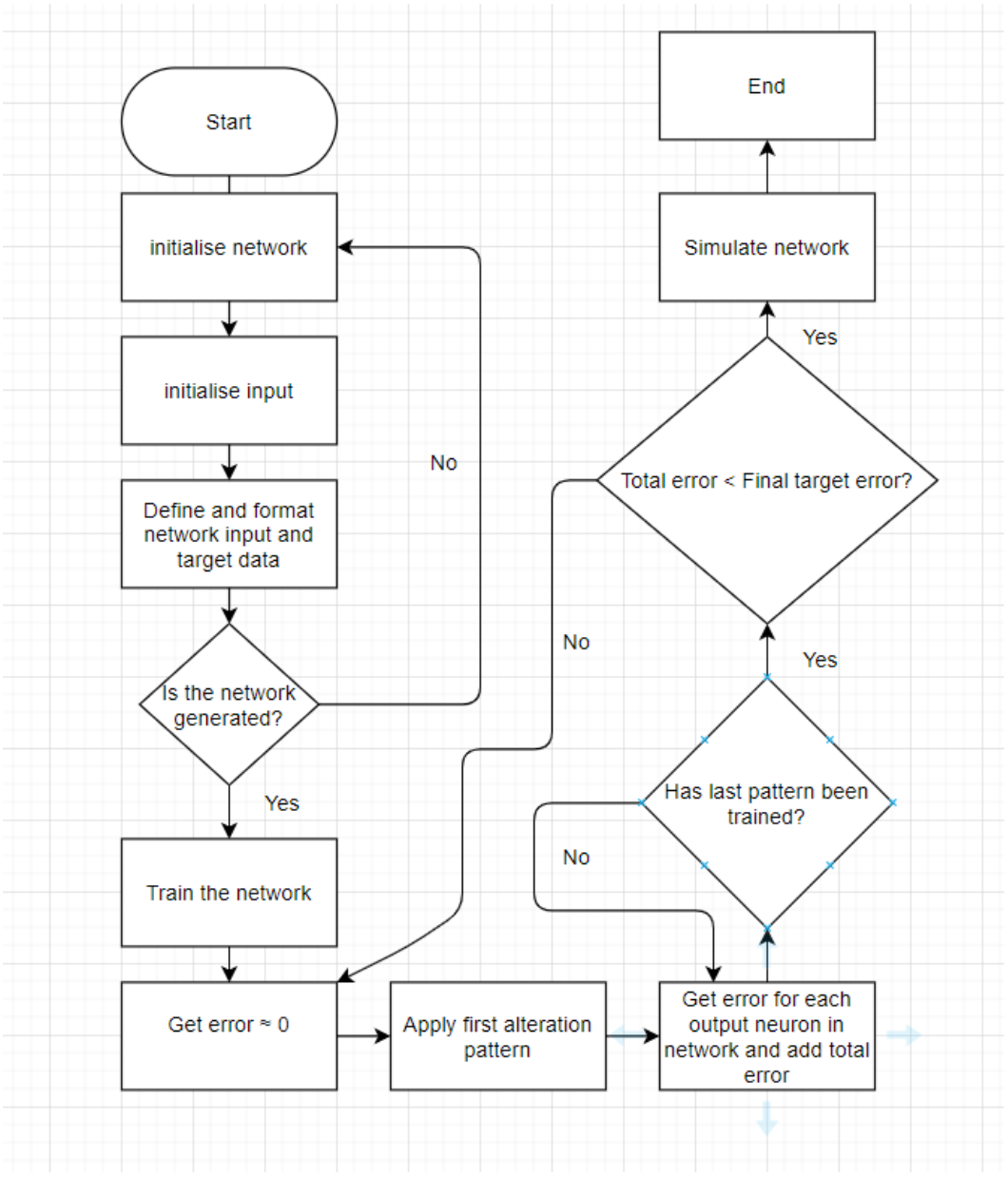


Figure 19

General overview model

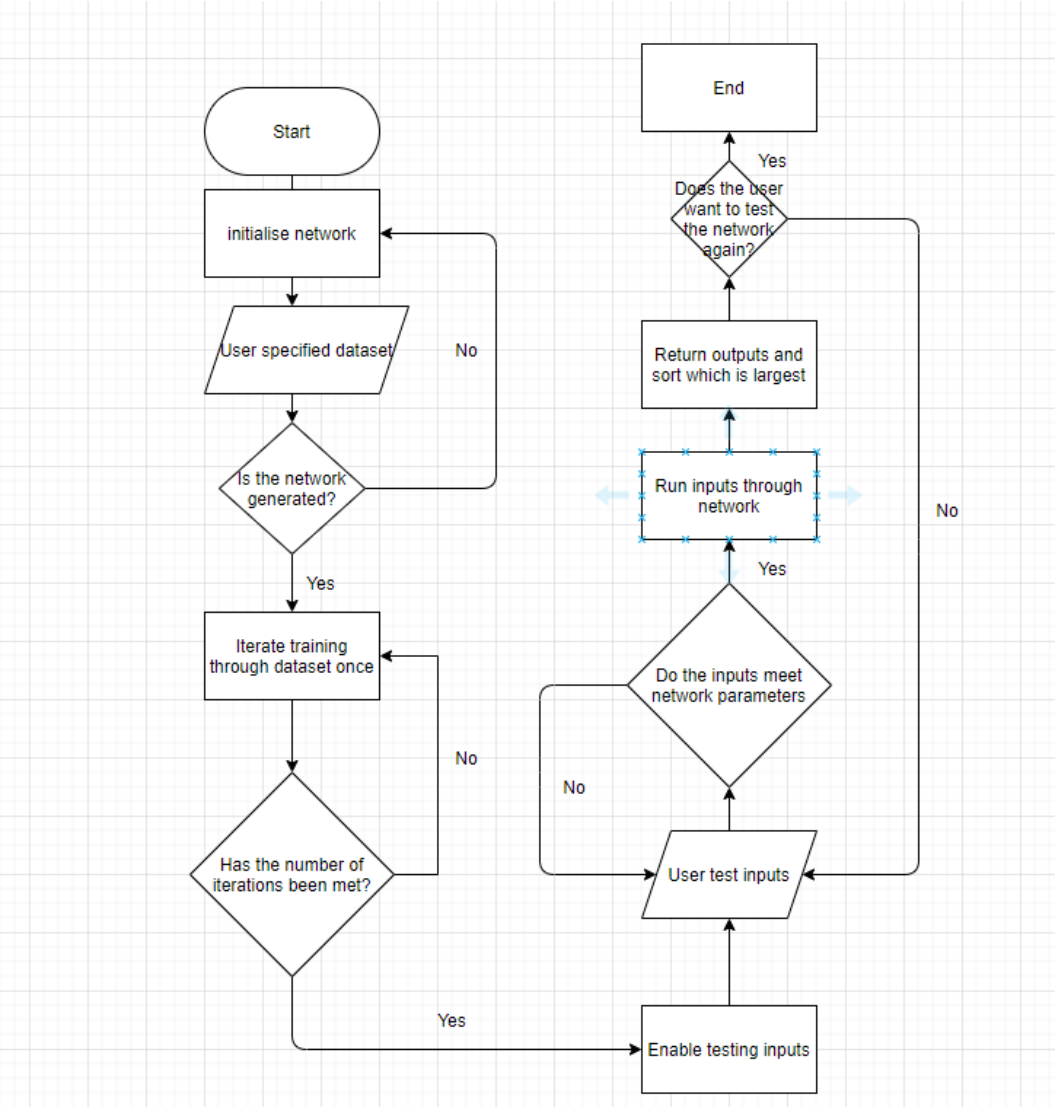


Figure 20

The figure 20 flowchart is a more accurate final flowchart for the general operations of the program as a whole and how the user interface will interact with the user and program. Developed flowcharts on specific operations such as network training and testing the network can be seen below.

Neural network flowchart

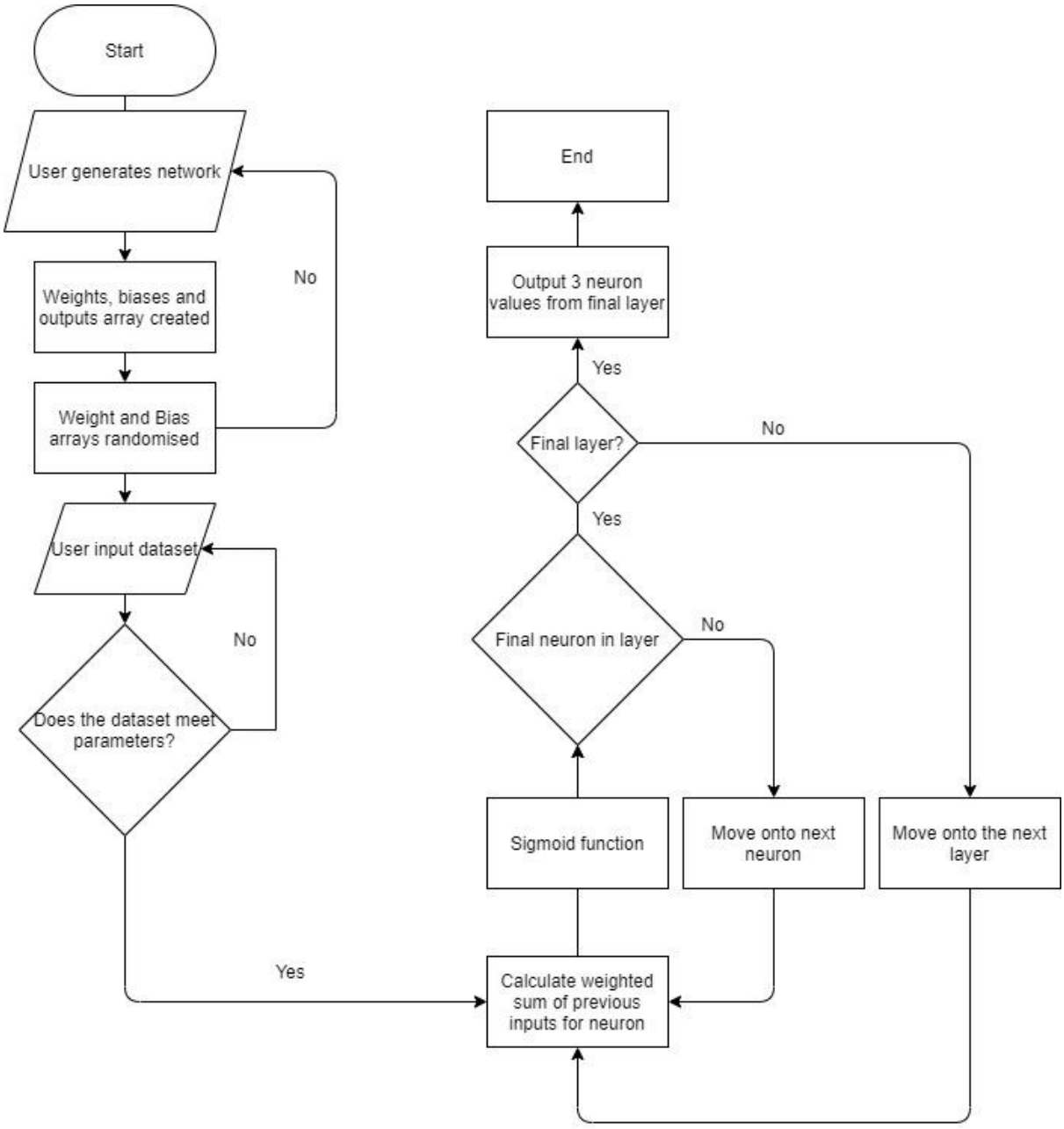
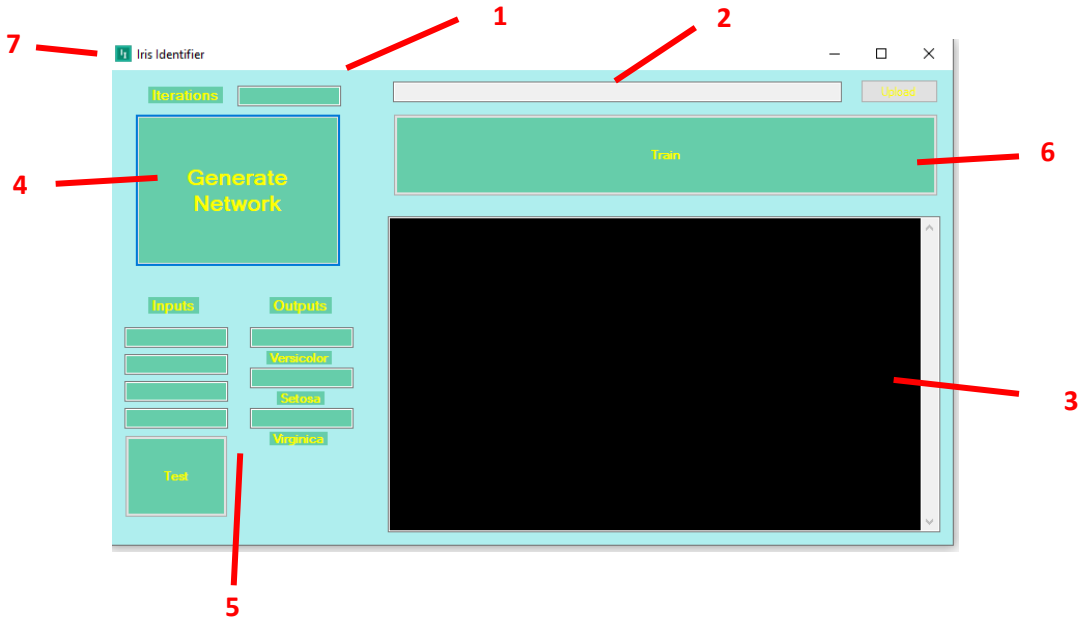


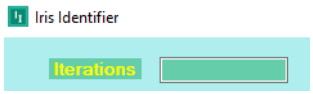
Figure 21

This is the flowchart design mapping the specific function of how the network iterates through training and generating weights and biases.

User interfaces

There is one form across the entire solution in order to ensure the network is as easy to use and despite being a complex algorithm is as visually simple as possible. The solution needs to be an effective and easily interpretable learning tool **[Objective 1.6]** therefore there shouldn't be too many buttons on screen at any given time. The colour scheme also needs to be simple, on top of this I used coloured labels with a contrasting background to draw attention to key interaction points for the user. A breakdown of individual components of the UI can be found below.



1. 

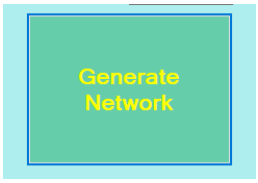
A single textbox will allow the user to enter a number from 1 – 50, where this number is the multiple of batches and in turn the multiple of 20 data entries, having the training process iterate finitely to this value. There is a label immedieatly to the left of the textbox where the role of the value is clearly denoted as the number of 'Iterations'.

2. 

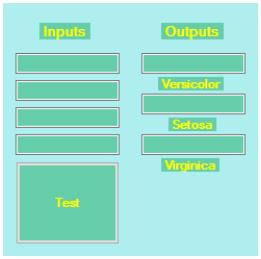
This large textbox is set to read only as it only needs to display the file location of the datset text file that the user selects for training. When the 'Upload' button is pressed it opens the file dialog menu for the user to interact with and select a file from their local storage.

3. 

The feed is a black background and multi lined layered textbox that has been placed so that it has a clear level of importance in the operation of the network. When the network is generated, the weight and bias of every neuron in the network will be printed layer by layer in the feed. This design makes the operation of the program more followable as the network is propagated forward (objective 1.1).

4. 

Before any operations of the program can be executed, the network must be generated. For this reason, the genrate network button is the largest on the form. If the network is attemtped to be tested or trained before the network is generated, an error message will be displayed. When the 'Generate Network' button is pressed the weights and biases will both be generated and printed simultaneously into the feed. At this point the network will be prepared for testing and training; if the button is pressed again the feed will be updated.

5. 

Two vertically listed collumns of 4 input textboxes and 3 output textboxes are used by the user to submit inputs for testing. The 'Test' button is listed beneath the inputs collumn as only inputs are required from the user in order to test the network, each of the output textboxes are read only for this reason. Each output textbox is labelled with the corresponding iris species that the textbox predicts (see objective 1.2).

6. 

The train button is another large component of the interface where based on the number of iterations specified by the user, will propagate forward utilising the network function iterating with respect to the iterations variable.

7. 

A logo is displayed as an icon to identify the program from taskbar easily. The form also follows the title 'Iris Identifier'.

Data dictionary

My neural network configuration is heavily variable dependant specifically on arrays as a temporary data structure. Many large data structures are used in the core of the network and have been assigned to a variable designating its name for reference in the algorithms and other processes. A list of the most integral variables to the operation of the solution is included with many temporary placeholders omitted e.g 'Temparray'.

Name	Privacy	Type	Description
Layer1	Public	NeuralNetwork	Each layer of the network is declared as an object so that it can be assigned respective values.
Layer2	Public	NeuralNetwork	Each layer of the network is declared as an object so that it can be assigned respective values.
Layer3	Public	NeuralNetwork	Each layer of the network is declared as an object so that it can be assigned respective values.
Inputs(4)	Private	String	The test inputs will be taken from user inputs into each of 4 textboxes.
Iterations	Public	Integer	When training the network, the user specifies the number of iterations that need to be gone through.
Transposed	Private	Boolean	If a matrix has been transposed from its original state, then the transposed = true.
SR	Public	StreamReader	A streamreader is created in the location selected by the user when uploading a dataset text file.
Values	Public	Array	This array isn't necessary but gives the option to implement backpropagation (objectives 2.1, 2.2).

Biases	Public	Array	An array declared respectively in the neural network class so it can be assigned to each layer used to store randomly generated biases for a neuron.
Weights	Public	Array	An array declared respectively in the neural network class so it can be assigned to each layer used to store randomly generated weights for a neuron.
Batches	Public	Array	An array in the neural network class where each line read from the user uploaded text file is broken up into 4 cells.
OutputsTest	Public	Array	Stores the weighted sum + bias and sigmoidal value of each neuron using user textbox inputs for testing the network.
OutputsTrain	Public	Array	Stores the weighted sum + bias and sigmoidal value of each neuron using batches from the user uploaded textfile.

Procedure dictionary

After considering different implementation methods for my solution, I decided to use 2010 Visual Studio to create a windows forms application. Due to the format, I have chosen, when the user is interacting with the network, it will take place through the use of the interface control available in visual studio's development environment. These controls are what allows the user to interact with the procedures of the network I the way that's intended.

Name	**Control**	**Event**	**Description**
Upload	OpenFileDialog		Opens the file dialog interaction menu for the user to select a text file to use as the training dataset.
Feed	Textbox		The main output for network activity. All weights and biases are printed layer by layer into the feed.
GenerateBtn	Button	_Click	Will activate a procedure of code which generates the network of weights and biases.
TrainBtn	Button	_Click	Will activate a procedure of code which propagates the network forward and print outputs and inputs into their corresponding textboxes for each iteration.
TestBtn	Button	_Click	Will activate a procedure of code which propagates once, giving 3 outputs and a message box returning the species with the largest output confidence.

Technical Solution

Technical Overview

This solution demonstrates the processes of a machine learning AI that operates through a neural network structure of 2 hidden layers containing 4 neurons each with 4 inputs and 3 outputs. It utilises weights and biases across the network to create a network function that can be iterated through different inputs and provide outputs.

It acts as a tool provide an interface for students to interact with a functional neural network in a way that effectively exposes the variance across weights and biases as the program operates. It will demonstrate how an AI is able to learn through training using the iris dataset. Once the network is trained it will allow the user to test by providing 4 inputs and returning 3 float outputs. Each output is representative in the computer's confidence that the input parameters correspond to that species of iris flower.

Iterative testing

As part of the iterative development strategy, regular testing and troubleshooting was done in order to achieve the final solution. As versions of the software are developed, a variety of tests were used to ensure functionality throughout versions of development. The following section describes the different troubleshooting errors that were encountered via this strategy and how each was resolved.

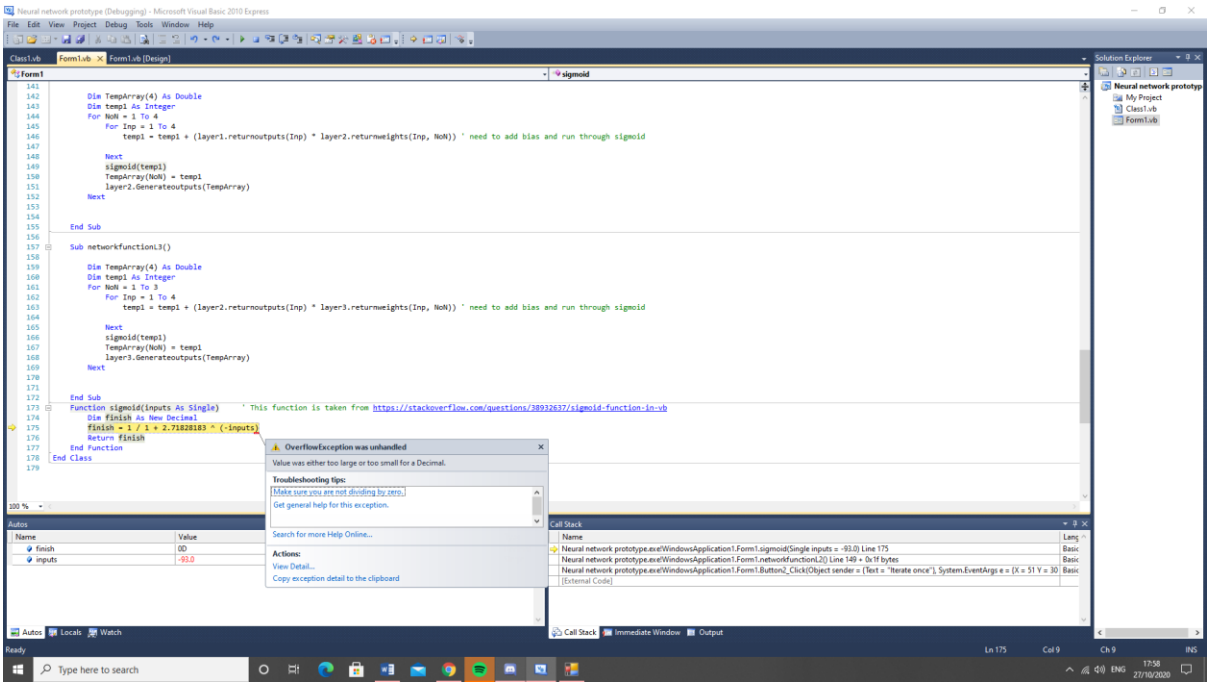
1. There was an Issue of declaring an array outside of a subroutine in the neural network class – initially I thought this could be solved by using ByVal inside of the subroutine which would bring a public variable. This solution was initially unsuccessful but after approaching the issue from another angle in another iteration of the design I found a solution. It was solved by declaring the global variables as an array then setting a temporary array equal to the global arrays.
2. The second issue I encountered was an error involving the number of neurons and previous inputs being outside the bounds of the arrays. This was solved by swapping the parameters of the array to (neurons, previous inputs). A similar problem arose later in the program where variables I was using would need to change in size. This is not possible in visual basic as arrays are a static data structure, however there was a simple workaround to this issue. Using an array of size (100,100) gives more than enough leeway to store the changing sizes of the matrices used in forward propagation calculations.
3. External code:

Complex mathematics is not always easy to implement into a program. Initially when implementing this function, I attempted to use the value of e to 3 significant figures as the sigmoid function is defined as y=1/(1+e^(-x)). This however did not work and so I turned to external online coding libraries for assistance on such a minor issue. Below is the code and its corresponding source reference.

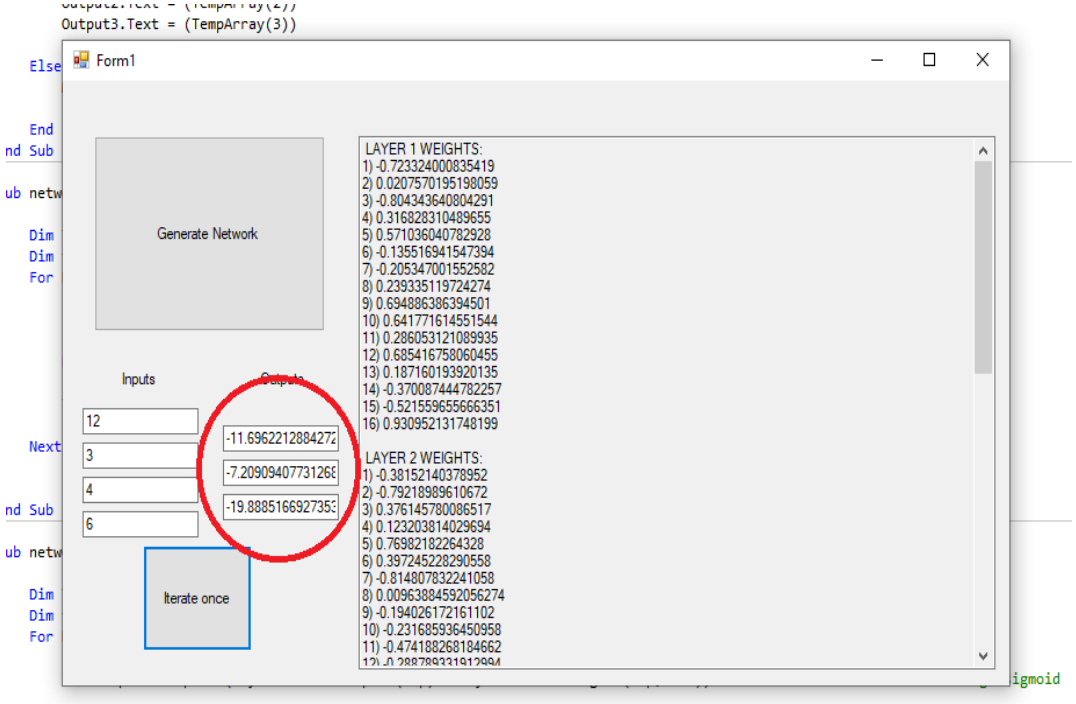
```
Function sigmoid(x As Double) As Double
    Return 1 / (1 + Math.Exp(-x))
End Function
```

https://stackoverflow.com/questions/38932637/sigmoid-function-in-vb

Iris flower species identifier Machine Learning Algorithm

4. 

Solved by converting variables to double



5. Next

Solved by using correct sigmoid function. There was also a logic error because I needed to make temp1 = to sigmoid (temp1) since it is a function not a subroutine where you can just call sigmoid (temp1).

6. I Used this website to test accuracy of matrices multiplication outputs: https://keisan.casio.com/exec/system/15157249643325

When generating batches array the program would sometimes not go below the 10th column, this was due to the way the stream reader in vb works. To solve this, I declared the stream reader in the first loop and closed it after the second loop so that the read head would reset its position each time. `For I As Integer = 1 To 20`

```
            Dim SR As New StreamReader(TextBoxUpload.Text) 'Creates a streamreader in the location selected by the
user
            rand = r.Next(0, 50)
            For x As Integer = 0 To rand
                SR.ReadLine()
                If x = rand Then
                    Dim T As String = SR.ReadLine   'Reads a random line from the textfile
                    For q = 1 To 4
                        If q = 1 Then
                            TempArray(q, I) = Mid(T, 1, 3)
                        End If
                        If q = 2 Then
                            TempArray(q, I) = Mid(T, 5, 3)
                        End If
                        If q = 3 Then
                            TempArray(q, I) = Mid(T, 9, 3)
                        End If
                        If q = 4 Then
                            TempArray(q, I) = Mid(T, 13, 3)        ' Splits each section of line into individual
cells in a temporary array
                        End If
                    Next
                End If
            Next
            SR.Close()
        Next
```

Variables list

Name	Privacy	Type	Description
Layer1	Public	NeuralNetwork	Each layer of the network is declared as an object so that it can be assigned respective values.
Layer2	Public	NeuralNetwork	Each layer of the network is declared as an object so that it can be assigned respective values.
Layer3	Public	NeuralNetwork	Each layer of the network is declared as an object so that it can be assigned respective values.
Inputs(4)	Private	String	The test inputs will be taken from user inputs into each of 4 textboxes.
Iterations	Public	Integer	When training the network, the user specifies the number of iterations that need to be gone through.
Transposed	Private	Boolean	If a matrix has been transposed from its original state, then the transposed = true.
SR	Public	StreamReader	A streamreader is created in the location selected by the user when uploading a

			dataset text file.
Values	Public	Array	This array isn't necessary but gives the option to implement backpropagation (objectives 2.1, 2.2).
Biases	Public	Array	An array declared respectively in the neural network class so it can be assigned to each layer used to store randomly generated biases for a neuron.
Weights	Public	Array	An array declared respectively in the neural network class so it can be assigned to each layer used to store randomly generated weights for a neuron.
Batches	Public	Array	An array in the neural network class where each line read from the user uploaded text file is broken up into 4 cells.
OutputsTest	Public	Array	Stores the weighted sum + bias and sigmoidal value of each neuron using user textbox inputs for testing the network.
OutputsTrain	Public	Array	Stores the weighted sum + bias and sigmoidal value of each neuron using batches from the user uploaded textfile.

Functions and sub procedures

There are two classes used in the solution, where the neural network contains sub procedures used to create arrays and functions to return the values in these arrays to the main class. Below is a full summary of functions and sub procedures used by the solution, included is every piece of associated code for reference (a chronological breakdown and annotation of the code can be found later in this section under 'chronologically annotated development of code').

Each section of the functions and sub procedures section will include all code associated with that algorithm along with an overview of the algorithm's objective and information on the time complexity of the program in big O notation.

Generate network

Before the user can use the network, they must generate the network via the 'GENERATE NETWORK' button. This emphasises how weights and biases are relevant to the process of forward propagation and shows exactly how they're laid out via the feed. The weights and biases are stared in arrays in order to propagate forward later.

```
Public Class Form1
    Dim Inputs(4) As Integer

    Dim layer1 As New NeuralNetwork
    Dim layer2 As New NeuralNetwork    ' Each layer of the network is declared as an
object
    Dim layer3 As New NeuralNetwork


Private Sub Button1_Click(sender As System.Object, e As System.EventArgs) Handles
GernerateBtn.Click

        ' Weights are generated for each layer and then printed into the feed
        Feed.Text = Nothing
        layer1.GenerateWeights(4, 4)
        layer2.GenerateWeights(4, 4)
        layer3.GenerateWeights(4, 3)                    ' generates arrays of random
weights given specified parameters for each object / layer
        Feed.AppendText(" LAYER 1 WEIGHTS: ")
        Feed.Text &= Environment.NewLine                ' outputs weights and biases for
each layer
        printweights(layer1)
        Feed.Text &= Environment.NewLine
        Feed.AppendText(" LAYER 2 WEIGHTS: ")
        Feed.Text &= Environment.NewLine
        printweights(layer2)
        Feed.Text &= Environment.NewLine
        Feed.AppendText(" LAYER 3 WEIGHTS: ")
        Feed.Text &= Environment.NewLine
        printweightslayer3()
        Feed.Text &= Environment.NewLine
        layer1.GenerateBiases(4)
        layer2.GenerateBiases(4)
        layer3.GenerateBiases(3)
        Feed.AppendText(" LAYER 1 BIASES: ")
        Feed.Text &= Environment.NewLine
        printbiases(layer1)
        Feed.Text &= Environment.NewLine
        Feed.AppendText(" LAYER 2 BIASES: ")
        Feed.Text &= Environment.NewLine
        printbiases(layer2)
        Feed.Text &= Environment.NewLine
        Feed.AppendText(" LAYER 3 BIASES: ")
        Feed.Text &= Environment.NewLine
        printbiaseslayer3()
        Feed.Text &= Environment.NewLine
    End Sub


Public Class NeuralNetwork
```

```
    Private Values As Array
    Private Batches As Array
    Private OutputsTest As Array
    Private OutputsTrain As Array
    Private Biases As Array
    Private Weights As Array
    Sub GenerateWeights(ByVal PreviousInputs, ByVal neurons)


        Randomize()

        Dim RandomArray(PreviousInputs, neurons) As Double
        'This sub creates a weight array depending on the layer object. the array is
size (PreviousInputs, neurons)
        For i = 1 To PreviousInputs
            For k = 1 To neurons
                Dim answer1 As Double = Rnd()
                Dim answer2 As Double = Rnd()
                If answer2 > 0.5 Then answer1 = answer1 * -1 ' Since the Rnd fucntion
generates a random number from 0 to 1 we used it to make a 50% chance that the value
will be negative (*-1)
                RandomArray(i, k) = answer1
            Next
        Next
        Weights = RandomArray ' The array just generated with random array is made
equal to the array that corresponds to the layer object.
    End Sub
    Sub GenerateBiases(ByVal neurons)

        Randomize()

        Dim RandomArray(neurons) As Double
        'This sub creates a weight array depending on the layer object. the array is
size (neurons)
        For k = 1 To neurons
            Dim answer1 As Double = Rnd()
            Dim answer2 As Double = Rnd()
            If answer2 > 0.5 Then answer1 = answer1 * -1 ' Since the Rnd fucntion
generates a random number from 0 to 1 we used it to make a 50% chance that the value
will be negative (*-1)
            RandomArray(k) = answer1
        Next

        Biases = RandomArray ' The array just generated with random array is made
equal to the array that corresponds to the layer object.
    End Sub


Sub printweights(ByVal Layer As NeuralNetwork)
        Dim integer1 As Integer = Nothing
        For i = 1 To 4
            For x = 1 To 4                                          ' This sub loops
across each cell via rows and collumns printing the contents of each in the weight
array for a specified layer
                integer1 = integer1 + 1  ' This counts up from one to the number of
weights outputted so far and was added to make sure there are 16 values for the first
two layers and 12 for the last
                Feed.AppendText(integer1 & ") ")
                Feed.AppendText(Layer.returnweights(i, x))
                Feed.Text &= Environment.NewLine
            Next
        Next
```

```
    End Sub
    Sub printweightslayer3()
        Dim integer1 As Integer = Nothing     ' This sub is the same as printweights
except since layer three only has 3 collumns it needs a different loop
        For i = 1 To 4
            For x = 1 To 3
                integer1 = integer1 + 1
                Feed.AppendText(integer1 & ") ")
                Feed.AppendText(layer3.returnweights(i, x))
                Feed.Text &= Environment.NewLine
            Next
        Next
    End Sub

    Sub printbiases(ByVal Layer As NeuralNetwork)            ' This sub prints the
biases line by line to the feed
        Dim integer1 As Integer = Nothing
        For i = 1 To 4
            integer1 = integer1 + 1
            Feed.AppendText(integer1 & ") ")
            Feed.AppendText(Layer.returnbiases(i))
            Feed.Text &= Environment.NewLine
        Next
    End Sub

    Sub printbiaseslayer3()
        Dim integer1 As Integer = Nothing                   ' This sub prints the
biases line by line to the feed
        For i = 1 To 3
            integer1 = integer1 + 1
            Feed.AppendText(integer1 & ") ")
            Feed.AppendText(layer3.returnbiases(i))
            Feed.Text &= Environment.NewLine
        Next

    End Sub
```

Generating the network is a complex algorithm that has multiple processes of O(n) and O(n^2) complexity where we take the worst-case scenario.

Batches created

The iteration method I chose is to create random batches of 20 entries from the dataset and train using matrices. This is repeated a specified number of times in order to complete network training.

```
Private Sub ButtonUpload_Click_1(sender As System.Object, e As System.EventArgs)
Handles ButtonUpload.Click
        If Upload.ShowDialog = Windows.Forms.DialogResult.OK Then     ' Using the open
file dialog menu, if the input is ok then select that file location in the textbox
            TextBoxUpload.Text = Upload.FileName ' Write the file location in the
textbox
        End If
    End Sub


Public Class NeuralNetwork
    Private Values As Array
    Private Batches As Array
    Private OutputsTest As Array
    Private OutputsTrain As Array
    Private Biases As Array
```

```
    Private Weights As Array

  Sub GenerateBatches(ByVal TempArray)
        Batches = TempArray ' The array is filled with rows inputs from 20 random
dataset entries.
    End Sub

Public Function returnbatches()
        Return Batches ' the return batches sub can be referenced in form1 class in
order to be used for calculations etc.
    End Function

    Sub Batches(ByVal TempArray)
        Dim r As New Random
        Dim rand As Integer


        For I As Integer = 1 To 20
            Dim SR As New StreamReader(TextBoxUpload.Text) 'Creates a streamreader in
the location selected by the user
            rand = r.Next(0, 149)
            For x As Integer = 0 To rand
                SR.ReadLine()
                If x = rand Then
                    Dim T As String = SR.ReadLine   'Reads a random line from the
textfile
                    For q = 1 To 4
                        If q = 1 Then
                            TempArray(q, I) = Mid(T, 1, 3)
                        End If
                        If q = 2 Then
                            TempArray(q, I) = Mid(T, 5, 3)
                        End If
                        If q = 3 Then
                            TempArray(q, I) = Mid(T, 9, 3)
                        End If
                        If q = 4 Then
                            TempArray(q, I) = Mid(T, 13, 3)        ' Splits each
section of line into individual cells in a temporary array
                        End If
                    Next
                End If
            Next
            SR.Close()
        Next
        Dim TempArray2(100, 100) As Double
        For o As Integer = 1 To 20
            For p As Integer = 1 To 4


                TempArray2(p, o) = CType(TempArray(p, o), Double)
            Next
        Next
        layer1.GenerateBatches(TempArray2) ' saves temp array as batches array

    End Sub
```

The time complexity of the generate batches algorithm is $O(n^2)$ including a nested for loop.

Train network

```
Private Sub Button1_Click_1(sender As System.Object, e As System.EventArgs) Handles
Button1.Click
        If TextBoxUpload.Text <> Nothing Then
```

```
            If Feed.Text <> Nothing Then    ' ensures that if these textboxes are
empty that the program won't run

                Dim TempArray(100, 100) As String ' Creates a matrix consisting of 20
random rows from the data called a 'batch' however 100 x 100 because of vb
limiatations without libraries
                Batches(TempArray)

                Dim transposed As Boolean = False
                If transposed = False Then
                    For i = 1 To 4
                        For j = 1 To 20
                            TempArray(j, i) = layer1.returnbatches(i, j) ' Transposes
the matrix and replaces the saved array with this transposed array
                        Next
                    Next
                    transposed = True
                End If


                layer1.Generatebatches(TempArray)

                Networkfunction()

            Else
                MsgBox("NETWORK MUST BE GENERATED BEFORE USE")
            End If
        Else
            MsgBox("CANNOT TRAIN NETWORK WITHOUT UPLOADING A DATASET TO TRAIN FROM")
        End If


    End Sub


Sub Networkfunction()
        Dim iterations As Integer = 10
        For x As Integer = 1 To iterations ' This loop gives optional functionality
later on where the number of iterations can be a user specified parameter
            networkfunctionL1()
            networkfunctionL2()
            networkfunctionL3()    ' This is the bulk of forward propagation where the
outputs for each neuron are generated
            For i As Integer = 1 To 20
                Threading.Thread.Sleep(10) ' A delay is added in order for the user to
observe how the network advances
                For j As Integer = 1 To 4        ' This prints each input to their
corresponding network output to the specified textbox
                    If j = 1 Then
                        Input1.Text = layer1.returnbatches(i, j)
                    ElseIf j = 2 Then
                        Input2.Text = layer1.returnbatches(i, j)
                    ElseIf j = 3 Then
                        Input3.Text = layer1.returnbatches(i, j)
                    ElseIf j = 4 Then
                        Input4.Text = layer1.returnbatches(i, j)
                    End If
                    Application.DoEvents()
                Next
                For r As Integer = 1 To 3
                    If r = 1 Then
                        Output1.Text = layer3.returnoutputstrainarray(i, r)
```

```
                    ElseIf r = 2 Then
                        Output2.Text = layer3.returnoutputstrainarray(i, r)
                    ElseIf r = 3 Then
                        Output3.Text = layer3.returnoutputstrainarray(i, r)
                    End If
                    Application.DoEvents()
                Next
            Next
        Next
    End Sub
```

The time complexity of the train network algorithm is O(n^3) including a triple nested for loop.

Forward propagation

```
Public Class NeuralNetwork
    Private Values As Array
    Private Batches As Array
    Private OutputsTest As Array
    Private OutputsTrain As Array
    Private Biases As Array
    Private Weights As Array

Sub Generateoutputstrain(ByVal TempArray)
        OutputsTrain = TempArray ' The array is filled with rows inputs from 20 random
dataset entries.
    End Sub

Public Function returnoutputstrainarray()
        Return OutputsTrain ' the return outputs sub can be referenced in form1 class
in order to be used for calculations etc.
    End Function


Sub networkfunctionL1()

        Dim Temparray2(100, 100) As Double ' Since array is a static data structure i
make an array with more cells than will be required
        Dim TempArray(,) As Double = MultiplyMatrices(layer1.returnbatches,
layer1.returnweightsarray) ' Temparray is set equal to the weighted sum of each input
for each data entry in the batch
        layer1.GenerateValues(TempArray) ' Values before sigmoid saved for back
propagation
        For x As Integer = 1 To 20
            For y As Integer = 1 To 4
                TempArray(x, y) = TempArray(x, y) + layer1.returnbiases(y) ' Adds on
the bias for each neuron's output value
                Temparray2(x, y) = sigmoid(TempArray(x, y)) ' A loop that sigmoids
every value generated previously and stores it in an identical array used to propogate
forward in the next layer
            Next
        Next

        layer1.Generateoutputstrain(Temparray2) ' This output array is stored for
layer 1
    End Sub

    Sub networkfunctionL2()

        Dim Temparray2(100, 100) As Double ' Since array is a static data structure i
make an array with more cells than will be required
        Dim TempArray(,) As Double = MultiplyMatrices(layer1.returnoutputstrainarray,
layer2.returnweightsarray) ' Temparray is set equal to the weighted sum of each output
from the previous layer
```

```
        layer2.GenerateValues(TempArray) ' Values before sigmoid saved for back
propagation
        For x As Integer = 1 To 20
            For y As Integer = 1 To 4
                TempArray(x, y) = TempArray(x, y) + layer2.returnbiases(y) ' Adds on
the bias for each neuron's output value
                Temparray2(x, y) = sigmoid(TempArray(x, y)) ' A loop that sigmoids
every value generated previously and stores it in an identical array used to propogate
forward in the next layer
            Next
        Next


        layer2.Generateoutputstrain(Temparray2) ' This output array is stored for
layer 2
    End Sub

    Sub networkfunctionL3()

        Dim Temparray2(100, 100) As Double ' Since array is a static data structure i
make an array with more cells than will be required
        Dim TempArray(,) As Double =
MultiplyMatricesAlt(layer2.returnoutputstrainarray, layer3.returnweightsarray) '
Temparray is set equal to the weighted sum of each output from the previous layer
        layer3.GenerateValues(TempArray) ' Values before sigmoid saved for back
propagation
        For x As Integer = 1 To 20
            For y As Integer = 1 To 3
                TempArray(x, y) = TempArray(x, y) + layer3.returnbiases(y) ' Adds on
the bias for each neuron's output value
                Temparray2(x, y) = sigmoid(TempArray(x, y))
            Next
        Next
        layer3.Generateoutputstrain(Temparray2) ' This output array is stored for
layer 3
    End Sub
    Function MultiplyMatrices(ByVal input1 As Array, ByVal input2 As Array) ' Matrix
multiplication allows more efficient and precise calculation
        Dim sum As Double
        Dim mat3(100, 100) As Double
        For v = 1 To 20
            For z = 1 To 4
                sum = 0
                For j = 1 To 4
                    sum = sum + input1(v, j) * input2(j, z)   ' A sum is calculated
for each new cell in the result
                Next
                mat3(v, z) = sum

            Next
        Next
        Return mat3 '  A new matrix is returned with (in this contex) the columns
representing neuron and the Rows representing each entry from the batch (up to 20)
where in each cell is the weighted sum for that neuron based on the inputs

    End Function

    Function MultiplyMatricesAlt(ByVal input1 As Array, ByVal input2 As Array) '
Alternate sub for different parameters as getting length of array not possible in this
scenario
        Dim sum As Double
        Dim mat3(100, 100) As Double
```

```
        For v = 1 To 20
            For z = 1 To 3 ' loop altered to 3 iterations as final layer only has 3
neurons
                sum = 0
                For j = 1 To 4
                    sum = sum + input1(v, j) * input2(j, z)
                Next
                mat3(v, z) = sum

            Next
        Next
        Return mat3

    End Function
    Function sigmoid(x As Double) As Double 'This code was taken from
https://stackoverflow.com/questions/38932637/sigmoid-function-in-vb
        Return 1 / (1 + Math.Exp(-x)) ' sigmoid function
    End Function
```

If we model the network to have n inputs, h neurons in the hidden layer and m outputs then in total we have h + m neurons. The amount of memory a neuron occupies is O(w) where w is the number of inputs the neuron receives. This is because each neuron has a weight for each input as well as additional information like bias output and error (also learning rate if that is added later). So if we think of it likes this then since these are all constants then the memory required by one neuron is proportional to the number of weights and thus inputs.

The hidden layer I've made has h neurons with n inputs each hence it needs h multiplied by O(n) units of memory to store all the values. The output layer I've made has m neurons with h inputs each hence it needs m multiplied by O(h) units of memory to store all the values. Using all of this knowledge, memory = h*O(n) + m*O(h) = O(h*n) + O(m*h) = O((n+m)*h).

Test network

```
Public Class NeuralNetwork
    Private Values As Array
    Private Batches As Array
    Private OutputsTest As Array
    Private OutputsTrain As Array
    Private Biases As Array
    Private Weights As Array

Sub Generateoutputstest(ByVal TempArray)
        OutputsTest = TempArray ' The array just generated with random array is made
equal to the array that corresponds to the layer object.
    End Sub

Public Function returnoutputstest()
        Return OutputsTest ' the return outputs sub can be referenced in form1 class
in order to be used for calculations etc.
    End Function
Public Function returnbiases(ByVal neurons)
        Return Biases(neurons) ' the return bias sub can be referenced in form1 class
in order to be used for calculations etc.
    End Function

Public Function returnweightsarray()
        Return Weights ' the return weights sub can be referenced in form1 class in
order to be used for calculations etc.
    End Function
```

```
Private Sub Button2_Click(sender As System.Object, e As System.EventArgs) Handles
TestBtn.Click
        If Feed.Text <> Nothing Then


            Inputs(1) = Input1.Text
            Inputs(2) = Input2.Text
            Inputs(3) = Input3.Text
            Inputs(4) = Input4.Text              ' when iterate button is pressed, the
text from each textbox is stored in an array called inputs
            networktestL1()                 ' These inputs are then referenced in this
subroutine as it is declared globally
            networktestL2()
            networktestL3()
            Dim TempArray() As Double = layer3.returnoutputstest
            LblVers.Visible = True
            LblSeto.Visible = True
            LblVirg.Visible = True
            Output1.Text = (TempArray(1))
            Output2.Text = (TempArray(2))
            Output3.Text = (TempArray(3))
            MsgBox(bubblesort(TempArray))


        Else
            MsgBox("NETWORK MUST BE GENERATED BEFORE USE")

        End If
    End Sub

    Sub networktestL1()

        Dim TempArray(4) As Double ' temp array equal to number of neurons in layer
        Dim temp1 As Double
        For NoN = 1 To 4    ' number of neurons
            For Inp = 1 To 4  ' number of previous layer outputs
                temp1 = temp1 + (Inputs(Inp) * layer1.returnweights(Inp, NoN)) ' need
to add bias and run through sigmoid

            Next
            temp1 = temp1 + layer1.returnbiases(NoN)
            TempArray(NoN) = sigmoid(temp1)    ' set this generated array equal to the
outputs of layer 1
            layer1.Generateoutputstest(TempArray)
        Next


    End Sub

    Sub networktestL2()

        Dim TempArray(4) As Double
        Dim temp1 As Double
        For NoN = 1 To 4
            For Inp = 1 To 4
                temp1 = temp1 + (layer1.returnoutputstest(Inp) *
layer2.returnweights(Inp, NoN)) ' This is the network function that multiplies each
weight by the corresponding input for each neuron

            Next
```

```
            temp1 = temp1 + layer2.returnbiases(NoN)         ' the bias is added to
the value for each neuron before sigmoid
            TempArray(NoN) = sigmoid(temp1)                  ' value is run through
sigmoid squishification function
            layer2.Generateoutputstest(TempArray)             ' the sigmoided value
is stored in an output array corresponding to the neurons of each layer
        Next


    End Sub

    Sub networktestL3()

        Dim TempArray(4) As Double             ' same as layer 2 network function but
for layer 3
        Dim temp1 As Double
        For NoN = 1 To 3
            For Inp = 1 To 4
                temp1 = temp1 + (layer2.returnoutputstest(Inp) *
layer3.returnweights(Inp, NoN)) ' need to add bias and run through sigmoid

            Next
            temp1 = temp1 + layer3.returnbiases(NoN)
            TempArray(NoN) = sigmoid(temp1)
            layer3.Generateoutputstest(TempArray)
        Next


    End Sub

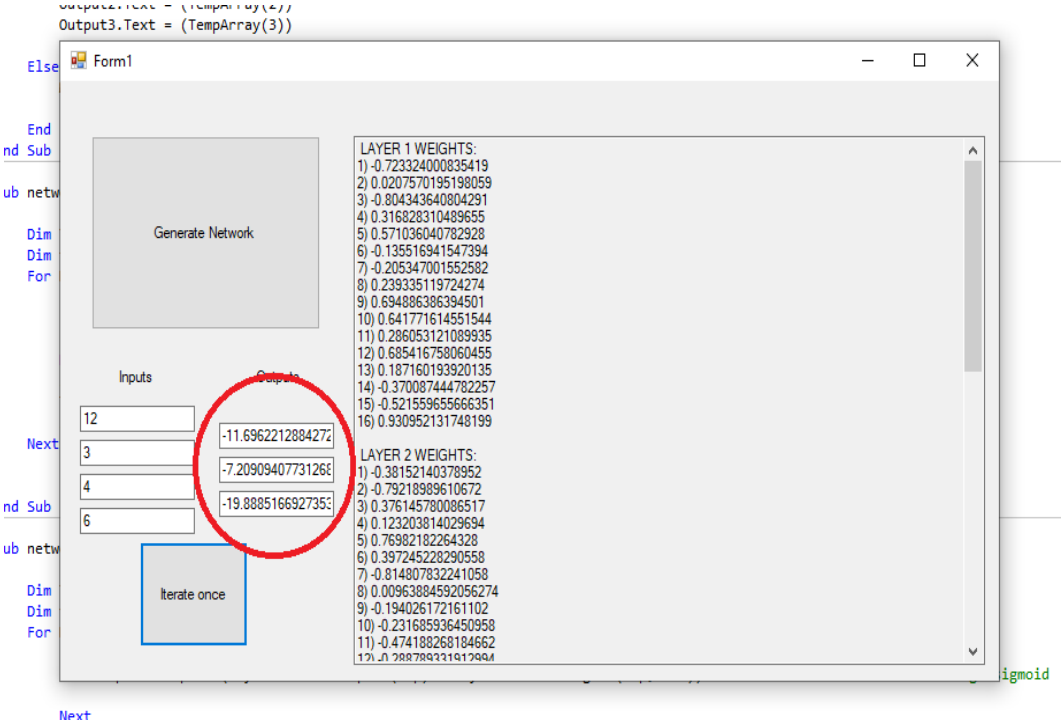
Function bubblesort(ByRef a) ' bubble sort where TempArray(1) is the largest value and
returns that value

        Dim b As Array = a
        Dim temp As Double
        For i As Integer = 1 To 3
            For k As Integer = 1 To 2
                If b(k) < b(k + 1) Then
                    temp = b(k + 1)
                    b(k + 1) = b(k)
                    b(k) = temp
                End If
            Next
        Next

        Return b(1)
    End Function
```

Chronologically annotated development of code

Pre-requisite to developing any code structure is a simple layout. Initially the base form looked like this:



With this design in place, production can start with the GenerateBtn_Click event. First however, as referenced in my design plan, I decided to implement object-oriented programming in order to generate my arrays. With this plan I could declare a new secondary class called 'NeuralNetwork' and use it to generate the arrays filled with weights and biases required for the network.

```
Public Class NeuralNetwork
    Private Biases As Array
    Private Weights As Array
```

Declaring these variables as private arrays allows me to use functions to return values later on, it also solves an early roadblock I faced in the development stage.

There was an issue of how to declare an array outside of the sub routine in the neural network class. I attempted to solve this error by using the ByVal reference inside of the sub routine in order to bringing a public variable. Eventually this was solved by declaring global variables as arrays then setting a temporary array equal to the global arrays.

This is the GenerateWeights sub procedure in the NeuralNetwork class. It takes the number of neurons in that layer and the number in the previous layer then generates weights connecting those layers.

```
Sub GenerateWeights(ByVal PreviousInputs, ByVal neurons)

        Randomize()

        Dim RandomArray(PreviousInputs, neurons) As Double
        'This sub creates a weight array depending on the layer object. the array is
size (PreviousInputs, neurons)
```

```
        For i = 1 To PreviousInputs
            For k = 1 To neurons
                Dim answer1 As Double = Rnd()
                Dim answer2 As Double = Rnd()
                If answer2 > 0.5 Then answer1 = answer1 * -1 ' Since the Rnd fucntion
generates a random number from 0 to 1 we used it to make a 50% chance that the value
will be negative (*-1)
                RandomArray(i, k) = answer1
            Next
        Next
        Weights = RandomArray ' The array just generated with random array is made
equal to the array that corresponds to the layer object.
    End Sub
```

Each weight cell needs to be filled with a float value from 1 to -1. In order to do this, I used the Rnd function which returns a random number from 0 to 1. This doesn't meet the range specified previously. To meet this parameter, I needed to use a second Rnd function where when the value is greater than 0.5 the value will be converted to its negative equivalent. This means 50% of the time the value will be between -1 and 0 meaning the total range is 1 to -1. This value is then set equal to the corresponding cell in the array, the process is iterated in a nested loop until all weights are generated.

```
    Sub GenerateBiases(ByVal neurons)

        Randomize()

        Dim RandomArray(neurons) As Double
        'This sub creates a weight array depending on the layer object. the array is
size (neurons)
        For k = 1 To neurons
            Dim answer1 As Double = Rnd()
            Dim answer2 As Double = Rnd()
            If answer2 > 0.5 Then answer1 = answer1 * -1 ' Since the Rnd fucntion
generates a random number from 0 to 1 we used it to make a 50% chance that the value
will be negative (*-1)
            RandomArray(k) = answer1
        Next

        Biases = RandomArray ' The array just generated with random array is made
equal to the array that corresponds to the layer object.
    End Sub
```

Similar but not identical to how each weight cell needs to be filled in the GenerateWeights sub procedure with a float value from 1 to -1, I used the Rnd function which returns a random number from 0 to 1. This doesn't meet the range specified previously. To meet this parameter, I needed to use a second Rnd function where when the value is greater than 0.5 the value will be converted to its negative equivalent. This means 50% of the time the value will be between -1 and 0 meaning the total range is 1 to -1. This value is then set equal to the corresponding cell in the array, the process is iterated in a nested loop until all biases are generated.

```
Public Function returnbiases(ByVal neurons)
        Return Biases(neurons) ' the return bias sub can be referenced in form1 class
in order to be used for calculations etc.
    End Function

    Public Function returnweights(ByVal PreviousInputs, ByVal neurons)
        Return Weights(PreviousInputs, neurons) ' the return weights function can be
referenced in form1 class in order to be used for calculations etc.
    End Function
```

After generating the weights and biases for each layer in the neural network class, I will need a way to access these values in the main class. To do this, public functions can be used to return specified values.

After generating the network, it is important to remember the solution is a learning tool and ensure it is visually accessible. The next step is to display the values generated in a feed where the order is expressed logically.

```
Private Sub Button1_Click(sender As System.Object, e As System.EventArgs) Handles
GernerateBtn.Click

        ' Weights are generated for each layer and then printed into the feed
        Feed.Text = Nothing
        layer1.GenerateWeights(4, 4)
        layer2.GenerateWeights(4, 4)
        layer3.GenerateWeights(4, 3)                    ' generates arrays of random
weights given specified parameters for each object / layer
        Feed.AppendText(" LAYER 1 WEIGHTS: ")
        Feed.Text &= Environment.NewLine              ' outputs weights and biases for
each layer
        printweights(layer1)
        Feed.Text &= Environment.NewLine
        Feed.AppendText(" LAYER 2 WEIGHTS: ")
        Feed.Text &= Environment.NewLine
        printweights(layer2)
        Feed.Text &= Environment.NewLine
        Feed.AppendText(" LAYER 3 WEIGHTS: ")
        Feed.Text &= Environment.NewLine
        printweightslayer3()
        Feed.Text &= Environment.NewLine
        layer1.GenerateBiases(4)
        layer2.GenerateBiases(4)
        layer3.GenerateBiases(3)
        Feed.AppendText(" LAYER 1 BIASES: ")
        Feed.Text &= Environment.NewLine
        printbiases(layer1)
        Feed.Text &= Environment.NewLine
        Feed.AppendText(" LAYER 2 BIASES: ")
        Feed.Text &= Environment.NewLine
        printbiases(layer2)
        Feed.Text &= Environment.NewLine
        Feed.AppendText(" LAYER 3 BIASES: ")
        Feed.Text &= Environment.NewLine
        printbiaseslayer3()
        Feed.Text &= Environment.NewLine



    End Sub
```

The values are printed layer by layer. This is the same order the program operates when propagating forward.

The printbiases subs utilise the returnbiases functions and so for the printweights subs.

After network generation was handled testing is the next logical step since it only requires iterating through the network once. Each network layer needs to be tested so three subroutines can be defined. In each, corresponding weights and biases will be returned in order to get the value output from each neuron. This utilises previously declared functions and so is not difficult to implement however, the output must be run through the sigmoid activation function. In order to do this, I used external code to create the subroutine:

```
Function sigmoid(x As Double) As Double 'This code was taken from
https://stackoverflow.com/questions/38932637/sigmoid-function-in-vb
        Return 1 / (1 + Math.Exp(-x)) ' sigmoid function
    End Function
```

There is also a bubble sort function that I used to return the largest of the three outputs when testing the network.

```
Function bubblesort(ByRef a) ' bubble sort where TempArray(1) is the largest value and
returns that value

        Dim b As Array = a
        Dim temp As Double
        For i As Integer = 1 To 3
            For k As Integer = 1 To 2
                If b(k) < b(k + 1) Then
                    temp = b(k + 1)
                    b(k + 1) = b(k)
                    b(k) = temp
                End If
            Next
        Next

        Return b(1)
    End Function
```

The network needs to output values, in order to do this, I would then need to declare an array to generate and return outputs for testing.

```
Public Class NeuralNetwork
    Private OutputsTest As Array

Sub Generateoutputstest(ByVal TempArray)
        OutputsTest = TempArray ' The array just generated with random array is made
equal to the array that corresponds to the layer object.
    End Sub

Public Function returnoutputstest()
        Return OutputsTest ' the return outputs sub can be referenced in form1 class
in order to be used for calculations etc.
    End Function
```

Now that all the values required to propagate forwards are declared we can test the network.

```
Sub networktestL1()

        Dim TempArray(4) As Double ' temp array equal to number of neurons in layer
        Dim temp1 As Double
        For NoN = 1 To 4    ' number of neurons
            For Inp = 1 To 4  ' number of previous layer outputs
                temp1 = temp1 + (Inputs(Inp) * layer1.returnweights(Inp, NoN)) ' need
to add bias and run through sigmoid

            Next
            temp1 = temp1 + layer1.returnbiases(NoN)
            TempArray(NoN) = sigmoid(temp1)    ' set this generated array equal to the
outputs of layer 1
            layer1.Generateoutputstest(TempArray)
        Next


    End Sub
```

```
    Sub networktestL2()

        Dim TempArray(4) As Double
        Dim temp1 As Double
        For NoN = 1 To 4
            For Inp = 1 To 4
                temp1 = temp1 + (layer1.returnoutputstest(Inp) *
layer2.returnweights(Inp, NoN)) ' This is the network function that multiplies each
weight by the corresponding input for each neuron

            Next
            temp1 = temp1 + layer2.returnbiases(NoN)         ' the bias is added to
the value for each neuron before sigmoid
            TempArray(NoN) = sigmoid(temp1)                  ' value is run through
sigmoid squishification function
            layer2.Generateoutputstest(TempArray)              ' the sigmoided value
is stored in an output array corresponding to the neurons of each layer
        Next


    End Sub

    Sub networktestL3()

        Dim TempArray(4) As Double             ' same as layer 2 network function but
for layer 3
        Dim temp1 As Double
        For NoN = 1 To 3
            For Inp = 1 To 4
                temp1 = temp1 + (layer2.returnoutputstest(Inp) *
layer3.returnweights(Inp, NoN)) ' need to add bias and run through sigmoid

            Next
            temp1 = temp1 + layer3.returnbiases(NoN)
            TempArray(NoN) = sigmoid(temp1)
            layer3.Generateoutputstest(TempArray)
        Next


    End Sub
```

TestBtn_Click handles the user request to test the network by taking 4 inputs from each of the four input textboxes. It will also print the three network outputs to the textboxes and display the labels corresponding to each.

```
Private Sub Button2_Click(sender As System.Object, e As System.EventArgs) Handles
TestBtn.Click
        If Feed.Text <> Nothing Then


            Inputs(1) = Input1.Text
            Inputs(2) = Input2.Text
            Inputs(3) = Input3.Text
            Inputs(4) = Input4.Text              ' when iterate button is pressed, the
text from each textbox is stored in an array called inputs
            networktestL1()                 ' These inputs are then referenced in this
subroutine as it is declared globally
            networktestL2()
            networktestL3()
            Dim TempArray() As Double = layer3.returnoutputstest
            LblVers.Visible = True
            LblSeto.Visible = True
```

```
            LblVirg.Visible = True
            Output1.Text = (TempArray(1))
            Output2.Text = (TempArray(2))
            Output3.Text = (TempArray(3))
            MsgBox(bubblesort(TempArray))


        Else
            MsgBox("NETWORK MUST BE GENERATED BEFORE USE")

        End If
    End Sub
```

Testing

Introduction

The testing section will be where I'll demonstrate that this solution works and is robust and furthermore will be used to provide evidence about the sophistication of the solution. The test description section will provide a focus on the more technically challenging parts of the solution and compare results of the testing to the objectives set out for the solution to achieve. The Test evidence section will include video evidence of the solution in operation by a user whilst the description will focus on verifying the values produced.

Testing strategy

In order to complete a testing process for ensured smooth functionality of my program, I will need to breakdown the testing process into sections. The key sections I've Identified are input validation, output verification, functionality, and efficiency. Input validation is simple sanitation of user inputs since this program is intensive on its dependence on user parameters. Output validation involves verifying that every value in the network is logically valid and generated without error. Functionality requires the perspective of the user outlined in my potential users' section, someone who has a background in computer science, neural network theory and no previous familiarity with the software / UI. When the user is propagating forward, the program must be efficient. Verifying this will involve checking timings of different processes and workloads.

Test descriptions

Input Validation

Test number	Description of purpose	Input data and type	Expected result
1.1.1 [Objective 1.5]	Test input validation on outer bounds of the number of iterations when training	999 **[Extreme]**	Program will train for the longest possible time
1.1.2 [Objective 1.5]	Test inputs for number of iterations that will take too long	1001 **[Erroneous]**	Message box asking for reasonable input
1.1.3 [Objective 1.5]	Test input validation on the number of iterations when training	10 **[Typical]**	Program will train for standard time
1.2.1 [Objective 1.2]	Test valid data input on the input value boxes	3.1,2.4,1.9,2.0 **[Typical]**	Program will output corresponding values to output boxes
1.2.2 [Objective 1.2]	Test unrealistic values on the input boxes	10,10,10,10 **[Extreme]**	Program will output corresponding values to output boxes
1.3.1 [Objective 1.3]	Test if only one dataset solution can be uploaded at a time	Selecting multiple solutions **[Erroneous]**	When selecting multiple solutions, only the most recent selection is stored in upload textbox
1.3.2 [Objective 1.3]	Test if incorrect file type can be uploaded	Selecting a .png file **[Erroneous]**	User is prompted to select another file type
1.3.3 [Objective 1.3]	Test file upload system is error free	Selecting a text file **[Typical]**	File location will appear in upload textbox
1.3.4 [Objective 1.3]	Test if network can train with the iris dataset	Selecting a text file and training **[Typical]**	The network will train outputting corresponding inputs and outputs based on the dataset
1.3.5 [Objective 1.3]	Test if network can train with an external dataset	Selecting a text file in the wrong format and training **[Erroneous]**	Network will not train
1.11 [Objective 1.1]	Test if user generates network multiple times	generating multiple networks **[Erroneous]**	Each newly generated network replaces the last in the text feed and provides different test results

Output Verification

The nature of the solution means many values will be printed without visible calculation. For this reason, the program is particularly susceptible to logic errors unless produced values are verified at various stages in the forward propagation process. It is important to ensure that each process produces correct values. In order to do this, I will insert breakpoints into the code and check variable and parameter values at key points in network training and testing:

Breakpoint number	Process	What is being checked	Desired output	How output will be verified
1.1 [Objective 1.1]	Weights network generation	Weights are generated correctly	There will be a total of 44 weights across all layers where each are unique and < |1|	The network will generate and print values to the feed that can be checked
1.2 [Objective 1.1]	Biases network generation	Biases are generated correctly	There will be a total of 11 biases across all layers where each are unique and < |1|	The network will generate and print values to the feed that can be checked
[Line 274] [Objective 1.5]	Multiply Matrices	Correct outputs from inputs	A new matrix is constructed where each value has been multiplied correctly	https://matrixcalc.org/en/
[Line 274] [Objective 1.5]	Multiply Matrices	Does the matrix generated have the correct columns and rows	A matrix of correct parameters than can be used to identify if the matrix needs to be transposed before next layer	Checked manually against prepared theory
[Line 372] [Objective 1.4]	Batches	Is each input from each collumn being split correctly into individual cells	Every row in temp array will have four cells each following the template (x.x)	Breakpoints after each loop allows the values in temparray to be checked
[Line 368] [Objective 1.4]	Batches	Are the inputs selected from the dataset random	Each time a row is complete another random row is selected	Check the value of the random value corresponds to the line selected in the text file and that this value never exceeds the number of lines in the text file
[Line 365] [Objective 1.4]	Batches	20 random lines are being	The array rows should each	Breakpoints after each loop allows the values in

		selected and stored appropriately	contain 4 random inputs where there are 20 columns in a batch	temparray to be checked
[Line 188] [Objective 1.5]	Sigmoid	The values should be run correctly through the sigmoid function	Refer to the sigmoid section in design for how value returned must be < |1|	Online sigmoid function
[Line 311] [Objective 1.2]	Bubble sort	Does the bubble sort return the highest confidence	array b(1) should contain the largest value where b(3) contains the smallest	Does array b(1) contain the largest value where b(3) contains the smallest

Functionality

Test number	Description of purpose	Input data and type	Expected result
1.1 [Objective 1.1 & 1.6]	Test 'GENERATE NETWORK' button on start-up	being pressed before network is generated **[Typical]**	The network will generate and print values to the feed
1.2.1 [Objective 1.3]	Test 'Train' button with 4 valid inputs	Being pressed with network generated **[Typical]**	The network will train outputting corresponding inputs and outputs based on the dataset
1.2.2 [Objective 1.3]	Test 'Train' button with 4 valid inputs	being pressed before network is generated **[Erroneous]**	Message box instructing user to generate network will appear
1.3.1 [Objective 1.2]	Test 'Test' button With missing inputs	Being pressed after network is generated **[Erroneous]**	Message box instructing user to provide network inputs
1.3.2 [Objective 1.2]	Test 'Test' button with 4 valid inputs	Being pressed with network generated **[Typical]**	3 outputs printed to each output box
1.3.3 [Objective 1.2]	Test 'Test' button with 4 valid inputs	being pressed before network is generated **[Erroneous]**	Message box instructing user to generate network will appear
1.4 [Objective 1.3]	Test 'Upload' button prints correct file path in upload box	Iris dataset text file uploaded [**Typical]**	Correct file location printed in upload box

Efficiency

Efficiency will be checked catergoricaly through workload analysis and the use of timings since the program uses real time mechanics when displaying the outputs of the network. For timings I will get an estimate of how the nubmer of iterations affects the time taken to train the network, also using this method to obtain a recommended number of iterations (usually this would be done by comparing timing against the accuracy produced from backpropagation however this solution only propagates forwards). For estimating workload and how well optimised the program is, I can use the integrated performance profiler provided by visual studio along with further checking performance through task manager.

The other measurement I've decided to list under efficiency is efficiency of use. How easy to use is the program **[Objective 1.6]**? Measuring how long it takes someone with a background in neural networks to train and test a network.

Test evidence

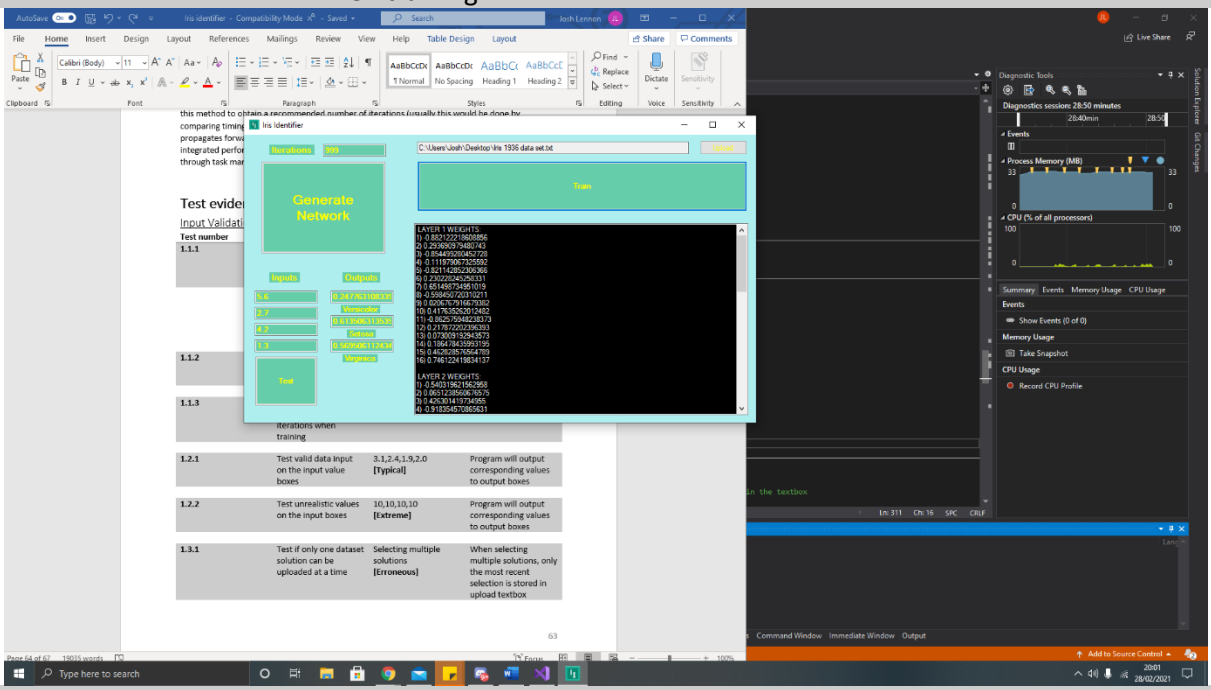
Input Validation

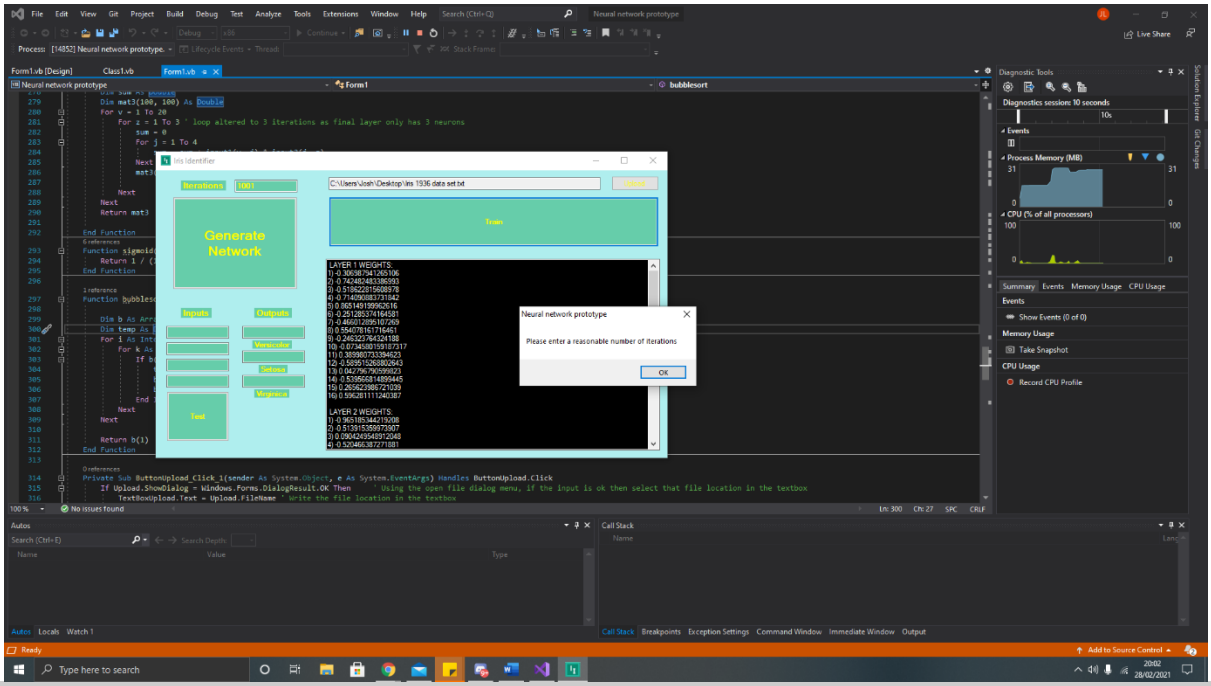
A VIDEO LINK FOR ALL TESTS IN THIS SECTION CAN BE FOUND HERE:
https://youtu.be/mgkgMS2qK2g

NARRATED VERSION:

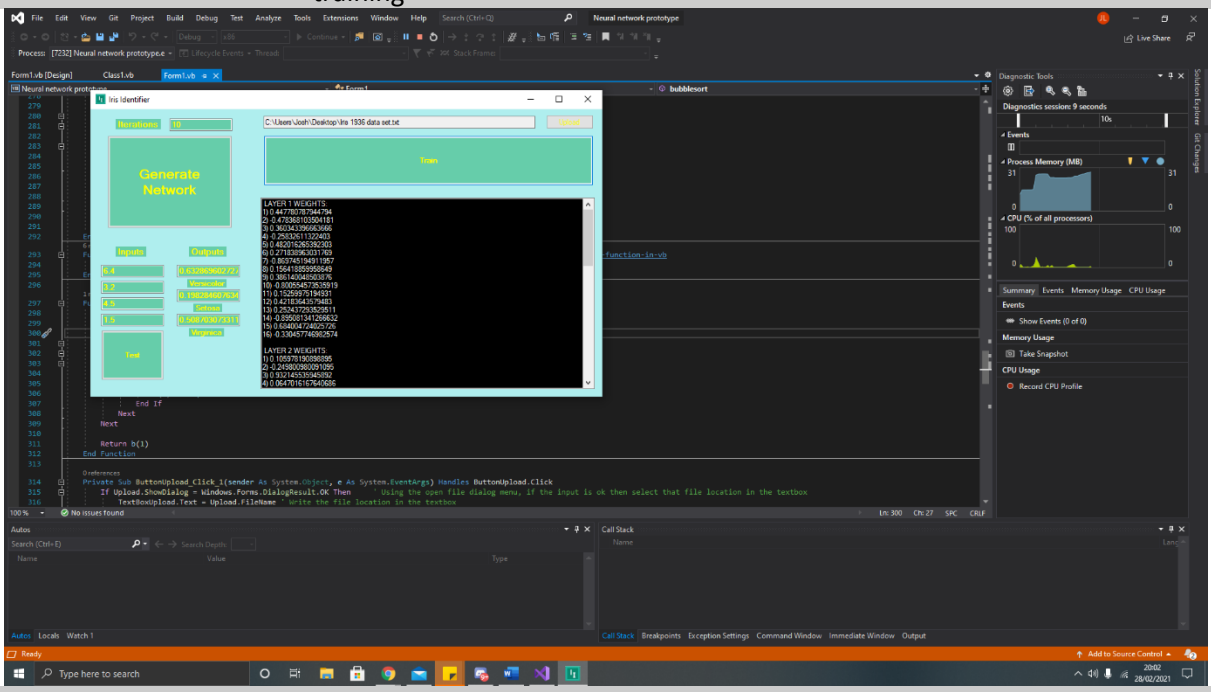
https://youtu.be/ePhTZiB92vs

Test number	Description of purpose	Input data and type	Expected result
1.1.1	Test input validation on outer bounds of the number of iterations when training	999 **[Extreme]**	Program will train for the longest possible time
1.1.2	Test inputs for number of iterations that will take too long	1001 **[Erroneous]**	Message box asking for reasonable input

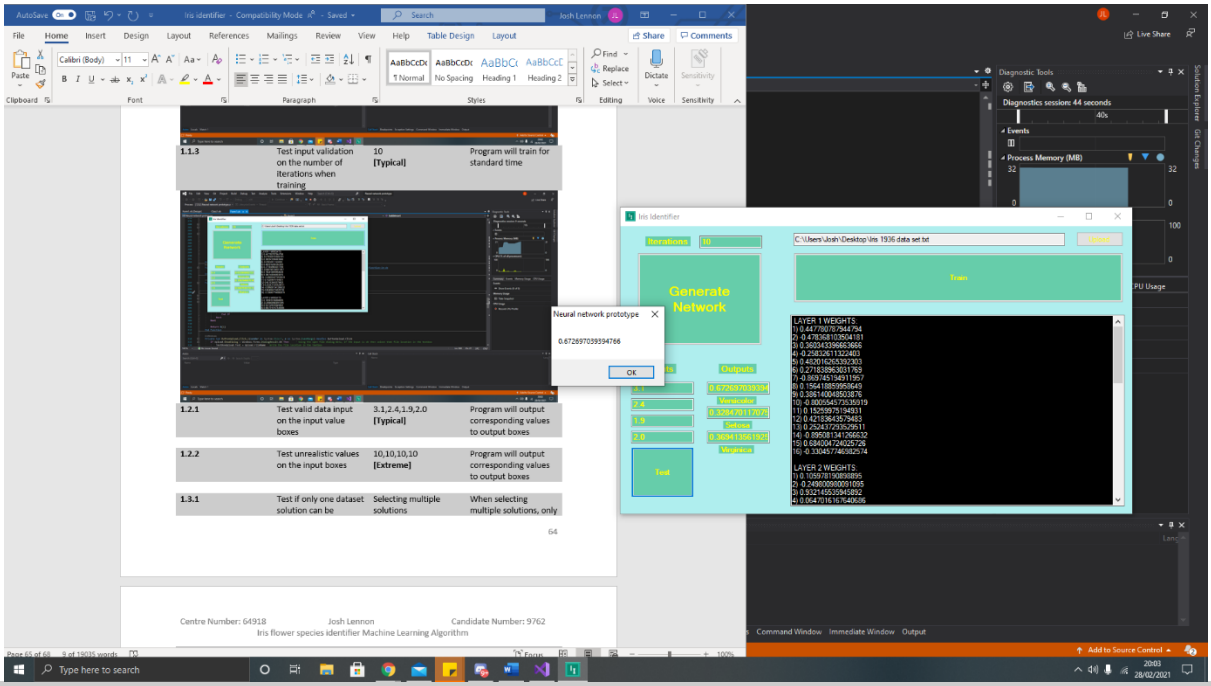




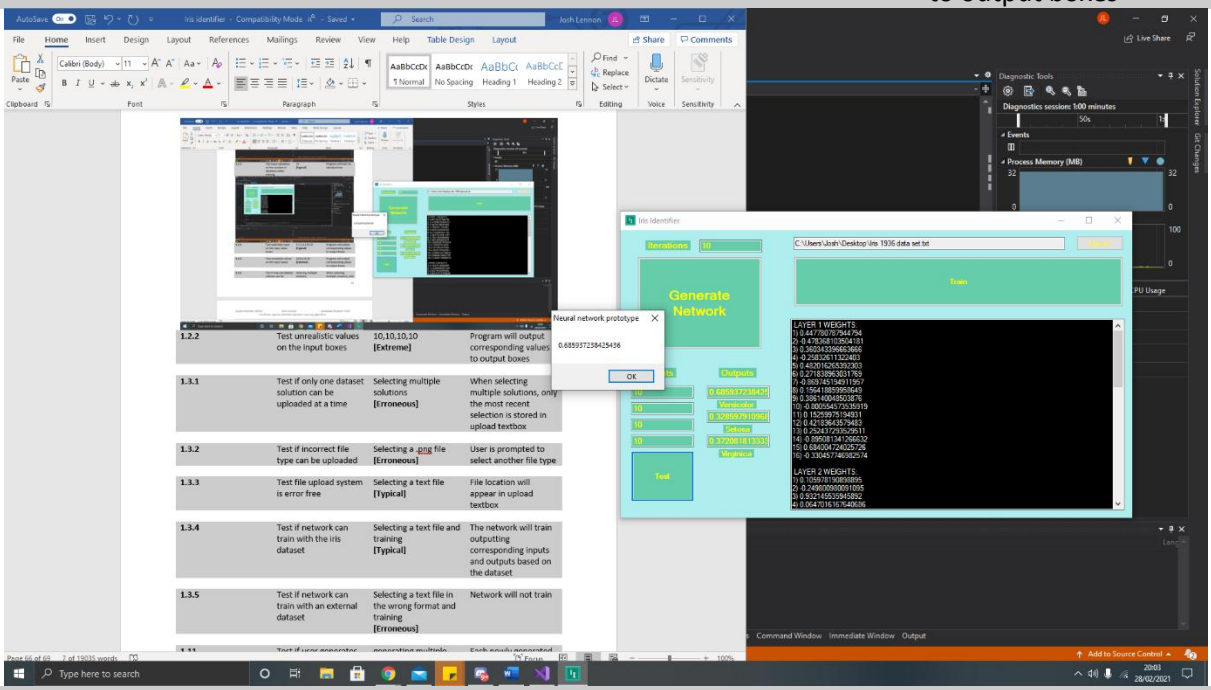
1.1.3	Test input validation on the number of iterations when training	10 **[Typical]**	Program will train for standard time



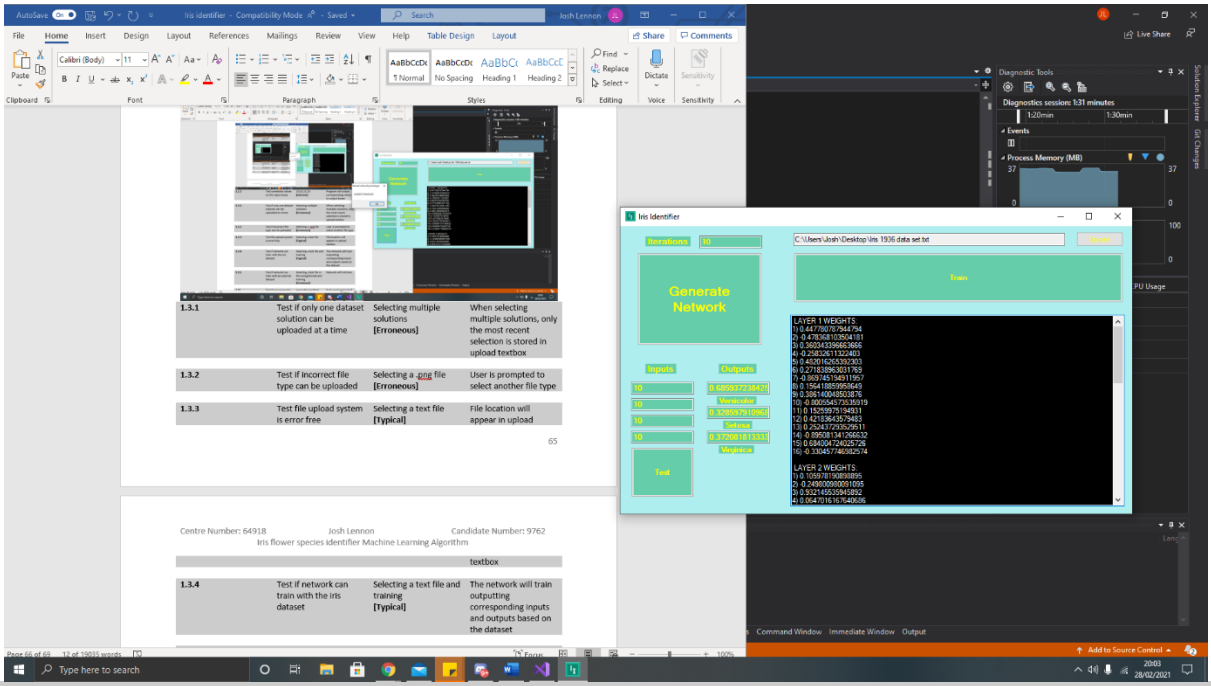
1.2.1	Test valid data input on the input value boxes	3.1,2.4,1.9,2.0 **[Typical]**	Program will output corresponding values to output boxes



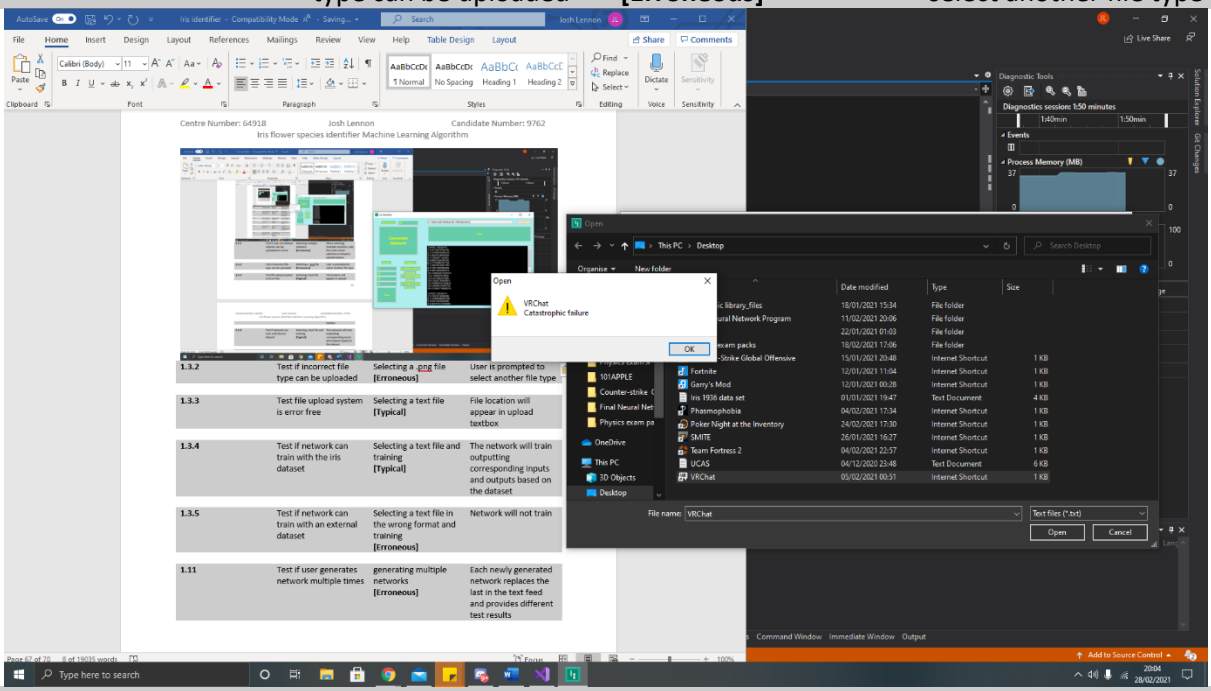
1.2.2	Test unrealistic values on the input boxes	10,10,10,10 **[Extreme]**	Program will output corresponding values to output boxes



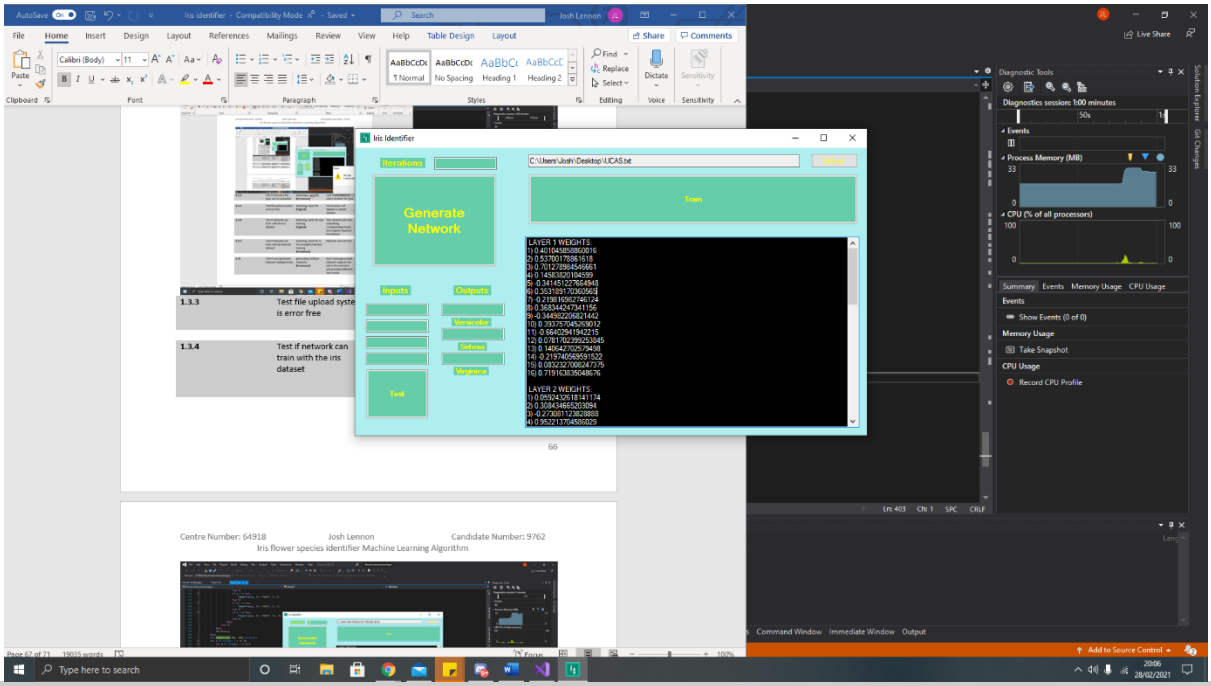
1.3.1	Test if only one dataset solution can be uploaded at a time	Selecting multiple solutions **[Erroneous]**	When selecting multiple solutions, only the most recent selection is stored in upload textbox



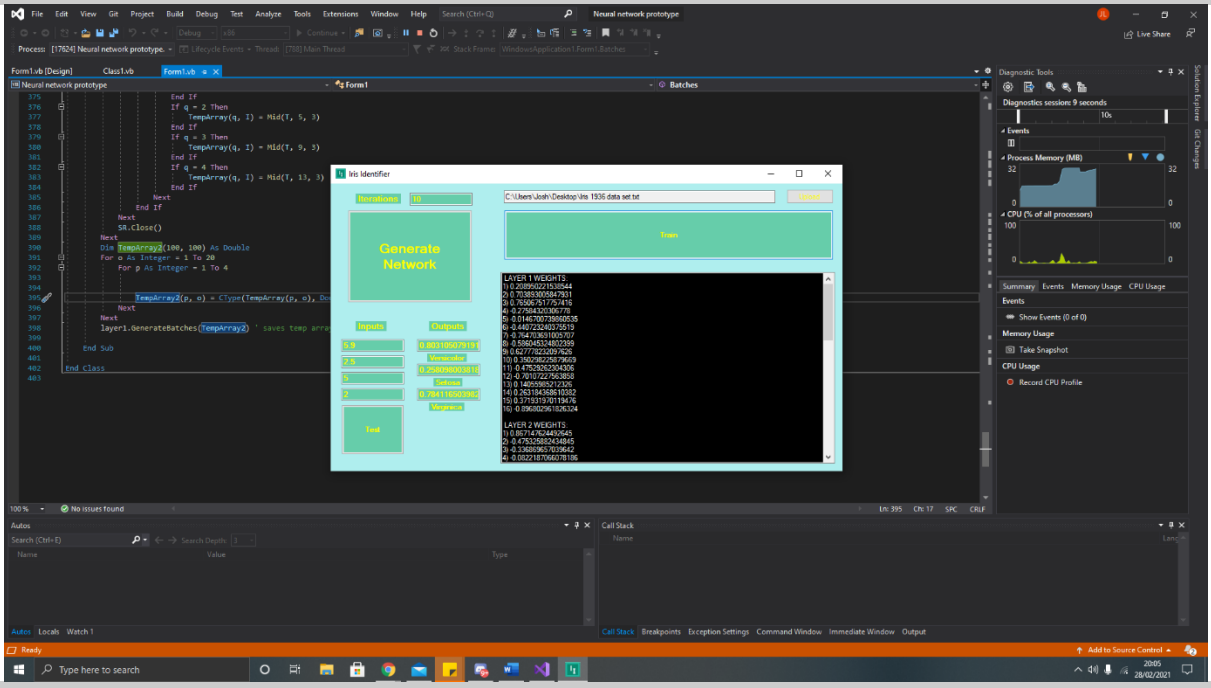
1.3.2	Test if incorrect file type can be uploaded	Selecting a .png file **[Erroneous]**	User is prompted to select another file type



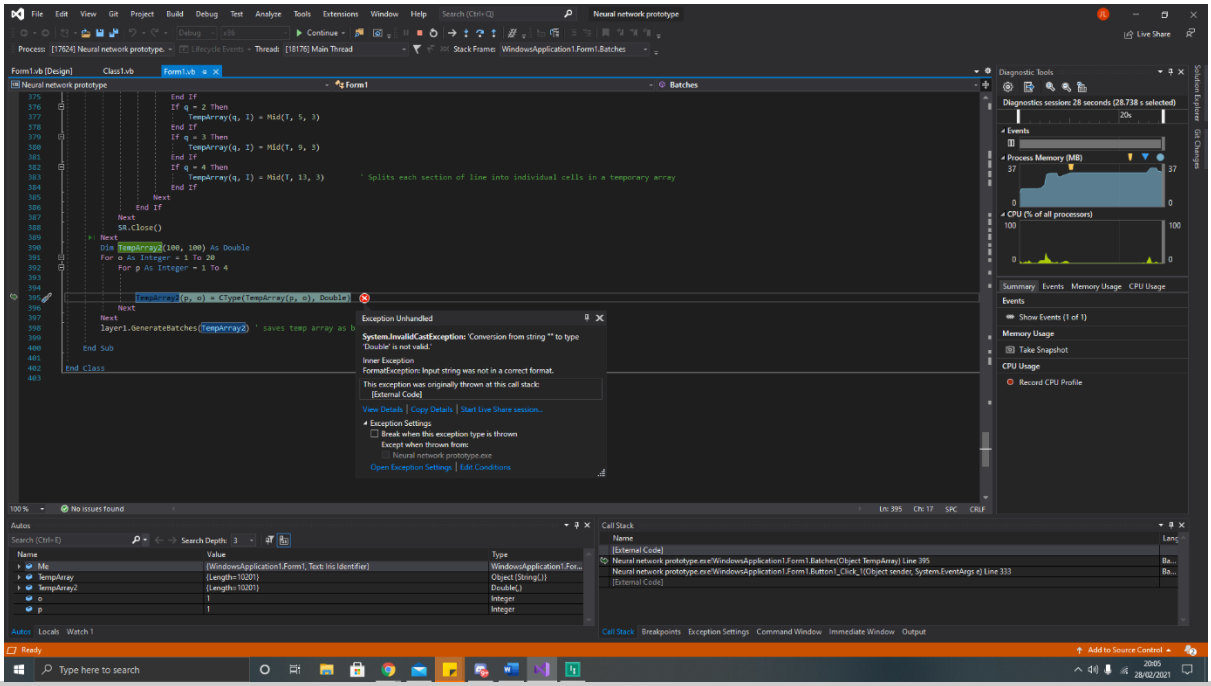
1.3.3	Test file upload system is error free	Selecting a text file **[Typical]**	File location will appear in upload textbox



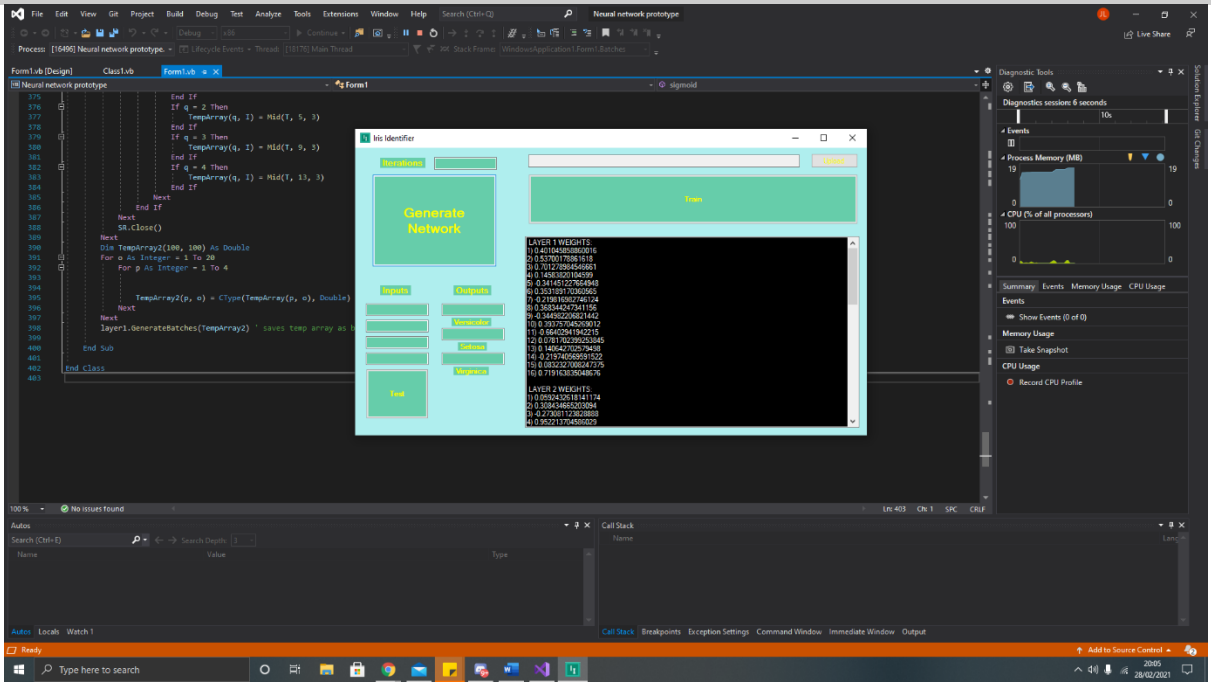
1.3.4	Test if network can train with the iris dataset	Selecting a text file and training **[Typical]**	The network will train outputting corresponding inputs and outputs based on the dataset



1.3.5	Test if network can train with an external dataset	Selecting a text file in the wrong format and training **[Erroneous]**	Network will not train



1.11	Test if user generates network multiple times	generating multiple networks **[Erroneous]**	Each newly generated network replaces the last in the text feed and provides different test results



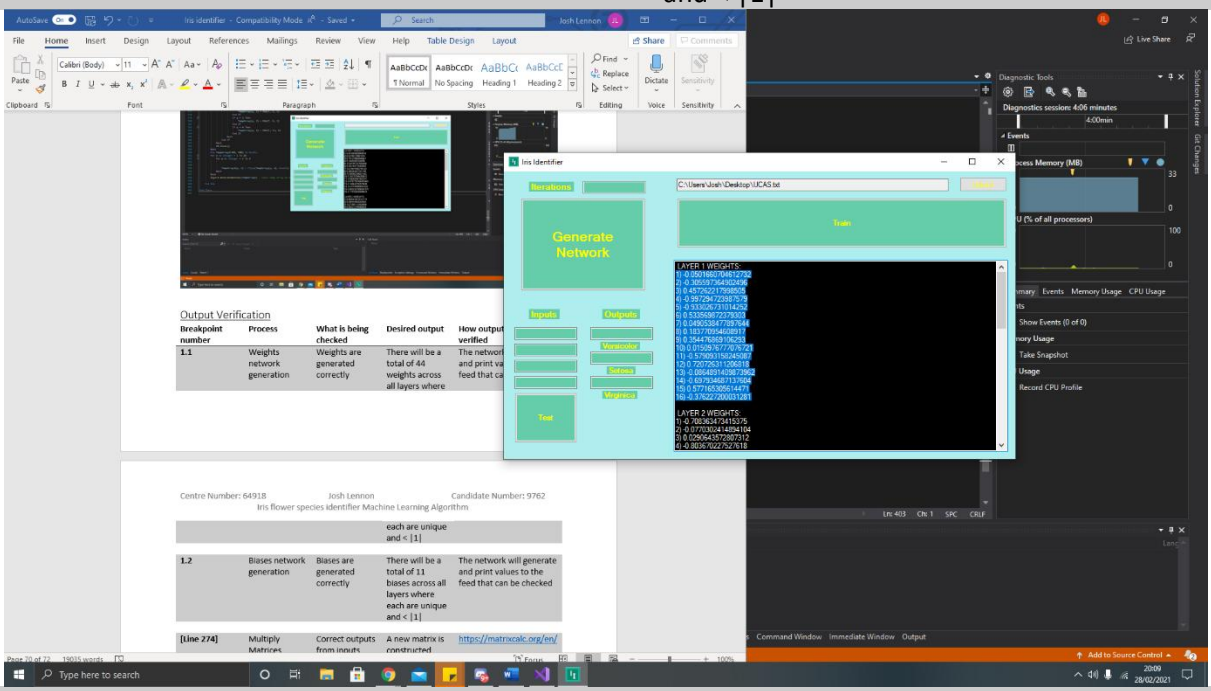
Output Verification

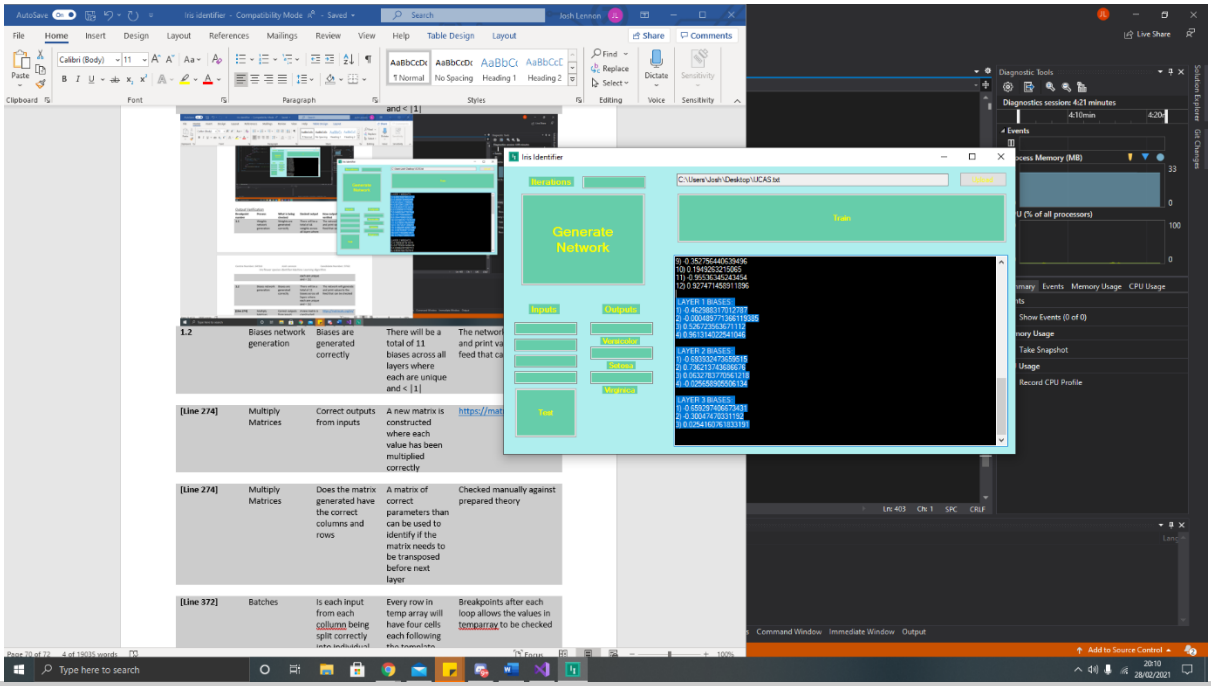
CLICK EACH TESTING IMAGE FOR A VIDEO OF THE CORRESPONDING TESTING PROCESS, A PLAYLIST LINK FOR ALL TESTS IN THIS SECTION CAN BE FOUND HERE:
https://youtube.com/playlist?list=PLSix9uGThuUOZc8j6nWpH4PP0SPfCWTH4

NARRATED VERSION:

https://youtube.com/playlist?list=PLSix9uGThuUN66Fl3UNc2cyDoTHWeUKSc

Breakpoint number	Process	What is being checked	Desired output	How output will be verified
1.1	Weights network generation	Weights are generated correctly	There will be a total of 44 weights across all layers where each are unique and < |1|	The network will generate and print values to the feed that can be checked
1.2	Biases network generation	Biases are generated correctly	There will be a total of 11 biases across all layers where each are unique and < |1|	The network will generate and print values to the feed that can be checked



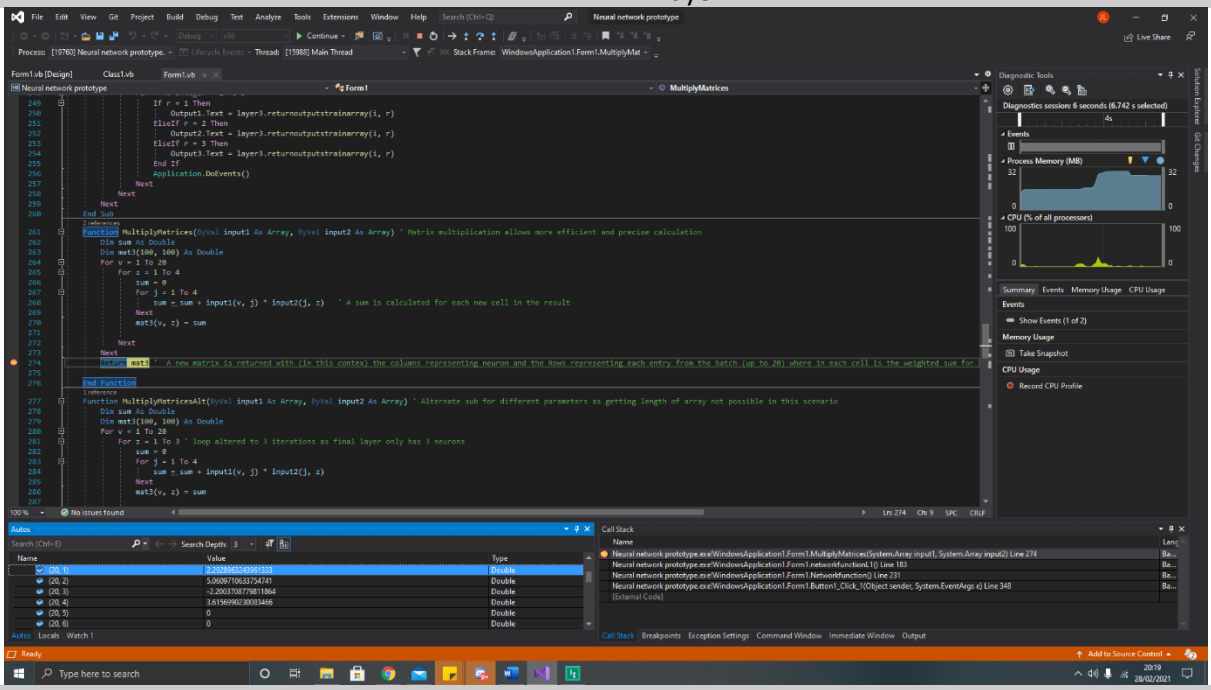


[Line 274]	Multiply Matrices	Correct outputs from inputs	A new matrix is constructed where each value has been multiplied correctly	https://matrixcalc.org/en/

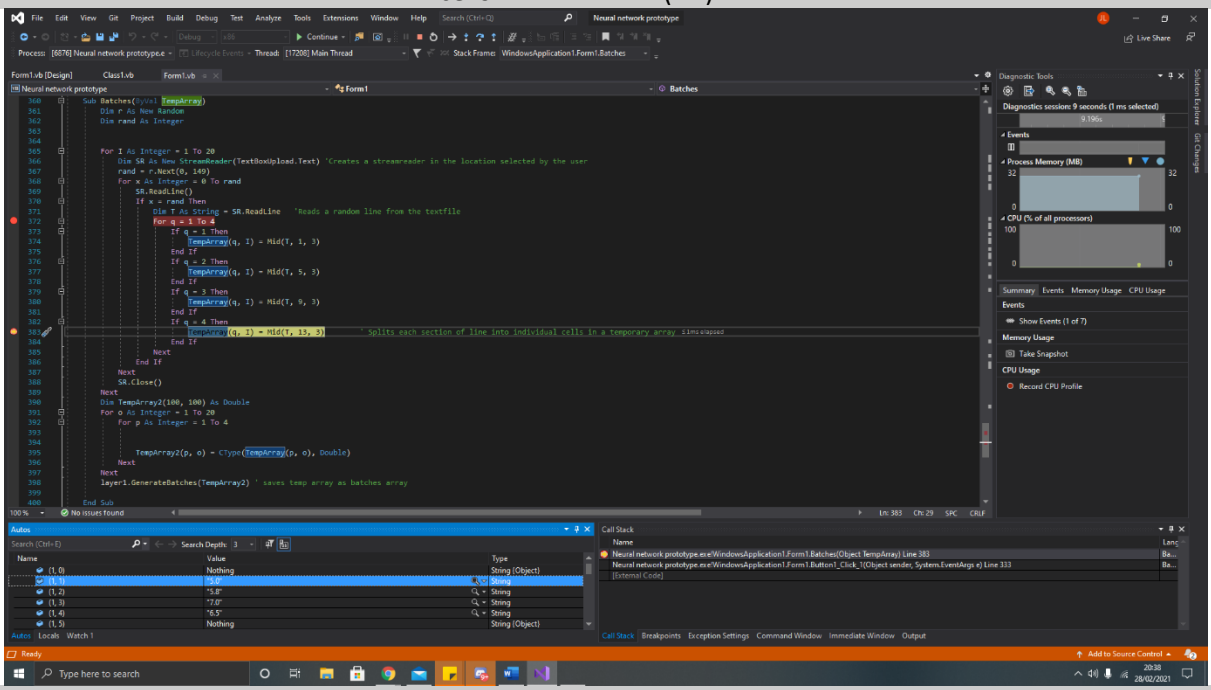


[Line 274]	Multiply Matrices	Does the matrix generated have the correct columns and rows	A matrix of correct parameters than can be used to identify if the matrix needs to be transposed before next	Checked manually against prepared theory

layer

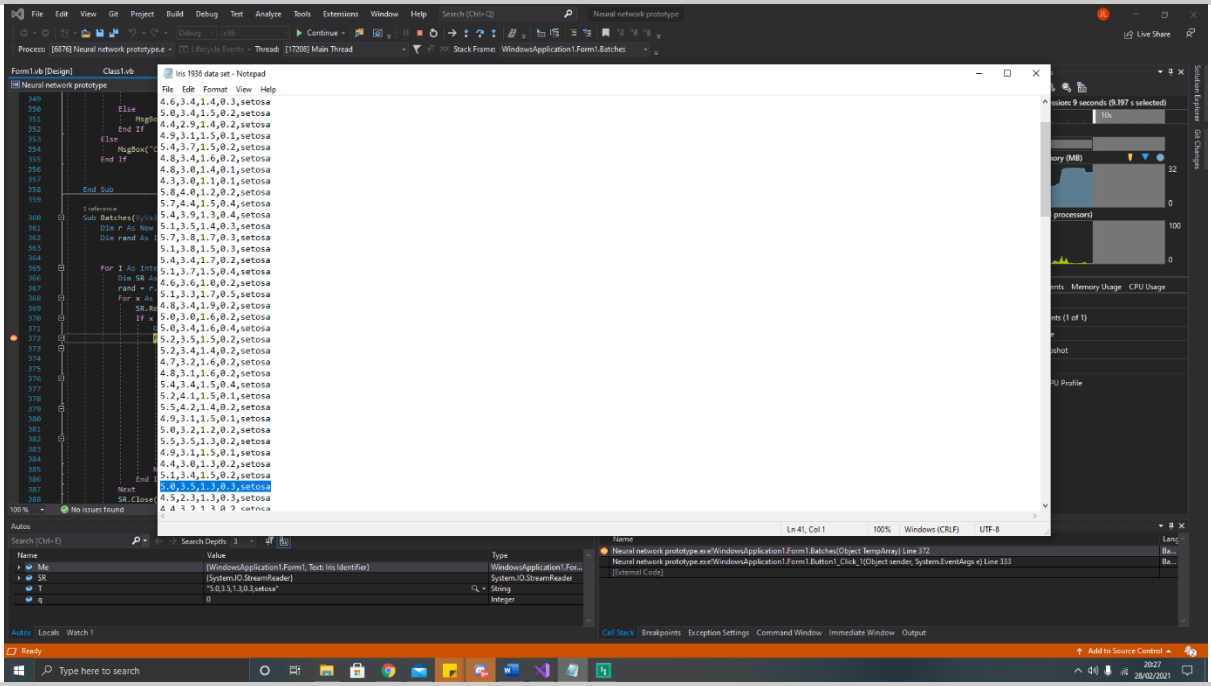


[Line 372]	Batches	Is each input from each collumn being split correctly into individual cells	Every row in temp array will have four cells each following the template (x.x)	Breakpoints after each loop allows the values in temparray to be checked

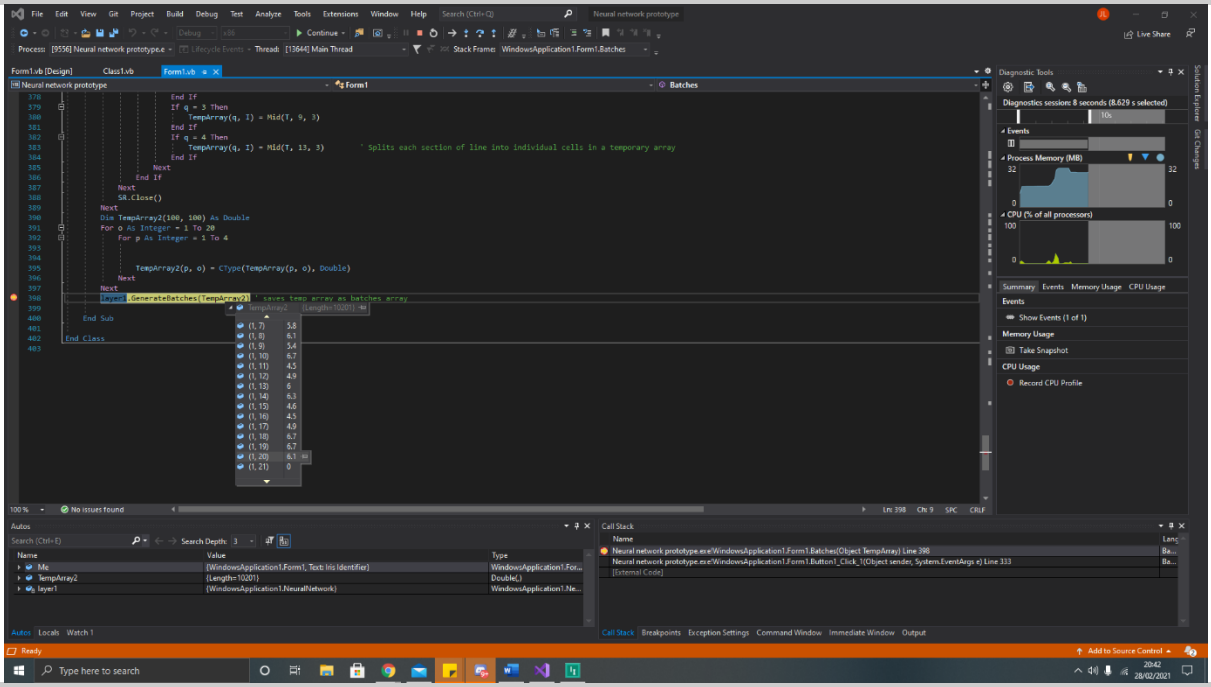


[Line 368]	Batches	Are the inputs selected from the dataset random	Each time a row is complete another random row is selected	Check the value of the random value corresponds to the line selected in the text file + 2 and that this value never exceeds the number of lines in the

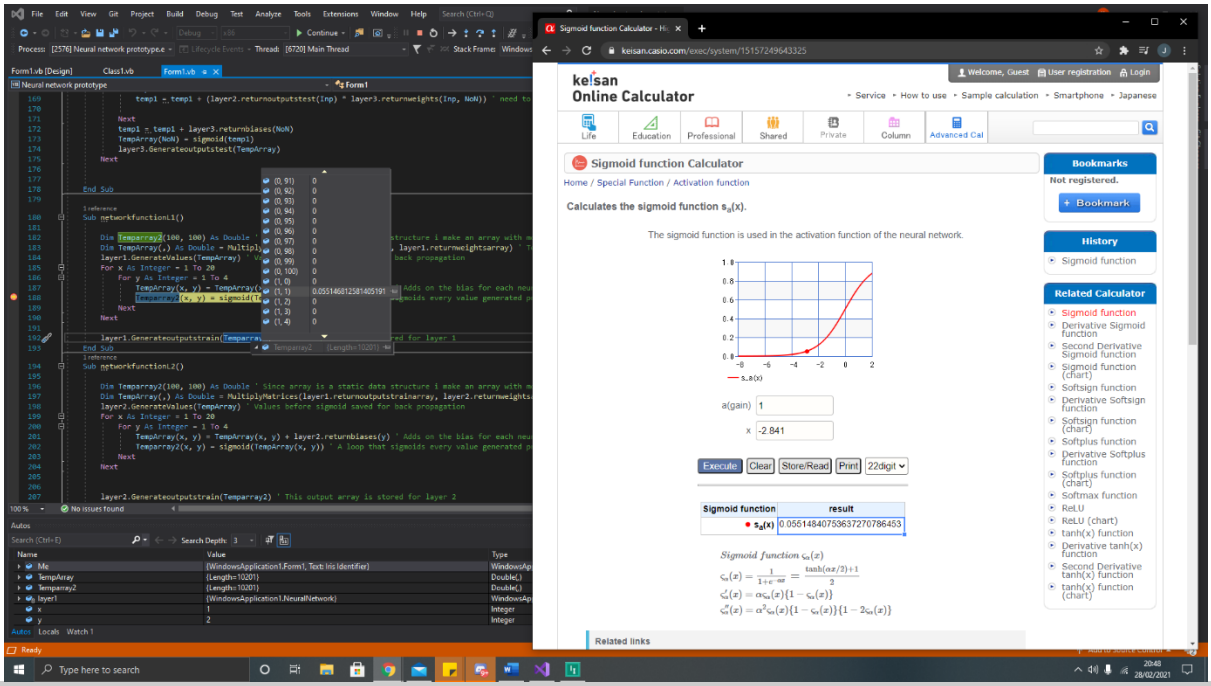
text file



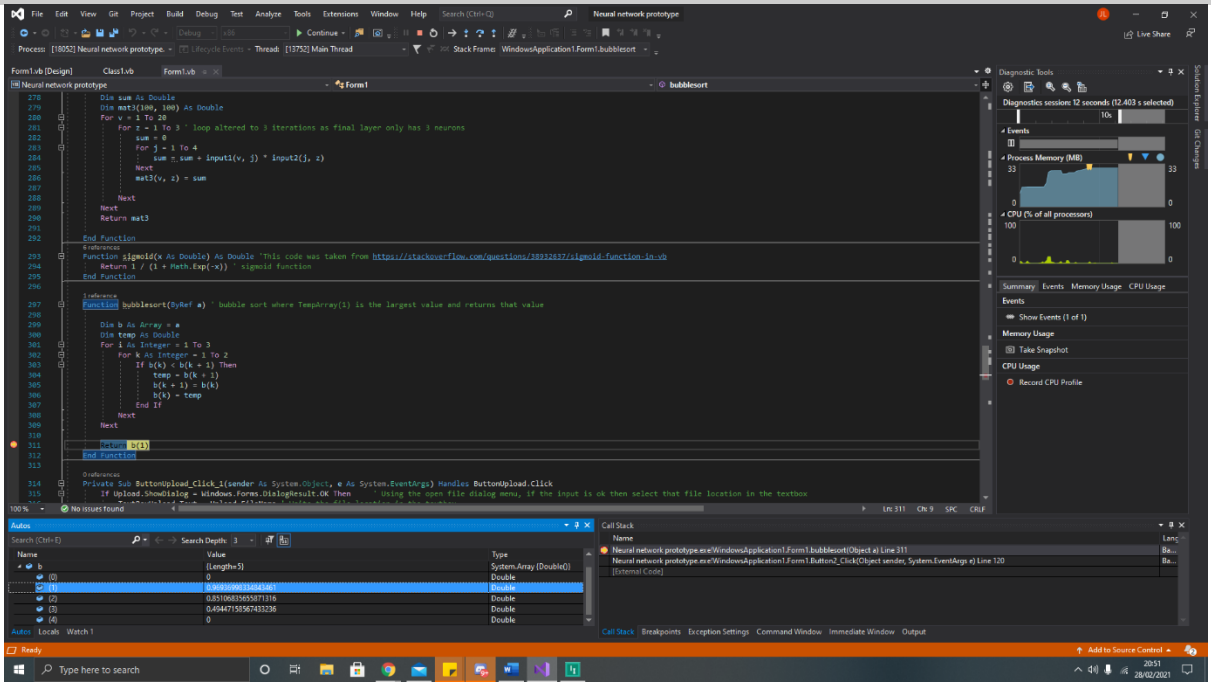
[Line 365]	Batches	20 random lines are being selected and stored appropriately	The array rows should each contain 4 random inputs where there are 20 columns in a batch	Breakpoints after each loop allows the values in temparray to be checked



[Line 188]	Sigmoid	The values should be run correctly through the sigmoid function	Refer to the sigmoid section in design for how value returned must be < |1|	Online sigmoid function



[Line 311]	Bubble sort	Does the bubble sort return the highest confidence	array b(1) should contain the largest value where b(3) contains the smallest	Does array b(1) contain the largest value where b(3) contains the smallest

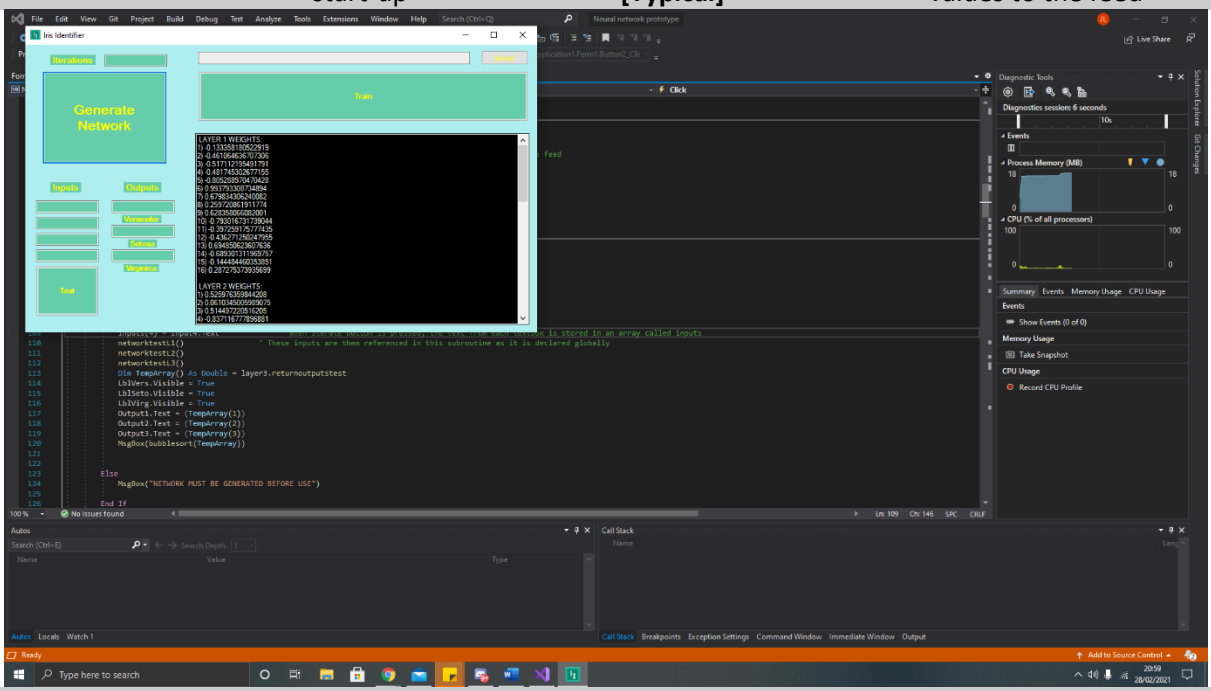


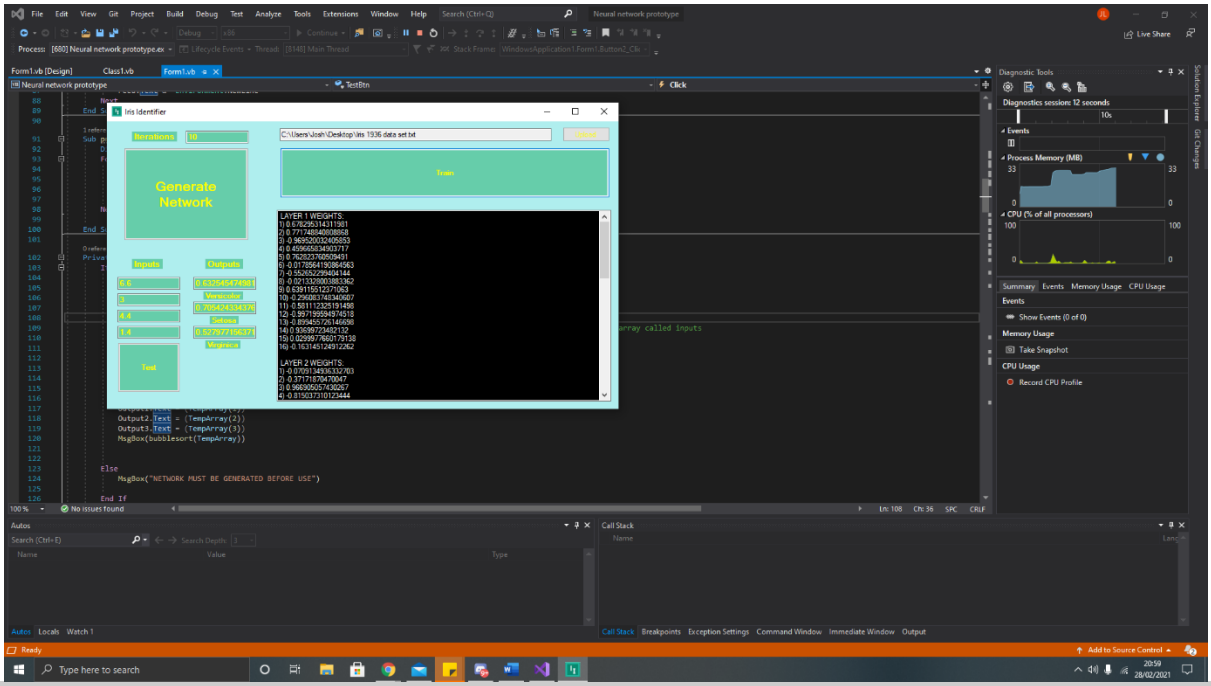
Functionality

A VIDEO LINK FOR ALL TESTS IN THIS SECTION CAN BE FOUND HERE: https://youtu.be/n9MagR-yEXQ

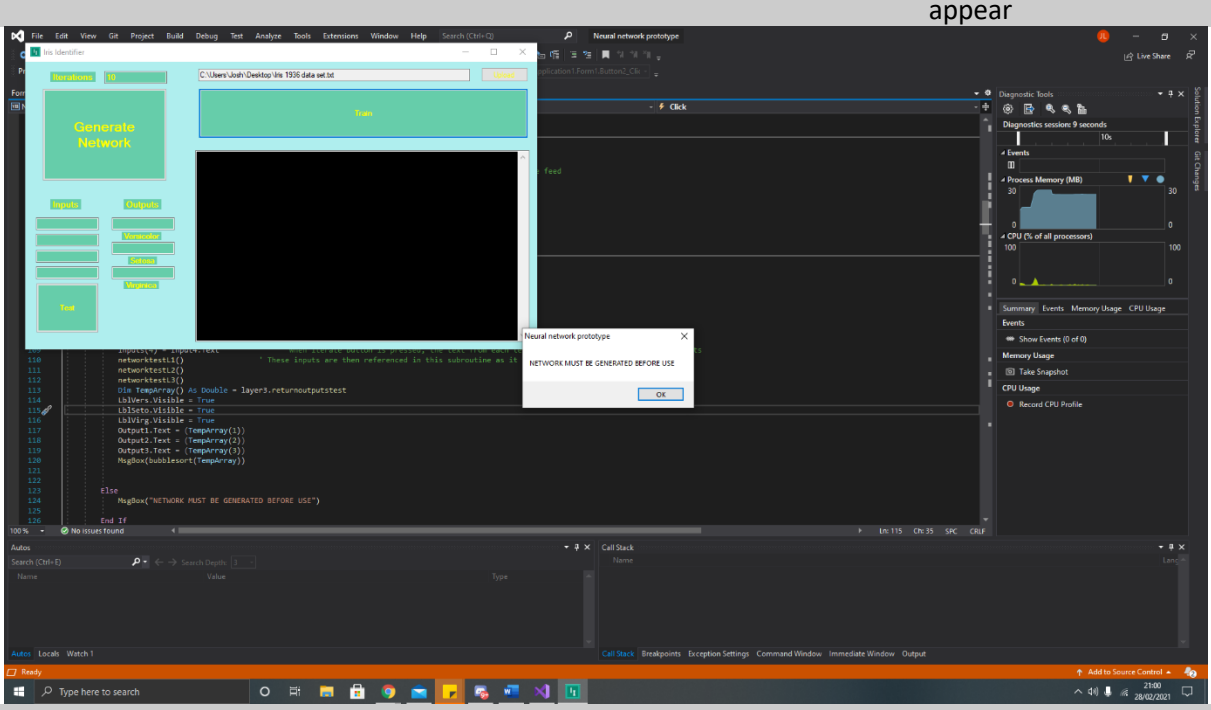
NARRATED VERSION:

https://youtu.be/Ug0VcTh6Mv4

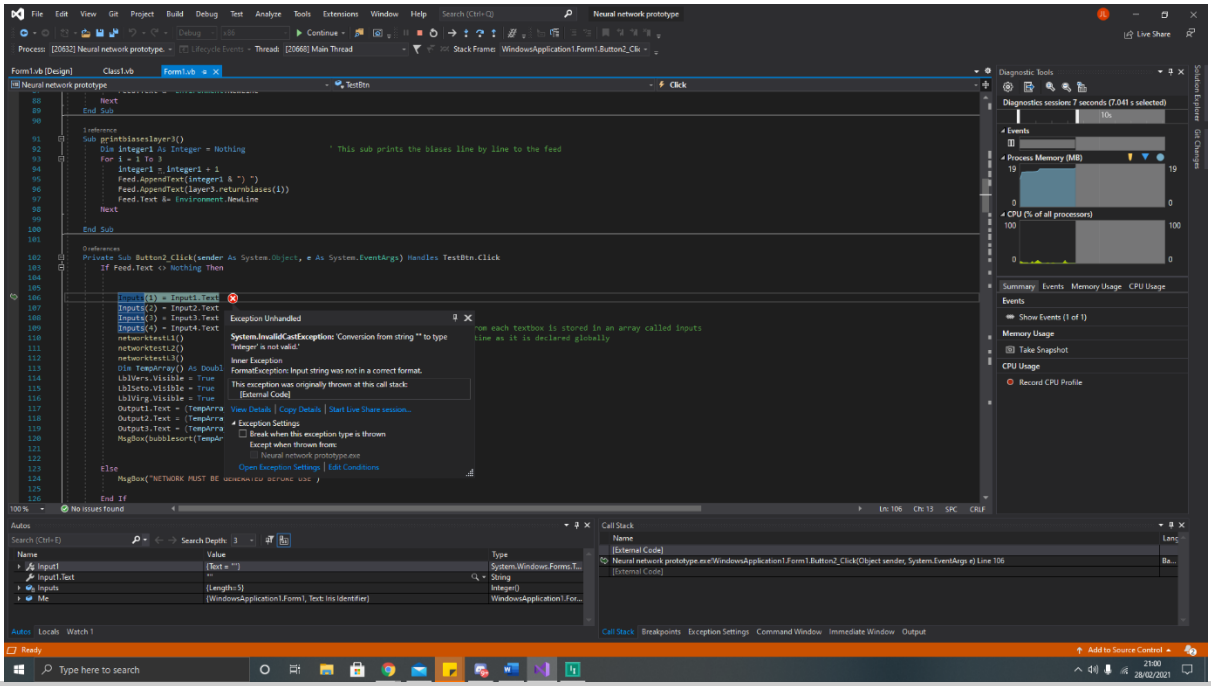
Test number	Description of purpose	Input data and type	Expected result
1.1	Test 'GENERATE NETWORK' button on start-up	being pressed before network is generated **[Typical]**	The network will generate and print values to the feed
			
1.2.1	Test 'Train' button with valid inputs	Being pressed with network generated **[Typical]**	The network will train outputting corresponding inputs and outputs based on the dataset



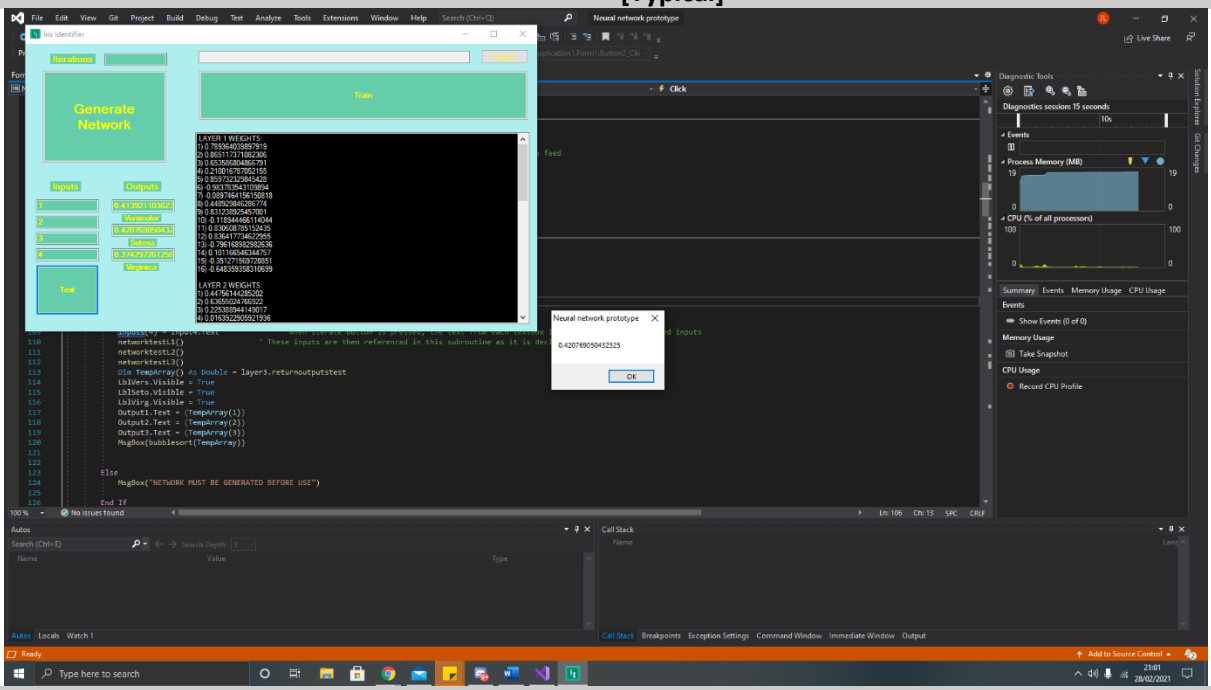
1.2.2	Test 'Train' button with valid inputs	being pressed before network is generated **[Erroneous]**	Message box instructing user to generate network will appear



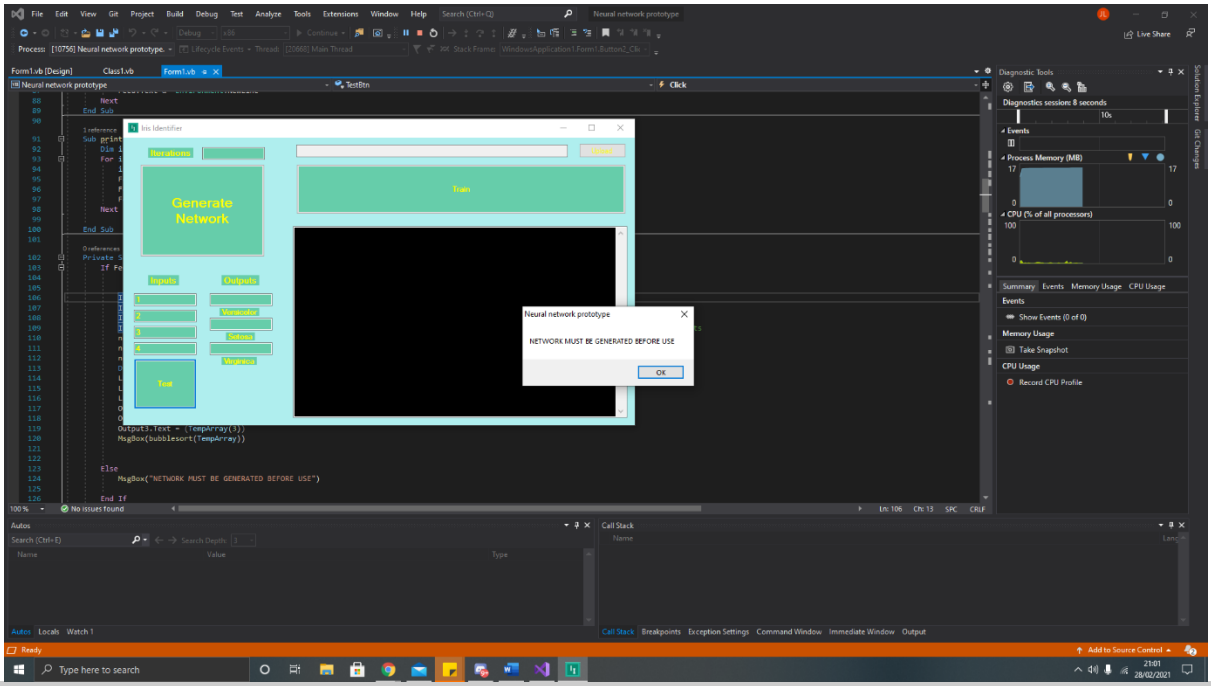
1.3.1	Test 'Test' button With missing inputs	Being pressed after network is generated **[Erroneous]**	Message box instructing user to provide network inputs



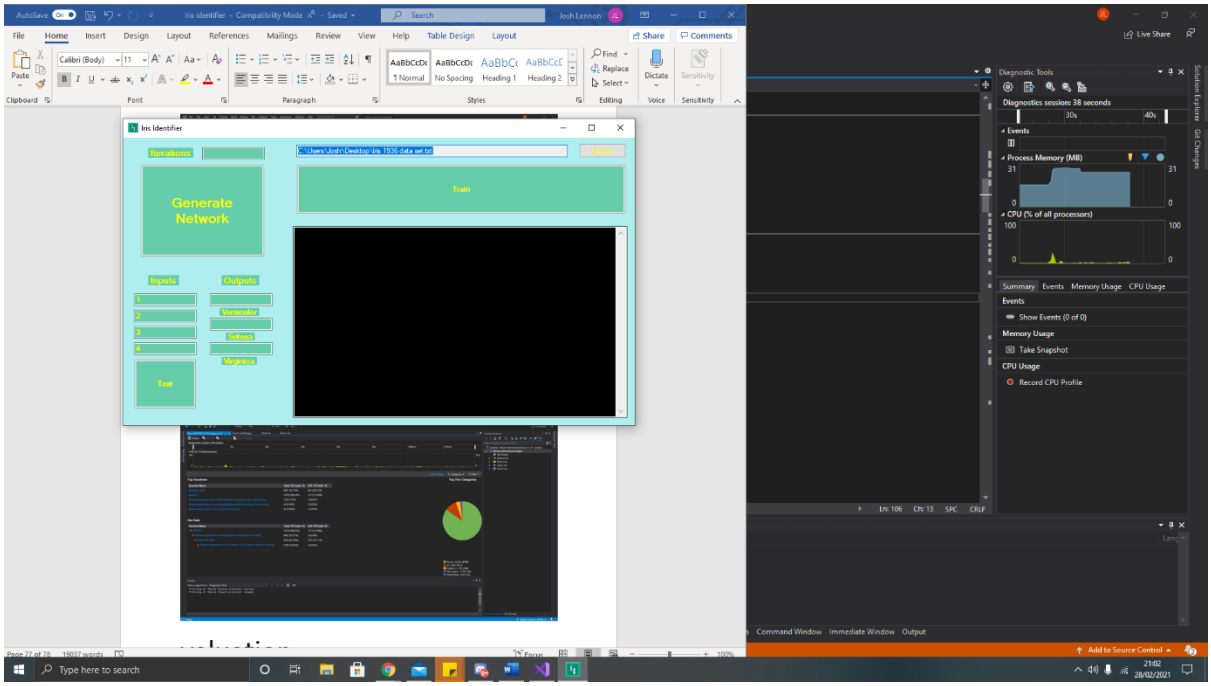
1.3.2	Test 'Test' button with 4 valid inputs	Being pressed with network generated **[Typical]**	3 outputs printed to each output box



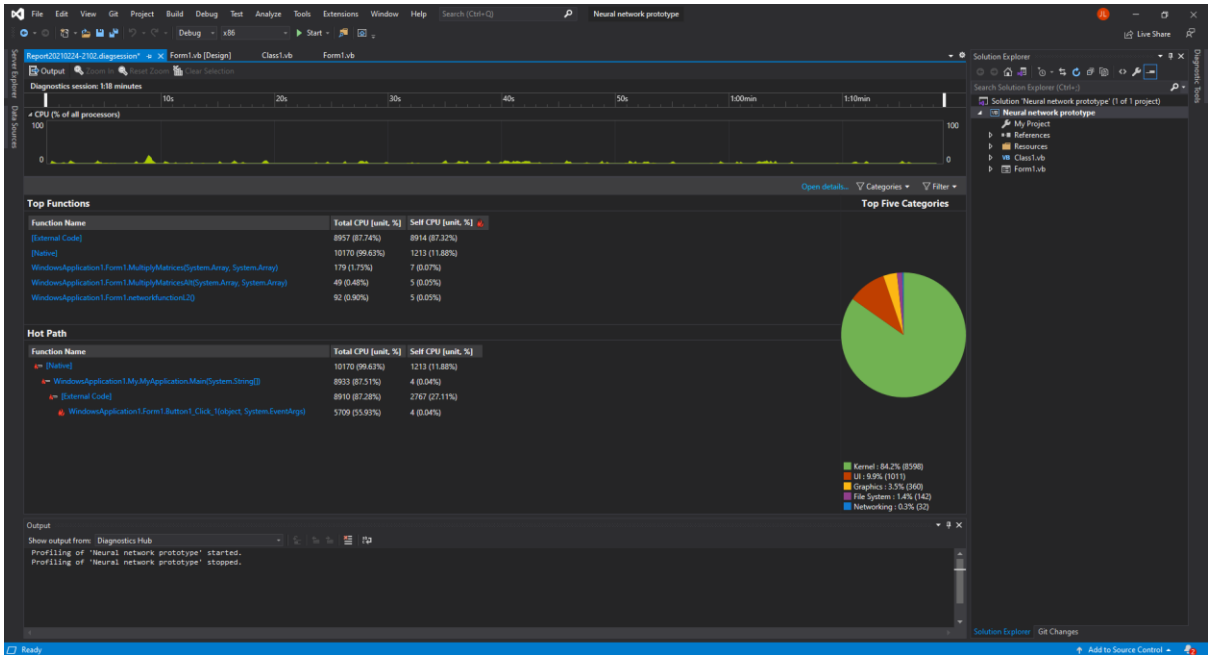
1.3.3	Test 'Test' button with 4 valid inputs	being pressed before network is generated **[Erroneous]**	Message box instructing user to generate network will appear

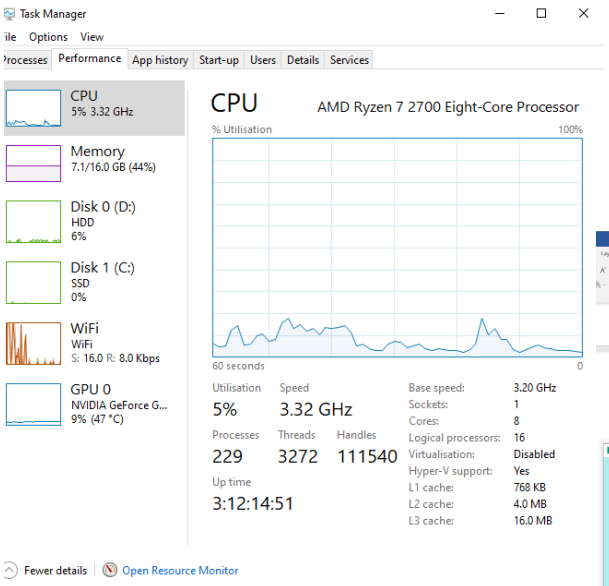


1.4	Test 'Upload' button prints correct file path in upload box	Iris dataset text file uploaded [**Typical**]	Correct file location printed in upload box

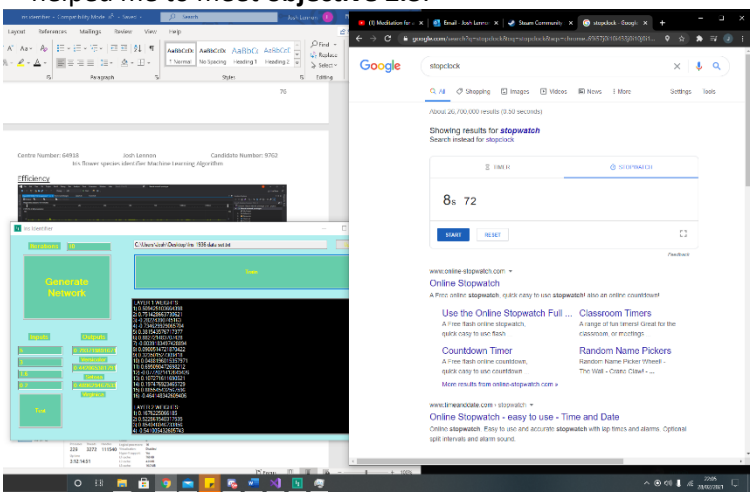


Efficiency





The efficiency of the program is mapped out in two different ways by these windows. We can see a low CPU Utilisation with a peak of 13%. From this testing evidence I can see that the implementation of matrices in my solution helped me to meet **objective 2.3**.



My real time investigations concluded that factoring in human error would result in 8 second for 10 iterations of training of average (results vary). In terms of timings for an intuitive UI, experimenting with users of different knowledge backgrounds in AI and computer science gave a variety of results. The people with a background in computer science had no problem with interpreting the feed after generating a network then both training and testing the network with an average of 51.5 seconds upon first use.

Actions required (as a result of testing)

The testing process resulted in an overall success however it did reveal improvements that could be made to the program to aid the user experience and better meet objectives outlined at the start of the project. Efficiency testing showed that users of the solution with less experience in computer science tended to take longer and occasionally over 2 minutes to interpret the UI. This means use of the program to test and train a network is not as intuitive as I'd hoped when starting the project. Whilst this is not a failure as it meets the original objective to be used by someone with a background in computer science, it could be improved. This could be solved by creating a user manual. A user manual would give the user a better idea of how to get effective use of the program and it's features. Another error was found in the output validation where uploading a text file of a different length would cause a fatal error message. It would be possible to make the custom file option support upload for text files of any length. This would allow the secondary objectives to be met more fully by making the network usable for a custom text file uploaded by the user.

Evaluation

High-level evaluation

Overall, I am happy with the solution produced as it meets initial requirements and is a useful tool for learning and as evidence of an understanding into the complex topic of machine learning. After completing and testing a solution produced on these guidelines, it meets most to all of the primary objectives outlined at the start. The program is a learning tool that effectively propagates forward and allows for quick implementation of back propagation using matrices multiplication.

For my High-level evaluation I'm going to do a full evaluation of the final product and its usability. I will rate the program in 3 categories each created in order to represent the original objectives of the solution. Each category is rated out of 5 based on how well they meet these original objectives. The first category is useability which consists of a combination of user interface review, solution sturdiness and efficiency of use. This category ensures a fair evaluation of the following objectives: 1.1, 1.2 and 1.6. The next category is functionality which will measure how accurately the final solution propagates forward. This category ensures a fair evaluation of the following objectives: 1.2, 1.3 and 1.4. Finally, efficiency is the category that represents how well optimised the program is including wait times / program down times and techniques used to reduce program workload.

Useability

The useability of the program is mainly dictated by how accessible the user interface is for a new user. The final solution has 3 main buttons that operate the network: generate network, train, and test. The generate network button is the largest as a network needs to be generated in order to use any other function and draws the user's attention intentionally for this reason. If the user attempts to press another button before generating the network, testing showed that an appropriate error message will be displayed. In general, the program sturdiness was pretty good however the upload file dialog produces a fatal error when uploading a file that isn't of the correct type. The colour scheme of the interface is a turquoise background with teal buttons and yellow text. This colour scheme was intended to be neutral and undistracting whilst also drawing attention to the feed via contrast in colours. This worked as the feed has a black solid background with white text.

From testing the useability of the solution, I can tell that it meets all the original requirements for efficiency of use. The steps for a user to take are made intuitive as there is not too much information displayed on the screen at once. The final program is easy to use and so I've decided it meets the demands of the solution at the start and evaluate it 5/5.

Functionality

The solution allows users to create a neural network. There is a feed printing weights and biases for each network generated (where only one can be generated at a time) meaning forward propagation is being demonstrated. testing both before and after the network has completed training. The user can test the network with 4 individual inputs that the network has not seen previously, responding with 3 outputs via 3 individual textboxes. This includes a message box that outputs the species the network predicts with the highest confidence. These features of functionality all meet the objectives outlined at the start and forward propagation has been verified through testing to take place 100% accurately meaning this section of the solution can be evaluated at 5/5.

Efficiency

Static data structures mean that the memory allocated to each array needs to be much larger than will ever be used at any given time in each dimension. For this reason, the program uses more memory than anticipated at the start of the project when running. Despite this, techniques such as matrices multiplication allow the solution to remain relatively efficient despite this. This meets the objectives outlined initially where I specified the original methodology of forward propagation would involve matrices multiplication for efficiency. These factor into making the final solution operate as quickly as possible, final loading times are short. There is a short down time when iterating forward and training which is proportional to the number of iterations of the network. Efficiency across the program is good and the workload is minimal when compared with a standard neural network AI, drawing these conclusions I'd give this section a 4/5.

Reflection

Initially I started researching how neural networks propagate forward and evolve. Immediately in my research I identified that there was an absence of resources for someone who was studying neural networks to see how one could be implemented. After researching these processes, I outlined objectives for what the program would need to do in order to meet these needs and be the requires solution.

After seeing how dependant the process of forward propagation is on the use of arrays, I would have instead used python as my programming language of choice. My program was forced to use arrays of size (100,100) despite never using every cell. This is a primary issue with static data structures and meant that memory would be wasted when my program was running. In order to counter this issue Python allows use of dynamic data structures effectively and libraries. Libraries are another important factor into making the programming process more efficient and can make the creation of a neural network through arrays significantly faster, overcoming multiple issues I faced early on due to the platform I was using.

After receiving early feedback from my user, I made the decision to delegate the objective of backpropagation to secondary as full functionality doesn't quantitatively return a user understanding. This process however would not be difficult to implement and with the solution being produced in a way which utilises matrices multiplication and having a pre-generated values array, the process would be a relatively frictionless addition (see extended section).

Objectives met?

In order to get a good measure of how extensively this project meets and solves the original issue I will compare the result to the objective I outlined at the start for my program (where checkmarks are areas that fully meet objective and unchecked are areas that are only partially met):

Primary objectives

[1.1] The solution allows users to follow the processes involved in the creation of a neural network as a learning tool where there is clear notation outlining the order of actions the program is executing. There will be a feed printing weights and biases for each network generated (only one can be generated at a time).

- The solution has a feed that displays the outputs of creation and therefore the processes of a neural network ✓
- Each weight and bias are generated accurately and output in an easy-to-understand way where the weights and biases for each layer are grouped together in order. ✓
- This objective has been fully met in each dimension of the requirements.

[1.2] The solution will be available for testing both before and after the network has completed training. The user can test the network with 4 individual inputs that the network has not seen previously, responding with 3 outputs via 3 individual textboxes. This includes a message box that outputs the species the network predicts with the highest confidence.

- As long as a network has been generated, the network can be tested before or after training and provide 3 unique outputs. ✓
- 4 individual inputs previously unseen by the network will be propagated once through the network forward to provide 3 different confidence outputs for each species. ✓

- A bubble sort algorithm is used to return the highest of the three confidences to the user which can be used to determine the species via the corresponding label. ✓
- This objective has been fully met in each dimension of the requirements.

[1.3] There will be an upload feature where the user will be able to select a text file before training of the network begins. The text file will be read from and utilised as the dataset for network training. A useful visualisation of the network architecture will be displayed to aid the user in selecting an appropriate dataset. While the network is training, the progress will be displayed via corresponding inputs and outputs being displayed in the textboxes on screen.

- There is an upload button that opens a file dialog menu which will allow the user to select any file they want to upload and use as that the dataset for training ✓
- When the network trains each input from the batches / dataset is displayed along with the corresponding output from propagating once through the network with those values ✓
- There is no diagram of network architecture after receiving feedback from my user ✓
- This objective has been fully met in each dimension of the requirements.

[1.4] When a dataset is uploaded by the user, the 'batches' training method will be used where 20 random lines will be read from the dataset text file in order to be used in forward propagation and saved as an array where each input from a line is split into 1 of 4 cells.

- The batches method was implemented via matrices multiplication allowing for large quantities of inputs to be run through the network at one time ✓
- Testing verified that 20 random lines are read from the dataset text file in order to be used for forward propagation ✓
- The batches are saved as an array where the previous testing section also demonstrated how each input from a line of the dataset text file is split into 1 of 4 cells in the array ✓
- This objective has been fully met in each dimension of the requirements.

[1.5] Forward propagation must take place in order to generate outputs where this process can be iterated a user specified number of times

- Forward propagation is used to generate outputs for the network through use of matrix multiplication using arrays as a data structure ✓
- The static neural network structure diagram was replaced by a textbox allowing the user to enter a custom parameter for the number of times the network should iterate through a new batch ✓
- This objective has been fully met in each dimension of the requirements.

[1.6] The user should be able to figure out how to use the program in under a minute and interpret the program with intuition given a background in both computer science and the theory behind how neural networks operate.

- My main perspective user was able to use the program to its intended function within a minute of using it for the first time meaning the user interface wasn't too complex ✓
- There was a clear colour scheme for the program where the generate network button was the largest feature with the largest text since it is the most important part of the network. This helps the program to be interpreted with intuition with a background in theory ✓
- This objective has been fully met in each dimension of the requirements.

Secondary objectives

The secondary objectives for my program outline optional features that go beyond the demands of A-Level but are required for a fully functioning AI as opposed to a learning tool.

[2.1] The program will initialise a network with random weights and biases and conclude with a network that gives accurate outputs. The inputs will be fed through the network and return several values for each node. The values produced in forward propagation will then be utilised in the backpropagation process to successfully alter the weight and bias of each node.

- A network can be generated with random weights and biases and returning outputs that are accurate for the network generated. The inputs return a weighted sum value for each neuron ✓
- Value arrays are produced however they are not used in the backpropagation process, so weights and biases are not altered throughout the training process however the network is propagated forward generating the correct values in the correct format to be used later for backpropagation
- This objective has been mostly met in each dimension of the requirements, excluding the use of values in backpropagation.

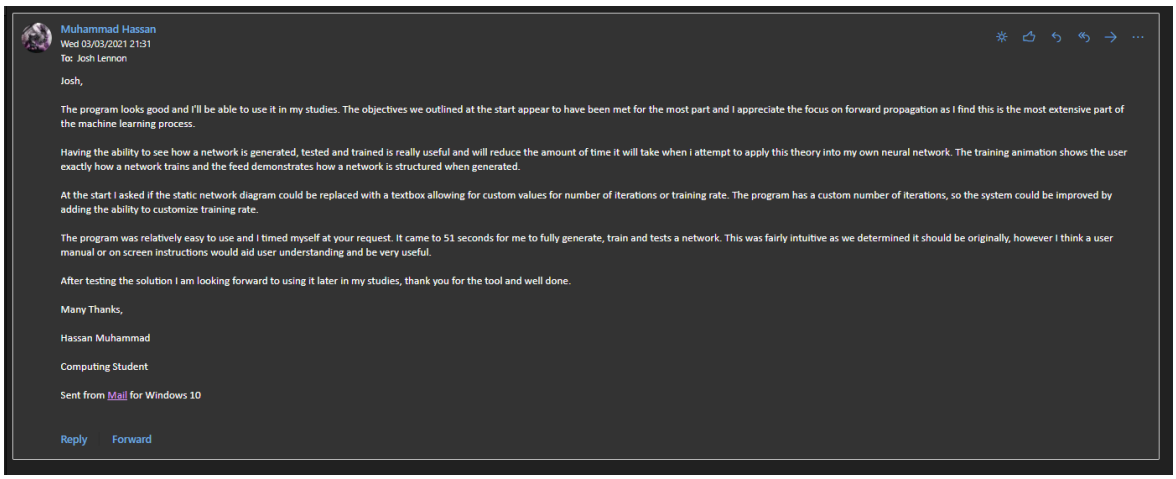
[2.2] The program will be able to identify the species of iris flower accurately from 4 inputs when tested. In theory, upon initialisation the program should be able to train a network with any dataset that matches the chosen architecture. This means it can be applied in other contexts to identifying species of iris flower.

- A network can be generated with random weights and biases and returning outputs that are accurate for the network generated. The inputs return a weighted sum value for each neuron ✓
- The network is able to train with an uploaded custom dataset, however the dataset must be of the exact same parameters and must be 150 lines in length
- The network does not adjust weights and biases when training and so will not be able to accurately predict the species of iris flower. Only forward propagation takes place as this is a tool in order to aid understanding of how a neural network can be implemented as a program.
- This objective has been partially met in each dimension of the requirements, excluding the adjustment of weights and biases as well as training custom datasets of all parameters.

[2.3] To ensure good performance, the feed outputting each weight and bias for each node will be updated via a 'refresh' button available to the user. The solution will also utilise matrices multiplication in frequent when propagating forward. This means as weights and bias are progressively altered in the training process, the change in each will only be displayed at the user's request.

- There is a feed that is static for each time a network is generated without a refresh button as the feed is never updated
- The solution uses matrices in every process of forward propagation to ensure good program performance and to prepare the program for simple implementation of backpropagation ✓
- Weights and biases remain static after the network is generated and when values are propagated through the network the outputs aren't used to train the weights and biases
- This objective has been fully met in each dimension of the requirements.

Feedback



Muhammad Hassan
Wed 03/03/2021 21:31
To: Josh Lennon

Josh,

The program looks good and I'll be able to use it in my studies. The objectives we outlined at the start appear to have been met for the most part and I appreciate the focus on forward propagation as I find this is the most extensive part of the machine learning process.

Having the ability to see how a network is generated, tested and trained is really useful and will reduce the amount of time it will take when i attempt to apply this theory into my own neural network. The training animation shows the user exactly how a network trains and the feed demonstrates how a network is structured when generated.

At the start I asked if the static network diagram could be replaced with a textbox allowing for custom values for number of iterations or training rate. The program has a custom number of iterations, so the system could be improved by adding the ability to customize training rate.

The program was relatively easy to use and I timed myself at your request. It came to 51 seconds for me to fully generate, train and tests a network. This was fairly intuitive as we determined it should be originally, however I think a user manual or on screen instructions would aid user understanding and be very useful.

After testing the solution I am looking forward to using it later in my studies, thank you for the tool and well done.

Many Thanks,

Hassan Muhammad

Computing Student

Sent from Mail for Windows 10

Reply Forward

'Josh,

The program looks good and I'll be able to use it in my studies. The objectives we outlined at the start appear to have been met for the most part and I appreciate the focus on forward propagation as I find this is the most extensive part of the machine learning process.

Having the ability to see how a network is generated, tested, and trained is really useful and will reduce the amount of time it will take when I attempt to apply this theory into my own neural network. The training animation shows the user exactly how a network trains and the feed demonstrates how a network is structured when generated.

At the start I asked if the static network diagram could be replaced with a textbox allowing for custom values for number of iterations or training rate. The program has a custom number of iterations, so the system could be improved by adding the ability to customize training rate.

The program was relatively easy to use and I timed myself at your request. It came to 51 seconds for me to fully generate, train and test a network. This was fairly intuitive as we determined it should be originally, however I think a user manual or on-screen instructions would aid user understanding and be very useful.

After testing the solution, I am looking forward to using it later in my studies, thank you for the tool and well done.

Many Thanks,

Hassan Muhammad

Computing Student'

Extended

In order to deduce in which direction, I would want to develop and extend this solution, I reflected using test data, user feedback and reflection. Test data revealed simple improvements that I have been able to implement for reduced friction from the user experience and result hopefully in a more effective overall solution. I can look at the feedback from the intended target demographic and create my own analysis of improvements to be made.

Backpropagation

A straightforward and obvious point to develop is to implement back propagation into the training process. This would mean the solution would meet all secondary objectives outlined at the start. However, implementation of this goes significantly beyond the A level specification. The program already has the values array set up in order to calculate delta and it wouldn't be hard to convert the species into binary values to compare to each output. This would give variables to use in the formula to calculate error: Target – Output (see backpropagation section). Back propagation would mean the UI would need to change in order to stay intuitive. For this, I can add a refresh button as outlined in objective 2.3. The training animation would remain the same, however each time the refresh button is pressed, the weights and biases printed into the feed will change value. Changing weights and biases also means that the network should provide realistically accurate outputs after training and be useable as a tool to identify iris species.

Learning rate

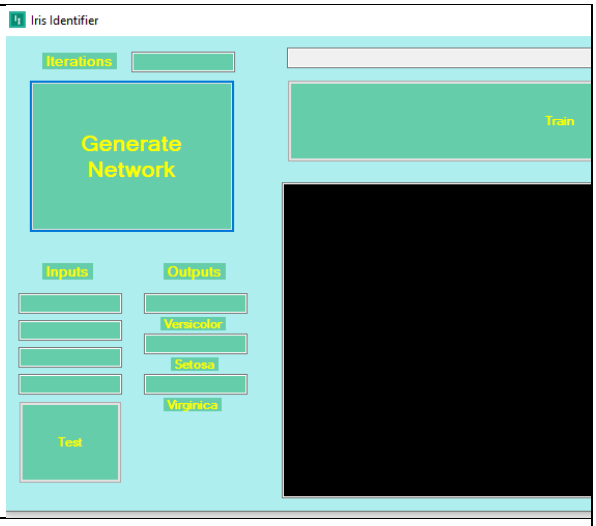
Another thing that could be implemented as an extension of the original project solution is to implement learning rate. This builds on top of backpropagation and allows the user to set a custom parameter which determines how much or how little the weights and biases should be altered in relation to the scale of the error value produced from forward propagation. The further away a network output is from the desired output the larger error value. This is where learning rate comes in and decides how much the weights and biases should be altered based on this value i.e., weighting the error value. A large value will mean training may be faster however too large means that the training process may be slower. A smaller value may take longer but be more accurate however so having the user input a custom parameter allow them to experiment with different values in order to find an optimal solution.

User manual

The program is an effective learning tool especially for users with a background in extensive mathematics, computer science and neural network theory. To make the program more accessible (which would be ideal) I could produce and implement a user manual as either a separate document or a separate form explaining either basic neural network theory, or how the program works. This would further reduce the time it takes a user to generate and train a network making the UI more intuitive and further meeting the objectives outlined at the start of the project.

Appendix A - Mapping

OBJECTIVE NUMBER	DESIGN	TESTING	EVALUATION	CODE
1.1	Page 23, 28	Pages 66, 67, 69 - 83	Page 88	Pages 49 - 51
1.2	Page 24, 30, 33	Pages 67, 69 - 83	Page 86, 88	Pages 56-58
1.3	Page 23	Pages 67, 69 - 83	Page 86, 89	Pages 52 - 54
1.4	Page 27	Pages 66, 69 - 83	Page 89	Page 51 - 53
1.5	Pages 25 - 26, 29 - 31, 34	Pages 66, 67, 69 - 83	Page 89	**Page 54 - 56**

				
1.6	Page 40, 36, 25	Page 68, 84	Page 87, 89	N/A
2.1	Page 28, 32 – 33, 42		Page 90	
2.2	Page 25 - 27		Page 90	
2.3	Page 25 - 27		Page 90	

END PAGE

(User manual would be a separate document)