|  |  |
| --- | --- |
| Apollo  A-Level NEA Project | Abstract  Creating music with LSTM networks  Jenson Cain  Computer Science 2023 |

**Contents**

[Analysis 5](#_Toc130382338)

[Hypothesis 5](#_Toc130382339)

[End User and Target Audience 5](#_Toc130382340)

[How was research conducted? 7](#_Toc130382341)

[Sources 7](#_Toc130382342)

[The Basics of a Neural Network 8](#_Toc130382343)

[How is a Neural Network trained? 9](#_Toc130382344)

[Why is this not done manually? 9](#_Toc130382345)

[The Cost Function 10](#_Toc130382346)

[Gradient Descent and Backpropagation 10](#_Toc130382347)

[Neural Network models used for generation 14](#_Toc130382348)

[Generative Adversarial Networks (GANs) 14](#_Toc130382349)

[Recurrent Neural Networks (RNNs) 14](#_Toc130382350)

[Transformer-Based Models 16](#_Toc130382351)

[Autoencoders 17](#_Toc130382352)

[Existing Solutions 17](#_Toc130382353)

[AIVA 17](#_Toc130382354)

[MuseNet 20](#_Toc130382355)

[Initial Modelling 21](#_Toc130382356)

[Data Flow Diagram 21](#_Toc130382357)

[Class Diagram 22](#_Toc130382358)

[Objectives for the Project 23](#_Toc130382359)

[Design 25](#_Toc130382360)

[Activation Functions 25](#_Toc130382361)

[The Sigmoid Function 25](#_Toc130382362)

[The Derivative of the Sigmoid Function 25](#_Toc130382363)

[The Derivative of Tanh 25](#_Toc130382364)

[The Softmax Function 25](#_Toc130382365)

[Matrices 25](#_Toc130382366)

[Element-wise Operations 26](#_Toc130382367)

[Specific Operations 26](#_Toc130382368)

[Constructors and creating a Matrix Object 28](#_Toc130382369)

[Weight Class 29](#_Toc130382370)

[Reading and Writing MIDI Files 29](#_Toc130382371)

[Converting MIDI to a string 30](#_Toc130382372)

[Converting a string into MIDI 31](#_Toc130382373)

[Writing Log Files 32](#_Toc130382374)

[Neural Network 32](#_Toc130382375)

[Forward Propagation 33](#_Toc130382376)

[Backwards Propagation 34](#_Toc130382377)

[Vocab Class 38](#_Toc130382378)

[Saving State 38](#_Toc130382379)

[Profile Class 40](#_Toc130382380)

[Profile Manager Class 40](#_Toc130382381)

[NeuralNetwork Class 40](#_Toc130382382)

[Top-down Diagram and Interface 41](#_Toc130382383)

[Program Start Up 42](#_Toc130382384)

[Training State 43](#_Toc130382385)

[Generation State 47](#_Toc130382386)

[Music Player 50](#_Toc130382387)

[Settings State 51](#_Toc130382388)

[Class Diagrams 54](#_Toc130382389)

[Neural Network Class Design 54](#_Toc130382390)

[File Management Class Design 55](#_Toc130382391)

[Frontend Class Design 55](#_Toc130382392)

[Technical Implementation 57](#_Toc130382393)

[(Apollo.IO) Reading and writing files 57](#_Toc130382394)

[The LogManager Class 57](#_Toc130382395)

[The Note Struct 59](#_Toc130382396)

[The MidiManager Class 60](#_Toc130382397)

[The TrainingFileManager Class 65](#_Toc130382398)

[(Apollo.MatrixMaths) Matrix arithmetic and operations 67](#_Toc130382399)

[The ActivationFuncs Class 67](#_Toc130382400)

[The Matrix Class 69](#_Toc130382401)

[Exceptions File 81](#_Toc130382402)

[(Apollo.NeuralNet) The Neural Network 82](#_Toc130382403)

[AdamParameters Class 82](#_Toc130382404)

[The Weight Class 82](#_Toc130382405)

[The Gate Class 84](#_Toc130382406)

[The LSTM Class 86](#_Toc130382407)

[The RNN Class 89](#_Toc130382408)

[Vocab Class 96](#_Toc130382409)

[Profile Class 98](#_Toc130382410)

[The NeuralNetwork Class 99](#_Toc130382411)

[Exceptions 104](#_Toc130382412)

[(Apollo) The Graphical User Interface 105](#_Toc130382413)

[StoredSettings class 105](#_Toc130382414)

[ProfileManager Class 107](#_Toc130382415)

[AssemblyInfo.cs 111](#_Toc130382416)

[App 111](#_Toc130382417)

[Startup Window 113](#_Toc130382418)

[Main Window 115](#_Toc130382419)

[Training Page 118](#_Toc130382420)

[Creation Page 121](#_Toc130382421)

[Queue.cs 124](#_Toc130382422)

[Stack.cs 125](#_Toc130382423)

[Listen Page 126](#_Toc130382424)

[Text Dialog 133](#_Toc130382425)

[Settings Page 134](#_Toc130382426)

[Testing 146](#_Toc130382427)

[Manual Test Table 146](#_Toc130382428)

[Unit Tests 155](#_Toc130382429)

[Apollo.MatrixMaths.Tests 155](#_Toc130382430)

[Apollo.NeuralNet.Tests 163](#_Toc130382431)

[Apollo.Tests 164](#_Toc130382432)

[Video Contents Table 167](#_Toc130382433)

[Evaluation 168](#_Toc130382434)

[User Questionnaire 168](#_Toc130382435)

[Section 1: Generation Quality 168](#_Toc130382436)

[Section 2: Listen Page 169](#_Toc130382437)

[Section 3: Training Page 172](#_Toc130382438)

[Section 4: Generation Page 174](#_Toc130382439)

[Section 5: Settings Page 177](#_Toc130382440)

[Section 6: Overall Evaluation 180](#_Toc130382441)

[Self-Evaluation 181](#_Toc130382442)

[Objective 1 181](#_Toc130382443)

[Objective 2 182](#_Toc130382444)

[Objective 3 182](#_Toc130382445)

[Objective 4 182](#_Toc130382446)

[Final Comments 183](#_Toc130382447)

# Analysis

## Hypothesis

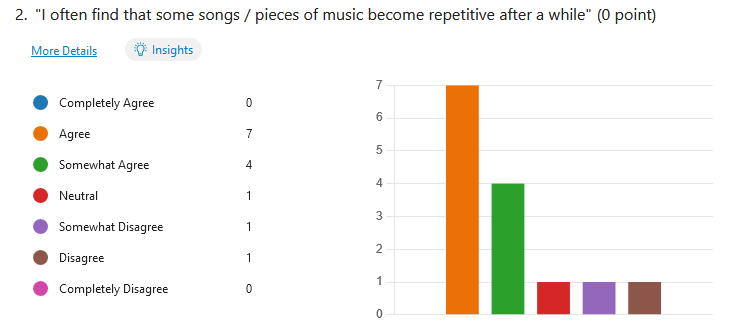
A neural network is a deep learning technique, which could be considered as a very basic, mathematical depiction of how scientists currently believe our brains work. On a very abstract level, (this will be discussed in further detail in the research section), neural networks are able to “learn from their mistakes”. They do this by taking an input, performing mathematical calculations on said input to get an answer which is then outputted, and compared to the real answer. The difference between the two answers is what is used to adjust the variables used in the calculations to make them increasingly accurate. As such, neural networks are very popular for solving intractable problems (problems which cannot be solved by algorithms with time complexity O(n2) or less).

On the contrary, music is (as defined by Henry Wadsworth Longfellow, an American poet and educator during the 19th Century) “the universal language of mankind”. This is because music can range from very simple (i.e., a few notes) to very complex because it can have a range of features such as key changes and beat drops which are only effective when they are timed well by the composer. Thus, I have decided to investigate how neural networks function and how a neural network could be created and adapted to generate content, with the aim of being able to use what I have investigated and learnt to create a neural network that generates music.

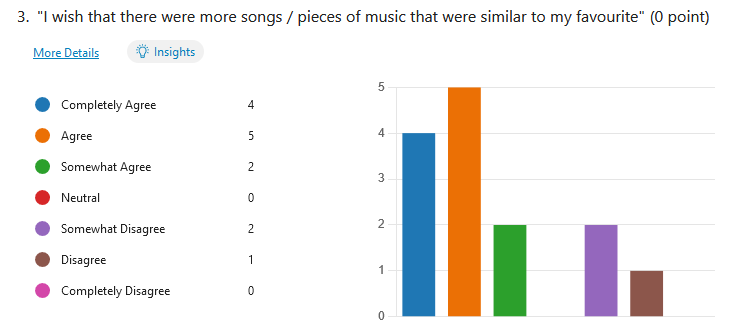
## End User and Target Audience

This project will target a wide audience due to its simplicity. However, it will be ideal for those who have an interest in users as well as those who work on indie projects. This is because indie projects are created by very small teams (as small as 1 person) and thus it is possible that nobody on the team has musical expertise. Therefore, this project will allow them to cheaply create music without the need to hire a composer or the need to figure it out for themselves.

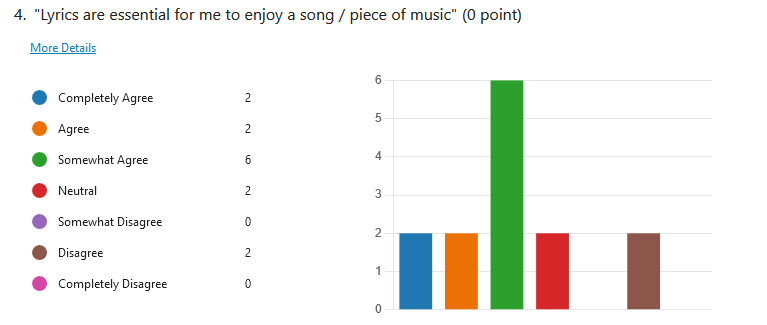
However, beyond very specific examples it is difficult to define who the program is created for, so I wrote a questionnaire that asks questions about music consumption but also people’s general experiences with applications.

The first section of the questionnaire, about music consumption, was useful in telling me about how the average person would find software like Apollo useful. 

The majority of people who took part in the survey stated that they find that music can become repetitive after a while. This is the problem that Apollo helps to solve since not only will it be able to generate *new* music, but it could also generate music that fits the music taste of the consumer.

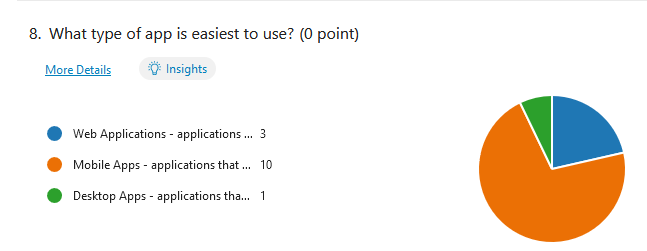


To extend my point about the last question about generating music similar to the user’s music taste. This is a feature that would be well perceived by the end users, since if they were able to pick to train the AI trained on their music taste, they would be able to generate music that they would like to listen to and thus find the software more engaging.



The response to the above question was disappointing to see since I am aware that the neural network will not be complex enough to create new pieces of music with lyrics, since creating an artificial voice to sing the lyrics would require a separate neural network which is out of the scope for this project.

On the bright side, the response to the question has also given me potential test users since those who find lyrics essential for them to enjoy a song would be more likely to be disappointed by the software and may potentially base their entire feedback on the fact that the generated music does not have lyrics. Therefore, it may be advantageous to target those who answered “Neutral” or “Disagree” to this question as test users since they are more likely to offer constructive criticism of the program.



Adding a GUI as a frontend for the AI was something that I intended to do since thinking of the project, however I was unsure how to implement this. That’s to say whether to make a web application, a very simple mobile application or something that runs natively on a computer. Of course, most people stated that they find it easier to use a mobile app, since this is what society is used to and what people use the most. However, on a later question I asked, “What was your reasoning behind your answer for question 8?” and those who stated that they preferred mobile apps all agreed on the fact that they preferred them due to “simple layouts” that make them “easy to use”, as well as mobile apps being generally more accessible.

At this stage, I would say that a desktop application would work best for this software since well-made mobile apps (or web applications optimised for mobile devices) are time consuming to make, and most of the creation time is spent planning the design so that it is easy to navigate, rather than creating the application itself. In addition, AI requires a lot of processing power, this would mean that it better suited for a PC, rather than a mobile device. While it is possible to create an API for the AI so that it could be used via a mobile or web application, this could potentially cause more problems. This is because the API would have to be running constantly on a machine and since each API call would require a large amount of processing, there could be potential problems with the response time of the API.

## How was research conducted?

The information found and used in the research section of this document was found from a variety of online resources, including but not limited to online courses, web articles and YouTube videos. Any diagrams taken from any of these sources will be referenced and any graphs will have been created using the Desmos Graphing Calculator (<https://www.desmos.com/calculator>).

### Sources

* <https://theblog.github.io/post/convolution-in-autoregressive-neural-networks/>
* <https://www.ted.com/talks/pierre_barreau_how_ai_could_compose_a_personalized_soundtrack_to_your_life>
* <https://www.youtube.com/watch?v=dPWYUELwIdM>
* <https://medium.com/@bks46/introduction-to-generative-ai-models-6a5ebcebc168>
* <https://openai.com/blog/musenet/>
* <https://www.seldon.io/neural-network-models-explained>
* <https://www.3blue1brown.com/topics/neural-networks>
* <https://www.unite.ai/what-is-an-autoencoder/>
* <https://developer.ibm.com/blogs/what-is-generative-ai-and-how-much-power-does-it-have/>

## The Basics of a Neural Network

A neural network consists of layers of neurones. These neurones are essentially a function, they receive inputs, and output a value. The neuron’s output is known as its activation. The first layer of a neural network is the input layer. This is the layer that receives the data to be processed by the neural network. The last layer is known as the output layer; the activations of these neurones are the output of the entire network. In between these two layers are the hidden layers. The main concept to understand here is that certain neurones firing, will cause other neurones to fire, essentially causing a butterfly effect to cause a specific output.

The layered structure decomposes the problem that the neural network is intended to solve. For example, if we were to create a neural network to classify handwritten digits. Whilst we cannot tell each layer *exactly* what to do (we could, it would just be very time consuming as I will explain later), we would hope that the last hidden layer would detect prominent features, such as loops (as you see in the digits 6, 8, 9) or long lines (1, 2, 4, etc.). As you go back through the network the features that are being looked for become smaller and smaller (be it a small segment of a curve or loop, or a segment of a line).

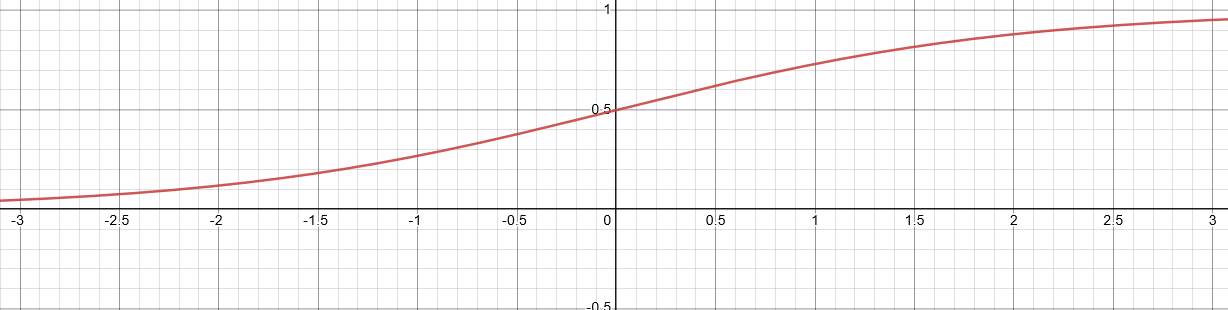
Unlike other programming solutions which are more logical, neural networks work based on a series of mathematical functions and ideas. This can be seen in how neural networks are trained (explained in a later section) as well as its structure. Neurones are connected between layers (i.e., no 2 connected neurones are in the same layer). These connections have parameters known as “weights” and “biases”. These are used to calculate the activation of the next neuron. This is done by finding the “weighted sum” of all the connections (the activation of the neuron multiplied by its weight) and adding on all the biases. This value is then passed through an “activation function” to find the activation of the next neuron. This can be calculated in one equation:

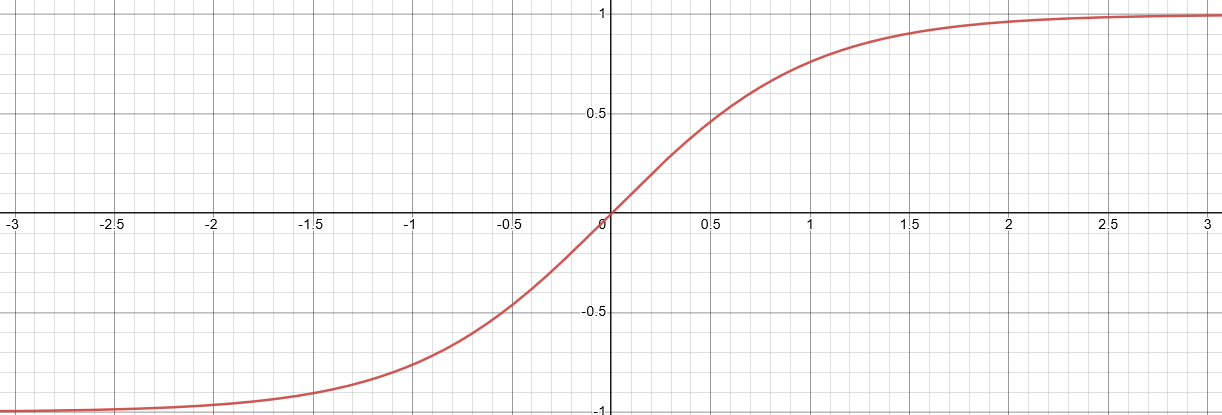
Where:

* al is the matrix of activations of a given layer in the network.
* **W** is the matrix of the weights of all the connections.
* **b** is the matrix of the weights of all the biases.
* σis the sigmoid activation function.

Activation functions are used to limit the range of possible activations. This is because adding together the weighted sum and the bias could yield a very large (positive or negative) number. Many neural networks have neuron activations between 0 and 1 and as such use activation functions such as the sigmoid function, or tanh to ensure the result is between these limits.

The Sigmoid Function:





Due to this, you can consider weights to be an indication of how a neuron in the first layer affects the activation of the neuron in the next. This is because, if the neuron in the first layer is activated and has a positive weight then this suggests that the neuron in the next layer should also be activated. Whereas if the same neuron has a negative weight, it suggests that the neuron in the next layer should not be activated. Then of course, if it has a weight of 0, then the activation of the first neuron has no influence on the activation of the second.

The biases essentially determine how large the weighted sum must be to activate the next function, since as you can see from the graphs of the functions, any weighted sum greater than 0 would cause the next neuron to also be active (to some extent).

## How is a Neural Network trained?

### Why is this not done manually?

Training refers to the process of the computer calculating optimal weights and biases for the network. The key question here is: why can’t we do this ourselves?

Consider a network with only three layers. If the first layer has 900 neurones (for each pixel in a 30x30 image for instance), and the next layer has 20 neurones. Every neuron in the first layer, is connected to every other neuron in the second layer. This means that there are 900x20 weights (one for each connection) and 20 biases (one for each neuron). This is obviously a lot to do by hand and would be incredibly time consuming and so therefore we leave it up to the computer to perform this task. This is also the reason why we cannot specifically tell the computer what each layer does since the weights and biases are what decide this.

### The Cost Function

Neural networks are trained using a training data, this is data intended directly for the training since it has example input data as well as the expected output from the network. Once the network processes the input data and generates an output. This output is compared to the expected value using a cost function, and the result from the cost function is what is used to adjust the weight and biases. This process is called gradient descent.

A cost function is a mathematical function which evaluates how different the network’s output and the expected output, as defined in the training data, are. The most common cost function is the mean squared loss function.

Where:

* n is the number of observations
* yi is the ith label in the training data
* is the corresponding output from the network

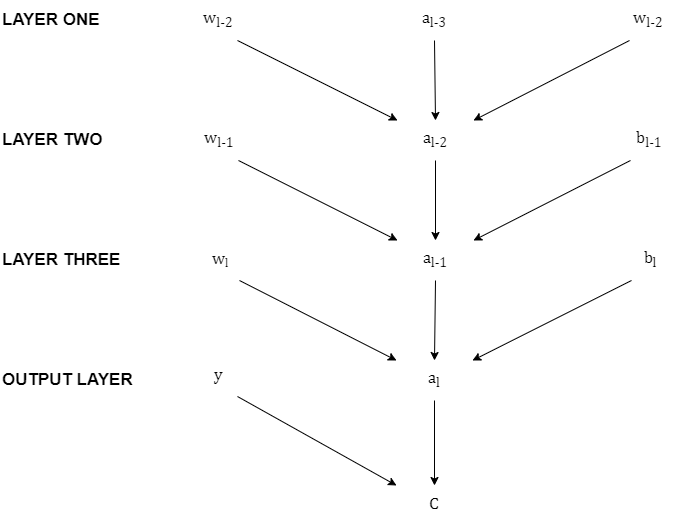
Once the cost function has been calculated, the aim of gradient descent is to find what weight and bias values will minimise the cost function. However, since there can be so many parameters (weights and biases) involved in a network, the graph of the cost function can be overcomplicated with many minima. This is why gradient descent is used.

### Gradient Descent and Backpropagation

Essentially, at the start of training, the weights and biases are randomised. Once the cost value has been calculated, we take derivatives. Specifically, the derivative of the cost function with respect to biases, and the derivative of the cost function with respect to weights. These tell us how a change in the weights and biases will affect the cost function. This is all calculated using backpropagation.

Backpropagation starts at the output layer and works backwards through the network to calculate the derivatives of the cost function with respect to every weight, and every bias by using the chain rule which is a method to find derivatives of composite functions (functions where another function is the input).

Consider a 4-layer network, each layer with only one neuron. We could draw this diagram showing each parameter in calculating the cost function.



Where:

* al is the lth neuron’s activation
* wl is the weight of the lth connection
* bl is the bias of the lth connection
* **ŷ is a3 which is the activation of the output neuron**
* **C is the cost function**

To make this easier to conceptualise using the chain rule, I am going to add an intermediary step in between neuron activations since:

This is a composite function. It is much easier to use the chain rule if you consider each function within a composite function as its own variable. Hence, I will divide this up such that:

To make this explanation simpler, I am only going to consider the last 2 layers of the network. As I mentioned before, the aim of gradient descent is to minimise the cost function by changing the weights and biases. We do this by calculating how each of these parameters affect the cost function by taking derivatives. These derivates tell us the direction of steepest increase, hence we take the opposite step. Hence, with the last two layers we can take the following derivatives using the chain rule:

The derivative of the cost with respect to the weight:

The derivative of the cost with respect to the bias

Where:

* C0 is the result of the cost function for a single training example
* al is the activation of the output neuron
* wl is the weight of the connection between the neuron on the last hidden layerand the output neuron
* bl is the bias of the connection between the neuron on the last hidden layer and the output neuron
* is the derivative of the sigmoid function

Note that C0 is the result of the cost function from a single training example, we must compute these derivatives for each training example. This means that the full cost function for a network, across all training examples would be:

Hence, we would also need to compute the derivative of the full cost of a network with respect to the weight and biases.

To compute the full gradient of the cost function, we need the derivatives of the cost function with respect to all the other weights and biases. The key to this is to find the derivative of the cost function with respect to the neuron’s activation, since as you can see from the diagram on the previous page, a neuron’s activation is calculated with the weights, biases, and activations of the previous layer.

This is given by:

We do not care about this derivative directly, however, we will use it to calculate the derivatives of the cost function with the previous weights and biases, and this is where the idea of “propagating backwards” comes into play. For instance, the derivative of the cost with respect to the penultimate weight would be given by:

This idea can be repeated throughout the network for every layer. This process is also more or less identical for more complicated networks with multiple neurones per layer. All these derivatives, of all the weights of biases, are stored in the gradient matrix.

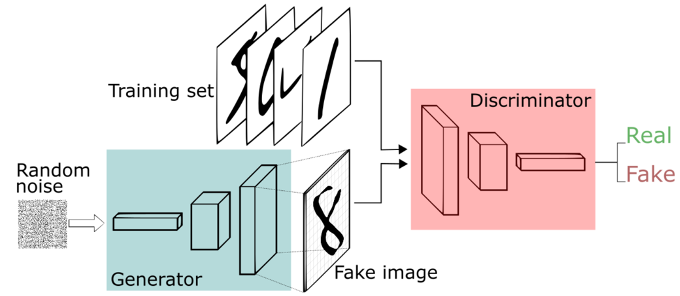
However, if we had to do this for every training example it would take a long time. So, it is more common for neural networks to use a process called Stochastic Gradient Descent, where the training examples are shuffled and split up into batches. Then the cost function is calculated, and the weights and biases are adjusted after every batch.

## Neural Network models used for generation

There is not a single neural network model though. Thorough research has given rise to many distinct types of neural networks which are more suited for particular tasks. These models may differ in complexity and structure, i.e., neurones may be modelled differently, and the connections between each neuron may be different, as well as how data flows through the network. There are neural network models designed specifically for data generation.

### Generative Adversarial Networks (GANs)

A GAN consists of two neural networks. Each neural network has their own role: either generator or discriminator. The generator network generates new data or content, which (hopefully) resembles the training data, and the discriminator distinguishes between the data generated by the generator network, and the training data.



Source: [IBM](https://developer.ibm.com/blogs/what-is-generative-ai-and-how-much-power-does-it-have/)

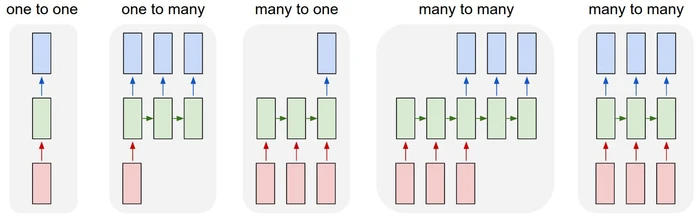
These networks are trained in alternating cycles: the generator learns to generate more realistic data and the discriminator gets better at differentiating between the generated data and the real data.

## Recurrent Neural Networks (RNNs)

RNNs specialise in processing sequences, which would make them effective for this use case since music, in a simple form, is a sequence of notes. Furthermore, they are often use for Natural Language Processing meaning that they can pick up intricacies in these patterns which would be important for generating a well-received piece of music.

They are different to vanilla neural networks since vanilla networks can only handle predefined input and output sizes, whereas RNNs allow for variable input and output lengths. This means that the user would be able to control how much music the AI generates and thus enhancing their experience with the software.

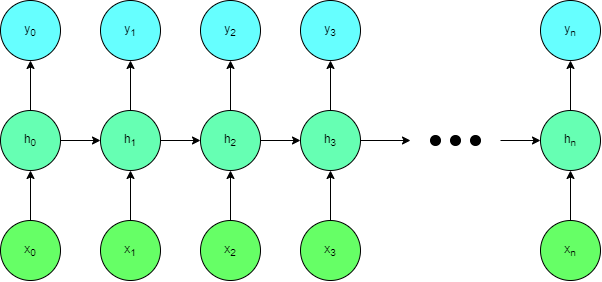
There are different types of RNNs, shown in the diagram below:



Source: [Victor Zhou](https://victorzhou.com/blog/intro-to-rnns/)

Each type comes with a different use case. For instance, sentiment analysis would be done using a many to one RNN since, for example, each word in the sentence could be inputted into the neural network (many), and the network would have a single (one) output which is the sentiment (e.g., “happy”, “sad”, “funny”).

Consider a many-to-many RNN.



The main takeaway from this diagram is that the activation of in each layer of the RNN has an effect on the activation of in the subsequent layers of the network.

Another thing that makes RNNs unique is the fact they work iteratively, updating a hidden state h each time. They also use the same sets of weights and biases for each step, whereas in a vanilla neural network there would be a different set of weights and biases for each layer.

There are 3 sets of weights for a given step :

* – used for the connection between and
* – used for the connection between and
* – used for the connection between and

There are 2 biases for a given step :

* – used for calculating
* – used for calculating

These can be broken down into the equations:

### Transformer-Based Models

Transformer based models transduce sequences into a form that can be “understood” and manipulated by a program which therefore makes them good for any task that manipulates an input into an output.

This makes them particularly good at understanding language and sentences, due to moderately strict grammar rules such as those in English that dictate that sentences are written in the order: subject, verb, object. Therefore, they are very popular amongst translator applications.

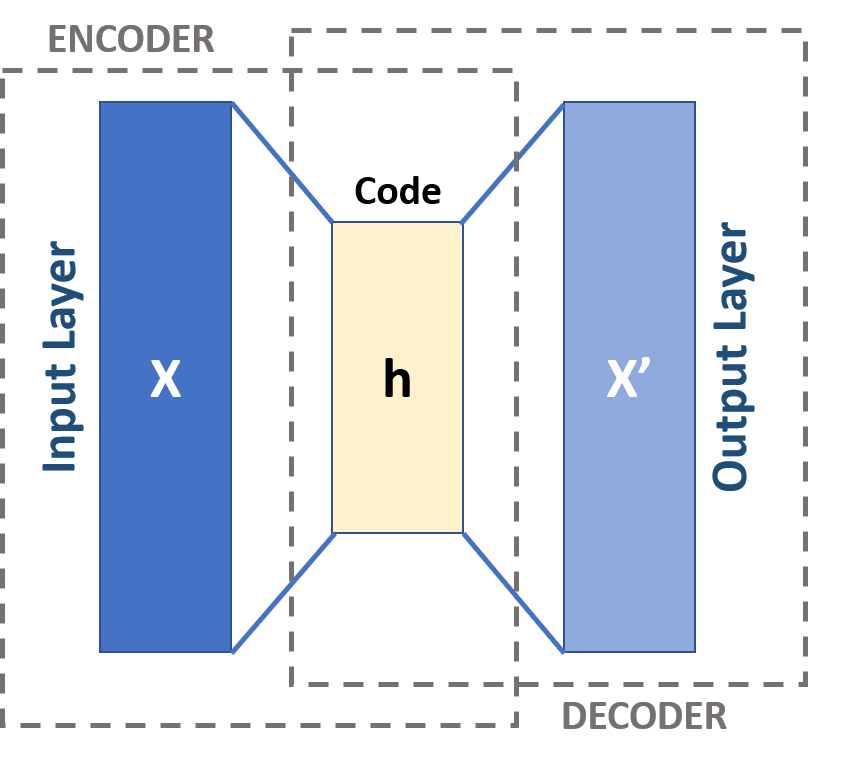


Source: [Medium](https://medium.com/@bks46/introduction-to-generative-ai-models-6a5ebcebc168)

### Autoencoders

Autoencoders is a type of unsupervised learning model which compresses and encodes input data, and then reconstructs the data from the encoded representation. Training the model reduces the data lost during the process of encoding and then decoding the data which allows it to “learn” the most significant parts of that data.

For the same reason, autoencoders are unique in the fact that its input and output layers of the neural network contain the exact same amounts of data. The network also contains a “bottleneck” which handles the compressed data.



Source: [unite.ai](https://www.unite.ai/what-is-an-autoencoder/)

The concept of an autoencoder may seem counterintuitive, why would you want a neural network which practically copies its input? Once, the network is trained, you can create a model which generates similar data and then adds or reduces target features identified by the autoencoder.

## Existing Solutions

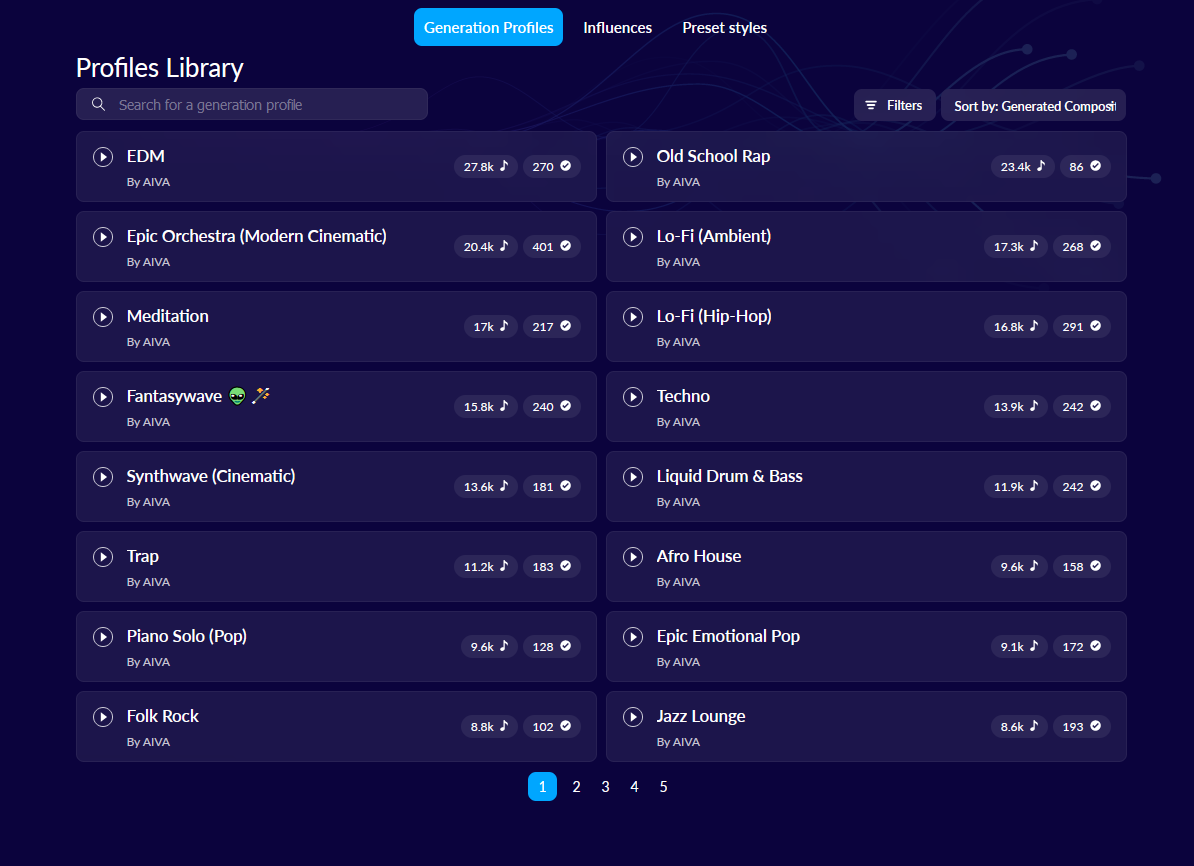
### AIVA

AIVA Technologies was founded in 2016 by Pierre and Vincent Barreau and Denis Shtefan in Luxembourg. Pierre Barreau is an engineer, raised in a family of musical artists and was inspired by a film called “Her”, in which an AI named Samantha created a piece of music to capture a moment in her life since she is unable to manifest herself physically and take a photograph like a human would.

The AI they created is called AIVA. This AI has been trained with over 30,000 musical scores from history’s greatest musicians such as Bach and Beethoven. With the use of deep neural networks, the AI can “learn” patterns in these pieces by trying to predict the next notes in the music from a given bar of the score. Each of these scores have been given category labels which denotes its emotion, genre and much more.

The aim of this technology is to augment human creativity. In the TED Talk (How AI could compose a personalized soundtrack to your life, 2018), Pierre Barreau stated that he wanted people to be able to create a “life soundtrack”. This is possible by the user inputting certain labels, which the AI was trained with, to generate a piece of music specific to them, or that moment in their life. He finished the same TED Talk with a piece of music influenced by the TED theme.

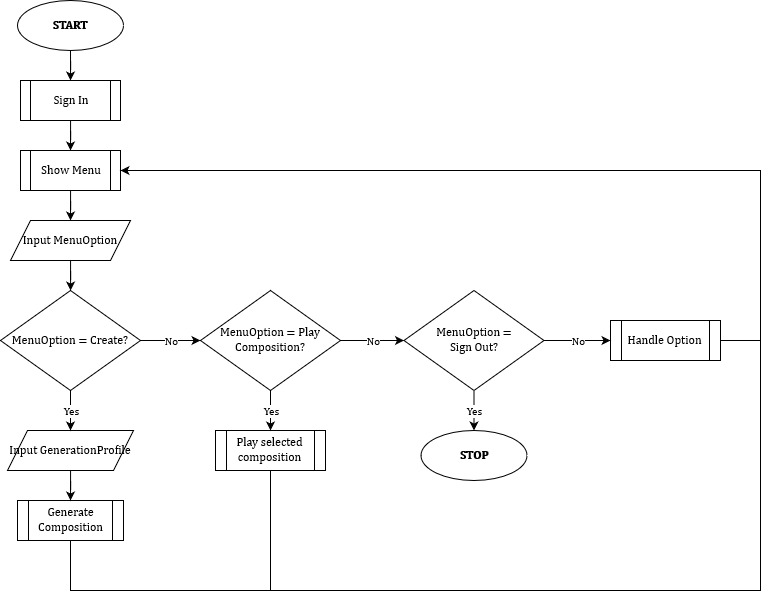
The way I train the network could be like the methods used by the creators of AIVA. Of course, this would have to be on a much smaller scale but training the network based on classical music (or at least older music), rather than more recent and modern music, may be easier as pre-existing datasets (such as <https://www.kaggle.com/datasets/soumikrakshit/classical-music-midi>) may already exist that can be split and used as training data.

****

AIVA’s website gives the user a nice, easy-to-use GUI so that they do not have to be concerned about how the AI itself is working. It clearly shows what pieces are currently being generated and allows the user to listen to the compositions without the need to download them or open a sound file separately. Therefore, I would also include a GUI in my program for the same reasons.

I was particularly interested in AIVA’s simplistic side bar which gives the users access to all important parts of the software. No matter what link you follow on this side bar, the green “Create Track” button remains. This encourages the user to actively engage with the software. For my implementation, I would like to include a side bar for easy access to the different parts of the program which at this stage will be the creator, the music player and the settings menu.

The overhead view of AIVA’s interface and system is simple, intuitive and easy for new users to understand. The following diagram is an overhead system flowchart for creating and playing compositions with AIVA. The features I was interested in while investigating AIVA was purely the creation and playing of compositions.



As you can see, the process of creating compositions with AIVA is extremely simple which makes this software accessible to many end-users. The following is an assumptive data dictionary for AIVA.

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Item** | **Data Type** | **Validation** | **Example Data** |
| Generation Profile | Profile structure   * Name * Author * NN Variables | N/A – selected from a list |  |
| Generation Name | STRING | Length > 0  Name doesn’t already exist | “Generated song 1” |
| Generation Length | REAL | N/A – selected from a list | Shown as: 2min 30sec Stored as: 2.5 |
| Number of Compositions | INTEGER | N/A – selected from a list | 5 |

### MuseNet

MuseNet is a neural network which can create compositions which lasts up to 4 minutes with up to ten different instruments. It can blend the different styles of composers and genres like EDM, Mozart and the Beatles.

Specifically, MuseNet is a large-scale Transformer model which was trained on thousands of MIDI files. It allows users to interact with the model in two different ways. Simple mode, which allows you to choose a composer or a style and gives you the option to choose a famous piece of music to start the generated piece. It also has an advanced mode which allows you to interact with the transformer model directly.

However, the official blog by OpenAI, the creators of MuseNet, highlight some limitations of the product. Highlighting that the user’s input are only “strong suggestions” and aren’t considered requirements by the model while generating the music. As well as the fact that it is difficult for the model to match up very different styles of music.

MuseNet is completely different to what I am aiming for. Since it uses a transformer rather than an RNN. However, it has given me some ideas to stay away from – particularly mixing or “taking suggestions” of what type of music to generate since if the creators themselves are critical of its ability to do this. Since my neural network model is of a completely different type, as well as being much smaller than that of MuseNet, it is extremely unlikely for it to work; especially when the creators of the superior network states that it struggles to do this.

## Initial Modelling

### Data Flow Diagram

This is a basic model of how the neural network will work with respect to its external inputs and outputs.

Graphical user interface, application

Description automatically generatedThe user will provide the networks with the required settings (regardless of whether the user is training the network or attempting to generate with it). The network will then use the same forward process to do a forward run with the network. If it is generating. It will not require the vectorised MIDI file.

It will keep passing its output to itself, until it has done as many forward passes as is needed. Then, if it is training, it will pass lists containing the value at each layer of the network to the backpropagation algorithm, which will calculate the derivatives of each layer with respect to the loss. These derivatives will be used by the optimisation algorithm to optimise the network parameters.

If the network is used for generating, then rather than passing the values and the error to the backpropagation algorithm. It will give the generated data to a process which interprets the network’s outputs so that it can be written to a MIDI file.

### Class Diagram

This is a basic class diagram for the neural network.

Text

Description automatically generated

I have divided the classes in this way in order to avoid the repetition of code since a lot of the equations used in neural networks – especially LSTMs – are the same, except they’re using different parameters.

One of the key areas where I have done this is by using the “weight class”. If I did not create a separate weight class, the LSTM and RNN classes would need 2 properties to represent a “weight” (a matrix for its value, and a matrix for its gradient). Furthermore, when writing the optimisation algorithm, the same calculation would be repeated over and over in code, due to the fact that the algorithm would be the same for every weight regardless of what it represents or where it is used.

## Objectives for the Project

1. To understand neural networks and be able to create a recurrent neural network from scratch.
   1. The use of 3rd party libraries should be kept to a minimum, apart from those used to create the GUI.
   2. Create a class library which contains classes used by the network. Specifically, a class for the RNN itself, as well as a class for the LSTM cell which are linked via composition.
   3. Use the Adaptive Moment Estimation optimisation algorithm in order to optimise the tuneable parameters of the network.
   4. Create a class library for matrix operations. The main matrix class will be based on a 2D array of doubles and will implement basic matrix arithmetic as well as the advanced functions required for neural networks such as tanh and the sigmoid function.
2. To allow the neural network to save and load its state to avoid it having to be trained every time it is used
   1. The state of the neural network should be saved in “state files” – this contains the current weights and biases of the network.
   2. The user should be able to create “profiles” for the AI, each profile has its own state files and its own set of training data.
   3. Users should be able to freely change “profiles”. The training data and the state files used by the application should correspond to the profile that the user has selected. These files will be organised such that they can be directly accessed by the program.
   4. Users should have the option to continue training the “profile” or create a new one.
   5. If a user chooses to continue training a profile, a new state file should be created such that there is now a “before training” state file and an “after training” state file.
   6. Users should be able to revert the state of the neural network to its state before the previous session of training.
   7. In a given training session, the state of the network should be saved twice. Once before training begins in the “before training” state file, and once after training has ended, in the “after training” state file.
   8. Once the program is closed, any “before training” files should be deleted.
3. To create a program which can process MIDI files
   1. MIDI files should be read and converted into data which can be inputted into the network
   2. Output data from the network should be read and converted into a MIDI file which is saved to a directory which is defined by the user
4. To create an easy-to-use graphical user interface
   1. This should be well-received by the end-users and those who test the program. They must consider it “readable” and “easy-to-use” such that it is accessible to many audiences.
   2. The GUI must allow the user to generate music (i.e., allow them to enter a name for the composition, maximum duration, etc.)
   3. The GUI must allow the user to play the sound files in-app.
   4. The sound player must use a queue in order to queue sounds to be played, and a stack in order to keep track of the recently played sounds.
   5. The user should be able to skip the playing sound (i.e. play the next sound in the queue) or play the previous sound (i.e. play the next sound in the stack).
   6. The GUI must allow the user to edit settings (i.e., save/load directories, default duration, etc.)
   7. If any unhandled exceptions arise whilst the application is running, the error should be displayed to the screen in a message box before the application is closed.
   8. Any errors should also be logged to an external file, where the user can see what exactly caused the error, and where it arose in the source code so it can be easily fixed by the developer.

# Design

## Activation Functions

There are three activation functions which are needed for the neural network: the sigmoid function, tanh and softmax. However, we also need the derivative of these two functions for backward propagation. Thus, I have chosen to create a static class with these 4 methods (tanh, the sigmoid function and their derivatives) even though tanh is implemented with the standard “Math” library, so that everything is kept together and thus the code is more readable.

### The Sigmoid Function

FUNCTION sigmoid(x)

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

ENDFUNCTION

### The Derivative of the Sigmoid Function

FUNCTION sigmoid\_prime(x)

return sigmoid(x) \* (1 – sigmoid(x))

ENDFUNCTION

### The Derivative of Tanh

FUNCTION tanh\_prime(x)

return 1 – math.pow(math.tanh(x), 2)

ENDFUNCTION

### The Softmax Function

FUNCTION softmax(element, matrix)

return math.exp(element) / sum(matrix.exp())

ENDFUNCTION

## Matrices

The use of matrices is fundamental to machine learning because while a neural network only using singular numbers is technically possible, this neural network would not be capable of learning anything since it is too basic. Matrices simply add to the complexity of weights and biases while opening the door to new uses of machine learning.

C# does not have any built-in matrix classes, and due to the number of different operations I need to perform upon these matrices, I opted to make a separate class for the matrix. The class is very basic, having a 2D double array as a private attribute, and public methods that perform certain operations on the numbers in the array.

### Element-wise Operations

An element-wise operation is a mathematical operation which is performed on each element in the matrix. In the program, they will be mostly controlled by a single “iterate\_content” method which takes in a function as a parameter this function takes in another double as a parameter, this will be an item in the matrix, and will return another double to replace the same item.

As C# it would look like this:

public void IterateContent(Func<double, double> contentAction)

{

for (var i = 0; i < Rows; i++)

for (var j = 0; j < Columns; j++)

Content[i,j] = contentAction(Content[i,j]);

}

This method is then used by different parts of the class to carry out different element-wise operations. This is how this method may be implemented and used in other class methods.

PROCEDURE sqrt()

iterate\_content(math.sqrt)

ENDPROCEDURE

PROCEDURE sigmoid()

iterate\_content(activations.sigmoid)

ENDPROCEDURE

PROCEDURE pow(power)

iterate\_content((value) => math.pow(value, power))

ENDPROCEDURE

#### Hadamard Operations

Hadamard Division and Multiplication means that the 2 matrices are multiplied element-by-element.

PROCEDURE hadamard\_multiplication(other\_matrix)

FOR i ß 0 TO Rows   
 FOR j ß 0 TO Columns   
 Content[i,j] \*= other\_matrix[i,j]

ENDFOR

ENDFOR

ENDPROCEDURE

### Specific Operations

#### Matrix Multiplication

Matrix multiplication is not element-wise, since there is a specific order in which matrices are multiplied, which means that:

1. Some matrices cannot be multiplied together

Consider . If A and B are matrices, then the number of columns in A must equal the number of rows in B.

1. might not be the same as , if A and B are matrices.

Matrix multiplication creates a new matrix which has the same amount of rows as A, and the same amount of columns as B. Each item in the new matrix is found by choosing the corresponding row in the first matrix, and the corresponding column in the second, then finding the sum of the products of each corresponding item (i.e. the first item in the row x the first item in the column). This process can be written as pseudocode:

PROCEDURE matmul(otherMatrix)

IF (Columns != otherMatrix.Rows)

OUTPUT “Invalid operation, the number of columns in A must equal the number of rows in B”

ENDIF

newContent ß new float[Rows,otherMatrix.Columns]

FOR i ß 0 TO Rows

FOR j ß otherMatrix.Columns

sum ß 0

FOR k ß 0 TO Columns

sum += Content[i,k] \* otherMatrix.Content[k,j]

ENDFOR

newContent[i,j] ß sum

ENDFOR

ENDFOR

Content = newContent

ENDPROCEDURE

#### Sum

This function finds the sum of every element in the matrix.

FUNCTION sum()

sum ß 0

FOR i ß 0 TO Rows

FOR j ß 0 TO Columns

sum += Content[i,j]

ENDFOR

ENDFOR

return sum

ENDFUNCTION

#### Transpose

Transpose turns all of the rows into columns, and all of the columns into rows (i.e. the first row becomes the first column and the first column becomes the first row).

PROCEDURE transpose()

newContent ß new float[Columns, Rows]

FOR i ß 0 TO Rows  
 FOR j ß 0 TO Columns  
 newContent[j,i] = Content[i,j]  
 ENDFOR  
 ENDFOR

ENDPROCEDURE

#### Stacking

“Stacking” appends one matrix to another. The class implements both vertical (the new matrix is added beneath the existing one) and horizontal (the new matrix is added “to the right” of the existing one) stacking.

PROCEDURE horizontal\_stack(other\_matrix)

IF Rows != other\_matrix.Rows  
 OUTPUT “Can only stack matrices with same number of rows”  
ENDIF

newContent ß new float[Rows, Columns + other\_matrix.Columns]

FOR i ß 0 TO Rows   
 FOR j ß 0 TO Columns + other\_matrix.Columns  
 IF j >= Columns   
 newContent[i,j] ß otherMat.Content[i,j – Columns]  
 ELSE  
 newContent[i,j] ß Content[i,j]  
 ENDIF  
 ENDFOR  
 ENDFOR

ENDPROCEDURE

### Constructors and creating a Matrix Object

There are different ways of creating matrix objects. One way is the same way you would create an instance of any class – through constructors. All these constructors do is set the public “Contents” attribute to an appropriate value.

One constructor will create an empty 2D array for the attribute, from the number of rows and columns that are passed in as parameters.

Another constructor takes a pre-existing 2D array as a parameter, and sets the attribute to that array.

A matrix object will also be created through static methods. While these could also be written as constructors, I believe this is the best way to implement them since it will make the program more readable in the long term.

The first alternative method to create a matrix object is to create the matrix based upon the shape of a pre-existing matrix.

FUNCTION Like(otherMatrix)

return Matrix.new(otherMatrix.Rows, otherMatrix.Columns)

ENDFUNCTION

Another way is to create a matrix filled with random values.

FUNCTION Random(rows, columns, seed=-1)

IF seed == -1

random = Random.new(seed)

ELSE

random = Random.new()

ENDIF

matrix = Matrix.new(rows, columns)

matrix.iterate\_content((value) => r.NextDouble)

return matrix

ENDFUNCTION

### Weight Class

Deriving from the matrix class will be a weight class. This will be used specifically by the network component as a tuneable matrix which can perform an optimisation algorithm on itself (the optimisation algorithm is later in the section).

It also has another matrix stored as a property which represents the gradient, which is then used in the optimisation algorithm.

## Reading and Writing MIDI Files

A MIDI file is essentially a list of event messages which contain data such as the note that should be played, how loud it should play and for how long, etc. Therefore, an algorithm is required to convert the contents of a MIDI file into a form that can be processed by the neural network, and another to convert what is processed and created by the neural network back into a MIDI file.

First, the MIDI file will be converted into text. This will be done using a library for C# called NAudio which allows for the reading and writing of MIDI files.

NAudio provides classes in order to represent the different MIDI events. Using these classes, the algorithm will retrieve the string representation for the note played. The absolute time of the file is represented by spaces - this means that any keys played simultaneously do not have a space in between them.

However, neural networks generally do not work well with exact values. So it is more common to input a “one-hot” vector into the neural network. A singular one-hot vector would represent a singular character in the converted string. It would be 89 characters long (for the 88 pitches and the space) and would contain a singular 1 (which identifies which character it is) and the other elements would be 0.

For example: if only the characters { A, B, C } were used then:

* A = [ 1 0 0 ]
* B = [ 0 1 0 ]
* C = [ 0 0 1 ]

Of course, a neural network would not generate a one-hot vector. So we would treat each element in the output as a probability, and regard the highest value as the 1 and turn the other values into 0s.

### Converting MIDI to a string

FUNCTION to\_string(midi\_path)

file = midi\_file.new(midi\_path)

data = “”

prev\_abs\_time = 0

FOR track\_num = 0 TO file.tracks

FOREACH event IN file.events[track\_num]

IF event.type != note\_on\_event  
 CONTINUE  
 ENDIF

note = event.note\_name

abs\_time = event.absolute\_time

time\_diff = abs\_time – prev\_abs\_time

data += (“ “ \* time\_diff) + note

prev\_abs\_time = abs\_time

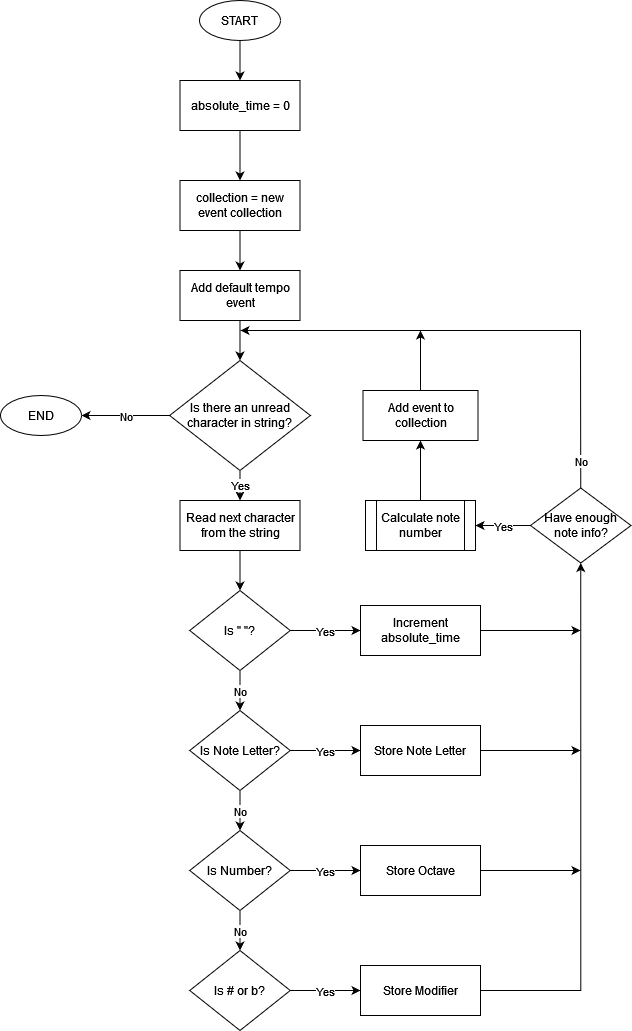
ENDFOREACH

ENDFOR

RETURN data

ENDFUNCTION

### Converting a string into MIDI



## Writing Log Files

The methods required for writing log files will be contained in a log manager class. Its main method will be “WriteLine” (mimicking Console.WriteLine). This will simply take in a string, and write it (via a StreamWriter) to the file.

However, it will have a private method which will be ran once throughout the entire program in order to create the log file itself as well and decide what its name will be. The naming structure is described as so in Backus-Naur form:

<log-name> ::= apollo\_logs\_<date>\_<id>.log  
<date> ::= <digit><date> | -<date> | <digit>  
<id> ::= <digit><id> | <digit>

The ID is required in case there are multiple log files created in one day. The ID will increment from 0 onwards based on how many existing log files there are from the same date.

## Neural Network

I have chosen to use a Recurrent Neural Network for this project, more specifically a Long Short-Term Memory (LSTM) network. This is because there is a prevalent issue with standard RNNs (gradient vanishing) when there is a large number of recurrences since information is lost. LSTMs prevent this since they are capable of identifying useless information in the network and getting rid of it, so that it “remembers” information for longer thus making more accurate predictions.

The concept behind the network is the same. However, rather than performing one simple calculation on each iteration, the input and the previous output, and previous “cell state” are passed into the LSTM cell, which contains many “gates”.

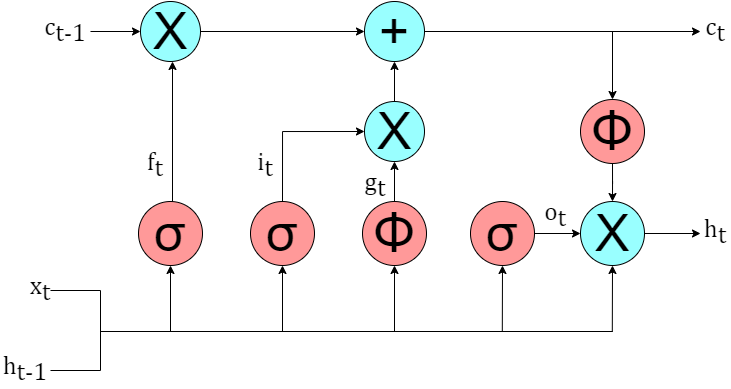
These gates manipulate the input data such that useless information can be “forgotten” and disregarded, and only the important parts of new information will be picked up.

These gates include:

* The forget gate which decides what information to keep from the previous cell state.
* The input gate which decides what new information to give to the cell state. This is combined with a “new information” matrix, which decides the extent of which the cell state will be changed.
* The output gate which uses the newly calculated cell state, and the cell’s inputs to determine the overall output of the cell.

The cell state can be thought of as the cell’s “long term memory” as it is retained and altered with every pass. The cell state at a given timestep is calculated using the previous cell state and the forget gate (to disregard any unneeded information) as well as the input gate, and the ‘new information’ matrix (to add any new information to the state).

The following is a diagram of an LSTM cell.



In this diagram:

* is the output of the cell
* is the input of the cell
* is the cell’s ‘state’
* represents the forget gate
* represents the input gate
* represents the matrix highlighting “new information” (known as the candidate state)
* represents the output gate
* is the timestep (e.g., is the previous cell output)
* Blue circles represent basic arithmetic operations
* Red circles represent activation functions

### Forward Propagation

Forward propagation refers to the main processing of a neural network. This refers to the algorithms that they use in order to calculate the output (given an input). In the case of an LSTM cell, this involves a series of calculations at each gate. The calculations required at this stage are as followed:

The LSTM cell is going to be contained in and used by a separate RNN class. The RNN class will be responsible for controlling the number of recurrences that occur as well as managing the inputs and outputs of the LSTM cell. However, as far as separate calculations are concerned, none will be performed by the RNN cell.

Since we are using one-hot vectors, the softmax function will be applied to the final output of the RNN. This means that each element in the output vector will represent a probability, and thus the outputs of the RNNs can be interpreted more easily.

#### LSTM Cell Pseudocode

FUNCTION Forward(input, prev\_output)

forget\_value ß forget.Calculate(input, prev\_output)

input\_value ß input.Calculate(input, prev\_output)

new\_value ß new\_info.Calculate(input, prev\_output)

cell\_state ß forget\_value \* cell\_state + input\_value \* new\_value

output\_value ß output.Calculate(input, prev\_output)

return output\_value \* tanh(cell\_state)

ENDFUNCTION

#### RNN Cell Pseudocode

FUNCTION Forward(initial\_input, num\_recurrences)

input ß initial\_input

lstm\_output ß new Matrix(newVocab, 1)

outputs ß []

FOR i ß 1 TO num\_recurrences

lstm\_output ß lstm\_cell.Forward(initial\_input)

input ß interpret\_onehot(lstm\_output)

outputs.append(input.softmax)

ENDFOR

return outputs

ENDFUNCTION

### Backwards Propagation

**Source used:** <https://www.youtube.com/watch?v=8rQPJnyGLlY>

Backwards Propagation is the process of decreasing the weight matrices such that the neural network performs better. This is done with gradient descent, but a lesser-known fact about machine learning is that there are different types of optimisers.

However, regardless of what optimisation algorithm is chosen, you still must calculate the required derivatives/gradients for every parameter in the neural network.

I am using a cross-entropy loss function for the neural network. This is because it takes into account the difference between the expected output from the network and the actual output. Since each note in the MIDI file is categorised and selected using one-hot vectors, this loss function is the most logical choice. The equation is as follows

Where:

* is the expected output
* is the actual output

In order to reduce this loss function, we need to calculate the derivatives of each tuneable variable with respect to the loss function. This includes the weights for the hidden state and input for each of the LSTM gates, as well as the weight in the softmax layer. We cannot do this immediately, so we must use the chain rule using the following derivatives:

Where (the unactivated softmax layer)

#### Derivatives of Gates and States

*Derivative of Input Weights*

*Derivative of Previous Output Weights*

Diagram

Description automatically generated

**Source Used:** <https://doi.org/10.48550/arXiv.1412.6980>

In terms of optimisation algorithm, I have chosen to use Adam (Adaptive Moment Estimation). This is because this is both a popular and recent algorithm (proposed in 2015 by Diederik Kingma from OpenAI and Jimmy Ba from the University of Toronto) in machine learning. This is because it combines two previously popular algorithms (AdaGrad and RMSProp), making Adam good for both sparse gradients and non-stationary objectives.

One thing that makes Adam effective is that the stepsizes are invariant to the calculated gradient and it is instead dictated by a hyperparameter. On top of this, Adam’s hyperparameters rarely require tuning thus the default values recommended in the paper in which the algorithm was proposed (“Adam: A Method for Stochastic Optimization”) are often enough.

The algorithm requires the following:

* – hyperparameter which determines the step size
* – hyperparameter which determines the exponential decay rate of the first moment vector
* – hyperparameter which determines the exponential decay rate of the infinity norm
* – the first moment vector (initialised as 0)
* – the infinity norm (also initialised as 0)
* – the calculated gradient
* – the parameter being optimised

The series of calculations in the algorithm are performed as follows:

Thus, my pseudocode for the Adam optimiser is as follows (given that the moment vector and infinity norm had been initialised somewhere else in the class):

PROCEDURE Adam(weight, gradient, beta1, beta2, alpha, epsilon)

m = beta1 \* m + (1 – beta1) \* gradient

v = beta2 \* v + (1 – beta2) \* gradient \* gradient

m\_hat = m / (1 – beta1)

v\_hat = v / (1 – beta2)

weight -= alpha \* m\_hat / (sqrt(v\_hat) + epsilon)

ENDPROCEDURE

Since the LSTM cell uses the concept of “gates”, the calculations on these gates are mostly the same for both forward propagation and backpropagation. Therefore, I will create a gate class which contains the necessary components in order to perform these calculations.

### Vocab Class

The neural network produces the vectorised form of a string, which will then be converted into a MIDI file. The vocab class will be used to restrict the characters which the neural network can output for the string, since if the neural network could output any character for the string, there is a higher chance that it will output meaningless characters or ultimately output something different to the training data.

The Vocab Class will essentially be a list which can only contain 1 copy of each character. It will be used to map each character to an integer ID and vice versa (using the index in the list). Using this ID, it will also be able to convert an individual character into a one-hot vector, and a one-hot vector to its corresponding character. The one-hot vectors will be stored as a matrix object.

FUNCTION interpret\_one\_hot(vec)

index\_of\_one = -1

FOR i = 0 TO vec.columns  
 IF vec[0,i] == 1  
 index\_of\_one= i  
 BREAK  
 ENDIF  
 ENDFOR

RETURN lookup\_table[index\_of\_one]

ENDFUNCTION

FUNCTION char\_to\_one\_hot(c)

vec = matrix.new(1, lookup\_table.length)

char\_id = lookup\_table.index\_of(c)

vec[0, c] = 1

RETURN vec

ENDFUNCTION

FUNCTION prepare\_training\_data(midi\_string)

training\_data = []

FOR i = 0 TO midi\_string.length  
 training\_data.append(char\_to\_one\_hot(midi\_string[i]))  
 ENDFOR

RETURN training\_data

ENDFUNCTION

### Saving State

Training is vital for the program to work as intended, since without it the neural network will generate random nonsense data which definitely be impossible to turn into a sound file. However, training is very processor intensive and thus not all end-users will have the capability to train the neural network. Thus, it is important to have a way to save and load network parameters that have already been trained.

State will be stored in a binary file. When the program creates the neural network, the binary file will be read. This is done as soon as the program opens. Then to ensure that training progress is not lost, on every epoch of training the state file will be saved in a new binary file (such that the training can be reverted), then once the program is closed the old state file is deleted.

C# has BinaryReader and BinaryWriter classes which assist with this. More helpfully, the BinaryReader class contains methods to read specific data types sequentially from the file. Since all the parameters are matrices, ReadMatrix and WriteMatrix subroutines are required.

PROCEDURE write\_matrix(mat, writer)

FOR i ß 0 TO mat.Rows

FOR j ß 0 TO mat.Columns

writer.write(mat.Content[i,j])

ENDFOR

ENDFOR

ENDPROCEDURE

FUNCTION read\_matrix(rows, columns, reader)

content ß new float[rows, columns]

FOR i ß 0 TO rows

FOR j ß 0 TO columns

content[i,j] = reader.read\_float()

ENDFOR

ENDFOR

return Matrix.new(content)

ENDFUNCTION

PROCEDURE save\_state(path)

writer = BinaryWriter.new(path)

write\_matrix(lstm\_cell.forget.weight, writer)

write\_matrix(lstm\_cell.forget.bias, writer)

write\_matrix(lstm\_cell.input.weight, writer)

write\_matrix(lstm\_cell.input.bias, writer)

write\_matrix(lstm\_cell.new\_info.weight, writer)

write\_matrix(lstm\_cell.new\_info.bias, writer)

write\_matrix(lstm\_cell.output.weight, writer)

write\_matrix(lstm\_cell.output.bias, writer)

ENDPROCEDURE

PROCEDURE read\_state(path)

reader = BinaryReader.new(path)

lstm\_cell.forget.weight = read\_matrix(lstm\_cell.forget.weight.Rows, lstm\_cell.forget.weight.Columns)

lstm\_cell.forget.bias = read\_matrix(lstm\_cell.forget.bias.Rows, lstm\_cell.forget.bias.Columns)

lstm\_cell.input.weight = read\_matrix(lstm\_cell.input.weight.Rows, lstm\_cell.input.weight.Columns)

lstm\_cell.input.bias = read\_matrix(lstm\_cell.input.bias.Rows, lstm\_cell.input.bias.Columns)

lstm\_cell.new\_info.weight = read\_matrix(lstm\_cell.new\_info.weight.Rows, lstm\_cell.new\_info.weight.Columns)

lstm\_cell.new\_info.bias = read\_matrix(lstm\_cell.new\_info.bias.Rows, lstm\_cell.new\_info.bias.Columns)

lstm\_cell.output.weight = read\_matrix(lstm\_cell.output.weight.Rows, lstm\_cell.output.weight.Columns)

lstm\_cell.output.bias = read\_matrix(lstm\_cell.output.bias.Rows, lstm\_cell.output.bias.Columns)

ENDPROCEDURE

### Profile Class

The Profile class will be a collection of data relating to each state profile for the network. It will contain the save paths for the before and after state files, as well as a string representation of a vocab list, and the path where the training data is stored for the profile.

Each profile will be a dedicated folder in the file system which will be where the state files and training data are stored by default as well as a JSON file which contains the data for the related class. I will use C#’s JsonSerializer in order to read and write to and from these files.

### Profile Manager Class

This is a class which will exist primarily on the front end in order to keep all the profiles in a single data structure so that they are easily accessible using a string identifier.

It will use a dictionary, using a string key and profile object value pair. It will have methods which can be called from the pages in the application in order to make a profile object on the fly as well as load the profiles that exist in the user’s file system.

It will use C#’s JsonSerializer in order to save to and read from the profiles’ JSON files which are stored on the user’s file system.

### NeuralNetwork Class

A lot of different classes are used for the RNN, which means that there are a lot of different objects to keep track of when it is eventually implemented in the front end. In order to make this implementation easier, I will create a NeuralNetwork class which will provide a higher-level interface with the RNN.

It will keep track of the current profile for the RNN (so that it can save and load its state), the vocab object used by the RNN to convert to and from one-hot vectors.

Furthermore, it will provide functions which allow content to be easily generated from the frontend, so that the code for the UI is purely for the UI, and nothing else.

FUNCTION create\_gen\_seed()

rows = []

FOR i = 0 TO rows.length

char\_id = random(0, vocab.size)

rows.append(vocab.create\_one\_hot(vocab[char\_id]))

ENDFOR

RETURN matrix.stack(rows)

ENDFUNCTION

PROCEDURE generate(num\_iterations, save\_path)

seed = create\_gen\_seed()

outputs = rnn.forward(seed, num\_iterations)

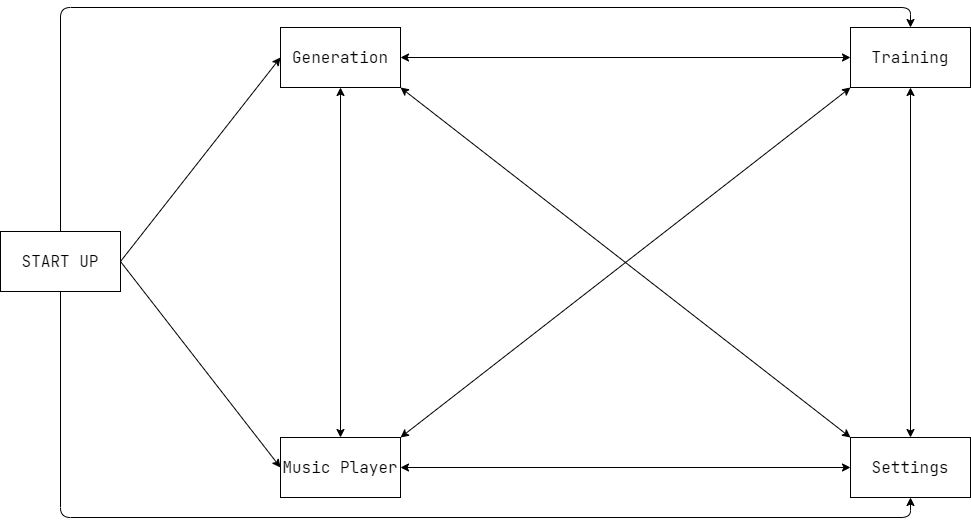
str\_output = vocab.interpret\_one\_hots(outputs)

write\_midi(str\_output, save\_path)

ENDPROCEDURE

## Top-down Diagram and Interface

This is a top-down diagram of how users will navigate to different parts of the program. The user will start at the “state up” state. This will include a button to any of the other 4 states. Then from any of the other 4, the user can use a navigation bar at the top of the application to navigate to any of the others (except from the start up screen).



This is the colour scheme I have chosen for the program. I chose these colours due to the fact that they are contrasting, solid colours, meaning that the program will be more accessible to people with colour-blindness.

## Program Start Up



This is the screen which is shown when the program is launched. It is a simple menu with buttons for quick access to each of the program “states”. It has no functionality nor purpose apart from design reasons.

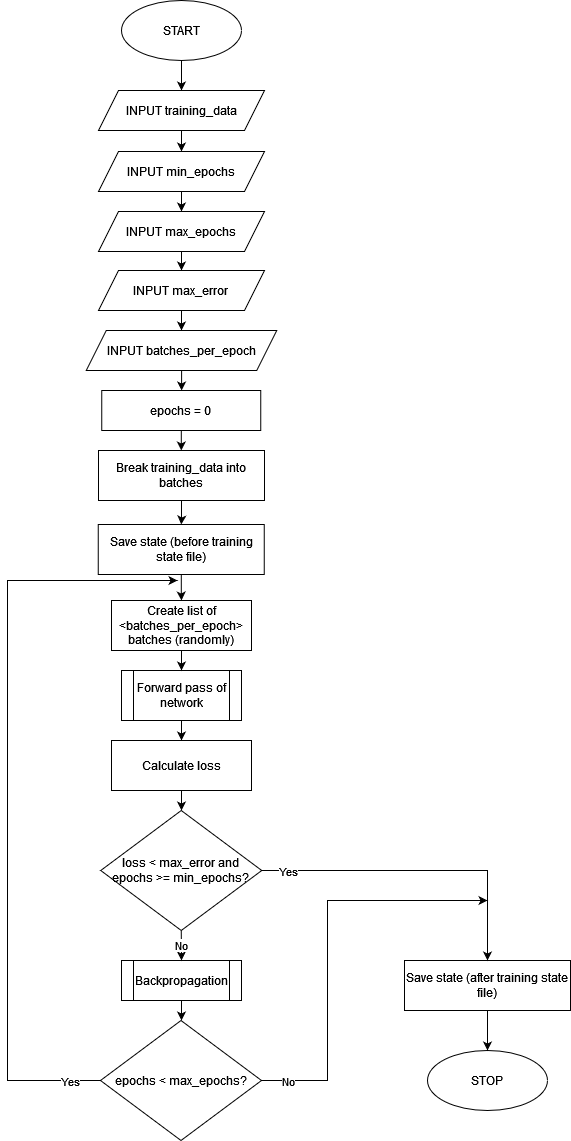
|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Train button pressed | Set main window’s page to training page | Open main window |
| Create button pressed | Set main window’s page to creation page | Open main window |
| Listen button pressed | Set main window’s page to listen page | Open main window |
| Settings button pressed | Set main window’s page to settings page | Open main window |

### Training State

Timeline

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Process** | **Storage** | **Output** |
| Minimum Epochs | Entered via a slider, so the main validation occurs here (Allowed values: 10-500)  When the user initially enters this page, it should show the default value which can be changed in settings. | Stored in a variable until it is passed to the NeuralNetwork object as a parameter | The neural network will train for *at least* the entered number of epochs. |
| Maximum Epochs | Entered via a slider, so the main validation occurs here (Allowed values: 100-2000)  When the user initially enters this page, it should show the default value which can be changed in settings. | Stored in a variable until it is passed to the NeuralNetwork object as a parameter | The neural network will train for *at most* the entered number of epochs. |
| Maximum Error | Entered via a slider, so the main validation occurs here (Allowed values: 0.0-0.5)  When the user initially enters this page, it should show the default value which can be changed in settings. | Stored in a variable until it is passed to the NeuralNetwork object as a parameter | The neural network will train until it has either done the maximum number of epochs, or it has done the minimum number of epochs and the estimated error is below the entered value |
| Batches per Epoch | Entered via a slider, so the main validation occurs here (Allowed values: 10-150)  When the user initially enters this page, it should show the default value which can be changed in settings. | Stored in a variable until it is passed to the NeuralNetwork object as a parameter | The neural network will use this number of batches per epoch.  The higher the value, the higher the amount of training done per epoch. |
| Revert Training Button | Prompts the user to ensure that they want to revert the training.  If the user picks yes then the NeuralNetwork should restore its state using the before state file.  If the user picks no then nothing should happen. | The before state file should be deleted. The after state file should be overwritten with the before state file’s contents. | N/A |
| Train Button | Starts network training.  The values of all the variables are as arguments to the neural network. | The NeuralNetwork will edit its state files during training. | Once training is complete, a message box should show to confirm training has ended. |



### Generation State

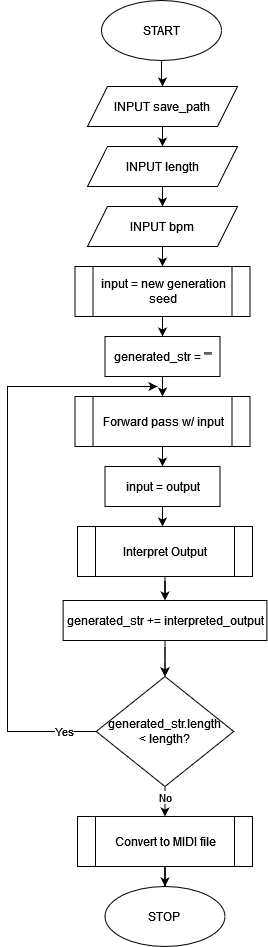
Since this is the main processing of the program and due to the fact that the output of the neural network is something which isn’t visibly outputted, the interface design for this stage of the program is also quite simple.

Just like the other states in the program, it includes a toolbar with links to the other parts of the program. For the neural network to generate anything, it requires a title and file path (in order to save the result) as well as the length of the generated piece. These are entered through textboxes (title and path) and through sliders (length).

A picture containing graphical user interface

Description automatically generated

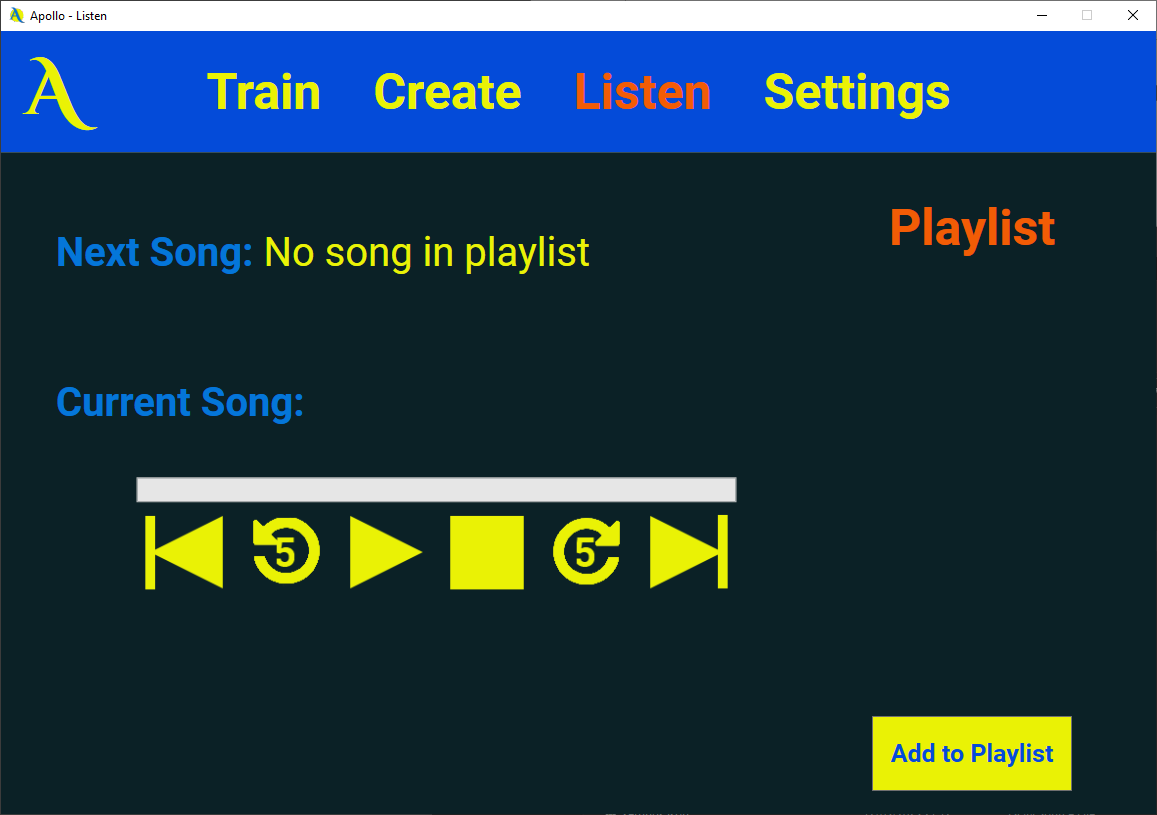
|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Process** | **Storage** | **Output** |
| Generation Length | Entered through a slider, so validation is managed by the UI element. (Allowed values: 5000-50000)   * When the user navigates to the generation screen, it starts on the number entered in settings | Stored in a variable until it is ready to be passed to the NeuralNetwork object as a parameter | The file generated from the neural network has a length proportionate to this value |
| Beats Per Minute | Entered through slider, so validation is managed by the UI element. (Allowed values: 30-120)   * When the user navigates to the generation screen, the slider is set to the default settings that can be changed in settings. | Stored in a variable until it is ready to be passed to the NeuralNetwork object as a parameter | The tempo of the file generated by the neural network is controlled by this input |
| Create Button | Opens a file dialog where the user can select the path where the generated file should be stored.  File dialog is apart of a library provided with WPF. Therefore the main bulk of validation is done here.  If the user closes the dialog without entering anything, then generation should not occur. | The values entered on the page via the sliders are passed to the NeuralNetwork object.  The NeuralNetwork generates the file. | The file generated by the network is saved to the path entered in the file dialog.  Once generation has ended, a message box should show confirming that generation has been completed. |



### Music Player

This state allows the user to play the generated MIDI files in app.

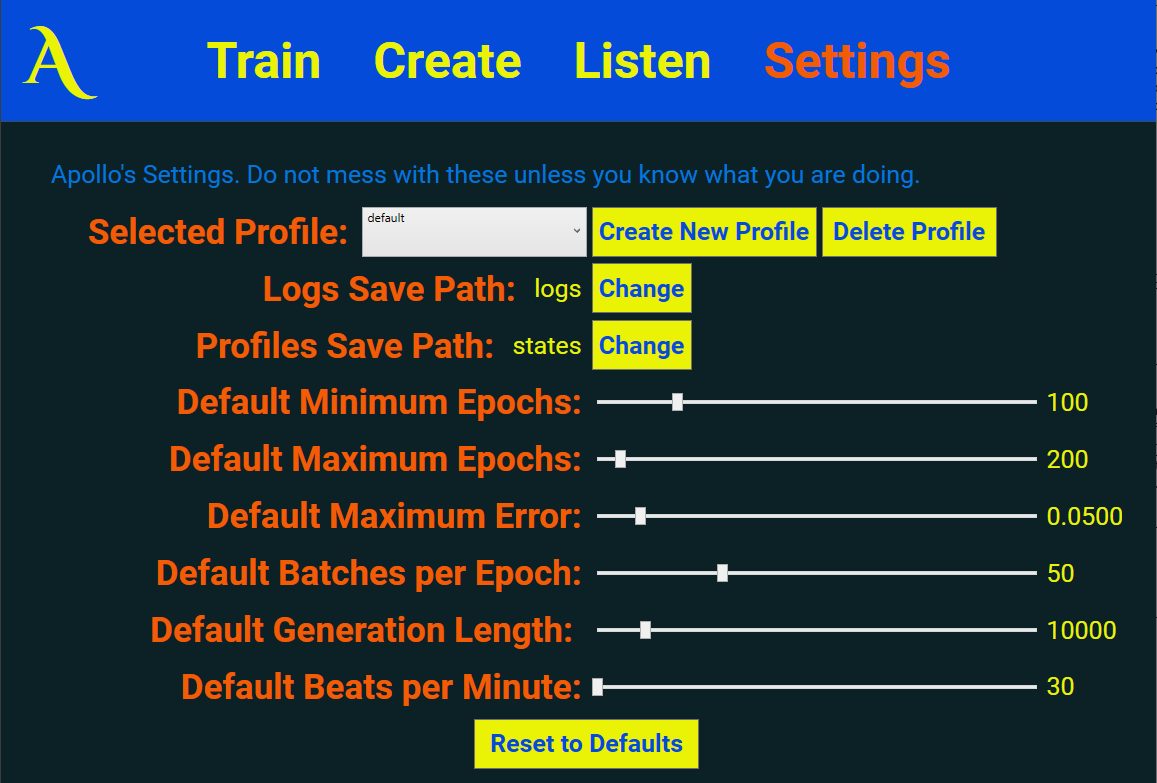
In terms of data structures, this page will use a queue and a stack. It will use a queue so that the user can select multiple songs to play, which will be played one after the other. The stack will be used to store the recently played songs, so that the user can play the previously (completed) song again.



|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Process** | **Storage** | **Output** |
| Add to Playlist | Open a File Dialog for the user to select a MIDI file to play. | The path of the selected MIDI file will be enqueued onto a queue representing the playlist | If there is nothing currently playing, then the file name will be shown next to the “Current song:” label.  Otherwise, it will be added to the appropriate place in the UI (either Next Song or under playlist depending on its position in the queue) |
| Previous Song Button | Pop the previous song off the stack and begin playing it. | The previous song will be removed from the stack | The previous song will begin playing |
| Rewind 5 Seconds Button | Rewind the playback by 5 seconds |  | The song will continue playing from 5 seconds behind its current position. |
| Play/Pause Button | If the icon shown is the pause icon – playback of the current song will be paused, and the icon will be switched for a play icon.  If the icon shown is the play icon – playback of the current song will be resumed, and the icon will be switched for a pause icon. |  | The button image will be correctly switched and the playback will be paused/played depending on what the clicked button was showing. |
| Stop Button | Stop playback + clear playlist | The queue representing the playlist will be cleared. The stack will be unchanged. | Playback is completely stopped. |
| Skip 5 Seconds Button | Advances playback position by 5 seconds |  | Playback will not be paused, but will continue playing from 5 seconds after its current position. |
| Skip Song Button | Playback of the current song is stopped | The path to the current song is added to the previous song stack | Playback of the next song begins. |

### Settings State

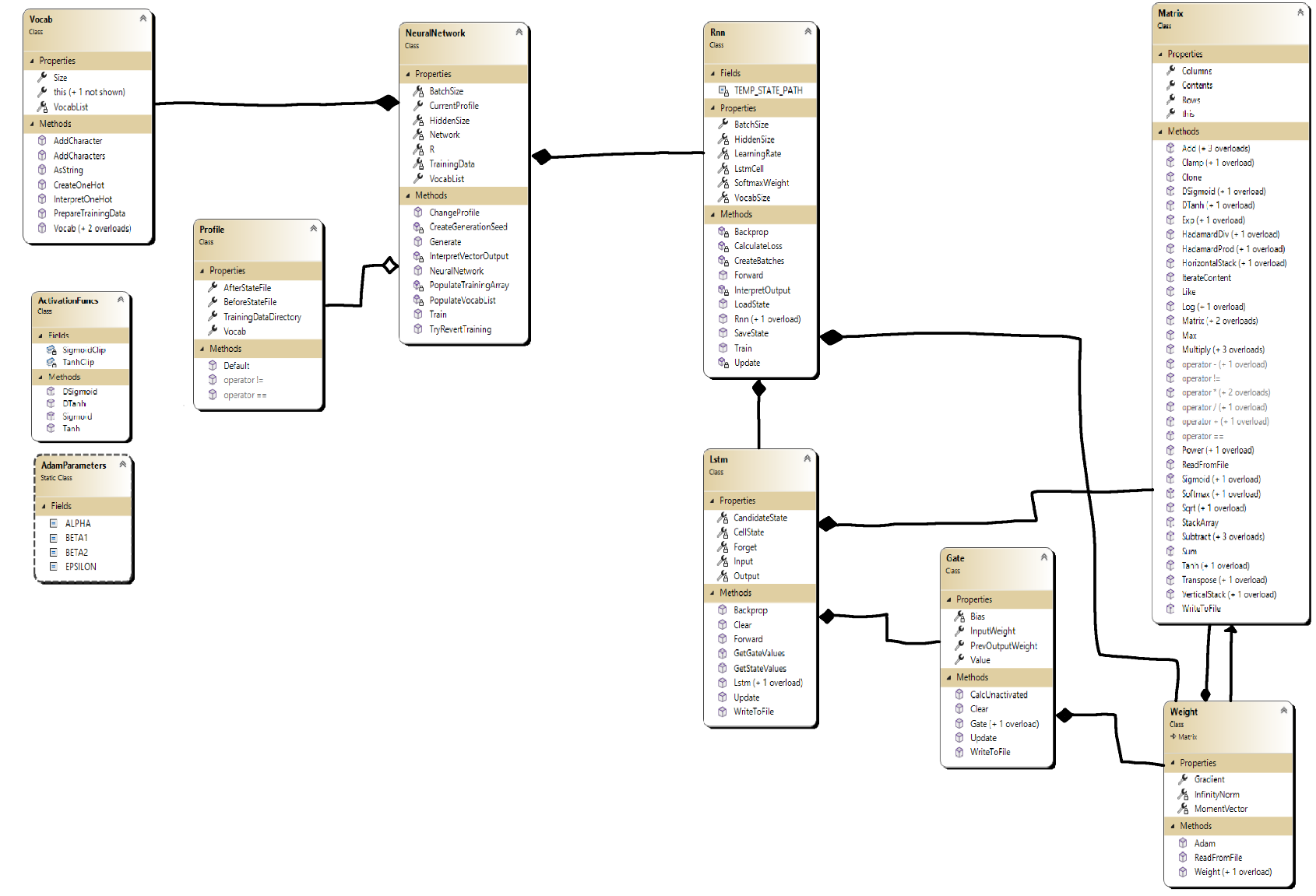
This state allows the user to edit the default settings of the application as well as change the “profile” (which state file they wish to use).



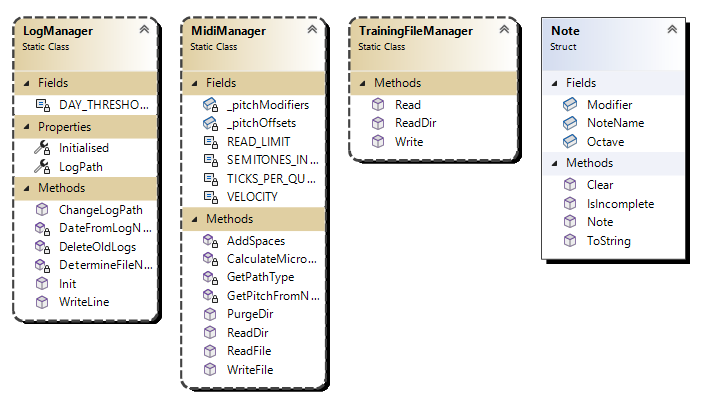
|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Process** | **Storage** | **Output** |
| Profile selected from “Selected Profile” combo box | The correct profile is loaded from the ProfileManager and passed to the NeuralNetwork object so that it can load the state file for that profile |  |  |
| Create new profile button | A text dialog opens to allow the user to enter the name of the profile (if the user closes without entering anything then nothing else happens)  Then a file dialog opens to allow the user to select the MIDI files for the profile.  The MIDI files are read in order to create a vocab list for the corresponding profile object  The new profile object is passed to the NeuralNetwork object so that it can load the state file for that profile | The ProfileManager object creates a profile directory to store the files associated with the profile  The MIDI files are copied over to the training data directory created by the ProfileManager object.  After the vocab list has been made, a string representation of the list is saved to the profile’s schema.json | A directory is created in the user’s file system with the data required for the profile  The combo box changes to the new profile |
| Delete profile button | Checks that the currently selected profile is not the only profile that the user has. If it is the only profile then nothing happens. | If it is allowed to be deleted, the directory associated with the profile is deleted from the user’s file system. | The combo box changes to the next profile in the list |
| Default Minimum Epochs | Entered via a slider, so the main validation occurs here (Allowed values: 10-500)  If the user attempts to input a value which is higher than maximum epochs, it will reset to its previous value.  If the entered value is valid, the value will be stored in the StoredSettings object. | If the user enters a valid value, this value will be saved to the settings.json file when the program is closed. | If the user attempts to input a value which is higher than maximum epochs, a message box will show stating that the change is not valid |
| Default Maximum Epochs | Entered via a slider, so the main validation occurs here (Allowed values: 100-2000)  If the user attempts to input a value which is lower than minimum epochs, it will reset to its previous value.  If the entered value is valid, the value will be stored in the StoredSettings object. | If the user enters a valid value, this value will be saved to the settings.json file when the program is closed. | If the user attempts to input a value which is lower than minimum epochs, a message box will show stating that the change is not valid |
| Default Maximum Error | Entered via a slider, so the main validation occurs here (Allowed values: 0.0-0.5) | If the user enters a valid value, this value will be saved to the settings.json file when the program is closed. |  |
| Default Batches per Epoch | Entered via a slider, so the main validation occurs here (Allowed values: 10-150) | If the user enters a valid value, this value will be saved to the settings.json file when the program is closed. |  |
| Default Generation Length | Entered through a slider, so validation is managed by the UI element. (Allowed values: 5000-50000) | If the user enters a valid value, this value will be saved to the settings.json file when the program is closed. |  |
| Default Beats per Minute | Entered through slider, so validation is managed by the UI element. (Allowed values: 30-120) | If the user enters a valid value, this value will be saved to the settings.json file when the program is closed. |  |

## Class Diagrams

### Neural Network Class Design

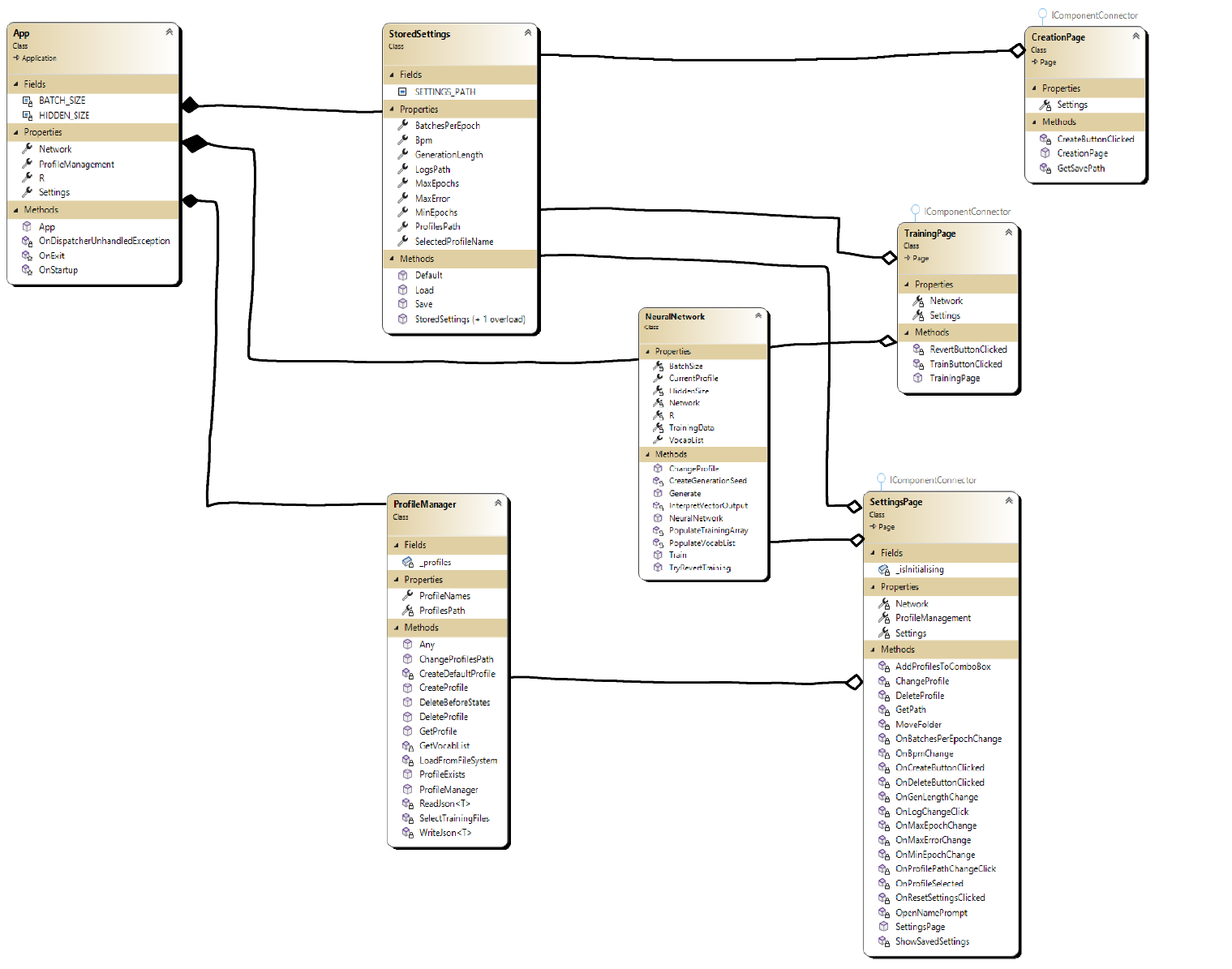
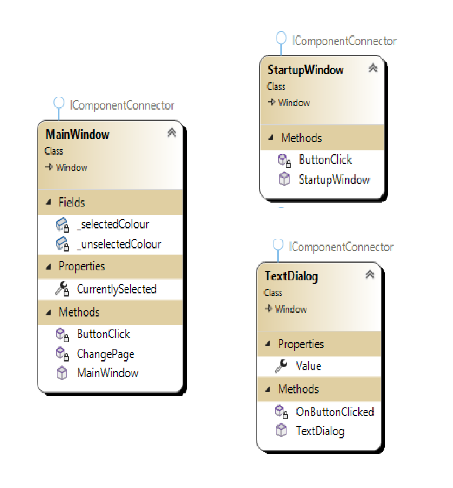


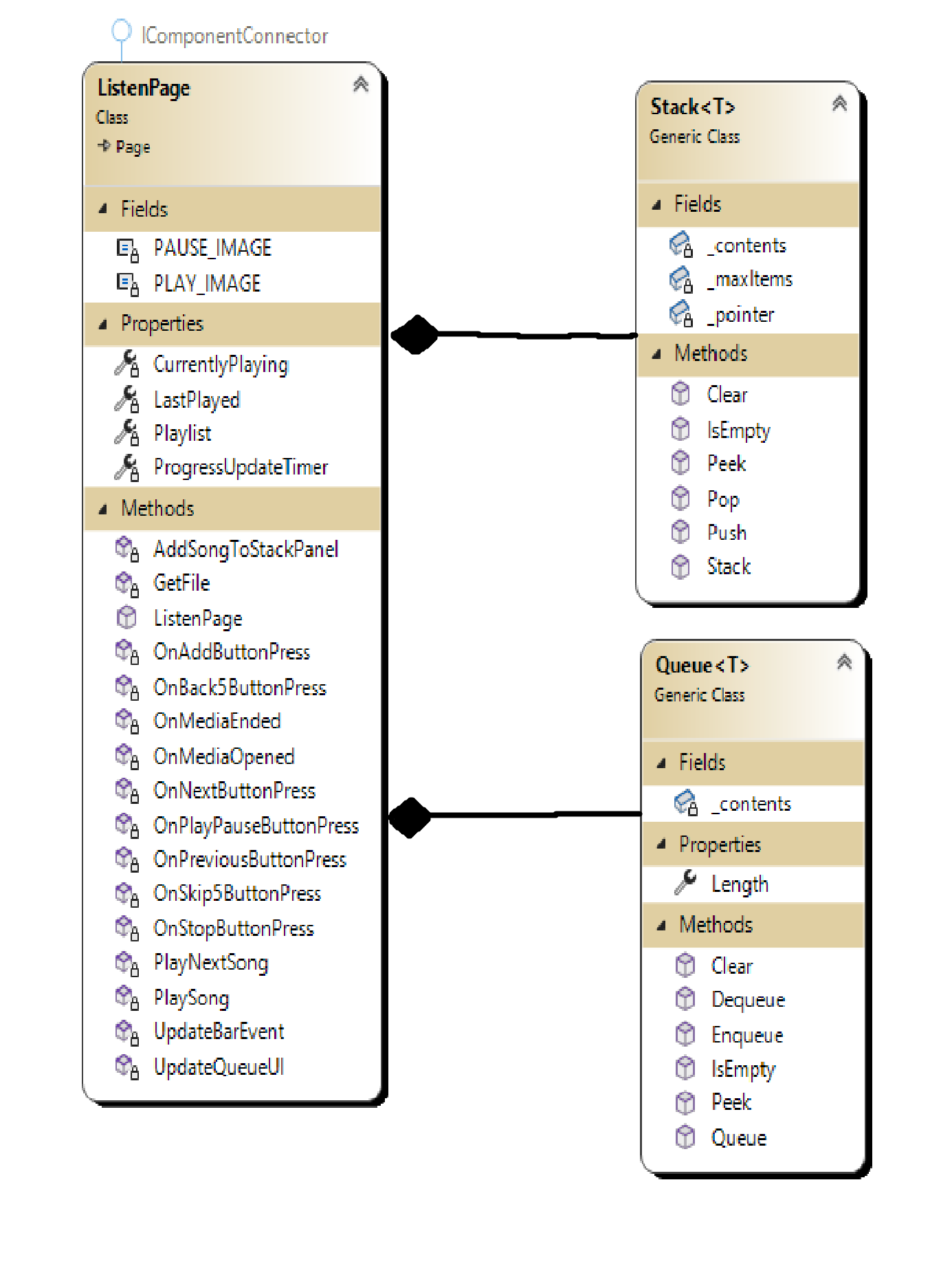
### File Management Class Design



I have used static classes here since the classes are used across all projects in the solution. This makes them easier to access without having to pass an object around methods.

### Frontend Class Design





# Technical Implementation

The implementation is written in C#, specifically .NET 7.0. It makes use of the WPF GUI framework (in the Apollo project), so can only be ran on a windows machine. The subheadings in this section represent the separate C# projects that make up the profile and the overall goal it tries to achieve.

## (Apollo.IO) Reading and writing files

This solution deals with the reading and writing of external files. Specifically, the writing of the log files as well as the reading and string interpretation of MIDI files.

### The LogManager Class

It was important to make this segment its own, separate class. Since the logs are used and accessed by various parts of the program, and thus I used OOP in order to prevent code duplication and redundancy. This class is also used by the other projects in order to make objective 4.6 possible.

using System.Globalization;  
  
namespace Apollo.IO;  
  
/// <summary>  
/// Class for managing and writing to a log text file  
/// </summary>  
public static class LogManager // Static class used so it is easily accessible across all libraries in the application  
{  
 private const int DAY\_THRESHOLD = 2;  
  
 private static string LogPath { get; set; }  
 private static bool Initialised { get; set; }  
  
 /// <summary>  
 /// Method ran once at the start of the program so that the LogManager can receive the path to the logs folder  
 /// </summary>  
 /// <param name="logPath">The path where the logs will be saved</param>  
 public static void Init(string logPath)  
 {  
 Initialised = true;  
  
 // Read from settings  
 LogPath = logPath;  
  
 // Create path for logs if it doesn't exist  
 if (!Directory.Exists(LogPath))  
 Directory.CreateDirectory(LogPath);  
  
 // Delete any old logs first  
 DeleteOldLogs();  
  
 // Get name for log file of current session   
 DetermineFileName();  
 }  
  
 /// <summary>  
 /// Change log path  
 /// </summary>  
 public static void ChangeLogPath(string newPath)  
 {  
 if (!Initialised)  
 {  
 Init(newPath);  
 return;  
 }  
  
 var logName = Path.GetFileName(LogPath);  
 LogPath = Path.Join(newPath, logName);  
 }  
  
 /// <summary>  
 /// Creates a filename which does not exist in the current log path  
 /// </summary>  
 /// <returns></returns>  
 private static void DetermineFileName()  
 {  
 if (!Initialised)  
 return;  
  
 // Do not continue if the LogPath is already to a file.   
 if (LogPath.EndsWith(".log")) return;  
  
 var date = DateTime.Now.ToString("yyyy-MM-dd");  
 var fileName = $"apollo\_logs\_{date}\_";  
  
 var currentId = -1;  
 var exists = true;  
 while (exists)  
 {  
 currentId++;  
 var path = Path.Join(LogPath, $"{fileName}{currentId}.log");  
 exists = File.Exists(path);  
 }  
  
 LogPath = Path.Join(LogPath, $"{fileName}{currentId}.log");  
 }  
  
 /// <summary>  
 /// Log some data  
 /// </summary>  
 /// <param name="data">The data to log</param>  
 public static void WriteLine(object data)  
 {  
 if (!Initialised)  
 return;  
  
 // Write the data to the console (if there is one)   
 Console.WriteLine(data);  
  
 // Create writer that does not overwrite the file  
 var writer = new StreamWriter(LogPath, true);  
  
 // Append the time (hours:minutes:seconds:miliseconds) to the data  
 var time = DateTime.Now.ToString("HH:mm:ss:fff");  
 data = $"[{time}]\n{data}";  
  
 writer.WriteLine(data);  
 writer.Flush();  
 writer.Close();  
 }  
  
 /// <summary>  
 /// Get the date from the name of the log file  
 /// </summary>  
 /// <param name="logName">The name of the log file</param>  
 /// <returns>A DateTime object of the date in the log file name</returns>  
 private static DateTime DateFromLogName(string logName)  
 {  
 // Example name: apollo\_logs\_2023-02-28\_8.log  
 // Splitting at \_'s will mean it is at index 2  
 var splitName = logName.Split('\_');  
 var stringDate = splitName[2];  
  
 return DateTime.ParseExact(stringDate, "yyyy-MM-dd", CultureInfo.CurrentCulture);  
 }  
  
 /// <summary>  
 /// Delete all old logs in the directory  
 /// </summary>  
 private static void DeleteOldLogs()  
 {  
 if (!Initialised)  
 return;  
  
 // Get all logs in the directory  
 var logFiles = Directory.GetFiles(LogPath).Where(fileName => fileName.EndsWith(".log"));  
  
 // Iterate through logs  
 foreach (var logPath in logFiles)  
 {  
 // Get date from filename   
 var logDate = DateFromLogName(Path.GetFileName(logPath));  
  
 // Check if created before threshold   
 var difference = DateTime.Now.Subtract(logDate);  
  
 if (difference.TotalDays >= DAY\_THRESHOLD)  
 // Delete if needed  
 File.Delete(logPath);  
 }  
 }  
}

### The Note Struct

The note struct is used to contain data about a single note. It is used by the MidiManager class in order to contain the related data and to reduce the amount of rogue variables.

namespace Apollo.IO;  
  
/// <summary>  
/// Struct to hold data about a single note  
/// </summary>  
internal struct Note  
{  
 public int Octave;  
 public char Modifier;  
 public char NoteName;  
  
 public override string ToString()  
 {  
 return $"{NoteName}{Octave}{Modifier}";  
 }  
  
 public Note()  
 {  
 Clear();  
 }  
  
 public void Clear()  
 {  
 Octave = -1;  
 Modifier = ' ';  
 NoteName = ' ';  
 }  
  
 public bool IsIncomplete()  
 {  
 // A note is "incomplete" if either the note name or the octave does not have an assigned value   
 return Octave == -1 || NoteName == ' ';  
 }  
}

### The MidiManager Class

This class handles the reading and writing of MidiFiles. It makes use of the NAudio library to facilitate both of these things. The code contained in this class does most of the work for objective 3. This is because the reading of files creates a string representation of the notes played within the file, and the writing of files deals with this same string representation in order to produce a MIDI file.

However, after a file is read, it is not yet ready to be inputted into the network hence why it does not fully satisfy 3.1, however this is satisfied by the Vocab class (in Apollo.NeuralNetwork) which converts the string representation into one-hot vectors and vice versa.

In order to simplify the application and learning process of the neural network, I decided to use constants to define the properties of each note played in the written file, so that the network can simply produce the notes themselves, rather than their duration, their velocity, etc. using NAudio.Midi;  
  
namespace Apollo.IO;  
  
/// <summary>  
/// Static class that handles anything to do with MidiFiles or music  
/// </summary>  
public static class MidiManager  
{  
 /// <summary>  
 /// Delete all MIDI files in a directory  
 /// </summary>  
 /// <param name="dirPath">The directory to delete from</param>  
 public static void PurgeDir(string dirPath)  
 {  
 foreach (var filePath in Directory.GetFiles(dirPath))  
 if (filePath.EndsWith(".mid") || filePath.EndsWith(".midi"))  
 File.Delete(filePath);  
 }  
  
 /// <summary>  
 /// Add a variable number of spaces to a string  
 /// </summary>  
 /// <param name="data">A reference to the string to add spaces to</param>  
 /// <param name="numSpaces">The number of spaces</param>  
 private static void AddSpaces(ref string data, int numSpaces)  
 {  
 var spaces = "";  
 Parallel.For(0, numSpaces, i => { spaces += " "; });  
 data += spaces;  
 }  
  
  
 /// <summary>  
 /// Calculate microseconds per quarter note from beats per minute  
 /// </summary>  
 /// <param name="bpm">Beats per minute</param>  
 /// <returns>Microseconds per quarter note</returns>  
 private static int CalculateMicrosecondsPerQuarterNote(int bpm)  
 {  
 // Number of microseconds in a minute ÷ bpm   
 // 60 seconds \* 1000 => milliseconds \* 1000 => microseconds  
 return 60 \* 1000 \* 1000 / bpm;  
 }  
  
 /// <summary>  
 /// Creates a MIDI value for passed in note  
 /// </summary>  
 /// <param name="note">String representation of note</param>  
 /// <returns>Integer of the MIDI representation of the note</returns>  
 private static int GetPitchFromNote(Note note)  
 {  
 // Start at the offsetted pitch for the letter (if there is one)   
 var noteValue = \_pitchOffsets.ContainsKey(note.NoteName) ? \_pitchOffsets[note.NoteName] : 0;  
  
 // Pitch modifiers can only be applied to notes w/ letter  
 if (\_pitchModifiers.ContainsKey(note.Modifier) && noteValue != 0)  
 noteValue += \_pitchModifiers[note.Modifier];  
  
 // -1 is used to state that there is no octave in the Note struct  
 if (note.Octave != -1)  
 noteValue += SEMITONES\_IN\_OCTAVE \* note.Octave;  
  
 return noteValue;  
 }  
  
 /// <summary>  
 /// Checks whether a path is valid  
 /// </summary>  
 /// <param name="path">The path to validate</param>  
 /// <returns>A character flag (d => directory, f => file, n => invalid path)</returns>  
 private static char GetPathType(string path)  
 {  
 var attributes = File.GetAttributes(path);  
  
 if (attributes == FileAttributes.Directory && Directory.Exists(path))  
 return 'd';  
  
 return File.Exists(path) ? 'f' : 'n';  
 }  
  
 #region Reading Files  
  
 private const int READ\_LIMIT = 10000;  
  
 /// <summary>  
 /// Read a MidiFile at the specified path  
 /// </summary>  
 /// <param name="path">Path to the MIDI file</param>  
 /// <returns>The string representation of the notes in the file</returns>  
 /// <exception cref="FileNotFoundException"></exception>  
 public static string ReadFile(string path)  
 {  
 if (GetPathType(path) != 'f')  
 throw new FileNotFoundException($"{path} is not a valid file path");  
  
 var file = new MidiFile(path, false);  
 var data = "";  
  
 var prevAbsoluteTime = 0L;  
 var numRead = 0;  
 var reachedMax = false;  
 for (var trackNum = 0; trackNum < file.Tracks && !reachedMax; trackNum++)  
 foreach (var e in file.Events[trackNum])  
 {  
 if (e.GetType() != typeof(NoteOnEvent))  
 continue;  
  
 var midiEvent = (NoteOnEvent)e;  
 var note = midiEvent.NoteName;  
 var absoluteTime = midiEvent.AbsoluteTime;  
  
 var timeDiff = Convert.ToInt32(absoluteTime - prevAbsoluteTime);  
 AddSpaces(ref data, timeDiff);  
 data += note;  
  
 prevAbsoluteTime = absoluteTime;  
 numRead++;  
  
 reachedMax = numRead > READ\_LIMIT;  
 }  
  
 return data;  
 }  
  
 /// <summary>  
 /// Reads all midi files in a given directory  
 /// </summary>  
 /// <param name="path">Path to the directory to read from</param>  
 /// <returns>List containing the string representation of every midi file in the directory</returns>  
 public static List<string> ReadDir(string path)  
 {  
 // Check path is a directory   
 if (GetPathType(path) != 'd')  
 throw new DirectoryNotFoundException($"{path} is not a valid directory");  
  
 var dirData = new List<string>();  
  
 // Iterate through all files in directory  
 // Call ReadFile with the path   
 // Add the result to a list  
 var dirInfo = new DirectoryInfo(path);  
 var dirFiles = dirInfo.GetFiles();  
  
 foreach (var fileInfo in dirFiles)  
 {  
 // Check that the file is a midi file  
 if (!fileInfo.Name.EndsWith(".mid") && !fileInfo.Name.EndsWith(".midi"))  
 continue;  
  
 var filePath = fileInfo.FullName;  
 var fileData = ReadFile(filePath);  
 dirData.Add(fileData);  
 }  
  
 return dirData;  
 }  
  
 #endregion  
  
 #region Writing Files  
  
 // There are 12 semitones in an octave  
 private const int SEMITONES\_IN\_OCTAVE = 12;  
  
 // MIDI file parameters  
 private const int TICKS\_PER\_QUARTER\_NOTE = 480;  
 private const int VELOCITY = 100;  
  
 // How each note offsets the pitch  
 private static readonly Dictionary<char, int> \_pitchOffsets = new()  
 {  
 { 'C', 0 },  
 { 'D', 2 },  
 { 'E', 4 },  
 { 'F', 5 },  
 { 'G', 7 },  
 { 'A', 9 },  
 { 'B', 11 }  
 };  
  
 // How sharps/flats affects the pitch  
 private static readonly Dictionary<char, int> \_pitchModifiers = new()  
 {  
 { '#', 1 },  
 { 'b', -1 }  
 };  
  
 /// <summary>  
 /// Writes a string representation to a MIDI file  
 /// </summary>  
 /// <param name="data">The string representation of the file</param>  
 /// <param name="path">The path of the file to write it to</param>  
 /// <param name="beatsPerMinute">Music's beat per minute</param>  
 public static void WriteFile(string data, string path, int beatsPerMinute)  
 {  
 // Magic number, change later  
 var collection = new MidiEventCollection(0, TICKS\_PER\_QUARTER\_NOTE);  
  
 // Notes last for 3/4 of a MIDI tick  
 var noteDur = 3 / 4 \* TICKS\_PER\_QUARTER\_NOTE;  
  
 var absoluteTime = 0L;  
  
 // Create tempo event so that the MidiFile is valid  
 var tempoEvent = new TempoEvent(CalculateMicrosecondsPerQuarterNote(beatsPerMinute), absoluteTime);  
 collection.AddEvent(tempoEvent, 1);  
  
 var currentNote = new Note();  
 foreach (var c in data)  
 {  
 // Space = increment absolute time   
 if (c == ' ')  
 {  
 absoluteTime++;  
 currentNote.Clear(); // Note components must be one "unit"   
 continue;  
 }  
  
 // Add information to note  
 if (char.IsUpper(c) && currentNote.NoteName == ' ')  
 currentNote.NoteName = c;  
 else if (\_pitchModifiers.ContainsKey(c) && currentNote.Modifier == ' ' && currentNote.NoteName == ' ')  
 currentNote.Modifier = c;  
 else if (char.IsNumber(c) && currentNote.Octave == -1 && currentNote.NoteName == ' ')  
 currentNote.Octave = (int)char.GetNumericValue(c);  
  
 // If the information about the note isn't complete then continue  
 if (currentNote.IsIncomplete())  
 continue;  
  
 // Gets here when enough information was gathered for a complete note  
 // Add note to collection   
 var pitch = GetPitchFromNote(currentNote);  
  
 var onEvent = new NoteOnEvent(absoluteTime, 1, pitch, VELOCITY, noteDur);  
 var offEvent = new NoteEvent(absoluteTime + noteDur, 1, MidiCommandCode.NoteOff, pitch, 0);  
  
 collection.AddEvent(onEvent, 1);  
 collection.AddEvent(offEvent, 1);  
  
 currentNote.Clear();  
 }  
  
 collection.PrepareForExport();  
 MidiFile.Export(path, collection);  
 }  
  
 #endregion  
}

### The TrainingFileManager Class

This class is responsible for the reading and writing of data from “training data files”. The training data files contain the vectorised form of the MIDI files. When the MIDI files are read and handled by the MIDI manager and the network. This static class writes the data to a binary file so that the entire MIDI file does not have to be re-read and re-converted when it is required again.

using Apollo.MatrixMaths;  
  
namespace Apollo.IO;  
  
/// <summary>  
/// Static class responsible for reading and writing Training Data files  
/// </summary>  
public static class TrainingFileManager  
{  
 /// <summary>  
 /// Write training data to a file  
 /// </summary>  
 /// <param name="trainingData">The training data to write to the file</param>  
 /// <param name="filePath">The path of the file to write it to</param>  
 public static void Write(Matrix[] trainingData, string filePath)  
 {  
 var vocabLength = trainingData[0].Columns;  
 using (var stream = File.Open(filePath, FileMode.Create))  
 {  
 using (var writer = new BinaryWriter(stream))  
 {  
 writer.Write(vocabLength);  
 writer.Write(trainingData.Length);  
 foreach (var vector in trainingData) vector.WriteToFile(writer);  
 }  
 }  
 }  
  
 /// <summary>  
 /// Read training data from a file  
 /// </summary>  
 /// <param name="filePath">The path of the file to read from</param>  
 /// <returns>The training data contained in the file</returns>  
 public static Matrix[] Read(string filePath)  
 {  
 if (!File.Exists(filePath) || !filePath.EndsWith(".td"))  
 throw new FileNotFoundException($"{filePath} is not a valid file");  
  
 Matrix[] trainingData;  
  
 using (var stream = File.Open(filePath, FileMode.Open))  
 {  
 using (var reader = new BinaryReader(stream))  
 {  
 var vocabSize = reader.ReadInt32();  
 var arrayLength = reader.ReadInt32();  
  
 trainingData = new Matrix[arrayLength];  
  
 for (var i = 0; i < arrayLength; i++) trainingData[i] = Matrix.ReadFromFile(reader, 1, vocabSize);  
 }  
 }  
  
 return trainingData;  
 }  
   
 /// <summary>  
 /// Calls TrainingFileManager.Read on all training data files in a directory.  
 /// </summary>  
 /// <param name="dirPath">The directory to read from</param>  
 /// <returns>An array of the data retrieved from the directory</returns>  
 public static Matrix[][] ReadDir(string dirPath)  
 {  
 if (!Directory.Exists(dirPath))  
 throw new DirectoryNotFoundException($"{dirPath} is not a valid directory");  
  
 var files = Directory.GetFiles(dirPath).Where(fileName => fileName.EndsWith(".td")).ToArray();  
 var trainingData = new Matrix[files.Length][];  
  
 for (var i = 0; i < trainingData.Length; i++)  
 {  
 var filePath = files[i];  
 var fileData = Read(filePath);  
 trainingData[i] = fileData;  
 }  
  
 return trainingData;  
 }  
}

## (Apollo.MatrixMaths) Matrix arithmetic and operations

This class library is responsible for the classes that make matrix operations possible. It contains the matrix class itself, as well as a class containing some of the commonly used, but previously undefined maths functions.

### The ActivationFuncs Class

This class contains the maths functions (specifically the activation functions) that are used by the network as well as their derivatives. Even though Tanh is already defined by the C# Maths module, it was reimplemented in order to clip the possible input values in order to avoid a NaN being returned.

This class targets the second half of objective 1.3.

using System;  
  
namespace Apollo.MatrixMaths;  
  
public class ActivationFuncs  
{  
 // Values taken from the graphs   
 private static readonly int TanhClip = 4; // tanh(x)  
  
 private static readonly int SigmoidClip = 6; // 1 / (1 + e^-x)  
  
 /// <summary>  
 /// Clipped Tanh (in order to avoid NaN)  
 /// </summary>  
 public static float Tanh(float x)  
 {  
 // Values taken from the tanh graph  
 if (x > TanhClip)  
 return 1;  
 if (x < -TanhClip)  
 return -1;  
  
 return (float)Math.Tanh(x);  
 }  
  
 /// <summary>  
 /// Derivative of hyperbolic tangent  
 /// </summary>  
 public static float DTanh(float x)  
 {  
 // tanh'(x) = 0 when tanh(x) = 1 or tanh(x) = -1 (since the function no longer increases or decreases)  
 if (x > TanhClip || x < -TanhClip)  
 return 0;  
  
 return 1 / (float)Math.Pow(Math.Cosh(x), 2);  
 }  
  
 /// <summary>  
 /// Clipped Sigmoid function (in order to avoid NaN)  
 /// </summary>  
 public static float Sigmoid(float x)  
 {  
 // Values taken from the graph of the function   
 if (x > SigmoidClip)  
 return 1;  
 if (x < -SigmoidClip)  
 return 0;  
  
 return 1 / (1 + (float)Math.Exp(-x));  
 }  
  
 /// <summary>  
 /// Derivative of the sigmoid function  
 /// </summary>  
 public static float DSigmoid(float x)  
 {  
 // sigmoid'(x) = 0 when sigmoid(x) = 1 or sigmoid(x) = -1 (since the function no longer increases or decreases)  
 if (x > SigmoidClip || x < -SigmoidClip)  
 return 0;  
  
 return (float)Math.Exp(-x) / (float)Math.Pow(1 + Math.Exp(-x), 2);  
 }  
}

### The Matrix Class

This class fulfils objective 1.3 since it implements all the matrix operations used by the network as well as the activations functions defined in the ActivationFuncs class. It is based upon a 2D array of floats which contains the elements in the matrix and an “IterateContents” is used throughout for operations which effect each individual item in the matrix.

Since matrices in the network are potentially very large, the more commonly used operations implement threading, in order to speed up the execution time of a single operation.

Each operation has a static, and a non-static variant. Its static variant performs the operation, and returns a new matrix so it avoids editing the operands. The non-static variant performs the operation upon the array it was called on, meaning its contents are changed.

using System;  
using System.IO;  
using System.Threading.Tasks;  
  
namespace Apollo.MatrixMaths;  
  
// Generic Matrix   
public class Matrix  
{  
 /// <summary>  
 /// Generate matrix full of zeros  
 /// </summary>  
 /// <param name="rows">Number of rows in matrix</param>  
 /// <param name="columns">Number of columns in matrix</param>  
 public Matrix(int rows, int columns)  
 {  
 Contents = new float[rows, columns];  
 }  
  
 /// <summary>  
 /// Create a matrix with already defined data  
 /// </summary>  
 /// <param name="defaultData">Data contained in the matrix</param>  
 public Matrix(float[,] defaultData)  
 {  
 Contents = defaultData;  
 }  
  
 /// <summary>  
 /// Create a matrix of random values  
 /// </summary>  
 /// <param name="rows">Number of rows in the matrix</param>  
 /// <param name="columns">Number of columns in the matrix</param>  
 /// <param name="r">Instance of Random</param>  
 /// <param name="min">Minimum value</param>  
 /// <param name="max">Maximum value</param>  
 public Matrix(int rows, int columns, Random r, int min = -2, int max = 2)  
 {  
 Contents = new float[rows, columns];  
 IterateContent(value => r.Next(min, max) + (float)r.NextDouble());  
 }  
  
 public float[,] Contents { get; private set; }  
  
 public float this[int i, int j]  
 {  
 get => Contents[i, j];  
 set => Contents[i, j] = value;  
 }  
  
 public int Rows => Contents.GetLength(0);  
 public int Columns => Contents.GetLength(1);  
  
 /// <summary>  
 /// Create a new matrix with the same shape as another matrix  
 /// </summary>  
 /// <param name="matrix">Matrix to copy the shape of</param>  
 public static Matrix Like(Matrix matrix)  
 {  
 return new Matrix(matrix.Rows, matrix.Columns);  
 }  
  
 /// <summary>  
 /// Apply a function over each element in the matrix (used for tanh, sqrt, etc.)  
 /// </summary>  
 /// <param name="contentAction">Function to apply to every value in the matrix</param>  
 public void IterateContent(Func<float, float> contentAction)  
 {  
 Parallel.For(0, Rows, i =>  
 {  
 for (var j = 0; j < Columns; j++)  
 Contents[i, j] = contentAction(Contents[i, j]);  
 });  
 }  
  
 /// <summary>  
 /// Apply the hyperbolic tangent to each element in the matrix  
 /// </summary>  
 public void Tanh()  
 {  
 IterateContent(ActivationFuncs.Tanh);  
 }  
  
 public static Matrix Tanh(Matrix mat)  
 {  
 var returnMat = mat.Clone();  
 returnMat.Tanh();  
 return returnMat;  
 }  
  
 /// <summary>  
 /// Apply the derivative of the hyperbolic tangent to each element in the matrix  
 /// </summary>  
 public void DTanh()  
 {  
 IterateContent(ActivationFuncs.DTanh);  
 }  
  
 public static Matrix DTanh(Matrix mat)  
 {  
 var returnMat = mat.Clone();  
 returnMat.DTanh();  
 return returnMat;  
 }  
  
 /// <summary>  
 /// Apply the sigmoid activation function to each element in the matrix  
 /// </summary>  
 public void Sigmoid()  
 {  
 IterateContent(ActivationFuncs.Sigmoid);  
 }  
  
 public static Matrix Sigmoid(Matrix mat)  
 {  
 var returnMat = mat.Clone();  
 returnMat.Sigmoid();  
 return returnMat;  
 }  
  
 /// <summary>  
 /// Apply the derivative of the sigmoid function to each element in the matrix  
 /// </summary>  
 public void DSigmoid()  
 {  
 IterateContent(ActivationFuncs.DSigmoid);  
 }  
  
 public static Matrix DSigmoid(Matrix mat)  
 {  
 var returnMat = mat.Clone();  
 returnMat.DSigmoid();  
 return returnMat;  
 }  
  
 /// <summary>  
 /// Square root each element in the matrix  
 /// </summary>  
 public void Sqrt()  
 {  
 IterateContent(MathF.Sqrt);  
 }  
  
 public static Matrix Sqrt(Matrix mat)  
 {  
 var returnMat = mat.Clone();  
 returnMat.Sqrt();  
 return returnMat;  
 }  
  
 /// <summary>  
 /// Put euler's number to the power of each element in the matrix  
 /// </summary>  
 public void Exp()  
 {  
 IterateContent(MathF.Exp);  
 }  
  
 public static Matrix Exp(Matrix mat)  
 {  
 var returnMat = mat.Clone();  
 returnMat.Exp();  
 return returnMat;  
 }  
  
 /// <summary>  
 /// Raise each element to a power  
 /// </summary>  
 /// <param name="power">The power to raise each element to</param>  
 public void Power(float power)  
 {  
 IterateContent(value => MathF.Pow(value, power));  
 }  
  
 public static Matrix Power(Matrix mat, float power)  
 {  
 var returnMat = mat.Clone();  
 returnMat.Power(power);  
 return returnMat;  
 }  
  
 /// <summary>  
 /// Multiply two matrices together using standard matrix multiplication  
 /// </summary>  
 /// <param name="otherMat">The other operand</param>  
 public void Multiply(Matrix otherMat)  
 {  
 // This matrix is multiplicatively conformable to otherMat if and only if:  
 // this.columns = otherMat.rows   
 if (Columns != otherMat.Rows)  
 throw new InvalidShapeException("First matrix isn't multiplicatively conformable to the other",  
 this, otherMat);  
  
 var newContents = new float[Rows, otherMat.Columns];  
  
 Parallel.For(0, Rows, row =>  
 {  
 for (var col = 0; col < otherMat.Columns; col++)  
 {  
 // Row in A \* Col in B   
 // A has as many columns as B has rows therefore A's columns has the same amount of numbers as B's rows   
 float sum = 0;  
  
 for (var i = 0; i < Columns; i++)  
 sum += Contents[row, i] \* otherMat.Contents[i, col];  
  
 newContents[row, col] = sum;  
 }  
 });  
  
 Contents = newContents;  
 }  
  
 public static Matrix Multiply(Matrix mat, Matrix otherMat)  
 {  
 var returnMat = mat.Clone();  
 returnMat.Multiply(otherMat);  
 return returnMat;  
 }  
  
 /// <summary>  
 /// Calculate the sum of every element in the matrix  
 /// </summary>  
 /// <returns>The sum of every element in the matrix</returns>  
 public float Sum()  
 {  
 float sum = 0;  
  
 for (var i = 0; i < Contents.GetLength(0); i++)  
 for (var j = 0; j < Contents.GetLength(1); j++)  
 sum += Contents[i, j];  
  
 return sum;  
 }  
  
 /// <summary>  
 /// Transpose the matrix  
 /// </summary>  
 public void Transpose()  
 {  
 var newContents = new float[Columns, Rows];  
  
 for (var i = 0; i < Rows; i++)  
 for (var j = 0; j < Columns; j++)  
 newContents[j, i] = Contents[i, j];  
  
 Contents = newContents;  
 }  
  
 public static Matrix Transpose(Matrix mat)  
 {  
 var returnMatrix = mat.Clone();  
 returnMatrix.Transpose();  
 return returnMatrix;  
 }  
  
 /// <summary>  
 /// Clamp each element of the matrix between two values  
 /// </summary>  
 /// <param name="min">The minimum possible value in the matrix</param>  
 /// <param name="max">The maximum possible value in the matrix</param>  
 public void Clamp(float min, float max)  
 {  
 IterateContent(value => Math.Clamp(value, min, max));  
 }  
  
 public static Matrix Clamp(Matrix mat, int min, int max)  
 {  
 var returnMat = mat.Clone();  
 returnMat.Clamp(min, max);  
 return returnMat;  
 }  
  
 /// <summary>  
 /// Concatenate two matrices horizontally  
 /// </summary>  
 /// <param name="otherMat">The matrix to concatenate to this one</param>  
 public void HorizontalStack(Matrix otherMat)  
 {  
 // Must have same amount of rows   
 if (otherMat.Rows != Rows)  
 throw new InvalidShapeException("You can only horizontally stack matrices with the same amount of rows",  
 this, otherMat);  
  
 // New shape = (same rows, sum of columns)  
 var newContents = new float[Rows, Columns + otherMat.Columns];  
  
 for (var i = 0; i < newContents.GetLength(0); i++)  
 for (var j = 0; j < newContents.GetLength(1); j++)  
 {  
 var insertData = j >= Columns ? otherMat.Contents[i, j - Columns] : Contents[i, j];  
 newContents[i, j] = insertData;  
 }  
  
 Contents = newContents;  
 }  
  
 public static Matrix HorizontalStack(Matrix mat, Matrix otherMat)  
 {  
 var returnMat = mat.Clone();  
 returnMat.HorizontalStack(otherMat);  
 return returnMat;  
 }  
  
 /// <summary>  
 /// Concatenate two matrices vertically  
 /// </summary>  
 /// <param name="otherMat">The matrix to concatenate to this one</param>  
 public void VerticalStack(Matrix otherMat)  
 {  
 // Must have same amount of columns   
 if (otherMat.Columns != Columns)  
 throw new InvalidShapeException("You can only horizontally stack matrices with the same amount of rows",  
 this, otherMat);  
  
 // New shape = (same rows, sum of columns)  
 var newContents = new float[Rows + otherMat.Rows, Columns];  
  
 for (var i = 0; i < newContents.GetLength(0); i++)  
 for (var j = 0; j < newContents.GetLength(1); j++)  
 {  
 var insertData = i >= Rows ? otherMat.Contents[i - Rows, j] : Contents[i, j];  
 newContents[i, j] = insertData;  
 }  
  
 Contents = newContents;  
 }  
  
 public static Matrix VerticalStack(Matrix mat, Matrix otherMat)  
 {  
 var returnMat = mat.Clone();  
 returnMat.VerticalStack(otherMat);  
 return returnMat;  
 }  
  
 /// <summary>  
 /// Add two matrices together  
 /// </summary>  
 /// <param name="otherMat">The matrix to add to this one</param>  
 public void Add(Matrix otherMat)  
 {  
 // Shapes must be the same   
 if (otherMat.Rows != Rows || otherMat.Columns != Columns)  
 throw new MatrixArithmeticException("Matrices aren't additively applicable (not the same shape)");  
  
 for (var i = 0; i < Rows; i++)  
 for (var j = 0; j < Columns; j++)  
 Contents[i, j] += otherMat.Contents[i, j];  
 }  
  
 public static Matrix Add(Matrix mat, Matrix otherMat)  
 {  
 var returnMat = mat.Clone();  
 returnMat.Add(otherMat);  
 return returnMat;  
 }  
  
 /// <summary>  
 /// Subtract two matrices  
 /// </summary>  
 /// <param name="otherMat">The matrix to take away from this one</param>  
 public void Subtract(Matrix otherMat)  
 {  
 // Shapes must be the same   
 if (otherMat.Rows != Rows || otherMat.Columns != Columns)  
 throw new MatrixArithmeticException("Matrices aren't additively applicable (not the same shape)");  
  
 for (var i = 0; i < Rows; i++)  
 for (var j = 0; j < Columns; j++)  
 Contents[i, j] -= otherMat.Contents[i, j];  
 }  
  
 public static Matrix Subtract(Matrix mat, Matrix otherMat)  
 {  
 var returnMat = mat.Clone();  
 returnMat.Subtract(otherMat);  
 return returnMat;  
 }  
  
 /// <summary>  
 /// Subtract a scalar quantity from each element in the matrix  
 /// </summary>  
 /// <param name="scalar">Quantity to subtract</param>  
 public void Subtract(float scalar)  
 {  
 for (var i = 0; i < Rows; i++)  
 for (var j = 0; j < Columns; j++)  
 Contents[i, j] -= scalar;  
 }  
  
 public static Matrix Subtract(Matrix mat, float scalar)  
 {  
 var returnMat = mat.Clone();  
 returnMat.Subtract(scalar);  
 return returnMat;  
 }  
  
 /// <summary>  
 /// Add a scalar quantity to each element in the matrix  
 /// </summary>  
 /// <param name="scalar">Quantity to add</param>  
 public void Add(float scalar)  
 {  
 for (var i = 0; i < Rows; i++)  
 for (var j = 0; j < Columns; j++)  
 Contents[i, j] += scalar;  
 }  
  
 public static Matrix Add(Matrix mat, float scalar)  
 {  
 var returnMat = mat.Clone();  
 returnMat.Add(scalar);  
 return returnMat;  
 }  
  
 /// <summary>  
 /// Multiply each element in the matrix by a scalar quantity  
 /// </summary>  
 /// <param name="scalar">Quantity to multiply by</param>  
 public void Multiply(float scalar)  
 {  
 for (var i = 0; i < Rows; i++)  
 for (var j = 0; j < Columns; j++)  
 Contents[i, j] \*= scalar;  
 }  
  
 public static Matrix Multiply(Matrix mat, float scalar)  
 {  
 var returnMat = mat.Clone();  
 returnMat.Multiply(scalar);  
 return returnMat;  
 }  
  
 /// <summary>  
 /// Element wise multiplication between two matrices  
 /// </summary>  
 /// <param name="otherMat">Matrix to multiply by</param>  
 public void HadamardProd(Matrix otherMat)  
 {  
 if (otherMat.Rows != Rows || otherMat.Columns != Columns)  
 throw new InvalidShapeException("Matrices must be the same shape for Hadamard multiplication",  
 this, otherMat);  
  
 for (var i = 0; i < Rows; i++)  
 for (var j = 0; j < Columns; j++)  
 Contents[i, j] \*= otherMat.Contents[i, j];  
 }  
  
 public static Matrix HadamardProd(Matrix mat1, Matrix mat2)  
 {  
 var returnMat = mat1.Clone();  
 returnMat.HadamardProd(mat2);  
 return returnMat;  
 }  
  
 /// <summary>  
 /// Element wise division between two matrices  
 /// </summary>  
 /// <param name="otherMat">Matrix to divide by</param>  
 public void HadamardDiv(Matrix otherMat)  
 {  
 if (otherMat.Rows != Rows || otherMat.Columns != Columns)  
 throw new InvalidShapeException("Matrices must be the same shape for Hadamard division", this, otherMat);  
  
 for (var i = 0; i < Rows; i++)  
 for (var j = 0; j < Columns; j++)  
 Contents[i, j] /= otherMat.Contents[i, j];  
 }  
  
 public static Matrix HadamardDiv(Matrix mat1, Matrix mat2)  
 {  
 var returnMat = mat1.Clone();  
 returnMat.HadamardDiv(mat2);  
 return returnMat;  
 }  
  
 public void Softmax()  
 {  
 // If not implemented properly, this function could produce NaNs   
 // This is because e^x is very large when x isn't that big, thus could produce an overflow error   
 // To counter-act this, we subtract the highest number in the matrix from every matrix element, and then   
 // calculate the softmax since this operation does not change the result   
  
 // Softmaxed matrices are considered row by row, so only apply softmax to individual rows then recompile the matrix  
  
 var matRows = new Matrix[Rows];  
  
 for (var i = 0; i < Rows; i++)  
 {  
 var rowContent = new float[1, Columns];  
 for (var j = 0; j < Columns; j++) rowContent[0, j] = Contents[i, j];  
  
 matRows[i] = new Matrix(rowContent);  
  
 var highestNumber = matRows[i].Max();  
 matRows[i].IterateContent(value => value - highestNumber);  
 matRows[i].Exp();  
 var sum = matRows[i].Sum();  
 matRows[i].Multiply(1 / sum);  
 }  
  
 // Compile the softmax rows into one matrix  
 var compiled = StackArray(matRows);  
 Contents = compiled.Contents;  
 }  
  
 public static Matrix Softmax(Matrix matrix)  
 {  
 var returnMat = matrix.Clone();  
 returnMat.Softmax();  
 return returnMat;  
 }  
  
 /// <summary>  
 /// Get the highest value in the matrix  
 /// </summary>  
 public float Max()  
 {  
 var max = float.MinValue;  
 for (var i = 0; i < Rows; i++)  
 for (var j = 0; j < Columns; j++)  
 if (Contents[i, j] > max)  
 max = Contents[i, j];  
  
 return max;  
 }  
  
 /// <summary>  
 /// Apply a logarithm to each element in the matrix  
 /// </summary>  
 /// <param name="logBase">The base of the logarithm</param>  
 public void Log(float logBase)  
 {  
 IterateContent(value => MathF.Log(value, logBase));  
 }  
  
 public static Matrix Log(Matrix matrix, float logBase)  
 {  
 var returnMat = matrix.Clone();  
 returnMat.Log(logBase);  
 return returnMat;  
 }  
  
 /// <summary>  
 /// Stack a list of matrices into a single matrix  
 /// </summary>  
 /// <param name="matrices">List of matrices to stack</param>  
 /// <param name="vertically">Flag variable specificing whether to stack vertically</param>  
 /// <returns>A compiled matrix</returns>  
 public static Matrix StackArray(Matrix[] matrices, bool vertically = true)  
 {  
 var mat = matrices[0].Clone();  
  
 for (var i = 1; i < matrices.Length; i++)  
 if (vertically)  
 mat.VerticalStack(matrices[i]);  
 else  
 mat.HorizontalStack(matrices[i]);  
  
 return mat;  
 }  
  
 /// <summary>  
 /// Creates a copy of the matrix  
 /// </summary>  
 public Matrix Clone()  
 {  
 return new Matrix((float[,])Contents.Clone());  
 }  
  
 /// <summary>  
 /// Write the matrix to a binary file  
 /// </summary>  
 /// <param name="writer">Instance of BinaryWriter to use for writing</param>  
 public void WriteToFile(BinaryWriter writer)  
 {  
 for (var i = 0; i < Rows; i++)  
 for (var j = 0; j < Columns; j++)  
 writer.Write(Contents[i, j]);  
 }  
  
 /// <summary>  
 /// Read a matrix from a binary file  
 /// </summary>  
 /// <param name="reader">Instance of BinaryReader to use for reading</param>  
 /// <param name="rows">Number of rows in the matrix</param>  
 /// <param name="columns">Number of columns in the matrix</param>  
 /// <returns></returns>  
 public static Matrix ReadFromFile(BinaryReader reader, int rows, int columns)  
 {  
 var readMat = new Matrix(rows, columns);  
  
 for (var i = 0; i < rows; i++)  
 for (var j = 0; j < columns; j++)  
 readMat.Contents[i, j] = reader.ReadSingle();  
  
 return readMat;  
 }  
  
 #region Operator overloading  
  
 public static bool operator ==(Matrix a, Matrix b)  
 {  
 return a.Contents == b.Contents;  
 }  
  
 public static bool operator !=(Matrix a, Matrix b)  
 {  
 return a.Contents != b.Contents;  
 }  
  
 public static Matrix operator +(Matrix a, Matrix b)  
 {  
 return Add(a, b);  
 }  
  
 public static Matrix operator +(Matrix a, float b)  
 {  
 return Add(a, b);  
 }  
  
 public static Matrix operator -(Matrix a, Matrix b)  
 {  
 return Subtract(a, b);  
 }  
  
 public static Matrix operator -(Matrix a, float b)  
 {  
 return Subtract(a, b);  
 }  
  
 public static Matrix operator \*(Matrix a, Matrix b)  
 {  
 return Multiply(a, b);  
 }  
  
 public static Matrix operator \*(Matrix mat, float scalar)  
 {  
 return Multiply(mat, scalar);  
 }  
  
 public static Matrix operator \*(float scalar, Matrix mat)  
 {  
 return Multiply(mat, scalar);  
 }  
  
 public static Matrix operator /(Matrix mat, float scalar)  
 {  
 return Multiply(mat, 1 / scalar);  
 }  
  
 public static Matrix operator /(float scalar, Matrix mat)  
 {  
 return Multiply(mat, 1 / scalar);  
 }  
  
 #endregion  
}

### Exceptions File

This file contains a collection of custom exceptions. This is so that it is easy to distinguish what type of error occurred in a matrix operation without having to read the full provided message.

using System;  
  
namespace Apollo.MatrixMaths;  
  
internal class MatrixArithmeticException : Exception  
{  
 public MatrixArithmeticException()  
 {  
 }  
  
 public MatrixArithmeticException(string message) : base(message)  
 {  
 }  
  
 public MatrixArithmeticException(string message, Exception inner) : base(message, inner)  
 {  
 }  
}  
  
internal class InvalidShapeException : Exception  
{  
 public InvalidShapeException()  
 {  
 }  
  
 public InvalidShapeException(string message, Matrix a, Matrix b) : base($"{message} " +  
 $"\nMatrix A Shape: ({a.Rows}x{a.Columns})" +  
 $"\nMatrix B Shape: ({b.Rows}x{b.Columns})")  
 {  
 }  
  
 public InvalidShapeException(string message, Exception inner) : base(message, inner)  
 {  
 }  
}

## (Apollo.NeuralNet) The Neural Network

This class library specifically targets objective 1, which deals with the creation and usage of the neural network itself

### AdamParameters Class

This class just contains the constants used by the ADAM optimiser. These parameters are commonly used amongst various machine learning libraries which implement it.

This class contains the vocabulary list used by the neural network and deals with the conversion between characters and one-hot vectors. The use of this class allows for data to be inputted into the network, and for the network’s output to be interpreted. Therefore, allowing for objective 3 to be possible.

namespace Apollo.NeuralNet;  
  
/// <summary>  
/// Static class for storing the recommended ADAM optimiser hyperparameters  
/// </summary>  
internal static class AdamParameters  
{  
 // Recommended parameters.  
 public const float ALPHA = 0.001f;  
 public const float BETA1 = 0.9f;  
 public const float BETA2 = 0.999f;  
 public const float EPSILON = 1e-8f;  
}

### The Weight Class

This class is used for the tuneable parameters in the network. It inherits from the matrix class (since the weight itself is a matrix) but also contains several other matrices via composition. This is because these matrices are used in order to optimise the content of the weight, in order to reduce the loss in the network.

It implements the ADAM optimiser.

using Apollo.MatrixMaths;  
  
namespace Apollo.NeuralNet;  
  
/// <summary>  
/// A matrix representing a weight in the network, can be optimised  
/// </summary>  
public class Weight : Matrix  
{  
 /// <param name="rows">Number of Rows in the weight matrix</param>  
 /// <param name="columns">Number of Columns in the weight matrix</param>  
 /// <param name="r">Random Instance</param>  
 public Weight(int rows, int columns, Random r) : base(rows, columns, r)  
 {  
 // Create gradient matrix of the same size  
 Gradient = new Matrix(rows, columns);  
  
 // Create ADAM matrices with same size  
 MomentVector = new Matrix(rows, columns);  
 InfinityNorm = new Matrix(rows, columns);  
 }  
  
 public Weight(float[,] defaultData) : base(defaultData)  
 {  
 // Create gradient matrix of the same size  
 Gradient = new Matrix(Rows, Columns);  
  
 // Create ADAM matrices with same size  
 MomentVector = new Matrix(Rows, Columns);  
 InfinityNorm = new Matrix(Rows, Columns);  
 }  
  
 public Matrix Gradient { get; set; }  
  
 /// <summary>  
 /// Optimise the weight using the Adaptive Moment Estimation Algorithm (ADAM)  
 /// </summary>  
 /// <param name="t">Backpropagation timestep</param>  
 public void Adam(int t)  
 {  
 // Change ADAM matrices  
 MomentVector = AdamParameters.BETA1 \* MomentVector + (1 - AdamParameters.BETA1) \* Gradient;  
 InfinityNorm = AdamParameters.BETA2 \* InfinityNorm + (1 - AdamParameters.BETA2) \* Power(Gradient, 2);  
  
 // Calculate m\_hat and v\_hat   
 var mHat = MomentVector / (1 - MathF.Pow(AdamParameters.BETA1, t));  
 var vHat = InfinityNorm / (1 - MathF.Pow(AdamParameters.BETA2, t));  
  
 // Update weight   
 var update = HadamardDiv(AdamParameters.ALPHA \* mHat, Sqrt(vHat) + AdamParameters.EPSILON);  
 Subtract(update);  
 }  
  
 public new static Weight ReadFromFile(BinaryReader reader, int rows, int columns)  
 {  
 var readData = Matrix.ReadFromFile(reader, rows, columns).Contents;  
 return new Weight(readData);  
 }  
  
 #region ADAM Matrices  
  
 private Matrix MomentVector { get; set; }  
 private Matrix InfinityNorm { get; set; }  
  
 #endregion  
}

### The Gate Class

This class represents the gates in the LSTM. It performs the standard maths operations of gate (before the activation function since these may vary depending on what gate it is). It contains the weight for the LSTM input and the previous output via composition as well as its own value and a randomly generated bias via the same means.

using Apollo.MatrixMaths;  
  
namespace Apollo.NeuralNet;  
  
/// <summary>  
/// Gate class responsible for optimising associated weight matrices and calculations in the LSTM cell  
/// </summary>  
public class Gate  
{  
 /// <summary>  
 /// Create an LSTM gate, with correctly sized weights and bias.  
 /// </summary>  
 public Gate(int vocabSize, int hiddenSize, int batchSize, Random r)  
 {  
 Value = new Matrix(batchSize, hiddenSize);  
  
 InputWeight = new Weight(vocabSize, hiddenSize, r);  
 PrevOutputWeight = new Weight(hiddenSize, hiddenSize, r);  
  
 Bias = new Matrix(batchSize, hiddenSize, r);  
 }  
  
 /// <summary>  
 /// Create an LSTM gate using the state file  
 /// </summary>  
 /// <param name="vocabSize">Vocab size (already read)</param>  
 /// <param name="hiddenSize">Hidden size (already read)</param>  
 /// <param name="batchSize">Batch size (already read)</param>  
 /// <param name="reader">Stream to the state file to read from</param>  
 public Gate(int vocabSize, int hiddenSize, int batchSize, BinaryReader reader)  
 {  
 Value = new Matrix(batchSize, hiddenSize);  
  
 InputWeight = Weight.ReadFromFile(reader, vocabSize, hiddenSize);  
 PrevOutputWeight = Weight.ReadFromFile(reader, hiddenSize, hiddenSize);  
 Bias = Matrix.ReadFromFile(reader, batchSize, hiddenSize);  
 }  
  
 public Matrix Value { get; set; } // Attribute to store the value of the gate   
 public Weight InputWeight { get; set; }  
 public Weight PrevOutputWeight { get; set; }  
 private Matrix Bias { get; }  
  
 /// <summary>  
 /// Calculate the value of the gate with a given input  
 /// </summary>  
 /// <param name="input">The input into the gate</param>  
 /// <param name="prevOutput">The previous output of the LSTM</param>  
 public void CalcUnactivated(Matrix input, Matrix prevOutput)  
 {  
 // All gate calculations take the form of   
 // Some activation function (W\_x \* x + W\_h \* h)  
 // Where  
 // input = x   
 // prev\_output = h  
  
 Value = Matrix.Multiply(input, InputWeight)  
 + Matrix.Multiply(prevOutput, PrevOutputWeight) + Bias;  
 }  
  
 /// <summary>  
 /// Set the value of the gate to 0  
 /// </summary>  
 public void Clear()  
 {  
 Value \*= 0;  
 }  
  
 /// <summary>  
 /// Update the gate's weights  
 /// </summary>  
 /// <param name="t">Backpropagation timestep</param>  
 public void Update(int t)  
 {  
 InputWeight.Adam(t);  
 PrevOutputWeight.Adam(t);  
 }  
  
 /// <summary>  
 /// Writes the values of the gate's weights to the binary file  
 /// </summary>  
 /// <param name="writer">Instance of BinaryWriter to use for writing</param>  
 public void WriteToFile(BinaryWriter writer)  
 {  
 InputWeight.WriteToFile(writer);  
 PrevOutputWeight.WriteToFile(writer);  
 Bias.WriteToFile(writer);  
 }  
}

### The LSTM Class

This class is responsible for the LSTM cell. This does most of the neural network processing. It contains the gates and cell state via composition, and has getters for the gate values, since these are the only values contained in the LSTM that is required outside of the class. This encapsulates the majority of the processing required for objective 2, however this class is managed and utilised by the RNN class.

using Apollo.MatrixMaths;  
  
namespace Apollo.NeuralNet;  
  
public class Lstm  
{  
 public Lstm(int vocabSize, int hiddenSize, int batchSize, Random r)  
 {  
 Forget = new Gate(vocabSize, hiddenSize, batchSize, r);  
 Input = new Gate(vocabSize, hiddenSize, batchSize, r);  
 CandidateState = new Gate(vocabSize, hiddenSize, batchSize, r);  
 Output = new Gate(vocabSize, hiddenSize, batchSize, r);  
  
 CellState = new Matrix(batchSize, hiddenSize);  
 }  
  
 public Lstm(int vocabSize, int hiddenSize, int batchSize, BinaryReader reader)  
 {  
 Forget = new Gate(vocabSize, hiddenSize, batchSize, reader);  
 Input = new Gate(vocabSize, hiddenSize, batchSize, reader);  
 CandidateState = new Gate(vocabSize, hiddenSize, batchSize, reader);  
 Output = new Gate(vocabSize, hiddenSize, batchSize, reader);  
  
 CellState = new Matrix(batchSize, hiddenSize);  
 }  
  
 // Gates   
 private Gate Forget { get; }  
 private Gate Input { get; }  
 private Gate Output { get; }  
  
 // States   
 private Matrix CellState { get; set; }  
  
 // Candidate state uses Gate class since it carries out the same mathematical operations   
 private Gate CandidateState { get; }  
  
 /// <summary>  
 /// Complete one pass through of the LSTM cell, given an input  
 /// </summary>  
 /// <param name="input">Column one-hot vector representing the input into the LSTM</param>  
 /// <param name="previousOutput">LSTM's previous output</param>  
 public Matrix Forward(Matrix input, Matrix previousOutput)  
 {  
 // Calculate forget gate value   
 Forget.CalcUnactivated(input, previousOutput);  
 Forget.Value.Sigmoid();  
  
 // Calculate input gate value  
 Input.CalcUnactivated(input, previousOutput);  
 Input.Value.Sigmoid();  
  
 // Calculate new info value   
 CandidateState.CalcUnactivated(input, previousOutput);  
 CandidateState.Value.Tanh();  
  
 // Calculate cell state  
 // Forget\_Gate x CellState + Input\_Gate x New\_Info\_Gate (using element-wise multiplication)   
 CellState = Matrix.HadamardProd(Forget.Value, CellState) +  
 Matrix.HadamardProd(Input.Value, CandidateState.Value);  
  
 // Calculate output gate  
 Output.CalcUnactivated(input, previousOutput);  
 Output.Value.Sigmoid();  
  
 // return output   
 return Matrix.HadamardProd(Output.Value, Matrix.Tanh(CellState));  
 }  
  
 /// <summary>  
 /// Perform backprop for a single timestep  
 /// </summary>  
 public void Backprop(Matrix input, Matrix dF, Matrix forgetValue, Matrix dI, Matrix inputGateValue, Matrix dO,  
 Matrix outputGateValue,  
 Matrix dG, Matrix candidateStateValue, Matrix lstmOutput)  
 {  
 Forget.InputWeight.Gradient += Matrix.Transpose(input) \* Matrix.HadamardProd(dF, Matrix.DSigmoid(forgetValue));  
 Forget.PrevOutputWeight.Gradient +=  
 Matrix.Transpose(dF) \* Matrix.HadamardProd(Matrix.DSigmoid(forgetValue), lstmOutput);  
  
 Input.InputWeight.Gradient +=  
 Matrix.Transpose(input) \* Matrix.HadamardProd(dI, Matrix.DSigmoid(inputGateValue));  
 Input.PrevOutputWeight.Gradient +=  
 Matrix.Transpose(dI) \* Matrix.HadamardProd(Matrix.DSigmoid(inputGateValue), lstmOutput);  
  
 Output.InputWeight.Gradient +=  
 Matrix.Transpose(input) \* Matrix.HadamardProd(dO, Matrix.DSigmoid(outputGateValue));  
 Output.PrevOutputWeight.Gradient +=  
 Matrix.Transpose(dO) \* Matrix.HadamardProd(Matrix.DSigmoid(outputGateValue), lstmOutput);  
  
 CandidateState.InputWeight.Gradient +=  
 Matrix.Transpose(input) \* Matrix.HadamardProd(dG, Matrix.DTanh(candidateStateValue));  
 CandidateState.PrevOutputWeight.Gradient +=  
 Matrix.Transpose(dG) \* Matrix.HadamardProd(Matrix.DTanh(candidateStateValue), lstmOutput);  
 }  
  
 /// <summary>  
 /// Update the gates of the LSTM  
 /// </summary>  
 /// <param name="t">Backpropagation timestep</param>  
 public void Update(int t)  
 {  
 Forget.Update(t);  
 Input.Update(t);  
 Output.Update(t);  
 CandidateState.Update(t);  
 }  
  
 /// <summary>  
 /// Returns the values of the forget, input and output gates  
 /// </summary>  
 /// <returns>An array containing the values. Index 0 is forget, 1 is input and 2 is output</returns>  
 public Matrix[] GetGateValues()  
 {  
 return new[] { Forget.Value.Clone(), Input.Value.Clone(), Output.Value.Clone() };  
 }  
  
 /// <summary>  
 /// Returns the values of the cell state, and candidate state  
 /// </summary>  
 /// <returns>An array containing the values. Index 0 is cell state, 1 is candidate state</returns>  
 public Matrix[] GetStateValues()  
 {  
 return new[] { CellState.Clone(), CandidateState.Value.Clone() };  
 }  
  
 /// <summary>  
 /// Clears the value of all states and gates in the LSTM  
 /// </summary>  
 public void Clear()  
 {  
 Forget.Clear();  
 Input.Clear();  
 Output.Clear();  
 CandidateState.Clear();  
 CellState \*= 0;  
 }  
  
 /// <summary>  
 /// Write the parameters of the LSTM cell to a binary file  
 /// </summary>  
 /// <param name="writer">Instance of BinaryWriter to use for writing</param>  
 public void WriteToFile(BinaryWriter writer)  
 {  
 Forget.WriteToFile(writer);  
 Input.WriteToFile(writer);  
 CandidateState.WriteToFile(writer);  
 Output.WriteToFile(writer);  
 }  
}

### The RNN Class

This class targets objective 1 in its entirety by combining the other classes in the same project. The RNN class uses the Forward function to generate new content. However, before this can be used it must be trained using the Train function, which makes use of the backpropagation algorithm and the update procedure.

Furthermore, the RNN deals with the saving to and loading from state files itself, handling objective 2.1. Also, when the network is trained, it produces a “before state file” before the training process as well as an “after state file” once the training process has been completed thus handling objective 2.7.

using Apollo.IO;  
using Apollo.MatrixMaths;  
  
namespace Apollo.NeuralNet;  
  
public class Rnn  
{  
 private const string TEMP\_STATE\_PATH = "temp\_state.state";  
  
 /// <param name="profile">Profile object containing state data for the network</param>  
 /// <param name="vocabSize">The amount of words in the vocabulary list</param>  
 /// <param name="hiddenSize">Size which depicts shape of hidden layer weights</param>  
 /// <param name="batchSize">The amount of words (from the vocab) passed in a single input</param>  
 /// <param name="r">Random Instance to instantiate weights</param>  
 public Rnn(int vocabSize, int hiddenSize,  
 int batchSize, Random r)  
 {  
 VocabSize = vocabSize;  
 HiddenSize = hiddenSize;  
 BatchSize = batchSize;  
  
 LstmCell = new Lstm(VocabSize, HiddenSize, BatchSize, r);  
  
 SoftmaxWeight = new Weight(HiddenSize, VocabSize, r);  
 }  
  
 /// <summary>  
 /// Create an RNN object from a state file  
 /// </summary>  
 /// <param name="stateFileToLoad">Path to the state file to load from</param>  
 public Rnn(string stateFileToLoad)  
 {  
 LoadState(stateFileToLoad);  
 }  
  
 // General Parameters  
 private int VocabSize { get; set; }  
 public int BatchSize { get; private set; } // Accessed when generating seeds for forward prop  
 private int HiddenSize { get; set; }  
 private float LearningRate { get; set; }  
  
 // LSTM Cell   
 private Lstm LstmCell { get; set; }  
  
 // Softmax layer weight   
 private Weight SoftmaxWeight { get; set; }  
  
 /// <summary>  
 /// Save network into a binary file  
 /// </summary>  
 public void LoadState(string name)  
 {  
 using (var stream = File.Open(name, FileMode.Open))  
 {  
 using (var reader = new BinaryReader(stream))  
 {  
 VocabSize = reader.ReadInt32();  
 BatchSize = reader.ReadInt32();  
 HiddenSize = reader.ReadInt32();  
 LearningRate = reader.ReadSingle();  
  
 LstmCell = new Lstm(VocabSize, HiddenSize, BatchSize, reader);  
  
 SoftmaxWeight = Weight.ReadFromFile(reader, HiddenSize, VocabSize);  
 }  
 }  
 }  
  
 /// <summary>  
 /// Save the network to a binary file  
 /// </summary>  
 public void SaveState(string name)  
 {  
 using (var stream = File.Open(name, FileMode.Create))  
 {  
 using (var writer = new BinaryWriter(stream))  
 {  
 writer.Write(VocabSize);  
 writer.Write(BatchSize);  
 writer.Write(HiddenSize);  
 writer.Write(LearningRate);  
 LstmCell.WriteToFile(writer);  
 SoftmaxWeight.WriteToFile(writer);  
 }  
 }  
 }  
  
 /// <summary>  
 /// Complete a full pass of the neural network, with the correct number of recurrences.  
 /// </summary>  
 /// <returns> An array containing the output from each recurrence of the network </returns>  
 public Matrix[] Forward(Matrix initialInput, int recurrenceAmount)  
 {  
 // Array to store outputs at each timestep  
 var outputs = new Matrix[recurrenceAmount];  
  
 // HiddenState (lstm output) = matrix of 0s when t=0   
 var lstmInput = initialInput;  
 var hiddenState = new Matrix(BatchSize, HiddenSize);  
  
 for (var i = 0; i < recurrenceAmount; i++)  
 {  
 hiddenState = LstmCell.Forward(lstmInput, hiddenState);  
 outputs[i] = hiddenState.Clone();  
  
 // Softmax layer  
 // Apply softmax to LSTM output and interpret  
 outputs[i] \*= SoftmaxWeight;  
 outputs[i].Softmax();  
 outputs[i] = InterpretOutput(outputs[i]);  
  
 // Output = next input  
 lstmInput = outputs[i];  
 }  
  
 return outputs;  
 }  
  
 /// <summary>  
 /// Interprets the softmaxed LSTM output as a one-hot vector  
 /// </summary>  
 /// <param name="softmax">The result after using the softmax function on the LSTM output</param>  
 /// <returns>A one-hot vector representing the LSTM's predictions</returns>  
 private Matrix InterpretOutput(Matrix softmax)  
 {  
 var r = new Random();  
 // Create matrix of 0s the same shape as softmax matrix  
 var interpreted = Matrix.Like(softmax);  
  
 // Find the index of the highest number on each row, and set the corresponding index in the interpreted  
 // matrix to 1  
 for (var i = 0; i < softmax.Rows; i++)  
 {  
 var highestProb = float.MinValue;  
 var highestProbIndex = -1;  
  
 var highestSucceed = float.MinValue;  
 var highestSucceedIndex = -1;  
  
 for (var j = 0; j < softmax.Columns; j++)  
 {  
 if (softmax[i, j] > highestProb)  
 highestProbIndex = j;  
 var rollQuota = (int)(softmax[i, j] \* 100);  
 var diceRoll = r.Next(100);  
  
 if (diceRoll <= rollQuota && softmax[i, j] > highestSucceed)  
 highestSucceedIndex = j;  
 }  
  
 // The 1 goes into the slot with the highest probability which succeeded the dice roll   
 // Or alternatively (if every probability failed), the highest probability in general  
 if (highestSucceedIndex == -1)  
 interpreted[i, highestProbIndex] = 1;  
 else  
 interpreted[i, highestSucceedIndex] = 1;  
 }  
  
 return interpreted;  
 }  
  
 /// <summary>  
 /// Perform the backpropagation algorithm on the neural network to optimise its parameters.  
 /// </summary>  
 /// <param name="forgetGates">The values of the forget gate at each timestep during training</param>  
 /// <param name="candidateStates">The values of the candidate state at each timestep during training</param>  
 /// <param name="cellStates">The values of the cell state at each timestep during training</param>  
 /// <param name="inputGates">The values of the input gate at each timestep during training</param>  
 /// <param name="outputGates">The values of the output gate at each timestep during training</param>  
 /// <param name="inputs">The inputs to the LSTM cell at each timestep during training</param>  
 /// <param name="lstmOutputs">The outputs straight from the LSTM at each timestep during training</param>  
 /// <param name="predictedOutputs">  
 /// The output from the overall network (after softmax layer) at each timestep  
 /// during training  
 /// </param>  
 /// <param name="expectedOutputs">The training data</param>  
 private void Backprop(List<Matrix> forgetGates, List<Matrix> candidateStates, List<Matrix> cellStates,  
 List<Matrix> inputGates, List<Matrix> outputGates, List<Matrix> inputs, List<Matrix> lstmOutputs,  
 List<Matrix> predictedOutputs, List<Matrix> expectedOutputs)  
 {  
 // t represents timestep  
 // Go until t > 1 since there is no timestep -1 (t-1 when t = 0)   
 for (var t = inputs.Count - 1; t > 1; t--)  
 {  
 // dL/dh(t) = (y\_hat - y)V^T  
 var dH = (predictedOutputs[t] - expectedOutputs[t]) \* Matrix.Transpose(SoftmaxWeight);  
  
 // dL/dc(t) = dL/dh(t) x o(t)tanh'(c\_t)   
 var dC = Matrix.HadamardProd(dH, Matrix.HadamardProd(outputGates[t], Matrix.DTanh(cellStates[t])));  
  
 // dL/dg(t) = dL/dc(t) x i(t)   
 var dG = Matrix.HadamardProd(dC, inputGates[t]);  
  
 // dL/do(t) = dL/dh x tanh(c(t))   
 var dO = Matrix.HadamardProd(dH, Matrix.Tanh(cellStates[t]));  
  
 // dL/di(t) = dL/dc(t) x g(t)  
 var dI = Matrix.HadamardProd(dC, candidateStates[t]);  
  
 // dL/df(t) = dL/dc(t) x c(t-1)   
 var dF = Matrix.HadamardProd(dC, cellStates[t - 1]);  
  
 // Increment gradient for weights   
 SoftmaxWeight.Gradient += Matrix.Transpose(lstmOutputs[t]) \* (predictedOutputs[t] - expectedOutputs[t]);  
  
 // Backprop through LSTM cell  
 LstmCell.Backprop(inputs[t], dF, forgetGates[t], dI, inputGates[t], dO, outputGates[t],  
 dG, candidateStates[t], lstmOutputs[t - 1]);  
  
 // Update network parameters  
 Update(t);  
 }  
 }  
  
 /// <summary>  
 /// Update the parameters of the neural network  
 /// </summary>  
 private void Update(int t)  
 {  
 SoftmaxWeight.Adam(t);  
 LstmCell.Update(t);  
 }  
  
 /// <summary>  
 /// Calculate the loss using categorical cross-entropy  
 /// </summary>  
 /// <param name="expected">The expected/desired output from the LSTM</param>  
 /// <param name="actual">The actual output of the LSTM</param>  
 /// <returns>A matrix representing the loss of the neural network</returns>  
 private float CalculateLoss(Matrix expected, Matrix actual)  
 {  
 var lossMatrix = -1 \* Matrix.HadamardProd(expected, Matrix.Log(actual, MathF.E));  
 return lossMatrix.Sum();  
 }  
  
 /// <summary>  
 /// Train the neural network on a single file  
 /// </summary>  
 /// <param name="trainingData">The one-hot vector representation of the file</param>  
 /// <param name="minimumEpochs">The minimum number of epochs to perform</param>  
 /// <param name="maximumEpochs">The maximum number of epochs to perform before stopping</param>  
 /// <param name="maximumError">The maximum error, training will stop early if the average error is below this</param>  
 /// <param name="batchesPerEpoch">The amount of batches per epoch, do not put to high or risk NaNs</param>  
 /// <param name="afterStatePath">The path at which to write the after-training state file</param>  
 /// <param name="r">An instance of random, to randomly select batches</param>  
 public void Train(Matrix[] trainingData, int minimumEpochs, int maximumEpochs, float maximumError,  
 int batchesPerEpoch, Random r)  
 {  
 var (inputData, expectedOutputs) = CreateBatches(trainingData);  
 LogManager.WriteLine($"Training input data length: {inputData.Count}");  
  
 // Previous gate/state values to be used in backpropagation  
 var usedInputs = new List<Matrix>();  
 var usedOutputs = new List<Matrix>();  
 var forgetGateValues = new List<Matrix>();  
 var candidateStateValues = new List<Matrix>();  
 var cellStateValues = new List<Matrix>();  
 var inputGateValues = new List<Matrix>();  
 var outputGateValues = new List<Matrix>();  
 var hiddenStateValues = new List<Matrix>();  
 var actualOutputValues = new List<Matrix>();  
  
 float totalLoss;  
 for (var epoch = 0; epoch < maximumEpochs; epoch++)  
 {  
 SaveState(TEMP\_STATE\_PATH);  
 if (batchesPerEpoch <= 0) // Stop if training with 0 batches per epoch  
 break;  
  
 // Clear stored values on each epoch   
 usedInputs.Clear();  
 usedOutputs.Clear();  
 forgetGateValues.Clear();  
 candidateStateValues.Clear();  
 cellStateValues.Clear();  
 inputGateValues.Clear();  
 outputGateValues.Clear();  
 hiddenStateValues.Clear();  
 actualOutputValues.Clear();  
  
 LstmCell.Clear();  
  
 // Create new matrix to store the hidden state (LSTM output)   
 // Instantiated as 0s since there is no outputted hidden state at t=0  
 var hiddenState = new Matrix(BatchSize, HiddenSize);  
 totalLoss = 0f;  
  
 // Simulate a forward pass of the network using a randomly selected training batch  
 var start = r.Next(inputData.Count - batchesPerEpoch);  
 var end = start + batchesPerEpoch;  
 for (var i = start; i < end; i++)  
 {  
 var input = inputData[i];  
 var expected = expectedOutputs[i];  
  
 usedInputs.Add(input);  
 usedOutputs.Add(expected);  
  
 hiddenState = LstmCell.Forward(input, hiddenState);  
  
 var actualOutput = hiddenState.Clone();  
 actualOutput \*= SoftmaxWeight;  
 actualOutput.Softmax();  
 actualOutputValues.Add(actualOutput);  
  
 hiddenStateValues.Add(hiddenState);  
  
 var gateValues = LstmCell.GetGateValues();  
 forgetGateValues.Add(gateValues[0]);  
 inputGateValues.Add(gateValues[1]);  
 outputGateValues.Add(gateValues[2]);  
  
 var stateValues = LstmCell.GetStateValues();  
 cellStateValues.Add(stateValues[0]);  
 candidateStateValues.Add(stateValues[1]);  
  
 totalLoss += CalculateLoss(expected, actualOutput);  
 }  
  
 // Work out and log the average loss at this point   
 var averageLoss = totalLoss / batchesPerEpoch;  
 LogManager.WriteLine($"Epoch: {epoch}\nLoss: {averageLoss}");  
  
 if (float.IsNaN(averageLoss))  
 {  
 batchesPerEpoch -= 5;  
 LoadState(TEMP\_STATE\_PATH);  
 continue;  
 }  
  
 // Break if training can stop  
 // i.e. the loss is lower than the maximum error, and we've done the minimum amount of epochs  
 if (averageLoss < maximumError && epoch >= minimumEpochs)  
 break;  
  
 // Backprop to reduce error  
 Backprop(forgetGateValues, candidateStateValues, cellStateValues, inputGateValues, outputGateValues,  
 usedInputs, hiddenStateValues, actualOutputValues, usedOutputs);  
  
 File.Delete(TEMP\_STATE\_PATH); // Delete temp state file  
 }  
 }  
  
 /// <summary>  
 /// Create valid batches from the training data  
 /// </summary>  
 /// <param name="trainingData">The training data to make the batches from</param>  
 /// <returns>A tuple in the form of (inputBatches, expectedBatches)</returns>  
 private Tuple<List<Matrix>, List<Matrix>> CreateBatches(Matrix[] trainingData)  
 {  
 var inputData = new List<Matrix>();  
 var expectedOutputs = new List<Matrix>();  
  
 for (var i = 0; i < trainingData.Length - BatchSize - 1; i++)  
 {  
 var input = trainingData.Skip(i).Take(BatchSize).ToArray();  
 var expected = trainingData.Skip(i + 1).Take(BatchSize).ToArray();  
  
 inputData.Add(Matrix.StackArray(input));  
 expectedOutputs.Add(Matrix.StackArray(expected));  
 }  
  
 return new Tuple<List<Matrix>, List<Matrix>>(inputData, expectedOutputs);  
 }  
}

### Vocab Class

This class acts as a lookup table for the characters which the RNN is allowed to generate. It contains a private list of characters which does not contain duplicates and allows methods outside of the class to convert between a character and its ID using an indexer. It also handles the conversion between characters and one hot vectors.

using Apollo.MatrixMaths;  
  
namespace Apollo.NeuralNet;  
  
public class Vocab  
{  
 public Vocab()  
 {  
 VocabList = new List<char>();  
 }  
  
 /// <summary>  
 /// Create a vocab list from a pre-existing list  
 /// </summary>  
 public Vocab(List<char> vocabList)  
 {  
 VocabList = new List<char>();  
  
 // Add each character one by one to avoid duplicates   
 foreach (var c in vocabList) AddCharacter(c); // So that duplicates are not added   
 }  
  
 /// <summary>  
 /// Create vocab list from a string of data  
 /// </summary>  
 public Vocab(string data)  
 {  
 VocabList = new List<char>();  
  
 // Add each character one by one to avoid duplicates   
 foreach (var c in data) AddCharacter(c); // So that duplicates are not added   
 }  
  
 private List<char> VocabList { get; }  
  
 /// <summary>  
 /// Retrieve a word from the vocab list using its ID  
 /// </summary>  
 /// <param name="id">The ID of the character to retrieve</param>  
 public char this[int id] => VocabList[id];  
  
 /// <summary>  
 /// Retrieve the ID of a character  
 /// </summary>  
 /// <param name="c">The character to retrieve the ID of</param>  
 public int this[char c] => VocabList.IndexOf(c);  
  
 public int Size => VocabList.Count;  
  
 public void AddCharacters(char[] characters)  
 {  
 foreach (var c in characters)  
 AddCharacter(c);  
 }  
  
 /// <summary>  
 /// Add a character to the vocabulary list  
 /// </summary>  
 /// <param name="c">The character to add to the vocabulary list</param>  
 public void AddCharacter(char c)  
 {  
 // Only add if it is not already in the vocabulary list  
 if (this[c] == -1)  
 VocabList.Add(c);  
 }  
  
 /// <summary>  
 /// Create a one-hot vector for a character in the vocab list  
 /// </summary>  
 /// <param name="c">The character to represent</param>  
 /// <param name="row">Flag variable to decide if the vector is a row vector, true by default</param>  
 /// <returns>A column or row one-hot vector</returns>  
 public Matrix CreateOneHot(char c, bool row = true)  
 {  
 var vector = new Matrix(1, Size);  
 var charIndex = this[c];  
  
 if (charIndex == -1)  
 throw new WordNotFoundException("Word does not exist in the vocab list");  
  
 vector[0, charIndex] = 1;  
  
 if (!row)  
 vector.Transpose();  
  
 return vector;  
 }  
  
 /// <summary>  
 /// Interpret a one-hot vector  
 /// </summary>  
 /// <param name="vector">The one-hot vector</param>  
 /// <returns>The character represented by the one-hot vector</returns>  
 public char InterpretOneHot(Matrix vector)  
 {  
 // A row or column vector should be passed, meaning the vector should have:  
 // 1 row/column  
 // vocabSize columns/rows  
 if ((vector.Rows > 1 && vector.Columns > 1) ||  
 (vector.Rows == 1 && vector.Columns != Size) ||  
 (vector.Columns == 1 && vector.Rows != Size))  
 throw new InvalidOneHotException("A one-hot vector should have shape 1 x vocabSize or vocabSize x 1");  
  
 if (vector.Columns == 1 && vector.Rows > 1) // Ensure it is a column vector   
 vector.Transpose();  
  
 // Find the index of the 1  
 var index = -1;  
 var numOnes = 0;  
 for (var i = 0; i < vector.Columns; i++)  
 if (vector[0, i] == 1f)  
 {  
 index = i;  
 numOnes++;  
 }  
  
 if (numOnes != 1)  
 throw new InvalidOneHotException("A one-hot vector should only contain one 1");  
  
 return this[index];  
 }  
  
 /// <summary>  
 /// Prepare training data  
 /// </summary>  
 /// <param name="midiString">String representation to convert into a series of one-hot vectors</param>  
 /// <returns>An array of one hot vectors</returns>  
 public Matrix[] PrepareTrainingData(string midiString)  
 {  
 var trainingData = new Matrix[midiString.Length];  
 for (var i = 0; i < midiString.Length; i++) trainingData[i] = CreateOneHot(midiString[i]);  
  
 return trainingData;  
 }  
  
 /// <summary>  
 /// Get all characters in the list as a string  
 /// </summary>  
 public string AsString()  
 {  
 return string.Join("", VocabList);  
 }  
}

### Profile Class

This class acts a structure to read from and write to the profile JSON files on the user’s file system.

namespace Apollo;  
  
/// <summary>  
/// Model for containing data about a network state profile (read from and written to a JSON file)  
/// </summary>  
public class Profile  
{  
 public string BeforeStateFile { get; set; }  
 public string AfterStateFile { get; set; }  
 public string TrainingDataDirectory { get; set; }  
 public string Vocab { get; set; }  
   
 /// <summary>  
 /// Returns the default profile   
 /// </summary>  
 /// <param name="pathToFiles">Path to the files associated with the new profile</param>  
 /// <returns>A profile containing default data</returns>  
 public static Profile Default(string pathToFiles)  
 {  
 var profile = new Profile();  
 profile.BeforeStateFile = Path.Join(pathToFiles, "apollo\_before.state");  
 profile.AfterStateFile = Path.Join(pathToFiles, "apollo\_after.state");  
 profile.TrainingDataDirectory = Path.Join(pathToFiles, "training");  
 profile.Vocab = "";  
  
 return profile;  
 }  
   
 /// <summary>  
 /// Evaluates whether two profiles are the same  
 /// </summary>  
 /// <param name="a">The first profile</param>  
 /// <param name="b">The second profile</param>  
 /// <returns>A boolean depicting whether the two contain the exact same data or not</returns>  
 public static bool operator ==(Profile a, Profile b)  
 {  
 return a.BeforeStateFile == b.BeforeStateFile  
 && a.AfterStateFile == b.AfterStateFile  
 && a.TrainingDataDirectory == b.TrainingDataDirectory  
 && a.Vocab == b.Vocab;  
 }  
   
 /// <summary>  
 /// Evaluates whether two profiles are different  
 /// </summary>  
 /// <param name="a">The first profile</param>  
 /// <param name="b">The second profile</param>  
 /// <returns>A boolean depicting whether the two contain different data or not</returns>  
 public static bool operator !=(Profile a, Profile b)  
 {  
 return !(a == b);  
 }  
}

### The NeuralNetwork Class

This class contains all the necessary objects and methods to use to RNN with ease. This makes the neural network easy to implement on the front end, so that the different pages and windows do not have to keep track of more than one object in order to use the RNN.

using System.Text;  
using Apollo.IO;  
using Apollo.MatrixMaths;  
  
namespace Apollo.NeuralNet;  
  
/// <summary>  
/// Wrapper class containing all necessary objects and methods for high level control of RNN  
/// </summary>  
public class NeuralNetwork  
{  
 public NeuralNetwork(Profile initialProfile, int hiddenSize, int batchSize, Random r)  
 {  
 HiddenSize = hiddenSize;  
 BatchSize = batchSize;  
 R = r;  
 TrainingData = null;  
 ChangeProfile(initialProfile);  
 PopulateVocabList();  
 }  
  
 private Rnn Network { get; set; } // RNN itself  
 public Vocab VocabList { get; set; } // The vocab used by the RNN   
 private Matrix[][]? TrainingData { get; set; } // TrainingData used during training   
 private int HiddenSize { get; } // The hidden size of the network  
 private int BatchSize { get; } // The batch size of the network  
 private Random R { get; } // Random instance used for ALL RNG in the network  
 public Profile CurrentProfile { get; private set; } // The profile currently being used by the network  
  
 /// <summary>  
 /// Populate the training data jagged array with the profile's training data  
 /// </summary>  
 private void PopulateTrainingArray()  
 {  
 var trainingDataFiles = Directory.GetFiles(CurrentProfile.TrainingDataDirectory)  
 .Where(filePath => filePath.EndsWith(".td")).ToArray();  
  
 if (trainingDataFiles.Length == 0) // If the folder doesn't have any training data files   
 {  
 // Convert the midi files into training data  
 var midiStrings = MidiManager.ReadDir(CurrentProfile.TrainingDataDirectory);  
 TrainingData = new Matrix[midiStrings.Count][];  
  
 Parallel.For(0, midiStrings.Count,  
 i => { TrainingData[i] = VocabList.PrepareTrainingData(midiStrings[i]); });  
  
 // Save the just-converted training data into training data files  
 for (var i = 0; i < midiStrings.Count; i++)  
 {  
 var fileName = Path.Join(CurrentProfile.TrainingDataDirectory, $"file{i}.td");  
 TrainingFileManager.Write(TrainingData[i], fileName);  
 }  
  
 // Delete the MIDI files  
 MidiManager.PurgeDir(CurrentProfile.TrainingDataDirectory);  
 }  
 else // If training data files exist, use them instead   
 {  
 TrainingData = TrainingFileManager.ReadDir(CurrentProfile.TrainingDataDirectory);  
 }  
 }  
  
 /// <summary>  
 /// Populate the Vocab object with the different characters used in the training data  
 /// </summary>  
 private void PopulateVocabList()  
 {  
 // If the profile does not already have a vocab list saved, then it cannot be used  
 // This is because the original MIDI files will have been replaced with the training data files before this point  
 if (CurrentProfile.Vocab.Length == 0)   
 {  
 var directory = Path.GetDirectoryName(CurrentProfile.AfterStateFile);  
 Directory.Delete(directory, true);  
 throw new Exception("The profile you are using is corrupted, please reopen the program and select a" +  
 "different profile");  
 }  
  
 VocabList = new Vocab(CurrentProfile.Vocab); // Create a vocab object from the stored list  
 }  
  
 /// <summary>  
 /// Train the network  
 /// </summary>  
 public void Train(int minEpochs, int maxEpochs, float maxError, int batchesPerEpoch)  
 {  
 // Populate the array if it is empty  
 if (TrainingData == null)  
 PopulateTrainingArray();  
  
 // Save before state  
 Network.SaveState(CurrentProfile.BeforeStateFile);  
  
 foreach (var vectorisedFile in TrainingData) // Perform a training loop for each file in the training data  
 Network.Train(vectorisedFile, minEpochs, maxEpochs, maxError, batchesPerEpoch, R);  
  
 // Save after state  
 Network.SaveState(CurrentProfile.AfterStateFile);  
 }  
  
 /// <summary>  
 /// Create a generation seed  
 /// </summary>  
 /// <returns>A matrix representing the initial input into the RNN when generating</returns>  
 private Matrix CreateGenerationSeed()  
 {  
 var rows = new Matrix[Network.BatchSize]; // Seed must be of size VocabSize x BatchSize  
  
 for (var i = 0; i < rows.Length; i++) // For each row, select a character at random to be the seed  
 {  
 var charId = R.Next(VocabList.Size);  
 rows[i] = VocabList.CreateOneHot(VocabList[charId]);  
 }  
  
 return Matrix.StackArray(rows); // Stack the array into one matrix so it can be inputted  
 }  
  
 /// <summary>  
 /// Interpret a matrix which was outputted from the RNN during generation  
 /// </summary>  
 /// <param name="outputs">What the RNN outputted during generation</param>  
 /// <returns>A string representation of the RNN output which can be passed to MidiManager</returns>  
 private string InterpretVectorOutput(Matrix[] outputs)  
 {  
 var stringOutput = new StringBuilder();  
  
 foreach (var output in outputs)  
 {  
 // The next character to be written is always the last row.   
 var rowContents = new float[1, output.Columns];  
 for (var j = 0; j < output.Columns; j++) rowContents[0, j] = output[output.Rows - 1, j];  
 var mat = new Matrix(rowContents);  
  
 // Interpret the last row of the output matrix, and add it to the string builder  
 stringOutput.Append(VocabList.InterpretOneHot(mat));  
 }  
  
 return stringOutput.ToString();  
 }  
  
 /// <summary>  
 /// Use the network to generate, outputting the created data to a MIDI file  
 /// </summary>  
 /// <param name="genLength">The amount of iterations of the RNN to do</param>  
 /// <param name="bpm">The beats per minute of the MIDI file</param>  
 /// <param name="savePath">The path to save the MIDI file to</param>  
 public void Generate(int genLength, int bpm, string savePath)  
 {  
 // Create generation seed   
 var seed = CreateGenerationSeed();  
  
 // Pass seed to RNN and interpret the output as a string   
 var networkOutputs = Network.Forward(seed, genLength);  
 var stringOutput = InterpretVectorOutput(networkOutputs);  
  
 // Log generation details and string representation   
 var logBuffer = $"Generating with:\nGeneration Length: {genLength}\nBPM: {bpm}\nSave Path: {savePath}" +  
 $"\nGenerated Text:\n{stringOutput}";  
 LogManager.WriteLine(logBuffer);  
  
 // Write and save the MIDI file using string representation  
 MidiManager.WriteFile(stringOutput, savePath, bpm);  
 }  
  
 /// <summary>  
 /// Change the state profile that the network is using  
 /// </summary>  
 /// <param name="profile">The profile to change to</param>  
 public void ChangeProfile(Profile profile)  
 {  
 // Change profile object here  
 CurrentProfile = profile;  
 // Wipe vocab list and populate it with relevant characters  
 VocabList = new Vocab();  
 PopulateVocabList();  
 // Wipe TrainingData since new profile may use different data  
 TrainingData = null;  
  
 Console.WriteLine(VocabList.Size);  
  
 // Create RNN with the data in the after state file (if it exists)   
 // Otherwise, create it from scratch  
 Network = File.Exists(CurrentProfile.AfterStateFile)  
 ? new Rnn(CurrentProfile.AfterStateFile)  
 : new Rnn(VocabList.Size, HiddenSize, BatchSize, R);  
 }  
  
 /// <summary>  
 /// Try to revert the training done to the network  
 /// </summary>  
 public bool TryRevertTraining()  
 {  
 if (!File.Exists(CurrentProfile.BeforeStateFile)) // If there is no before state, training cannot be reverted  
 return false;  
  
 // Load the before state file and overwrite the after state file  
 Network.LoadState(CurrentProfile.BeforeStateFile);  
 Network.SaveState(CurrentProfile.AfterStateFile);  
  
 return true;  
 }  
}

### Exceptions

This file contains a collection of custom exceptions. This is so that it is easy to distinguish what type of error occurred while the neural network was in use without having to read the full provided message.

namespace Apollo.NeuralNet;  
  
internal class LstmInputException : Exception  
{  
 public LstmInputException()  
 {  
 }  
  
 public LstmInputException(string message) : base(message)  
 {  
 }  
  
 public LstmInputException(string message, Exception inner) : base(message, inner)  
 {  
 }  
}  
  
internal class WordNotFoundException : Exception  
{  
 public WordNotFoundException()  
 {  
 }  
  
 public WordNotFoundException(string message) : base(message)  
 {  
 }  
  
 public WordNotFoundException(string message, Exception inner) : base(message, inner)  
 {  
 }  
}  
  
internal class InvalidOneHotException : Exception  
{  
 public InvalidOneHotException()  
 {  
 }  
  
 public InvalidOneHotException(string message) : base(message)  
 {  
 }  
  
 public InvalidOneHotException(string message, Exception inner) : base(message, inner)  
 {  
 }  
}

## (Apollo) The Graphical User Interface

This class project uses the WPF GUI framework in order to provide the user with an interface with the neural network. When the program is run, it opens on the start up window. However, once the user selects an option, it opens the main window, which is separated into different pages: creation, listen, settings and training.

Each window, page or application is split into two files: a xaml file and a xaml.cs file

### StoredSettings class

This class stores the settings used across the application. It handles the reading and writing of the settings to and from the “settings.json” file.

using System;  
using System.IO;  
using System.Text.Json;  
  
namespace Apollo;  
  
/// <summary>  
/// Class passed to Json Deserializer to store settings  
/// </summary>  
public class StoredSettings  
{  
 public const string SETTINGS\_PATH = "settings.json";  
  
 /// <summary>  
 /// Blank Constructor for JsonSerializer  
 /// </summary>  
 public StoredSettings()  
 {  
 }  
  
 /// <summary>  
 /// Constructor used in StoredSettings.Default()  
 /// </summary>  
 public StoredSettings(string selectedProfile, string profilesPath, string logsPath, int minEpochs, int maxEpochs,  
 float maxError, int batchesPerEpoch, int generationLength, int bpm)  
 {  
 SelectedProfileName = selectedProfile;  
 ProfilesPath = profilesPath;  
 LogsPath = logsPath;  
 MinEpochs = minEpochs;  
 MaxEpochs = maxEpochs;  
 MaxError = maxError;  
 BatchesPerEpoch = batchesPerEpoch;  
 GenerationLength = generationLength;  
 Bpm = bpm;  
 }  
  
 public string SelectedProfileName { get; set; } // Currently selected state profile  
 public string ProfilesPath { get; set; } // Path where all the state profiles are stored  
 public string LogsPath { get; set; } // Path where the logs are saved  
 public int MinEpochs { get; set; } // Default Minimum Epochs [TRAINING SETTINGS]   
 public int MaxEpochs { get; set; } // Default Maximum Epochs [TRAINING SETTINGS]  
 public float MaxError { get; set; } // Default Maximum Error [TRAINING SETTINGS]  
 public int BatchesPerEpoch { get; set; } // Default Batches per Epoch [TRAINING SETTINGS]  
 public int GenerationLength { get; set; } // Default Generation Length [GENERATION SETTINGS]  
 public int Bpm { get; set; } // Default Beats per Minute [GENERATION SETTINGS]  
  
 /// <summary>  
 /// Get default settings  
 /// </summary>  
 public static StoredSettings Default()  
 {  
 return new StoredSettings("default", "states", "logs", 100,  
 200, 0.05f, 50, 10000, 30);  
 }  
  
 /// <summary>  
 /// Generate StoredSettings object from JSON file  
 /// </summary>  
 /// <returns>StoredSettings object containing data in JSON file</returns>  
 public static StoredSettings Load()  
 {  
 var str = File.ReadAllText(SETTINGS\_PATH);  
 Console.WriteLine(str);  
 return JsonSerializer.Deserialize<StoredSettings>(str);  
 }  
  
 public void Save()  
 {  
 var jsonString = JsonSerializer.Serialize(this, new JsonSerializerOptions { WriteIndented = true });  
 File.WriteAllText(SETTINGS\_PATH, jsonString);  
 }  
}

### ProfileManager Class

This class is responsible for maintaining a dictionary containing the existing profiles and their corresponding names/identifiers. It also simplifies the process of making a new profile object when the user wants to create a new one. Thus, this class helps to fulfil objectives 2.2 and 2.3. This is also contains the methods which the Settings Page calls in order to dynamically create profile objects

using System;  
using System.Collections.Generic;  
using System.IO;  
using System.Linq;  
using System.Text.Json;  
using System.Threading.Tasks;  
using System.Windows;  
using System.Windows.Input;  
using Apollo.IO;  
using Apollo.NeuralNet;  
using Microsoft.Win32;  
  
namespace Apollo;  
  
/// <summary>  
/// Class which handles the creation and switching between profiles  
/// </summary>  
public class ProfileManager  
{  
 private readonly Dictionary<string, Profile> \_profiles;  
  
 /// <summary>  
 /// Instantiate a ProfileManager object  
 /// </summary>  
 /// <param name="profilesPath">The path where all the profiles are stored</param>  
 public ProfileManager(string profilesPath)  
 {  
 \_profiles = new Dictionary<string, Profile>();  
 ProfilesPath = profilesPath;  
 LoadFromFileSystem(); // Fill the dictionary with already existing profiles   
  
 if (ProfileNames.Length == 0)  
 CreateDefaultProfile();  
 }  
  
 private string ProfilesPath { get; set; } // The path where all profiles will be found   
  
 public string[] ProfileNames => \_profiles.Keys.ToArray(); // Getter for array of dictionary keys  
  
 private void CreateDefaultProfile()  
 {  
 MessageBox.Show("No training profiles found. Please close this box to create one.");  
 CreateProfile("default", true);  
 }  
  
 /// <summary>  
 /// Check if a profile with a name exists  
 /// </summary>  
 /// <param name="name">The name to check</param>  
 /// <returns>Boolean depicting whether a profile with the passed name exists</returns>  
 public bool ProfileExists(string name)  
 {  
 return \_profiles.ContainsKey(name);  
 }  
  
 /// <summary>  
 /// Returns a profile from the dictionary  
 /// </summary>  
 /// <returns></returns>  
 public Profile Any()  
 {  
 return \_profiles.Values.ToArray()[0];  
 }  
  
 public void ChangeProfilesPath(string newPath)  
 {  
 ProfilesPath = newPath;  
 }  
  
 /// <summary>  
 /// Opens a file dialog for the user to select multiple files  
 /// </summary>  
 /// <returns>A list of the paths to the selected training files</returns>  
 private string[] SelectTrainingFiles(bool mandatory = false)  
 {  
 var fileDialog = new OpenFileDialog();  
 fileDialog.Multiselect = true;  
 fileDialog.Title = "Select multiple training files";  
 fileDialog.Filter = "MIDI File|\*.mid;\*.midi";  
  
 if (fileDialog.ShowDialog() == true) return fileDialog.FileNames;  
  
 if (mandatory)  
 throw new Exception("You cannot use the program without selecting the training files");  
  
 return Array.Empty<string>();  
 }  
  
 /// <summary>  
 /// Loads existing profiles from the file system and adds them to the dictionary.  
 /// If no profiles are found, then it will create one.  
 /// </summary>  
 private void LoadFromFileSystem()  
 {  
 // If the root profile path doesn't exist, create it   
 if (!Directory.Exists(ProfilesPath))  
 Directory.CreateDirectory(ProfilesPath);  
  
 var profileDirectories = Directory.GetDirectories(ProfilesPath);  
  
 // Stop if no profiles exist  
 if (profileDirectories.Length == 0)  
 return;  
  
 // Check for valid profiles (profiles with a schema.json file)   
 foreach (var dir in profileDirectories)  
 {  
 var schemaPath = Path.Join(dir, "schema.json");  
  
 if (!File.Exists(schemaPath)) // Continue if invalid   
 continue;  
  
 // Read the schema + add to dictionary  
 var profile = ReadJson<Profile>(schemaPath);  
  
 \_profiles.TryAdd(Path.GetFileName(dir), profile);  
 }  
 }  
  
 /// <summary>  
 /// Create a profile  
 /// </summary>  
 public void CreateProfile(string name, bool mandatory = false)  
 {  
 Mouse.SetCursor(Cursors.Wait);  
 // Get user to select training files  
 var trainingFiles = SelectTrainingFiles(mandatory);  
  
 // Do not continue if no files are selected   
 if (trainingFiles.Length == 0)  
 {  
 MessageBox.Show("You must select training files in order to create a profile");  
 return;  
 }  
  
 // Create directory where profile info is stored  
 var profilePath = Path.Join(ProfilesPath, name);  
 Directory.CreateDirectory(profilePath);  
  
 // Create the profile and add it to the dictionary  
 var profile = Profile.Default(profilePath);  
 \_profiles.TryAdd(name, profile);  
  
 // Copy the training data into the profile training data directory  
 Directory.CreateDirectory(profile.TrainingDataDirectory);  
 foreach (var file in trainingFiles)  
 {  
 var currentPath = Path.GetFullPath(file);  
 var fileName = Path.GetFileName(file);  
 var newPath = Path.Join(profile.TrainingDataDirectory, fileName);  
  
 File.Copy(currentPath, newPath);  
 }  
  
 // Create schema.json file   
 profile.Vocab = GetVocabList(profile);  
 var schemaPath = Path.Join(profilePath, "schema.json");  
 WriteJson(profile, schemaPath);  
 Mouse.SetCursor(null);  
 }  
  
 private string GetVocabList(Profile profile)  
 {  
 var vocabList = new Vocab();  
   
 // Retrieve string representations of the training data and pass them to the vocab list   
 var stringReps = MidiManager.ReadDir(profile.TrainingDataDirectory);  
 Parallel.ForEach(stringReps, rep => { vocabList.AddCharacters(rep.ToCharArray()); });  
  
 return vocabList.AsString();  
 }  
  
 /// <summary>  
 /// Retrieve a profile from the profile manager  
 /// </summary>  
 /// <param name="name">The name of the profile to search for</param>  
 /// <returns>The queried profile if it exists, otherwise null</returns>  
 public Profile? GetProfile(string name)  
 {  
 return \_profiles.TryGetValue(name, out var profile) ? profile : null;  
 }  
  
 /// <summary>  
 /// Delete a profile  
 /// </summary>  
 /// <param name="name">Name of the profile to delete</param>  
 public void DeleteProfile(string name)  
 {  
 // Do not continue if the profile does not exist  
 if (!ProfileExists(name))  
 return;  
  
 // Get the directory of the profile   
 var profilePath = Path.Join(ProfilesPath, name);  
  
 // Delete directory and its contents  
 Directory.Delete(profilePath, true);  
 }  
  
 /// <summary>  
 /// Read a JSON file  
 /// </summary>  
 /// <param name="path">Path to JSON file</param>  
 /// <typeparam name="T">Model class type to store the data</typeparam>  
 private T ReadJson<T>(string path)  
 {  
 var str = File.ReadAllText(path);  
 return JsonSerializer.Deserialize<T>(str);  
 }  
  
 /// <summary>  
 /// Write to A JSON file  
 /// </summary>  
 /// <param name="obj">The object you want to write the data from</param>  
 /// <param name="path">The path to the JSON file</param>  
 /// <typeparam name="T">The class type which you are writing the data from</typeparam>  
 private void WriteJson<T>(T obj, string path)  
 {  
 var jsonString = JsonSerializer.Serialize(obj, new JsonSerializerOptions { WriteIndented = true });  
 File.WriteAllText(path, jsonString);  
 }  
  
 /// <summary>  
 /// Delete all "before training" state files [across all profiles]  
 /// </summary>  
 public void DeleteBeforeStates()  
 {  
 foreach (var profile in \_profiles.Values)  
 if (File.Exists(profile.BeforeStateFile))  
 File.Delete(profile.BeforeStateFile);  
 }  
}

### AssemblyInfo.cs

This file is automatically generated by the framework

using System.Windows;  
  
[assembly: ThemeInfo(  
 ResourceDictionaryLocation.None, //where theme specific resource dictionaries are located  
 //(used if a resource is not found in the page,  
 // or application resource dictionaries)  
 ResourceDictionaryLocation.SourceAssembly //where the generic resource dictionary is located  
 //(used if a resource is not found in the page,  
 // app, or any theme specific resource dictionaries)  
)]

### App

The C# file oversees the running of the GUI across all windows and pages. I have added no extra functionality apart from “OnDispatcherUnhandledException” which is ran upon an unhandled exception. If there is an unhandled exception, then the user is notified, the error is logged and the application closes. I also added “OnExit” which saves the settings.json file and deletes any “before training state files” (objective 2.8)

This feature fulfils objective 4.5

#### XAML File

<Application x:Class="Apollo.App"  
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
 StartupUri="StartupWindow.xaml">  
 <Application.Resources>  
 <FontFamily x:Key="Roboto">pack://application:,,,/Fonts/#Roboto</FontFamily>  
 <FontFamily x:Key="CinzelDecorative">pack://application:,,,/Fonts/#Cinzel Decorative</FontFamily>  
 </Application.Resources>  
</Application>

#### C# File

using System;  
using System.Windows;  
using System.Windows.Threading;  
using Apollo.IO;  
using Apollo.NeuralNet;  
  
namespace Apollo;  
  
/// <summary>  
/// Interaction logic for App.xaml  
/// </summary>  
public partial class App : Application  
{  
 private const int BATCH\_SIZE = 4;  
 private const int HIDDEN\_SIZE = 32;  
  
 public App()  
 {  
 // Add unhandled exception handler   
 Dispatcher.UnhandledException += OnDispatcherUnhandledException;  
  
 R = new Random();  
 }  
  
 /// <summary>  
 /// Objects used across all parts of the application  
 /// </summary>  
 public ProfileManager ProfileManagement { get; private set; }  
  
 public StoredSettings Settings { get; set; }  
 public Random R { get; }  
 public NeuralNetwork Network { get; private set; }  
  
 protected override void OnStartup(StartupEventArgs e)  
 {  
 base.OnStartup(e);  
  
 // Load settings from settings.json if it exists and is valid   
 try  
 {  
 Settings = StoredSettings.Load();  
 LogManager.Init(Settings.LogsPath); // Initialise LogManager to use correct path   
 }  
 // If it does not exist or it isn't valid, start with fresh settings  
 catch (Exception exception)  
 {  
 Settings = StoredSettings.Default();  
 Settings.Save();  
 LogManager.Init(Settings.LogsPath); // Initialise LogManager to use correct path   
 LogManager.WriteLine($"Settings set to default:\n{e}");  
 }  
  
 ProfileManagement = new ProfileManager(Settings.ProfilesPath);  
 var initialProfile = ProfileManagement.GetProfile(Settings.SelectedProfileName);  
  
 if (ReferenceEquals(initialProfile, null))  
 initialProfile = ProfileManagement.Any();  
  
 Network = new NeuralNetwork(initialProfile, HIDDEN\_SIZE, BATCH\_SIZE, R);  
  
 Console.WriteLine(Settings.MinEpochs);  
 }  
  
 /// <summary>  
 /// Handling unhandled exceptions  
 /// </summary>  
 private void OnDispatcherUnhandledException(object sender, DispatcherUnhandledExceptionEventArgs e)  
 {  
 // Create messages for the logs and message box  
 var boxMsg = $"Unhandled exception occurred: {e.Exception.Message}";  
 var logMsg = boxMsg + $"\n{e.Exception.StackTrace}";  
  
 // Log message + stack trace  
 LogManager.WriteLine(logMsg);  
  
 // Show message to the user   
 MessageBox.Show(boxMsg, "Error", MessageBoxButton.OK, MessageBoxImage.Error);  
  
 e.Handled = true;  
 Shutdown(); // Close the app since the exception may have an effect on how the app runs.  
 }  
  
 /// <summary>  
 /// Is called when the application is closed  
 /// </summary>  
 protected override void OnExit(ExitEventArgs e)  
 {  
 try  
 {  
 // Save settings  
 Settings.Save();  
  
 // Delete before states  
 ProfileManagement.DeleteBeforeStates();  
 }  
 finally  
 {  
 base.OnExit(e);  
 }  
 }  
}

### Startup Window

This is the window that user sees when first loading the program

#### XAML file

<Window x:Class="Apollo.StartupWindow"  
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
 xmlns:d="http://schemas.microsoft.com/expression/blend/2008"  
 xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"  
 mc:Ignorable="d"  
 Title="Apollo Music Generation" Height="822" Width="1171" ResizeMode="CanMinimize">  
 <Grid>  
 <Grid.ColumnDefinitions>  
 <ColumnDefinition Width="418" />  
 <ColumnDefinition />  
 </Grid.ColumnDefinitions>  
  
 <!-- Column 0 -->  
 <Rectangle Grid.Column="0" Fill="#044bd9" />  
 <StackPanel Orientation="Vertical" Grid.Column="0" VerticalAlignment="Center" HorizontalAlignment="Center">  
 <Label Content="Apollo" FontFamily="{StaticResource CinzelDecorative}" FontSize="50pt" Foreground="#eaf205"  
 FontWeight="Bold" HorizontalAlignment="Center" />  
 <Label Content="generate music with ai" FontFamily="{StaticResource CinzelDecorative}" FontSize="20pt"  
 FontWeight="Bold" HorizontalAlignment="Center" Foreground="#f25c05" />  
 </StackPanel>  
  
 <!-- Column 1 -->  
 <Rectangle Grid.Column="1" Fill="#0b2126" />  
 <StackPanel Orientation="Vertical" Grid.Column="1" VerticalAlignment="Center" HorizontalAlignment="Center">  
 <Button Name="TrainButton" Content="Train" FontSize="46.74" FontFamily="{StaticResource Roboto}"  
 FontWeight="Bold"  
 Foreground="#eaf205" Background="#0476d9" Width="378" Height="116" Click="ButtonClick" />  
 <Separator Height="50" Opacity="0" />  
 <Button Name="CreateButton" Content="Create" FontSize="46.74" FontFamily="{StaticResource Roboto}"  
 FontWeight="Bold"  
 Foreground="#eaf205" Background="#0476d9" Width="378" Height="116" Click="ButtonClick" />  
 <Separator Height="50" Opacity="0" />  
 <Button Name="ListenButton" Content="Listen" FontSize="46.74" FontFamily="{StaticResource Roboto}"  
 FontWeight="Bold"  
 Foreground="#eaf205" Background="#0476d9" Width="378" Height="116" Click="ButtonClick" />  
 <Separator Height="50" Opacity="0" />  
 <Button Name="SettingsButton" Content="Settings" FontSize="46.74" FontFamily="{StaticResource Roboto}"  
 FontWeight="Bold"  
 Foreground="#eaf205" Background="#0476d9" Width="378" Height="116" Click="ButtonClick" />  
 </StackPanel>  
 </Grid>  
</Window>

#### C# File

The only user interaction that happens on that page is the selection of a button, which opens the corresponding page on the main window.

using System;  
using System.Windows;  
using System.Windows.Controls;  
using System.Windows.Input;  
  
namespace Apollo;  
  
/// <summary>  
/// Interaction logic for MainWindow.xaml  
/// </summary>  
public partial class StartupWindow : Window  
{  
 public StartupWindow()  
 {  
 InitializeComponent();  
 }  
  
 private void ButtonClick(object sender, RoutedEventArgs e)  
 {  
 var button = (Button)sender;  
  
  
 // Create new window, and make it the "main window" of the application  
 Mouse.OverrideCursor = Cursors.Wait;  
 var mainWindow = new MainWindow(Convert.ToString(button.Content));  
 var app = (App)Application.Current;  
 app.MainWindow = mainWindow;  
 app.MainWindow.Show();  
 Close();  
 }  
}

### Main Window

This is opened after the user selects an option from the Startup window

#### XAML file

<Window x:Class="Apollo.MainWindow"  
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
 xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"  
 xmlns:d="http://schemas.microsoft.com/expression/blend/2008"  
 mc:Ignorable="d"  
 Title="Apollo - Train" Height="822" Width="1171" ResizeMode="CanMinimize">  
 <Grid>  
 <Grid.RowDefinitions>  
 <RowDefinition Height="121.17" />  
 <RowDefinition />  
 </Grid.RowDefinitions>  
  
 <!-- Navigation Bar (Row 0) -->  
 <Rectangle Grid.Row="0" Fill="#044bd9" />  
  
 <Label Grid.Row="0" Width="100" Content="A" FontFamily="{StaticResource CinzelDecorative}" FontWeight="Bold"  
 FontSize="75"  
 Foreground="#eaf205" VerticalAlignment="Center" HorizontalAlignment="Left"  
 HorizontalContentAlignment="Center" />  
  
 <StackPanel Grid.Row="0" Orientation="Horizontal" HorizontalAlignment="Center" VerticalAlignment="Center">  
 <Button Name="TrainButton" Content="Train" FontFamily="{StaticResource Roboto}" FontWeight="Bold"  
 FontSize="50"  
 Foreground="#eaf205" VerticalAlignment="Center" HorizontalAlignment="Center"  
 Background="Transparent" BorderThickness="0" Click="ButtonClick" />  
 <Separator Width="50" Opacity="0" />  
 <Button Name="CreateButton" Content="Create" FontFamily="{StaticResource Roboto}" FontWeight="Bold"  
 FontSize="50"  
 Foreground="#eaf205" VerticalAlignment="Center" HorizontalAlignment="Center"  
 Background="Transparent" BorderThickness="0" Click="ButtonClick" />  
 <Separator Width="50" Opacity="0" />  
 <Button Name="ListenButton" Content="Listen" FontFamily="{StaticResource Roboto}" FontWeight="Bold"  
 FontSize="50"  
 Foreground="#eaf205" VerticalAlignment="Center" HorizontalAlignment="Center"  
 Background="Transparent" BorderThickness="0" Click="ButtonClick" />  
 <Separator Width="50" Opacity="0" />  
 <Button Name="SettingsButton" Content="Settings" FontFamily="{StaticResource Roboto}" FontWeight="Bold"  
 FontSize="50"  
 Foreground="#eaf205" VerticalAlignment="Center" HorizontalAlignment="Center"  
 Background="Transparent" BorderThickness="0" Click="ButtonClick" />  
 </StackPanel>  
  
 <!-- Frame for showing the different pages of the application -->  
 <Frame Name="PageFrame" Grid.Row="1" Background="#0b2126" NavigationUIVisibility="Hidden" />  
 </Grid>  
</Window>

#### C# file

This window is not just responsible for the user interface that it provides, but also the ability for the other pages shown within the window to access the various features of the backend (i.e. the network).

using System;  
using System.Windows;  
using System.Windows.Controls;  
using System.Windows.Input;  
using System.Windows.Media;  
  
namespace Apollo;  
  
public partial class MainWindow : Window  
{  
 public MainWindow(string startingPage)  
 {  
 InitializeComponent();  
  
 // Open on the correct page  
 CurrentlySelected = TrainButton;  
  
 var buttonToHighlight = TrainButton;  
 switch (startingPage.ToLower())  
 {  
 case "train":  
 buttonToHighlight = TrainButton;  
 break;  
 case "create":  
 buttonToHighlight = CreateButton;  
 break;  
 case "listen":  
 buttonToHighlight = ListenButton;  
 break;  
 case "settings":  
 buttonToHighlight = SettingsButton;  
 break;  
 }  
  
 ChangePage(buttonToHighlight);  
  
 Mouse.OverrideCursor = null; // Change the mouse back to default  
 }  
  
 #region Page Selection  
  
 private Button CurrentlySelected { get; set; }  
  
 private readonly SolidColorBrush \_selectedColour =  
 new((Color)ColorConverter.ConvertFromString("#f25c05"));  
  
 private readonly SolidColorBrush \_unselectedColour =  
 new((Color)ColorConverter.ConvertFromString("#eaf205"));  
  
 private void ChangePage(Button newSelected)  
 {  
 // Change window title  
 Title = $"Apollo - {newSelected.Content}";  
  
 // Highlight text  
 CurrentlySelected.Foreground = \_unselectedColour;  
 newSelected.Foreground = \_selectedColour;  
 CurrentlySelected = newSelected;  
  
 // Change page  
 switch (newSelected.Name)  
 {  
 case "TrainButton":  
 PageFrame.Source = new Uri("TrainingPage.xaml", UriKind.Relative);  
 break;  
 case "CreateButton":  
 PageFrame.Source = new Uri("CreationPage.xaml", UriKind.Relative);  
 break;  
 case "ListenButton":  
 PageFrame.Source = new Uri("ListenPage.xaml", UriKind.Relative);  
 break;  
 case "SettingsButton":  
 PageFrame.Source = new Uri("SettingsPage.xaml", UriKind.Relative);  
 break;  
 }  
 }  
  
 private void ButtonClick(object sender, RoutedEventArgs e)  
 {  
 var clickedButton = (Button)sender;  
 ChangePage(clickedButton);  
 }  
  
 #endregion  
}

### Training Page

This page is shown within the main window and allows the user to train the network, hence fulfilling objective part of 2.4 (since the ability to change between profiles is possible through the settings page).

It also fulfils objective 2.6 because it allows the user to revert training with a click of a button.

#### XAML file

<Page x:Class="Apollo.TrainingPage"  
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
 xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"  
 xmlns:d="http://schemas.microsoft.com/expression/blend/2008"  
 mc:Ignorable="d"  
 Title="TrainingPage" Height="700.83" Width="1171"  
 Background="Transparent">  
 <Grid Margin="50 25 50 50">  
 <Grid.ColumnDefinitions>  
 <ColumnDefinition Width="500" />  
 <ColumnDefinition />  
 </Grid.ColumnDefinitions>  
 <Grid.RowDefinitions>  
 <RowDefinition />  
 <RowDefinition />  
 <RowDefinition />  
 <RowDefinition />  
 <RowDefinition />  
 <RowDefinition />  
 </Grid.RowDefinitions>  
  
 <TextBlock Grid.Row="0" Grid.Column="0" Grid.ColumnSpan="2"  
 VerticalAlignment="Center"  
 Foreground="#0476d9" FontFamily="{StaticResource Roboto}" FontSize="35">  
 Train the neural network. Be warned that this process can also hinder the <LineBreak />  
 performance of the network.  
 </TextBlock>  
  
 <!-- Minimum Epochs -->  
 <Label Grid.Row="1" Grid.Column="0"  
 VerticalAlignment="Center" HorizontalAlignment="Right"  
 FontFamily="{StaticResource Roboto}" FontWeight="Bold" FontSize="35"  
 Foreground="#f25c05" Content="Minimum Epochs:" />  
 <StackPanel Grid.Row="1" Grid.Column="1" Orientation="Horizontal">  
 <Slider Name="MinEpochSlider" VerticalAlignment="Center" Width="450"  
 Minimum="10" Maximum="500" IsSnapToTickEnabled="True"  
 SmallChange="1" />  
 <Label VerticalAlignment="Center" HorizontalAlignment="Center"  
 FontFamily="{StaticResource Roboto}" Foreground="#eaf205" FontSize="35"  
 Content="{Binding ElementName=MinEpochSlider, Path=Value}" />  
 </StackPanel>  
  
 <!-- Maximum Epochs -->  
 <Label Grid.Row="2" Grid.Column="0"  
 VerticalAlignment="Center" HorizontalAlignment="Right"  
 FontFamily="{StaticResource Roboto}" FontWeight="Bold" FontSize="35"  
 Foreground="#f25c05" Content="Maximum Epochs:" />  
 <StackPanel Grid.Row="2" Grid.Column="1" Orientation="Horizontal">  
 <Slider Name="MaxEpochSlider" VerticalAlignment="Center" Width="450"  
 Minimum="100" Maximum="2000" IsSnapToTickEnabled="True"  
 SmallChange="1" />  
 <Label VerticalAlignment="Center" HorizontalAlignment="Center"  
 FontFamily="{StaticResource Roboto}" Foreground="#eaf205" FontSize="35"  
 Content="{Binding ElementName=MaxEpochSlider, Path=Value}" />  
 </StackPanel>  
  
 <!-- Maximum Error -->  
 <Label Grid.Row="3" Grid.Column="0"  
 VerticalAlignment="Center" HorizontalAlignment="Right"  
 FontFamily="{StaticResource Roboto}" FontWeight="Bold" FontSize="35"  
 Foreground="#f25c05" Content="Maximum Error:" />  
 <StackPanel Grid.Row="3" Grid.Column="1" Orientation="Horizontal">  
 <Slider Name="MaxErrorSlider" VerticalAlignment="Center" Width="450"  
 Minimum="0" Maximum="0.5" SmallChange="0.001" />  
 <Label VerticalAlignment="Center" HorizontalAlignment="Center"  
 FontFamily="{StaticResource Roboto}" Foreground="#eaf205" FontSize="35"  
 Content="{Binding ElementName=MaxErrorSlider, Path=Value}"  
 ContentStringFormat="{}{0:N4}" />  
 </StackPanel>  
  
 <!-- Batches per Epoch -->  
 <Label Grid.Row="4" Grid.Column="0"  
 VerticalAlignment="Center" HorizontalAlignment="Right"  
 FontFamily="{StaticResource Roboto}" FontWeight="Bold" FontSize="35"  
 Foreground="#f25c05" Content="Batches per Epoch:" />  
 <StackPanel Grid.Row="4" Grid.Column="1" Orientation="Horizontal">  
 <Slider Name="BatchesPerEpochSlider" VerticalAlignment="Center" Width="450"  
 Minimum="10" Maximum="150" IsSnapToTickEnabled="True"  
 SmallChange="1" />  
 <Label VerticalAlignment="Center" HorizontalAlignment="Center"  
 FontFamily="{StaticResource Roboto}" Foreground="#eaf205" FontSize="35"  
 Content="{Binding ElementName=BatchesPerEpochSlider, Path=Value}" />  
 </StackPanel>  
  
 <!-- Train and Revert Training Buttons -->  
 <StackPanel Grid.Row="5" Grid.Column="0" Grid.ColumnSpan="2" VerticalAlignment="Center"  
 HorizontalAlignment="Center" Orientation="Horizontal">  
 <Button HorizontalAlignment="Center" VerticalAlignment="Center"  
 Width="375" Height="75"  
 Background="#eaf205" Foreground="#044bd9"  
 FontFamily="{StaticResource Roboto}" FontWeight="Bold" FontSize="50"  
 Content="Revert Training" Click="RevertButtonClicked" />  
 <Separator Opacity="0" Width="5" />  
 <Button HorizontalAlignment="Center" VerticalAlignment="Center"  
 Width="150" Height="75"  
 Background="#eaf205" Foreground="#044bd9"  
 FontFamily="{StaticResource Roboto}" FontWeight="Bold" FontSize="50"  
 Content="Train" Click="TrainButtonClicked" />  
 </StackPanel>  
 </Grid>  
</Page>

#### C# file

using System;  
using System.Windows;  
using System.Windows.Controls;  
using Apollo.NeuralNet;  
  
namespace Apollo;  
  
public partial class TrainingPage : Page  
{  
 public TrainingPage()  
 {  
 Network = (Application.Current as App).Network;  
 Settings = (Application.Current as App).Settings;  
  
 InitializeComponent();  
  
 MinEpochSlider.Value = Settings.MinEpochs;  
 MaxEpochSlider.Value = Settings.MaxEpochs;  
 MaxErrorSlider.Value = Settings.MaxError;  
 BatchesPerEpochSlider.Value = Settings.BatchesPerEpoch;  
 }  
  
 private NeuralNetwork Network { get; }  
 private StoredSettings Settings { get; }  
  
 /// <summary>  
 /// Collect values from slider and send them to neural network  
 /// </summary>  
 private void TrainButtonClicked(object sender, RoutedEventArgs e)  
 {  
 var minEpochs = Convert.ToInt32(MinEpochSlider.Value);  
 var maxEpochs = Convert.ToInt32(MaxEpochSlider.Value);  
  
 // minEpochs cannot be bigger than maxEpochs  
 // Avoid training if this is the case  
 if (minEpochs > maxEpochs)  
 {  
 MessageBox.Show("Minimum Epochs cannot be greater than Maximum Epochs", "Invalid Data Entry",  
 MessageBoxButton.OK, MessageBoxImage.Exclamation);  
 return;  
 }  
  
 var maxError = (float)MaxEpochSlider.Value;  
 var batchesPerEpoch = Convert.ToInt32(BatchesPerEpochSlider.Value);  
  
 Network.Train(minEpochs, maxEpochs, maxError, batchesPerEpoch);  
  
 MessageBox.Show("Training is complete.");  
 }  
  
 private void RevertButtonClicked(object sender, RoutedEventArgs e)  
 {  
 var confirmation = MessageBox.Show("Are you sure?", "Revert Training Confirmation",  
 MessageBoxButton.YesNo);  
  
 if (confirmation == MessageBoxResult.No)  
 return;  
  
 var reverted = Network.TryRevertTraining();  
  
 if (!reverted)  
 MessageBox.Show("The attempt to revert training failed");  
 }  
}

### Creation Page

This page is shown within the main window and allows the user to create music using the neural network. Thus, fulfilling objective 4.2. It also allows the user to define where to store the generated file, thus fulfilling objective 3.2

#### XAML file

<Page x:Class="Apollo.CreationPage"  
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
 xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"  
 xmlns:d="http://schemas.microsoft.com/expression/blend/2008"  
 mc:Ignorable="d"  
 Title="CreationPage" Height="700.83" Width="1171"  
 Background="Transparent">  
 <Grid Margin="50 25 50 50">  
 <Grid.RowDefinitions>  
 <RowDefinition Height="150" />  
 <RowDefinition Height="100" />  
 <RowDefinition Height="100" />  
 <RowDefinition />  
 </Grid.RowDefinitions>  
 <Grid.ColumnDefinitions>  
 <ColumnDefinition Width="500" />  
 <ColumnDefinition />  
 </Grid.ColumnDefinitions>  
  
 <TextBlock Grid.Row="0" Grid.Column="0" Grid.ColumnSpan="2"  
 VerticalAlignment="Center"  
 Foreground="#0476d9" FontFamily="{StaticResource Roboto}" FontSize="35">  
 Create music with Apollo. Apollo creates music by generating <LineBreak />  
 characters. So increasing the number of characters generated will <LineBreak />  
 increase the length of the generated piece of music.  
 </TextBlock>  
  
 <!-- Generation Length -->  
 <Label Grid.Row="1" Grid.Column="0"  
 VerticalAlignment="Center" HorizontalAlignment="Right"  
 FontFamily="{StaticResource Roboto}" FontWeight="Bold" FontSize="35"  
 Foreground="#f25c05" Content="Generation Length:" />  
 <StackPanel Grid.Row="1" Grid.Column="1" Orientation="Horizontal">  
 <Slider Name="GenerationLenSlider" VerticalAlignment="Center" Width="450"  
 Minimum="5000" Maximum="50000" IsSnapToTickEnabled="True"  
 SmallChange="100" />  
 <Label VerticalAlignment="Center" HorizontalAlignment="Center"  
 FontFamily="{StaticResource Roboto}" Foreground="#eaf205" FontSize="30"  
 Content="{Binding ElementName=GenerationLenSlider, Path=Value}" />  
 </StackPanel>  
  
 <!-- Beats per Minute -->  
 <Label Grid.Row="2" Grid.Column="0"  
 VerticalAlignment="Center" HorizontalAlignment="Right"  
 FontFamily="{StaticResource Roboto}" FontWeight="Bold" FontSize="35"  
 Foreground="#f25c05" Content="Beats per Minute:" />  
 <StackPanel Grid.Row="2" Grid.Column="1" Orientation="Horizontal">  
 <Slider Name="BpmSlider" VerticalAlignment="Center" Width="450"  
 Minimum="30" Maximum="120" IsSnapToTickEnabled="True"  
 SmallChange="100" />  
 <Label VerticalAlignment="Center" HorizontalAlignment="Center"  
 FontFamily="{StaticResource Roboto}" Foreground="#eaf205" FontSize="30"  
 Content="{Binding ElementName=BpmSlider, Path=Value}" />  
 </StackPanel>  
  
 <!-- Create Button -->  
 <Button Grid.Row="3" Grid.Column="0" Grid.ColumnSpan="2"  
 HorizontalAlignment="Center" VerticalAlignment="Center"  
 Width="200" Height="200"  
 Background="#ff0000" Foreground="#000000"  
 FontFamily="{StaticResource Roboto}" FontWeight="Bold" FontSize="50"  
 Content="Create" Click="CreateButtonClicked">  
 <Button.Resources>  
 <Style TargetType="Border">  
 <Setter Property="CornerRadius" Value="100" />  
 </Style>  
 </Button.Resources>  
 </Button>  
 </Grid>  
</Page>

#### C# file

using System;  
using System.IO;  
using System.Windows;  
using System.Windows.Controls;  
using Microsoft.Win32;  
  
namespace Apollo;  
  
public partial class CreationPage : Page  
{  
 public CreationPage()  
 {  
 Settings = (Application.Current as App).Settings;  
  
 InitializeComponent();  
  
 GenerationLenSlider.Value = Settings.GenerationLength;  
 BpmSlider.Value = Settings.Bpm;  
 }  
  
 private StoredSettings Settings { get; }  
  
 /// <summary>  
 /// Opens a SaveFileDialog for the user  
 /// </summary>  
 /// <returns>The result of the file dialog</returns>  
 private string GetSavePath()  
 {  
 var dialog = new SaveFileDialog();  
 dialog.Filter = "MIDI File|\*.mid";  
 dialog.Title = "Save the Generated Piece";  
  
 if (dialog.ShowDialog() == true)  
 return Path.GetFullPath(dialog.FileName);  
  
 return "";  
 }  
  
 /// <summary>  
 /// Called when the create button is clicked  
 /// </summary>  
 private void CreateButtonClicked(object sender, RoutedEventArgs e)  
 {  
 // Get the generation length + bpm from the sliders + prompt the user for the save path  
 var generationLength = Convert.ToInt32(GenerationLenSlider.Value);  
 var bpm = Convert.ToInt32(BpmSlider.Value);  
 var savePath = GetSavePath();  
  
 // Is null or empty if the user closes the dialog rather than selecting a save path   
 if (string.IsNullOrEmpty(savePath))  
 return;  
  
 // Start generating with the provided parameters  
 (Application.Current as App).Network.Generate(generationLength, bpm, savePath);  
 MessageBox.Show("Creation Complete.");  
 }  
}

### Queue.cs

This class is used by the listen page to allow the user to queue songs to be played.

using System.Collections.Generic;  
  
namespace Apollo;  
  
internal class Queue<T>  
{  
 private readonly List<T> \_contents;  
  
 public Queue()  
 {  
 \_contents = new List<T>();  
 }  
  
 public int Length => \_contents.Count;  
  
 public void Enqueue(T item)  
 {  
 \_contents.Add(item);  
 }  
  
 public T Peek()  
 {  
 if (IsEmpty())  
 return default;  
  
 return \_contents[0];  
 }  
  
 public T Dequeue()  
 {  
 if (IsEmpty())  
 return default;  
  
 var dequeued = Peek();  
 \_contents.RemoveAt(0);  
 return dequeued;  
 }  
  
 public bool IsEmpty()  
 {  
 return \_contents.Count == 0;  
 }  
  
 public void Clear()  
 {  
 \_contents.Clear();  
 }  
}

### Stack.cs

This class is used by the listen page in order to keep track of the most recent songs that have been played by the user (in order of most recent)

using System.Collections.Generic;  
  
namespace Apollo;  
  
public class Stack<T>  
{  
 private readonly List<T> \_contents;  
 private readonly int \_maxItems;  
 private int \_pointer;  
  
 public Stack(int maxItems)  
 {  
 \_contents = new List<T>();  
 \_pointer = -1;  
 \_maxItems = maxItems;  
 }  
  
 public void Push(T item)  
 {  
 // Do not add item if it is at capacity   
 if (\_contents.Count == \_maxItems)  
 return;  
   
 \_contents.Add(item);  
 \_pointer++;  
 }  
  
 public T Peek()  
 {  
 if (IsEmpty())  
 return default;  
  
 return \_contents[\_pointer];  
 }  
  
 public T Pop()  
 {  
 if (IsEmpty())  
 return default;  
  
 var popped = Peek();  
 \_contents.RemoveAt(\_pointer);  
 \_pointer--;  
 return popped;  
 }  
  
 public bool IsEmpty()  
 {  
 return \_contents.Count == 0;  
 }  
  
 public void Clear()  
 {  
 \_contents.Clear();  
 }  
}

### Listen Page

This page targets objective 4.3 since it allows the user to queue and play any MIDI file (not just the ones that have been generated by the network). It implements a queue and a stack in order to achieve this.

#### XAML file

<Page x:Class="Apollo.ListenPage"  
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
 xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"  
 xmlns:d="http://schemas.microsoft.com/expression/blend/2008"  
 mc:Ignorable="d"  
 Title="ListenPage" Height="700.83" Width="1171"  
 Background="Transparent">  
 <Grid Margin="50 25 50 50">  
 <Grid.ColumnDefinitions>  
 <ColumnDefinition />  
 <ColumnDefinition Width="300" />  
 </Grid.ColumnDefinitions>  
  
 <MediaElement Name="MusicPlayer" LoadedBehavior="Manual" MediaOpened="OnMediaOpened"  
 MediaEnded="OnMediaEnded" MaxHeight="0" MaxWidth="0" />  
  
 <!-- Music Player -->  
 <Grid Grid.Column="0">  
 <Grid.RowDefinitions>  
 <RowDefinition Height="150" />  
 <RowDefinition Height="150" />  
 <RowDefinition />  
 </Grid.RowDefinitions>  
  
 <StackPanel Grid.Row="0" Orientation="Horizontal" VerticalAlignment="Center" HorizontalAlignment="Left">  
 <Label Content="Next Song:" VerticalAlignment="Center"  
 FontFamily="{StaticResource Roboto}" FontSize="40" FontWeight="Bold" Foreground="#0476d9" />  
 <Label Name="NextSongLabel" Content="No song in playlist" VerticalAlignment="Center"  
 FontFamily="{StaticResource Roboto}" FontSize="40" Foreground="#eaf205" />  
 </StackPanel>  
  
 <StackPanel Grid.Row="1" Orientation="Horizontal" VerticalAlignment="Center" HorizontalAlignment="Left">  
 <Label Content="Current Song:" VerticalAlignment="Center"  
 FontFamily="{StaticResource Roboto}" FontSize="40" FontWeight="Bold" Foreground="#0476d9" />  
 <Label Name="CurrentSongLabel" Content="" VerticalAlignment="Center"  
 FontFamily="{StaticResource Roboto}" FontSize="40" Foreground="#eaf205" />  
 </StackPanel>  
  
 <StackPanel Grid.Row="2" Orientation="Vertical" HorizontalAlignment="Center" VerticalAlignment="Top">  
 <ProgressBar Name="MediaProgressBar" Width="600" Height="25" />  
 <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">  
  
 <Button Width="100" Height="100" Click="OnPreviousButtonPress"  
 Background="Transparent" BorderThickness="0">  
 <Image Source="Images/PrevSong.png" />  
 </Button>  
  
 <Button Width="100" Height="100" Click="OnBack5ButtonPress"  
 Background="Transparent" BorderThickness="0">  
 <Image Source="Images/Prev5Sec.png" />  
 </Button>  
  
 <Button Width="100" Height="100" Click="OnPlayPauseButtonPress"  
 Background="Transparent" BorderThickness="0">  
 <Image Name="PlayPauseImage" Source="Images/Play.png" />  
 </Button>  
  
 <Button Width="100" Height="100" Click="OnStopButtonPress"  
 Background="Transparent" BorderThickness="0">  
 <Image Source="Images/Stop.png" />  
 </Button>  
  
 <Button Width="100" Height="100" Click="OnSkip5ButtonPress"  
 Background="Transparent" BorderThickness="0">  
 <Image Source="Images/Skip5Sec.png" />  
 </Button>  
  
 <Button Width="100" Height="100" Click="OnNextButtonPress"  
 Background="Transparent" BorderThickness="0">  
 <Image Source="Images/SkipSong.png" />  
 </Button>  
  
 </StackPanel>  
 </StackPanel>  
 </Grid>  
  
 <!-- Playlist Panel -->  
 <Grid Grid.Column="1">  
 <Grid.RowDefinitions>  
 <RowDefinition Height="100" />  
 <RowDefinition />  
 <RowDefinition Height="100" />  
 </Grid.RowDefinitions>  
  
 <Label Grid.Row="0" Content="Playlist" VerticalAlignment="Center" HorizontalAlignment="Center"  
 FontFamily="{StaticResource Roboto}" FontSize="50" FontWeight="Bold" Foreground="#f25c05" />  
  
 <StackPanel Grid.Row="1" Name="PlaylistPanel" Orientation="Vertical" CanVerticallyScroll="True" />  
  
 <Button Grid.Row="2" Content="Add to Playlist" VerticalAlignment="Center" HorizontalAlignment="Center"  
 Width="200" Height="75" FontSize="25" FontFamily="{StaticResource Roboto}" FontWeight="Bold"  
 Foreground="#044bd9" Background="#eaf205"  
 Click="OnAddButtonPress" />  
 </Grid>  
 </Grid>  
</Page>

#### C# file

using System;  
using System.IO;  
using System.Windows;  
using System.Windows.Controls;  
using System.Windows.Media;  
using System.Windows.Media.Imaging;  
using System.Windows.Threading;  
using Microsoft.Win32;  
  
namespace Apollo;  
  
public partial class ListenPage : Page  
{  
 private const string PLAY\_IMAGE = "Images/Play.png";  
 private const string PAUSE\_IMAGE = "Images/Pause.png";  
  
 public ListenPage()  
 {  
 InitializeComponent();  
 LastPlayed = new Stack<string>(15);  
 Playlist = new Queue<string>();  
 CurrentlyPlaying = null;  
 }  
  
 private Stack<string> LastPlayed { get; } // Stores last 15 songs that have been played  
 private Queue<string> Playlist { get; } // Stores songs that are upcoming   
 private string? CurrentlyPlaying { get; set; } // Stores path to media which is currently playing   
 private DispatcherTimer? ProgressUpdateTimer { get; set; }  
  
 /// <summary>  
 /// Update the UI elements associated with the playlist queue  
 /// (The Playlist Stack Panel and the Next Song Label)  
 /// </summary>  
 private void UpdateQueueUI()  
 {  
 // If there is nothing in the playlist then wipe the UI and display appropriate message  
 if (Playlist.IsEmpty())  
 {  
 NextSongLabel.Content = "No song in playlist";  
 PlaylistPanel.Children.Clear();  
 return;  
 }  
  
 // Nothing to update if the length of the queue at least 1 more than the number of children  
 if (Playlist.Length >= PlaylistPanel.Children.Count + 1)  
 return;  
  
 // Otherwise remove top child of stack panel and change next song label   
 PlaylistPanel.Children.RemoveAt(0);  
 NextSongLabel.Content = Path.GetFileName(Playlist.Peek());  
 }  
  
 private void PlaySong(string songPath)  
 {  
 LastPlayed.Push(songPath);  
 CurrentlyPlaying = songPath;  
  
 // Set music player source to the stored path  
 MusicPlayer.Source = new Uri(songPath);  
 MusicPlayer.Play();  
  
 // Ensure play/pause button says pause  
 PlayPauseImage.Source = new BitmapImage(new Uri(PAUSE\_IMAGE, UriKind.Relative));  
 }  
  
 /// <summary>  
 /// Begin playing the next song 3  
 /// </summary>  
 private void PlayNextSong()  
 {  
 // Return if there is nothing to play  
 if (Playlist.IsEmpty())  
 return;  
  
 var nextSong = Playlist.Dequeue();  
 PlaySong(nextSong);  
 }  
  
 /// <summary>  
 /// Adds a song to the stack panel  
 /// </summary>  
 private void AddSongToStackPanel(string filePath)  
 {  
 // Create label to add to stack panel  
 var label = new Label();  
 label.Content = Path.GetFileName(filePath);  
 label.Foreground = new SolidColorBrush((Color)ColorConverter.ConvertFromString("#eaf205"));  
 label.FontFamily = new FontFamily("file:///Fonts/Roboto");  
 label.FontWeight = FontWeights.Bold;  
 label.FontSize = 25;  
 label.HorizontalAlignment = HorizontalAlignment.Left;  
  
 // Add label to stack panel  
 PlaylistPanel.Children.Add(label);  
 }  
  
 /// <summary>  
 /// Opens a File Dialog for the user to select a file to play  
 /// </summary>  
 /// <returns>The path to the file to play</returns>  
 private string GetFile()  
 {  
 var dialog = new OpenFileDialog();  
 dialog.Filter = "MIDI File|\*.mid;\*.midi";  
 dialog.Title = "Select a file to add to the playlist";  
  
 if (dialog.ShowDialog() == true)  
 return dialog.FileName;  
  
 return string.Empty;  
 }  
  
 #region Playlist Events  
  
 /// <summary>  
 /// Event which is called when the "Add to Playlist" button is pressed  
 /// </summary>  
 private void OnAddButtonPress(object sender, RoutedEventArgs e)  
 {  
 var filePath = GetFile();  
  
 // Do not continue if no file is provided  
 if (string.IsNullOrEmpty(filePath))  
 return;  
  
 // Add to Playlist Queue   
 Playlist.Enqueue(filePath);  
  
 // Add file name to stack panel  
 AddSongToStackPanel(filePath);  
  
 // Play if nothing is playing   
 if (string.IsNullOrEmpty(CurrentlyPlaying)) PlayNextSong();  
  
 // Update the UI in case it needs updating  
 UpdateQueueUI();  
 }  
  
 #endregion  
  
 #region Playback Button Events  
  
 /// <summary>  
 /// Event which is called when previous song button is pressed  
 /// </summary>  
 private void OnPreviousButtonPress(object sender, RoutedEventArgs e)  
 {  
 if (LastPlayed.IsEmpty())  
 {  
 MessageBox.Show("You haven't played any songs recently");  
 return;  
 }  
  
 MusicPlayer.Stop();  
  
 var song = LastPlayed.Pop();  
 PlaySong(song);  
 // Do not need to update queue UI as nothing was taken from the queue  
 }  
  
 /// <summary>  
 /// Event which is called when rewind 5 seconds button is pressed  
 /// </summary>  
 private void OnBack5ButtonPress(object sender, RoutedEventArgs e)  
 {  
 var timespanChange = TimeSpan.FromSeconds(-5);  
 var newPos = timespanChange.Add(MusicPlayer.Position);  
 MusicPlayer.Position = newPos;  
 }  
  
 /// <summary>  
 /// Event which is called when the play/pause button is pressed  
 /// </summary>  
 private void OnPlayPauseButtonPress(object sender, RoutedEventArgs e)  
 {  
 // Do not handle event if nothing is playing  
 if (string.IsNullOrEmpty(CurrentlyPlaying))  
 return;  
  
 var displayedImageUri = (PlayPauseImage.Source as BitmapImage).UriSource.OriginalString;  
 Console.WriteLine(displayedImageUri);  
  
 // If the button says play, play  
 if (displayedImageUri == PLAY\_IMAGE)  
 {  
 PlayPauseImage.Source = new BitmapImage(new Uri(PAUSE\_IMAGE, UriKind.Relative));  
 MusicPlayer.Play();  
 }  
 // Otherwise pause  
 else  
 {  
 PlayPauseImage.Source = new BitmapImage(new Uri(PLAY\_IMAGE, UriKind.Relative));  
 MusicPlayer.Pause();  
 }  
 }  
  
 /// <summary>  
 /// Event which is called when stop playback button is pressed  
 /// </summary>  
 private void OnStopButtonPress(object sender, RoutedEventArgs e)  
 {  
 // Stop playback  
 MusicPlayer.Stop();  
 MusicPlayer.Close();  
  
 CurrentlyPlaying = null;  
 CurrentSongLabel.Content = "";  
  
 // Wipe playlist   
 Playlist.Clear();  
 UpdateQueueUI();  
 }  
  
 /// <summary>  
 /// Event which is called when the fast forward 5 seconds button is pressed  
 /// </summary>  
 private void OnSkip5ButtonPress(object sender, RoutedEventArgs e)  
 {  
 if (!MusicPlayer.NaturalDuration.HasTimeSpan)  
 return;  
  
 var secondsUntilEnd = MusicPlayer.NaturalDuration.TimeSpan.Subtract(MusicPlayer.Position).TotalSeconds;  
  
 if (secondsUntilEnd > 5)  
 {  
 var timespanChange = TimeSpan.FromSeconds(5);  
 var newPos = timespanChange.Add(MusicPlayer.Position);  
 MusicPlayer.Position = newPos;  
 }  
 else  
 {  
 PlayNextSong();  
 UpdateQueueUI();  
 }  
 }  
  
 /// <summary>  
 /// Event which is called when the next song button is pressed  
 /// </summary>  
 private void OnNextButtonPress(object sender, RoutedEventArgs e)  
 {  
 MusicPlayer.Stop();  
 PlayNextSong();  
  
 if (Playlist.IsEmpty())  
 CurrentSongLabel.Content = "";  
  
 UpdateQueueUI();  
 }  
  
 #endregion  
  
 #region MediaElement Events  
  
 /// <summary>  
 /// Event which is called when the MediaElement has opened and loaded a piece of media  
 /// </summary>  
 private void OnMediaOpened(object sender, RoutedEventArgs e)  
 {  
 // Update currently playing variable and label  
 CurrentlyPlaying = MusicPlayer.Source.ToString();  
 CurrentSongLabel.Content = Path.GetFileName(CurrentlyPlaying);  
  
 // Set up maximum   
 MediaProgressBar.Maximum = MusicPlayer.NaturalDuration.TimeSpan.TotalMilliseconds;  
  
 // Set up timer   
 ProgressUpdateTimer = new DispatcherTimer();  
 ProgressUpdateTimer.Interval = TimeSpan.FromSeconds(1);  
 ProgressUpdateTimer.Tick += UpdateBarEvent;  
 ProgressUpdateTimer.Start();  
 }  
  
 /// <summary>  
 /// Event which is called when the MediaElement has finished playing a piece of media  
 /// </summary>  
 private void OnMediaEnded(object sender, RoutedEventArgs e)  
 {  
 CurrentSongLabel.Content = "";  
 CurrentlyPlaying = null;  
  
 // Start playing next song and update the UI   
 PlayNextSong();  
 UpdateQueueUI();  
 }  
  
 /// <summary>  
 /// Event which is called by the DispatcherTimer to update the progress bar  
 /// </summary>  
 private void UpdateBarEvent(object? sender, EventArgs e)  
 {  
 if (CurrentlyPlaying == null) // If nothing is playing then empty the progress bar   
 {  
 ProgressUpdateTimer.Stop();  
 MediaProgressBar.Value = 0;  
 return;  
 }  
   
 // Otherwise, make it match the progress in the song  
 MediaProgressBar.Value = MusicPlayer.Position.TotalMilliseconds;  
 }  
  
 #endregion  
}

### Text Dialog

Since one is not provided by WPF, I also created a text prompt for the user. This is a dialog which prompts the user to enter a string into a textbox.

#### XAML file

<Window x:Class="Apollo.TextDialog"  
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
 Title="User Action Required"  
 Width="500"  
 Height="200"  
 ResizeMode="NoResize"  
 Background="#0b2126">  
 <StackPanel VerticalAlignment="Center">  
 <Label Name="PromptLabel" Content="Hello There" HorizontalAlignment="Center"  
 FontSize="30" FontFamily="{StaticResource Roboto}" FontWeight="Bold" Foreground="#eaf205" />  
 <TextBox Name="ResponseBox" FontFamily="{StaticResource Roboto}" FontSize="20" Width="450" />  
 <Separator Opacity="0" Height="2" />  
 <Button Content="Submit" Click="OnButtonClicked" HorizontalAlignment="Center"  
 FontFamily="{StaticResource Roboto}" Foreground="#044bd9" Background="#eaf205"  
 FontSize="20" Height="30" Width="75" FontWeight="Bold" />  
 </StackPanel>  
</Window>

#### C# file

using System.Windows;  
  
namespace Apollo;  
  
public partial class TextDialog : Window  
{  
 public TextDialog(string promptLabel)  
 {  
 InitializeComponent();  
 PromptLabel.Content = promptLabel;  
 }  
  
 public string Value => ResponseBox.Text;  
  
 private void OnButtonClicked(object sender, RoutedEventArgs e)  
 {  
 DialogResult = true;  
 }  
}

### Settings Page

This section completely tackles objective 4.4, allowing the user to edit the save directories of logs and profiles as well as the default settings for the generation and training parameters.

It also allows the user to create and change between profiles, which fulfils objectives 2.2, 2.3 and 2.4.

In terms of programming techniques, it implements recursion in its “MoveFolder” procedure.

#### XAML file

<Page x:Class="Apollo.SettingsPage"  
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
 xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"  
 xmlns:d="http://schemas.microsoft.com/expression/blend/2008"  
 mc:Ignorable="d"  
 Title="SettingsPage" Height="700.83" Width="1171"  
 Background="Transparent">  
 <Grid Margin="50 25 50 50">  
 <Grid.RowDefinitions>  
 <RowDefinition />  
 <RowDefinition />  
 <RowDefinition />  
 <RowDefinition />  
 <RowDefinition />  
 <RowDefinition />  
 <RowDefinition />  
 <RowDefinition />  
 <RowDefinition />  
 <RowDefinition />  
 <RowDefinition />  
 </Grid.RowDefinitions>  
 <Grid.ColumnDefinitions>  
 <ColumnDefinition />  
 <ColumnDefinition />  
 </Grid.ColumnDefinitions>  
  
 <TextBlock Grid.Row="0" Grid.Column="0" Grid.ColumnSpan="2"  
 VerticalAlignment="Center"  
 Foreground="#0476d9" FontFamily="{StaticResource Roboto}" FontSize="25">  
 Apollo's Settings. Do not mess with these unless you know what you are doing.  
 </TextBlock>  
  
 <!-- Profile Selection -->  
 <StackPanel Grid.Column="0" Grid.Row="1" Orientation="Horizontal" HorizontalAlignment="Right">  
 <Label VerticalAlignment="Center" HorizontalAlignment="Right"  
 FontFamily="{StaticResource Roboto}" FontWeight="Bold" FontSize="35"  
 Foreground="#f25c05" Content="Selected Profile: " />  
 <ComboBox Height="50" Width="225"  
 VerticalAlignment="Center" HorizontalAlignment="Right"  
 Name="ProfileComboBox"  
 SelectionChanged="OnProfileSelected" />  
 </StackPanel>  
 <StackPanel Grid.Column="1" Grid.Row="1" Orientation="Horizontal" HorizontalAlignment="Left">  
 <Separator Opacity="0" Width="5" />  
 <Button Width="225" Height="50"  
 Background="#eaf205" Foreground="#044bd9"  
 FontFamily="{StaticResource Roboto}" FontWeight="Bold" FontSize="25"  
 Content="Create New Profile" Click="OnCreateButtonClicked" />  
 <Separator Opacity="0" Width="5" />  
 <Button Width="175" Height="50"  
 Background="#eaf205" Foreground="#044bd9"  
 FontFamily="{StaticResource Roboto}" FontWeight="Bold" FontSize="25"  
 Content="Delete Profile" Click="OnDeleteButtonClicked" />  
 </StackPanel>  
  
 <!-- Logs File Path -->  
 <StackPanel Grid.Column="0" Grid.Row="2" Orientation="Horizontal" HorizontalAlignment="Right">  
 <Label VerticalAlignment="Center" HorizontalAlignment="Right"  
 FontFamily="{StaticResource Roboto}" FontWeight="Bold" FontSize="35"  
 Foreground="#f25c05" Content="Logs Save Path: " />  
 <Label Name="LogPathLabel" Content="logs/" VerticalAlignment="Center"  
 FontFamily="{StaticResource Roboto}" FontSize="25" Foreground="#eaf205" />  
 </StackPanel>  
 <StackPanel Grid.Column="1" Grid.Row="2" Orientation="Horizontal" HorizontalAlignment="Left">  
 <Separator Opacity="0" Width="5" />  
 <Button Width="100" Height="50"  
 Background="#eaf205" Foreground="#044bd9"  
 FontFamily="{StaticResource Roboto}" FontWeight="Bold" FontSize="25"  
 Content="Change" Click="OnLogChangeClick" />  
 </StackPanel>  
  
 <!-- Profile Save Path -->  
 <StackPanel Grid.Column="0" Grid.Row="3" Orientation="Horizontal" HorizontalAlignment="Right">  
 <Label VerticalAlignment="Center" HorizontalAlignment="Right"  
 FontFamily="{StaticResource Roboto}" FontWeight="Bold" FontSize="35"  
 Foreground="#f25c05" Content="Profiles Save Path: " />  
 <Label Name="ProfilePathLabel" Content="logs/" VerticalAlignment="Center"  
 FontFamily="{StaticResource Roboto}" FontSize="25" Foreground="#eaf205" />  
 </StackPanel>  
 <StackPanel Grid.Column="1" Grid.Row="3" Orientation="Horizontal" HorizontalAlignment="Left">  
 <Separator Opacity="0" Width="5" />  
 <Button Width="100" Height="50"  
 Background="#eaf205" Foreground="#044bd9"  
 FontFamily="{StaticResource Roboto}" FontWeight="Bold" FontSize="25"  
 Content="Change" Click="OnProfilePathChangeClick" />  
 </StackPanel>  
  
 <!-- Minimum Epochs -->  
 <Label Grid.Row="4" Grid.Column="0"  
 VerticalAlignment="Center" HorizontalAlignment="Right"  
 FontFamily="{StaticResource Roboto}" FontWeight="Bold" FontSize="35"  
 Foreground="#f25c05" Content="Default Minimum Epochs:" />  
 <StackPanel Grid.Row="4" Grid.Column="1" Orientation="Horizontal">  
 <Separator Opacity="0" Width="5" />  
 <Slider Name="MinEpochSlider" VerticalAlignment="Center" Width="450"  
 Minimum="10" Maximum="500" IsSnapToTickEnabled="True"  
 SmallChange="1" ValueChanged="OnMinEpochChange" />  
 <Label VerticalAlignment="Center" HorizontalAlignment="Center"  
 FontFamily="{StaticResource Roboto}" Foreground="#eaf205" FontSize="25"  
 Content="{Binding ElementName=MinEpochSlider, Path=Value}" />  
 </StackPanel>  
  
 <!-- Maximum Epochs -->  
 <Label Grid.Row="5" Grid.Column="0"  
 VerticalAlignment="Center" HorizontalAlignment="Right"  
 FontFamily="{StaticResource Roboto}" FontWeight="Bold" FontSize="35"  
 Foreground="#f25c05" Content="Default Maximum Epochs:" />  
 <StackPanel Grid.Row="5" Grid.Column="1" Orientation="Horizontal">  
 <Separator Opacity="0" Width="5" />  
 <Slider Name="MaxEpochSlider" VerticalAlignment="Center" Width="450"  
 Minimum="100" Maximum="2000" IsSnapToTickEnabled="True"  
 SmallChange="1" ValueChanged="OnMaxEpochChange" />  
 <Label VerticalAlignment="Center" HorizontalAlignment="Center"  
 FontFamily="{StaticResource Roboto}" Foreground="#eaf205" FontSize="25"  
 Content="{Binding ElementName=MaxEpochSlider, Path=Value}" />  
 </StackPanel>  
  
 <!-- Maximum Error -->  
 <Label Grid.Row="6" Grid.Column="0"  
 VerticalAlignment="Center" HorizontalAlignment="Right"  
 FontFamily="{StaticResource Roboto}" FontWeight="Bold" FontSize="35"  
 Foreground="#f25c05" Content="Default Maximum Error:" />  
 <StackPanel Grid.Row="6" Grid.Column="1" Orientation="Horizontal">  
 <Separator Opacity="0" Width="5" />  
 <Slider Name="MaxErrorSlider" VerticalAlignment="Center" Width="450"  
 Minimum="0" Maximum="0.5" SmallChange="0.001"  
 ValueChanged="OnMaxErrorChange" />  
 <Label VerticalAlignment="Center" HorizontalAlignment="Center"  
 FontFamily="{StaticResource Roboto}" Foreground="#eaf205" FontSize="25"  
 Content="{Binding ElementName=MaxErrorSlider, Path=Value}"  
 ContentStringFormat="{}{0:N4}" />  
 </StackPanel>  
  
 <!-- Batches per Epoch -->  
 <Label Grid.Row="7" Grid.Column="0"  
 VerticalAlignment="Center" HorizontalAlignment="Right"  
 FontFamily="{StaticResource Roboto}" FontWeight="Bold" FontSize="35"  
 Foreground="#f25c05" Content="Default Batches per Epoch:" />  
 <StackPanel Grid.Row="7" Grid.Column="1" Orientation="Horizontal">  
 <Separator Opacity="0" Width="5" />  
 <Slider Name="BatchesPerEpochSlider" VerticalAlignment="Center" Width="450"  
 Minimum="10" Maximum="150" IsSnapToTickEnabled="True"  
 SmallChange="1" ValueChanged="OnBatchesPerEpochChange" />  
 <Label VerticalAlignment="Center" HorizontalAlignment="Center"  
 FontFamily="{StaticResource Roboto}" Foreground="#eaf205" FontSize="25"  
 Content="{Binding ElementName=BatchesPerEpochSlider, Path=Value}" />  
 </StackPanel>  
  
 <!-- Generation Length -->  
 <Label Grid.Row="8" Grid.Column="0"  
 VerticalAlignment="Center" HorizontalAlignment="Right"  
 FontFamily="{StaticResource Roboto}" FontWeight="Bold" FontSize="35"  
 Foreground="#f25c05" Content="Default Generation Length: " />  
 <StackPanel Grid.Row="8" Grid.Column="1" Orientation="Horizontal">  
 <Separator Opacity="0" Width="5" />  
 <Slider Name="GenerationLenSlider" VerticalAlignment="Center" Width="450"  
 Minimum="5000" Maximum="50000" IsSnapToTickEnabled="True"  
 SmallChange="100" ValueChanged="OnGenLengthChange" />  
 <Label VerticalAlignment="Center" HorizontalAlignment="Center"  
 FontFamily="{StaticResource Roboto}" Foreground="#eaf205" FontSize="25"  
 Content="{Binding ElementName=GenerationLenSlider, Path=Value}" />  
 </StackPanel>  
  
 <!-- Beats per Minute -->  
 <Label Grid.Row="9" Grid.Column="0"  
 VerticalAlignment="Center" HorizontalAlignment="Right"  
 FontFamily="{StaticResource Roboto}" FontWeight="Bold" FontSize="35"  
 Foreground="#f25c05" Content="Default Beats per Minute:" />  
 <StackPanel Grid.Row="9" Grid.Column="1" Orientation="Horizontal">  
 <Separator Opacity="0" Width="5" />  
 <Slider Name="BpmSlider" VerticalAlignment="Center" Width="450"  
 Minimum="30" Maximum="120" IsSnapToTickEnabled="True"  
 SmallChange="100" ValueChanged="OnBpmChange" />  
 <Label VerticalAlignment="Center" HorizontalAlignment="Center"  
 FontFamily="{StaticResource Roboto}" Foreground="#eaf205" FontSize="25"  
 Content="{Binding ElementName=BpmSlider, Path=Value}" />  
 </StackPanel>  
  
 <!-- Reset button -->  
 <Button Grid.Row="10" Grid.Column="0" Grid.ColumnSpan="2"  
 Width="225" Height="50"  
 Background="#eaf205" Foreground="#044bd9"  
 FontFamily="{StaticResource Roboto}" FontWeight="Bold" FontSize="25"  
 Content="Reset to Defaults" Click="OnResetSettingsClicked" />  
 </Grid>  
</Page>

#### C# File

using System.IO;  
using System.Windows;  
using System.Windows.Controls;  
using Apollo.IO;  
using Apollo.NeuralNet;  
using Microsoft.WindowsAPICodePack.Dialogs;  
  
namespace Apollo;  
  
public partial class SettingsPage : Page  
{  
 private readonly bool \_isInitialising;  
  
 public SettingsPage()  
 {  
 // Retrieve objects from App.xaml.cs (AGGREGATION)   
 ProfileManagement = (Application.Current as App).ProfileManagement;  
 Network = (Application.Current as App).Network;  
 Settings = (Application.Current as App).Settings;  
  
 \_isInitialising = true;  
 InitializeComponent();  
 \_isInitialising = false;  
  
 // Load the existing profiles as options in the combo box and select the currently selected one   
 AddProfilesToComboBox();  
  
 ShowSavedSettings();  
 }  
  
 /// <summary>  
 /// Objects which are used throughout the page  
 /// </summary>  
 private ProfileManager ProfileManagement { get; }  
  
 private NeuralNetwork Network { get; }  
 private StoredSettings Settings { get; }  
  
 #region Helper Subroutines  
  
 /// <summary>  
 /// Change all labels/sliders to current settings  
 /// </summary>  
 private void ShowSavedSettings()  
 {  
 LogPathLabel.Content = Settings.LogsPath;  
 ProfilePathLabel.Content = Settings.ProfilesPath;  
 MinEpochSlider.Value = Settings.MinEpochs;  
 MaxEpochSlider.Value = Settings.MaxEpochs;  
 MaxErrorSlider.Value = Settings.MaxError;  
 BatchesPerEpochSlider.Value = Settings.BatchesPerEpoch;  
 GenerationLenSlider.Value = Settings.GenerationLength;  
 BpmSlider.Value = Settings.Bpm;  
 }  
  
 /// <summary>  
 /// Procedure which adds profiles already in the ProfileManager to the combo box  
 /// </summary>  
 private void AddProfilesToComboBox()  
 {  
 for (var i = 0; i < ProfileManagement.ProfileNames.Length; i++)  
 {  
 var profileName = ProfileManagement.ProfileNames[i];  
 ProfileComboBox.Items.Add(profileName);  
  
 // Will never be a null reference, we are retrieving profileName from the list of keys of the dictionary  
 // If it is the selected profile, set it as the selected option in the combo box  
 if (Network.CurrentProfile == ProfileManagement.GetProfile(profileName))  
 ProfileComboBox.SelectedIndex = i;  
 }  
 }  
  
 /// <summary>  
 /// Change the profile across all objects used in the application  
 /// </summary>  
 /// <param name="profileName">The name of the profile that was selected</param>  
 /// <param name="changeComboBox">Whether to change the currently selected value of the combo box</param>  
 private void ChangeProfile(string profileName, bool changeComboBox = false)  
 {  
 // Retrieve profile from ProfileManager  
 var profile = ProfileManagement.GetProfile(profileName);  
  
 // Do not continue if the profile is not found   
 if (ReferenceEquals(profile, null))  
 return;  
  
 // Change the profile used in the NeuralNetwork object  
 Network.ChangeProfile(profile);  
  
 // Change combo box if required  
 if (changeComboBox)  
 {  
 var comboIndex = ProfileComboBox.Items.IndexOf(profileName);  
 ProfileComboBox.SelectedIndex = comboIndex;  
 }  
  
 // Change settings so that the selected profile changes on start up   
 Settings.SelectedProfileName = profileName;  
 Settings.Save();  
 }  
  
  
 /// <summary>  
 /// Delete a profile from the file system  
 /// </summary>  
 private void DeleteProfile(string profileName)  
 {  
 ProfileManagement.DeleteProfile(profileName); // Delete from file system  
 ProfileComboBox.Items.Remove(profileName); // Remove from ComboBox  
 }  
  
 /// <summary>  
 /// Move a folder to a new destination  
 /// </summary>  
 /// <param name="oldPath">The path to the directory you wish to move</param>  
 /// <param name="newPath">The path to where you wish to move the directory</param>  
 private void MoveFolder(string oldPath, string newPath)  
 {  
 Directory.CreateDirectory(newPath); // Create new directory in new location  
  
 var files = Directory.GetFiles(oldPath);  
  
 foreach (var file in files) // Move files to new destination one by one   
 {  
 var filename = Path.GetFileName(file);  
 var newFilePath = Path.Join(newPath, filename);  
  
 File.Copy(file, newFilePath);  
 File.Delete(file);  
 }  
  
 var dirs = Directory.GetDirectories(oldPath);  
 foreach (var dir in dirs)  
 {  
 var dirname = Path.GetFileName(dir);  
 var newDir = Path.Join(newPath, dirname);  
 MoveFolder(dir, newDir); // Use recursion to move contents of every directory in the parent directory  
 }  
 }  
  
 #endregion  
  
 #region Events  
  
 /// <summary>  
 /// Event which is called when the Create Profile button is clicked  
 /// </summary>  
 private void OnCreateButtonClicked(object sender, RoutedEventArgs e)  
 {  
 // Retrieve the name via a prompt   
 var name = OpenNamePrompt();  
  
 // Do not continue if no name was provided  
 if (name == "")  
 {  
 MessageBox.Show("Must enter a profile name in order to create a profile");  
 return;  
 }  
  
 // If it already exists do not continue   
 if (ProfileManagement.ProfileExists(name))  
 {  
 MessageBox.Show("A profile already exists with that name");  
 return;  
 }  
  
 // Go through necessary steps to create the profile   
 ProfileManagement.CreateProfile(name);  
  
 ProfileComboBox.Items.Add(name);  
  
 // Change the profile and value in the combo box (since it was not selected from the combo box)   
 ChangeProfile(name, true);  
 }  
  
 /// <summary>  
 /// Event which is called when the Delete Profile button is clicked  
 /// </summary>  
 private void OnDeleteButtonClicked(object sender, RoutedEventArgs e)  
 {  
 if (ProfileComboBox.Items.Count == 1) // Do not allow the user to delete if there is only 1 profile  
 {  
 MessageBox.Show("You cannot delete the only profile");  
 return;  
 }  
   
 // Retrieve the identifier for the profile the user wants to delete  
 var selectedIndex = ProfileComboBox.SelectedIndex;  
 var selectedName = (string)ProfileComboBox.Items.GetItemAt(selectedIndex);  
  
 DeleteProfile(selectedName);  
  
 // Select the profile at the top of the combo box  
 ProfileComboBox.SelectedIndex = 0;  
 }  
  
 /// <summary>  
 /// Event which is called when the Logs Save Path change button is clicked  
 /// </summary>  
 private void OnLogChangeClick(object sender, RoutedEventArgs e)  
 {  
 // Get path from file dialog   
 var path = GetPath(Settings.LogsPath);  
  
 if (string.IsNullOrEmpty(path))  
 return;  
  
 // Move logs that are already there  
 MoveFolder(Settings.LogsPath, path);  
  
 // Change the path   
 Settings.LogsPath = path;  
 LogManager.ChangeLogPath(Settings.LogsPath);  
  
 // Change log path in the UI  
 LogPathLabel.Content = path;  
 }  
  
 /// <summary>  
 /// Event which is called when the Logs Save Path change button is clicked  
 /// </summary>  
 private void OnProfilePathChangeClick(object sender, RoutedEventArgs e)  
 {  
 // Get path from file dialog   
 var path = GetPath(Settings.ProfilesPath);  
  
 if (string.IsNullOrEmpty(path))  
 return;  
  
 // Move existent profiles to new location  
 MoveFolder(Settings.ProfilesPath, path);  
  
 // Change the path  
 Settings.ProfilesPath = path;  
 ProfileManagement.ChangeProfilesPath(Settings.ProfilesPath);  
  
 // Change profile path in the UI   
 LogPathLabel.Content = path;  
 }  
  
 /// <summary>  
 /// Event which is called when the min epochs slider is changed  
 /// </summary>  
 private void OnMinEpochChange(object sender, RoutedEventArgs e)  
 {  
 if (\_isInitialising)  
 return;  
  
 var newEpochs = (int)MinEpochSlider.Value;  
  
 if (newEpochs >= Settings.MaxEpochs) // Do not allow the user to select a value higher than maximum epochs  
 {  
 MessageBox.Show("Minimum epochs must be less than maximum epochs");  
 MinEpochSlider.Value = Settings.MinEpochs;  
 return;  
 }  
  
 Settings.MinEpochs = newEpochs;  
 }  
  
 /// <summary>  
 /// Event which is called when the max epoch slider is changed  
 /// </summary>  
 private void OnMaxEpochChange(object sender, RoutedEventArgs e)  
 {  
 if (\_isInitialising)  
 return;  
  
 var newEpochs = (int)MaxEpochSlider.Value;  
  
 if (newEpochs <= Settings.MinEpochs) // Do not allow the user to select a value lower than minimum epochs  
 {  
 MessageBox.Show("Maximum epochs must be less than minimum epochs");  
 MaxEpochSlider.Value = Settings.MaxEpochs;  
 return;  
 }  
  
 Settings.MaxEpochs = newEpochs;  
 }  
  
 /// <summary>  
 /// Event which is called when the max error slider is changed  
 /// </summary>  
 private void OnMaxErrorChange(object sender, RoutedEventArgs e)  
 {  
 if (\_isInitialising)  
 return;  
  
 var newError = (float)MaxErrorSlider.Value;  
 Settings.MaxError = newError;  
 }  
  
 /// <summary>  
 /// Event which is called when the batches per epoch slider is changed  
 /// </summary>  
 private void OnBatchesPerEpochChange(object sender, RoutedEventArgs e)  
 {  
 if (\_isInitialising)  
 return;  
  
 var newBatches = (int)BatchesPerEpochSlider.Value;  
 Settings.BatchesPerEpoch = newBatches;  
 }  
  
 /// <summary>  
 /// Event which is called when the characters to generate slider is changed  
 /// </summary>  
 private void OnGenLengthChange(object sender, RoutedEventArgs e)  
 {  
 if (\_isInitialising)  
 return;  
  
 var newLength = (int)GenerationLenSlider.Value;  
 Settings.GenerationLength = newLength;  
 }  
  
 /// <summary>  
 /// Event which is called when the bpm slider is changed  
 /// </summary>  
 private void OnBpmChange(object sender, RoutedEventArgs e)  
 {  
 if (\_isInitialising)  
 return;  
  
 var newBpm = (int)BpmSlider.Value;  
 Settings.Bpm = newBpm;  
 }  
  
 /// <summary>  
 /// Event which is called when the combo box's selected value changed  
 /// </summary>  
 private void OnProfileSelected(object sender, RoutedEventArgs e)  
 {  
 var selectedIndex = ProfileComboBox.SelectedIndex;  
  
 if (selectedIndex == -1)  
 return;  
  
 var selectedName = (string)ProfileComboBox.Items.GetItemAt(selectedIndex);  
  
 ChangeProfile(selectedName);  
 }  
  
 /// <summary>  
 /// Event which is called when the Reset to Defaults button is clicked  
 /// </summary>  
 private void OnResetSettingsClicked(object sender, RoutedEventArgs e)  
 {  
 var defaultSettings = StoredSettings.Default(); // Create new settings object with default settings  
   
 // Move profiles and logs to default location  
 MoveFolder(Settings.LogsPath, defaultSettings.LogsPath);   
 MoveFolder(Settings.ProfilesPath, defaultSettings.ProfilesPath);  
  
 (Application.Current as App).Settings = defaultSettings;  
 ShowSavedSettings();  
 }  
  
 #endregion  
  
 #region Prompts + Dialogs  
  
 /// <summary>  
 /// Function which opens a text dialog and returns the entered value  
 /// </summary>  
 /// <returns>The value entered into the prompt</returns>  
 private string OpenNamePrompt()  
 {  
 var dialog = new TextDialog("Enter Profile Name:");  
  
 if (dialog.ShowDialog() == true)  
 return dialog.Value;  
  
 return "";  
 }  
  
 /// <summary>  
 /// Open a File Dialog for the user to select a directory path  
 /// </summary>  
 /// <returns>The path selected</returns>  
 private string GetPath(string currentDirectory)  
 {  
 var dialog = new CommonOpenFileDialog();  
 dialog.InitialDirectory = currentDirectory;  
 dialog.IsFolderPicker = true;  
  
 if (dialog.ShowDialog() == CommonFileDialogResult.Ok) return dialog.FileName;  
  
 return string.Empty;  
 }  
  
 #endregion  
}

# Testing

## Manual Test Table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test ID** | **Corresponding Objective** | **Aim of Test** | **Test Data** | **Expected Output** | **Actual Output** |
| 1 |  | Test that a profile can be created, if none previously exist | (Any MIDI file) | The profile should be created. This involves creating a folder which contains a json file, as well as another directory called “training” which contains the selected file | The profile was created successfully |
| 2 |  | Test training screen (with erroneous data) | Min Epochs = 300  Max Epochs = 100  (This should create an error)  Max Error = default  Batches per epoch = default | Training should not begin, and the user should be displayed a message stating that the entered data is invalid. | The user receives a message stating “Minimum Epochs cannot be greater than Maximum Epochs” and training does not begin |
| 3 |  | Test training screen (with normal data) | Min Epochs = 50  Max Epochs = Maximum Possible  Max Error = 0.0142  Batches per Epoch = 10 | The application should train the network using the given settings, and provide a prompt once the training is completed | The application trained the network and a prompt was provided stating that the training was complete. |
| 4 |  | Test create screen (normal data) | Leave both values as default  Save in any directory which is accessible by the program (e.g. Documents) | The network should generate music which is stored in a midi file in the provided location. The midi file should be playable (and actually plays something). Once creation is complete, a prompt should be shown | Music was generated in a playable midi file at the provided path. A prompt was shown once creation was complete. |
| 5 |  | Test create screen (erroneous data) | Leave both values as default  Save in any directory which would not be accessible by the program (e.g. C:\Windows) | No creation should occur, and a prompt should be provided once the directory is selected. | A prompt was provided when the C:\Windows directory was selected. |
| 6 |  | Test Listen Screen.  Test that selected file will immediately start playing when there is nothing in the playlist | Open the program  Go to the listen screen  There should be nothing in the playlist since the program has just been opened.  Click Add to Playlist and select any sound file. | The current song label should update with the file name of the file and the file should begin playing. | The selected sound file immediately started playing, and the label updated as expected. |
| 7 |  | Test Listen Screen.  Test that a selected file will be added to the playlist when there is something currently playing. | Complete steps for step 6  Attempt to add 3 files | The file name of the first selected file should appear next to the “Next Song:” label.  The file name of the second and third files should appear under the “Playlist” label | The UI was updated as expected. |
| 8 |  | Test Listen Screen.  Test that the next song in the playlist will immediately begin playing once the currently playing file has ended | Open the program, queue 2 songs on the listen screen and wait for the first one to end | The second file should begin playing immediately after the first file has ended | The next file in the playlist began playing as expected. |
| 9 |  | Test Listen Screen buttons.  Test that the “previous song” button begins playing the last song that was played. | Open the program.  Select at least one file.  Once the first file has finished playing, press the left most button under the progress bar. | The first file that was selected should begin playing, and the queue should be unaltered | The first file began playing and the queue was unaltered |
| 10 |  | Test Listen Screen buttons.  Test that the previous button does not throw any errors when no songs have been played | Open the program.  Do not select any files.  Click the left most button under the progress bar | A message box should pop up stating “you haven’t played any songs recently” | The appropriate message was shown |
| 11 |  | Test Listen Screen buttons  Test that the skip button does not throw any errors when no songs have been played | Open the program  Do not select any files  Click the right most button under the progress bar | Nothing should happen | Nothing happened |
| 12 |  | Test Listen Screen buttons  Test that the playback buttons (rewind, fast forward, play/pause and stop buttons) do not throw any errors when no songs are being played | Open the program  Do not select any files  Click all the buttons except the first and last buttons. | Nothing should happen | Nothing happened |
| 13 |  | Test Listen Screen buttons.  Test that the rewind button functions as expected during playback (when the file has been playing for at least 5 seconds) | Open the program  Select a file  Wait 10 seconds  Press the second button from the left | The file should continue playing, but from 5 seconds behind where it is | The file rewinds 5 seconds as expected |
| 14 |  | Test Listen Screen buttons.  Test that the rewind button functions as expected during playback (when the file has been playing for less than 5 seconds) | Open the program  Select a file  Wait 2 seconds  Press the second button from the left | The file should begin playing from the beginning | The file plays from the beginning as expected |
| 15 |  | Test Listen Screen buttons.  Test that the fast forward button functions as expected during playback (when there is more than 5 seconds until the end of the file) | Open the program  Select a file  Wait 10 seconds  Press the second button from the right | The file should continue playing, but from 5 seconds after where it is | The file rewinds 5 seconds as expected |
| 16 |  | Test Listen Screen buttons.  Test that the rewind button functions as expected during playback (when the file has less than 5 seconds left) | Open the program  Select a file  Wait 2 seconds  Press the second button from the right | The file should stop playing as the end has been reached | The file stops playing as expected |
| 17 |  | Test Listen Screen buttons  Test the pause/play button | Open the program  Select a file  Press the third button from the left  Wait 3 seconds  Press the button again | The first time the button is pressed, the playback should stop, but the next song should not be played. The button should also change from a pause button to a play button.  The second time the button is pressed, playback should resume from the same position from which it was paused. The button should also change from a play button to a pause button | The button changes, and affects the playback of the file as expected |
| 18 |  | Test Listen Screen buttons  Test the stop button | Open the program  Select 5 files  Press the third button from the right | The playlist should be cleared, and the current song should stop playing |  |
| 19 |  | Test that a profile can be created and saved from the settings page | Open the program  Create default profile (if required)  Go to the settings page  Create a new profile  Close the program  Reopen the program | The newly created profile should be selected by default and present in the combo box | Changes as expected |
| 20 |  | Test that the logs path can be successfully changed | Open the program  Change the logs folder from the settings menu  Reopen the program | The existing logs files should be moved to the new location, where the new log file should also be created in the new path | Existing logs copied over, but the new log file |
| 21 |  | **Test 20 Retest** | **Test 20 Retest** | **Test 20 Retest** | Existing logs copied over, new log file is created in the correct place this time. |
| 21 |  | Test that the profiles path can be successfully changed | Open the program  Change the profiles path from the settings menu  Create a new profile | The newly created, and the already existing, profiles should be present in the new path | Existing profiles are successfully copied over, and the new profile was created in the new path |
| 22 |  | Test that the default minimum epochs can be successfully changed | Open the program  Change the minimum epochs slider in the settings page  Close the program and check the settings.json file | The minimum epoch value in the json file should match the value that it was changed to in the UI | The value in settings.json reflected the change made via the UI |
| 23 |  | Test that the default maximum epochs can be successfully changed | Open the program  Change the maximum epochs slider in the settings page  Close the program and check the settings.json file | The maximum epoch value in the json file should match the value that it was changed to in the UI | The value in settings.json reflected the change made via the UI |
| 24 |  | Test that invalid min and max epochs cannot be entered on the settings page | Open the program  Try to change the minimum epochs and maximum epochs slider such that the value on the minimum epoch slider >= the value on the maximum epoch slider | When the illegal change is being made, a message box should appear on the screen telling the user that this is invalid.  The illegal change should be reverted. | The relevant message box was shown when the user attempted to make an invalid change |
| 25 |  | Test that the default maximum error can be successfully changed | Open the program  Change the maximum error slider in the settings page  Close the program and check the settings.json file | The maximum error value in the json file should match the value that it was changed to in the UI | The value in settings.json reflected the change made via the UI |
| 26 |  | Test that the default batches per epoch can be successfully changed | Open the program  Change the batches per epoch slider in the settings page  Close the program and check the settings.json file | The batches per epoch value in the json file should match the value that it was changed to in the UI | The value in settings.json reflected the change made via the UI |
| 27 |  | Test that the default generation length can be successfully changed | Open the program  Change the generation length slider in the settings page  Close the program and check the settings.json file | The generation length value in the json file should match the value that it was changed to in the UI | The value in settings.json reflected the change made via the UI |
| 28 |  | Test that the default beats per minute can be successfully changed | Open the program  Change the beats per minute slider in the settings page  Close the program and check the settings.json file | The BPM value in the json file should match the value that it was changed to in the UI | The value in settings.json reflected the change made via the UI |
| 29 |  | Test that the edited settings are correctly saved and loaded | Open the program  Change the default settings  Close and reopen the program and go to the settings page | The settings page should still show the values that the default settings were changed to before the program was closed | The settings page when the page was reopened had the same values as the settings page after the changes were made and before the program was shut |
| 30 |  | Test that the default settings are correctly displayed on the generation page | Open the program  Change the default generation length and bpm.  Go to the generation page | The values shown by the sliders on the generation page when it is opened, should be the same as the default values in the settings | The changed default settings were shown by default on the generation page |
| 31 |  | Test that the default settings are correctly displayed on the generation page | Open the program  Change the default generation length and bpm.  Close and reopen the program  Go to the generation page | The values shown by the sliders on the generation page when it is opened, should be the same as the default values in the settings | The changed default settings were shown by default on the generation page even after closing |
| 32 |  | Test that the default settings are correctly displayed on the training page | Open the program  Change the default minimum and maximum epochs, maximum error and batches per epoch  Go to the training page | The values shown by the sliders on the training page when it is opened, should be the same as the default values in the settings | The changed default settings were shown by default on the training page |
| 33 |  | Test that the default settings are correctly displayed on the training page | Open the program  Change the default minimum and maximum epochs, maximum error and batches per epoch  Close and reopen the program  Go to the training page | The values shown by the sliders on the training page when it is opened, should be the same as the default values in the settings | The changed default settings were shown by default on the training page, even after closing the application |

## Unit Tests

Unit Tests were created and carried out using C#’s XUnit Library.

### Apollo.MatrixMaths.Tests

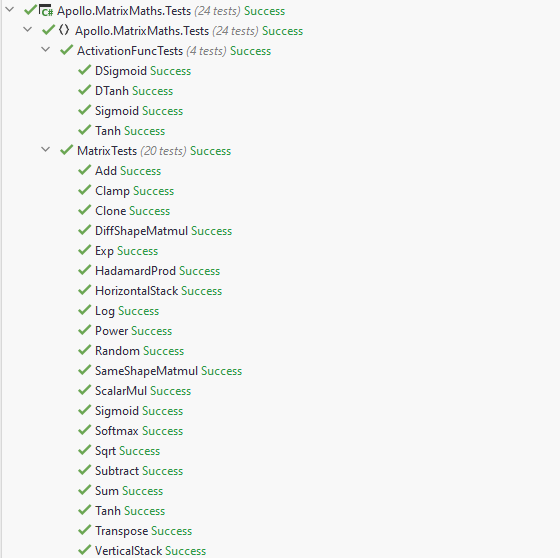
#### MatrixTests.cs Source Code

namespace Apollo.MatrixMaths.Tests;  
  
public class MatrixTests  
{  
 /// <summary>  
 /// Test data used in the majority of the unit tests  
 /// </summary>  
 private readonly float[,] defaultData =  
 {  
 { 0, 1, 4 },  
 { 9, 18, 99 },  
 { 1000, 2500, 1990 }  
 };  
  
 private readonly float[,] defaultData2 =  
 {  
 { 24, 3, 12 },  
 { 1, 0, -5 },  
 { -25, 1, 22 }  
 };  
  
 [Fact]  
 // Tests that the power operation is correctly performed  
 // i.e. each item in the matrix is taken and put to the specified power separately   
 public void Power()  
 {  
 float[,] expectedData =  
 {  
 { 0, 1, 16 },  
 { 81, 324, 9801 },  
 { 1000000, 6250000, 3960100 }  
 };  
  
 var actualMatrix = new Matrix(defaultData);  
 actualMatrix.Power(2);  
 var actualData = actualMatrix.Contents;  
  
 Assert.Equal(expectedData, actualData);  
 }  
  
 [Fact]  
 // Tests that the square root operation is correctly performed  
 // i.e. the square root of each item in the matrix is found separately  
 public void Sqrt()  
 {  
 var expectedData = new[,]  
 {  
 { 0, 1, 2 },  
 { 3, 4.2426406871192848f, 9.9498743710662f },  
 { 31.622776601683793f, 50, 44.609416046390926f }  
 };  
  
 var actualMatrix = new Matrix(defaultData);  
 actualMatrix.Sqrt();  
 var actualData = actualMatrix.Contents;  
  
 Assert.Equal(expectedData, actualData);  
 }  
  
 [Fact]  
 // Tests that the exp operation is correctly performed  
 // i.e. Euler's number is put to the power of each item in the matrix separately   
 public void Exp()  
 {  
 var expectedData = new[,]  
 {  
 { 1, 2.71828175f, 54.5981483f },  
 { 8103.08398f, 65659968, float.PositiveInfinity },  
 { float.PositiveInfinity, float.PositiveInfinity, float.PositiveInfinity }  
 };  
  
 var actualMatrix = new Matrix(defaultData);  
 actualMatrix.Exp();  
 var actualData = actualMatrix.Contents;  
  
 Assert.Equal(expectedData, actualData);  
 }  
  
 [Fact]  
 // Tests that the tanh operation is correctly performed  
 // i.e. tanh is applied to each item in the matrix separately   
 public void Tanh()  
 {  
 var expectedData = new[,]  
 {  
 { 0, 0.761594176f, 0.999329329f },  
 { 1, 1, 1 },  
 { 1, 1, 1 }  
 };  
  
 var actualMatrix = new Matrix(defaultData);  
 actualMatrix.Tanh();  
 var actualData = actualMatrix.Contents;  
  
 Assert.Equal(expectedData, actualData);  
 }  
  
 [Fact]  
 // Tests that standard matrix multiplication is correctly applied to matrices of the same shape  
 public void SameShapeMatmul()  
 {  
 var expectedData = new float[,]  
 {  
 { -99, 4, 83 },  
 { -2241, 126, 2196 },  
 { -23250, 4990, 43280 }  
 };  
  
 var defaultMat1 = new Matrix(defaultData);  
 var defaultMat2 = new Matrix(defaultData2);  
 defaultMat1.Multiply(defaultMat2);  
 var actualData = defaultMat1.Contents;  
  
 Assert.Equal(expectedData, actualData);  
 }  
  
 [Fact]  
 // Tests that standard matrix multiplication is correctly applied to matrices of different but valid shapes  
 public void DiffShapeMatmul()  
 {  
 var expectedData = new[,]  
 {  
 { -7f },  
 { -1f }  
 };  
  
 var mat1Data = new[,]  
 {  
 { 1f, -2f },  
 { 3f, 4f }  
 };  
  
 var mat2Data = new[,]  
 {  
 { -3f },  
 { 2f }  
 };  
  
 var mat1 = new Matrix(mat1Data);  
 var mat2 = new Matrix(mat2Data);  
 var result = Matrix.Multiply(mat1, mat2);  
 var actualData = result.Contents;  
  
 Assert.Equal(expectedData, actualData);  
 }  
  
 [Fact]  
 // Tests that multiplying a matrix by a scalar works as expected  
 // i.e. each item in the matrix is independently multiplied by the scalar quantity   
 public void ScalarMul()  
 {  
 var scalar = 2;  
 var expectedData = new float[,]  
 {  
 { 0, 2, 8 },  
 { 18, 36, 198 },  
 { 2000, 5000, 3980 }  
 };  
  
 var defaultMat = new Matrix(defaultData);  
 defaultMat.Multiply(scalar);  
 var actualData = defaultMat.Contents;  
  
 Assert.Equal(expectedData, actualData);  
 }  
  
 [Fact]  
 // Checks that summing the elements in the matrix works as expected  
 public void Sum()  
 {  
 var expectedData = 5621.0;  
  
 var defaultMat = new Matrix(defaultData);  
 var actualData = defaultMat.Sum();  
  
 Assert.Equal(expectedData, actualData);  
 }  
  
 [Fact]  
 // Checks that the transpose operation works as expected  
 // i.e. the first row becomes the first column, and the second row becomes the second column, etc.  
 public void Transpose()  
 {  
 var startingData = new float[,]  
 {  
 { 0, 1, 4 },  
 { 9, 18, 99 }  
 };  
  
 var expectedData = new float[,]  
 {  
 { 0, 9 },  
 { 1, 18 },  
 { 4, 99 }  
 };  
  
 var startMat = new Matrix(startingData);  
 startMat.Transpose();  
 var actualData = startMat.Contents;  
  
 Assert.Equal(expectedData, actualData);  
 }  
  
 [Fact]  
 // Checks that the clamp operation works as expected  
 // i.e. all the elements in the matrix after the operation should be in the interval [min, max]  
 public void Clamp()  
 {  
 var min = 2;  
 var max = 10;  
  
 var expectedData = new float[,]  
 {  
 { 2, 2, 4 },  
 { 9, 10, 10 },  
 { 10, 10, 10 }  
 };  
  
 var defaultMat = new Matrix(defaultData);  
 defaultMat.Clamp(min, max);  
 var actualData = defaultMat.Contents;  
  
 Assert.Equal(expectedData, actualData);  
 }  
  
 [Fact]  
 // Checks that matrices can be stacked horizontally  
 // i.e. pushed next to one another   
 public void HorizontalStack()  
 {  
 var expectedData = new float[,]  
 {  
 { 0, 1, 4, 24, 3, 12 },  
 { 9, 18, 99, 1, 0, -5 },  
 { 1000, 2500, 1990, -25, 1, 22 }  
 };  
  
 var defaultMat1 = new Matrix(defaultData);  
 var defaultMat2 = new Matrix(defaultData2);  
 defaultMat1.HorizontalStack(defaultMat2);  
 var actualData = defaultMat1.Contents;  
  
 Assert.Equal(expectedData, actualData);  
 }  
  
 [Fact]  
 // Checks that matrices can be stacked vertically  
 // i.e. stacked one on top of the other   
 public void VerticalStack()  
 {  
 var expectedData = new float[,]  
 {  
 { 0, 1, 4 },  
 { 9, 18, 99 },  
 { 1000, 2500, 1990 },  
 { 24, 3, 12 },  
 { 1, 0, -5 },  
 { -25, 1, 22 }  
 };  
  
 var defaultMat1 = new Matrix(defaultData);  
 var defaultMat2 = new Matrix(defaultData2);  
 defaultMat1.VerticalStack(defaultMat2);  
 var actualData = defaultMat1.Contents;  
  
 Assert.Equal(expectedData, actualData);  
 }  
  
 [Fact]  
 // Tests that matrices can be randomly generated   
 public void Random()  
 {  
 var mat1R = new Random(5);  
 var mat2R = new Random(120);  
  
 var mat1 = new Matrix(5, 5, mat1R);  
 var mat2 = new Matrix(5, 5, mat2R);  
  
 Assert.NotEqual(mat1.Contents, mat2.Contents);  
 }  
  
 [Fact]  
 // Tests that two matrices can be added together correctly  
 // (Addition is element-wise)   
 public void Add()  
 {  
 var expectedData = new float[,]  
 {  
 { 24, 4, 16 },  
 { 10, 18, 94 },  
 { 975, 2501, 2012 }  
 };  
  
 var mat1 = new Matrix(defaultData);  
 var mat2 = new Matrix(defaultData2);  
 mat1.Add(mat2);  
 var actualData = mat1.Contents;  
  
 Assert.Equal(expectedData, actualData);  
 }  
  
 [Fact]  
 // Tests that two matrices can be subtracted correctly  
 // (Subtraction is element-wise)  
 public void Subtract()  
 {  
 var expectedData = new float[,]  
 {  
 { -24, -2, -8 },  
 { 8, 18, 104 },  
 { 1025, 2499, 1968 }  
 };  
  
 var mat1 = new Matrix(defaultData);  
 var mat2 = new Matrix(defaultData2);  
 mat1.Subtract(mat2);  
 var actualData = mat1.Contents;  
  
 Assert.Equal(expectedData, actualData);  
 }  
  
 [Fact]  
 // Tests that the sigmoid function is correctly applied to the matrix  
 // i.e. the sigmoid function is carried out on each item in the matrix separately   
 public void Sigmoid()  
 {  
 var expectedData = new[,]  
 {  
 { 0.5f, 0.731058598f, 0.982013762f },  
 { 1, 1, 1 },  
 { 1, 1, 1 }  
 };  
  
 var defaultMatrix = new Matrix(defaultData);  
 defaultMatrix.Sigmoid();  
 var actualData = defaultMatrix.Contents;  
  
 Assert.Equal(expectedData, actualData);  
 }  
  
 [Fact]  
 // Tests element-wise multiplication is correctly carried out  
 public void HadamardProd()  
 {  
 var expectedOutput = new[,]  
 {  
 { 3f, 2f, 25f },  
 { 54f, 130f, 36f },  
 { 16f, 352f, 132f }  
 };  
  
 var matContent1 = new[,]  
 {  
 { 1f, 0.5f, 5f },  
 { 2f, 13f, 12f },  
 { 4f, 22f, 11f }  
 };  
  
 var matContent2 = new[,]  
 {  
 { 3f, 4f, 5f },  
 { 27f, 10f, 3f },  
 { 4f, 16f, 12f }  
 };  
  
 var mat1 = new Matrix(matContent1);  
 var mat2 = new Matrix(matContent2);  
 var mat1x2 = Matrix.HadamardProd(mat1, mat2);  
 var mat2x1 = Matrix.HadamardProd(mat2, mat1);  
  
 Assert.Equal(expectedOutput, mat1x2.Contents); // Check if right answer obtained   
 Assert.Equal(expectedOutput, mat2x1.Contents); // Check if order matters  
 }  
  
 [Fact]  
 // Tests the softmax activation function is correctly carried out on the matrix  
 public void Softmax()  
 {  
 var data = new[,]  
 {  
 { 0.59474933f, 0.6371081f, 0.62671316f, 0.7615537f }  
 };  
  
 var expectedOutput = new[,]  
 {  
 { 0.23489222f, 0.2450557f, 0.24252155f, 0.27753058f }  
 };  
  
 var matrix = new Matrix(data);  
 matrix.Softmax();  
 var actualOutput = matrix.Contents;  
  
 Assert.Equal(expectedOutput, actualOutput);  
 }  
  
 [Fact]  
 // Tests logarithms are correctly performed on the matrix  
 // i.e. the logarithm should be carried out on each element separately   
 public void Log()  
 {  
 var logBase = 5;  
  
 var data = new[,]  
 {  
 { 2f, 5.5f },  
 { 25f, 125.7556f }  
 };  
 var expectedOutput = new[,]  
 {  
 { 0.43067655f, 1.0592195f },  
 { 2f, 3.003744539f }  
 };  
  
 var matrix = new Matrix(data);  
 matrix.Log(logBase);  
 var actualOutput = matrix.Contents;  
  
 Assert.Equal(expectedOutput, actualOutput);  
 }  
  
 [Fact]  
 // Tests that the matrix can be cloned properly   
 // i.e. the clone should not be affected when the original is changed   
 public void Clone()  
 {  
 var mat1 = new Matrix(defaultData);  
 var mat2 = mat1.Clone();  
  
 mat1.Add(mat2);  
  
 Assert.False(mat1.Contents == mat2.Contents);  
 }  
}

#### ActivationFuncTests.cs Source Code

namespace Apollo.MatrixMaths.Tests;  
  
public class ActivationFuncTests  
{  
 /// <summary>  
 /// Test data used for the unit tests  
 /// </summary>  
   
 // Boundary Data  
 private readonly float \_sigmoidBoundary = 6.0001f;  
 private readonly float \_tanhBoundary = 4.0001f;  
   
 // Normal Data  
 private readonly float \_testData = 3;   
  
  
 [Fact]  
 // Tests tanh is found correctly for a single value  
 public void Tanh()  
 {  
 var expectedData = 0.9950547537f; // Expected data for "testData"  
 var actualData = ActivationFuncs.Tanh(\_testData); // Actual data for "testData"  
 Assert.Equal(expectedData, actualData);  
  
 // Boundary tests   
 Assert.Equal(1, ActivationFuncs.Tanh(\_tanhBoundary));  
 Assert.Equal(-1, ActivationFuncs.Tanh(-\_tanhBoundary));  
 }  
  
 [Fact]  
 // Test the derivative of tanh is correctly found for a given value   
 public void DTanh()  
 {  
 var expectedData = 0.009866037165f;  
 var actualData = ActivationFuncs.DTanh(\_testData);  
 Assert.Equal(expectedData, actualData);  
  
 // Boundary tests   
 Assert.Equal(0, ActivationFuncs.DTanh(\_tanhBoundary));  
 Assert.Equal(0, ActivationFuncs.DTanh(-\_tanhBoundary));  
 }  
  
 [Fact]  
 // Test the sigmoid function is correctly found for a given value   
 public void Sigmoid()  
 {  
 var expectedData = 0.9525741268f;  
 var actualData = ActivationFuncs.Sigmoid(\_testData);  
 Assert.Equal(expectedData, actualData);  
  
 // Boundary Tests   
 Assert.Equal(1, ActivationFuncs.Sigmoid(\_sigmoidBoundary));  
 Assert.Equal(0, ActivationFuncs.Sigmoid(-\_sigmoidBoundary));  
 }  
  
 [Fact]  
 // Test the derivative of the sigmoid function is correctly found for a given value   
 public void DSigmoid()  
 {  
 var expectedData = 0.045176655f;  
 var actualData = ActivationFuncs.DSigmoid(\_testData);  
 Assert.Equal(expectedData, actualData);  
  
 // Boundary Tests   
 Assert.Equal(0, ActivationFuncs.DSigmoid(\_sigmoidBoundary));  
 Assert.Equal(0, ActivationFuncs.DSigmoid(-\_sigmoidBoundary));  
 }  
}

#### Test Results

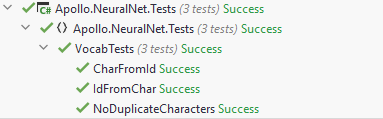


### Apollo.NeuralNet.Tests

#### VocabTests.cs Source Code

namespace Apollo.NeuralNet.Tests;  
  
public class VocabTests  
{  
 // The vocab list which is used to instantiate the vocab class for testing  
 private readonly List<char> \_testingVocabList = new() { 'a', 'b', 'c', 'd', 'e' };  
  
 [Fact]  
 // Testing that vocab gives the correct character given an ID  
 public void CharFromId()  
 {  
 var vocab = new Vocab(\_testingVocabList);  
  
 Assert.Equal('a', vocab[0]);  
 Assert.Equal('b', vocab[1]);  
 Assert.Equal('c', vocab[2]);  
 Assert.Equal('d', vocab[3]);  
 Assert.Equal('e', vocab[4]);  
 }  
  
 [Fact]  
 // Testing that vocab gives the correct ID given a character  
 public void IdFromChar()  
 {  
 var vocab = new Vocab(\_testingVocabList);  
  
 Assert.Equal(0, vocab['a']);  
 Assert.Equal(1, vocab['b']);  
 Assert.Equal(2, vocab['c']);  
 Assert.Equal(3, vocab['d']);  
 Assert.Equal(4, vocab['e']);  
 }  
  
 [Fact]  
 // Testing that AddCharacter() does not add duplicate characters.  
 public void NoDuplicateCharacters()  
 {  
 var vocab = new Vocab(\_testingVocabList);  
  
 vocab.AddCharacter('e');  
 vocab.AddCharacter('f');  
  
 // If the duplicate e was added, vocab[5] would be e  
 Assert.Equal('f', vocab[5]);  
 }  
}

#### Test Results



### Apollo.Tests

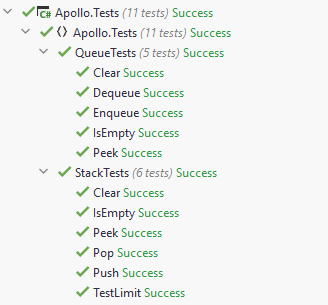
#### QueueTests Source Code

namespace Apollo.Tests;  
  
public class QueueTests  
{  
 // Helper subroutine to create queue for testing   
 public Queue<int> MakeQueue(int numItems)  
 {  
 var queue = new Queue<int>();  
   
 for (var i = 0; i < numItems; i++)   
 queue.Enqueue(i);  
  
 return queue;  
 }  
  
 [Fact]  
 // Test that items can be added to the queue  
 public void Enqueue()  
 {  
 var queue = new Queue<int>();  
   
 queue.Enqueue(1);  
 queue.Enqueue(2);   
 queue.Enqueue(3);  
  
 Assert.Equal(1, queue.Dequeue());  
 Assert.Equal(2, queue.Dequeue());  
 Assert.Equal(3, queue.Dequeue());  
 }  
  
 [Fact]  
 // Test that the next item to be dequeued is returned from peek  
 public void Peek()  
 {  
 var queue = MakeQueue(5);  
   
 Assert.Equal(0, queue.Peek());  
 queue.Dequeue();  
 Assert.Equal(1, queue.Peek());  
 queue.Dequeue();  
 Assert.Equal(2, queue.Peek());  
 }  
  
 [Fact]  
 // Test that the first item to be added is returned from dequeue  
 public void Dequeue()  
 {  
 var queue = MakeQueue(5);  
   
 Assert.Equal(0, queue.Dequeue());  
 Assert.Equal(1, queue.Dequeue());  
 Assert.Equal(2, queue.Dequeue());  
 Assert.Equal(3, queue.Dequeue());  
 Assert.Equal(4, queue.Dequeue());  
 }  
  
 [Fact]  
 // Test that queue correctly identifies whether or not it is empty   
 public void IsEmpty()  
 {  
 var queue = new Queue<int>();  
 Assert.True(queue.IsEmpty());  
 queue = MakeQueue(5);  
 Assert.False(queue.IsEmpty());  
 }  
  
 [Fact]  
 // Test that the queue is correctly cleared   
 public void Clear()  
 {  
 var queue = MakeQueue(5);  
 Assert.False(queue.IsEmpty());  
 queue.Clear();  
 Assert.True(queue.IsEmpty());  
 }  
}

#### StackTests Source Code

using Apollo;  
  
namespace Apollo.Tests;  
  
public class StackTests  
{  
 // Helper function to create stack to use for testing  
 private Stack<int> MakeStack(int maxItems)  
 {  
 // Stack will contain incrementing numbers   
 var stack = new Stack<int>(maxItems);  
  
 for (var i = 0; i < maxItems; i++)  
 stack.Push(i);  
  
 return stack;  
 }  
  
 [Fact]  
 // Test items can be pushed to the stack  
 public void Push()  
 {  
 var stack = new Stack<int>(3);  
  
 stack.Push(1);  
 Assert.Equal(1, stack.Peek());  
  
 stack.Push(2);  
 Assert.Equal(2, stack.Peek());  
  
 stack.Push(3);  
 Assert.Equal(3, stack.Peek());  
 }  
  
 [Fact]  
 // Test the top item is returned by peek  
 public void Peek()  
 {  
 var stack = MakeStack(5);  
 var expected = 4;  
 Assert.Equal(expected, stack.Peek());  
 }  
   
 [Fact]  
 // Test items can be removed from the stack  
 public void Pop()  
 {  
 var stack = MakeStack(5);  
 var expected = 4;  
 Assert.Equal(expected, stack.Pop());  
 Assert.NotEqual(expected, stack.Peek());  
 }  
  
 [Fact]  
 // Check that the stack correctly identifies if it is empty   
 public void IsEmpty()  
 {  
 var stack = new Stack<int>(2);  
 Assert.True(stack.IsEmpty());  
   
 stack = MakeStack(5);  
 Assert.False(stack.IsEmpty());  
 }  
  
 [Fact]  
 // Check that the contents of the stack can be cleared  
 public void Clear()  
 {  
 var stack = MakeStack(5);   
 Assert.False(stack.IsEmpty());  
 stack.Clear();  
 Assert.True(stack.IsEmpty());  
 }  
  
 [Fact]  
 // Test that only a certain amount of items can be added to the stack  
 public void TestLimit()  
 {  
 var stack = MakeStack(10);  
  
 for (var i = 0; i < 10; i++)  
 stack.Push(i);  
   
 stack.Push(10);  
   
 Assert.NotEqual(10, stack.Peek());  
 }  
}

#### Test Results



## Video Contents Table

|  |  |  |
| --- | --- | --- |
| **Corresponding Objective** | **Contents of Video** | **Video URL** |
| N/A | Full Cycle | <https://youtu.be/4rpOZ7QyHjA> |
| 2.1 / 2.2 / 2.3 / 2.4 / 2.5 / 2.7 2.8 | Profile testing | <https://youtu.be/TxWDkeswJxw> |
| 2.6 | Training Page | <https://youtu.be/agxDINyq8gk> |
| 4.5 / 4.6 | Error Handling & Logging | <https://youtu.be/rMG_IMOpuz8> |

# Evaluation

## User Questionnaire

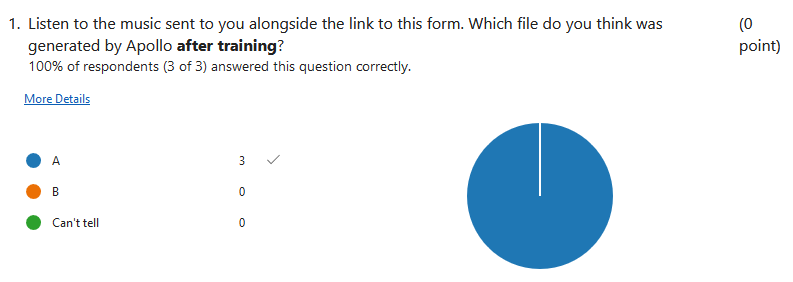
I asked 3 users to try to use the various components of the program before answering the corresponding section of the questionnaire. The main theme of the questionnaire is about “readability” and “ease of use” (Objective 4.1)

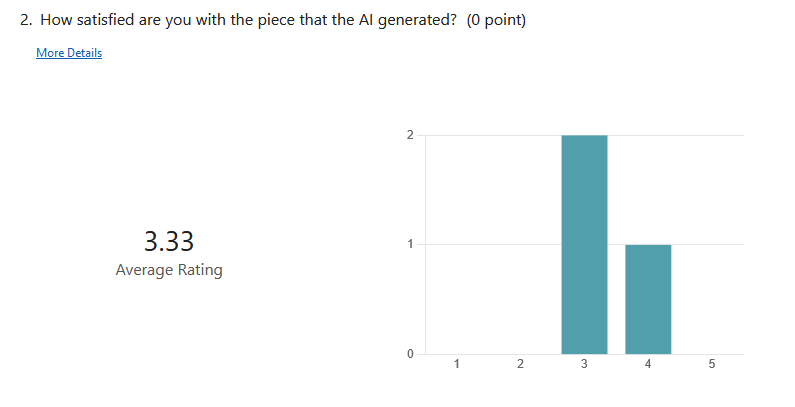
### Section 1: Generation Quality

#### Synopsis

This section was used to confirm that training the network, in fact, makes a difference to the quality of the generated sound. This section asked the users to listen to two pieces of music generated by the network. One which was generated prior to training and one which was generated after training. The users were asked to decide which one they think was generated after training. They were then asked about their satisfaction with the piece, after listening to what it was trained with.

#### Responses





#### Evaluation

This section confirms that the neural network can be trained successfully, since the end users were easily able to identify which piece was generated after training. I’m not surprised nor disappointed that none of the 3 users rated the piece a 5 out of 5 considering that the machine learning model is very small compared to that of AIVA nor does it have the same level of customisability. Given the time frame of the project, and my lack of real machine learning experience before this project, it was certain that I would not create a network of that nature.

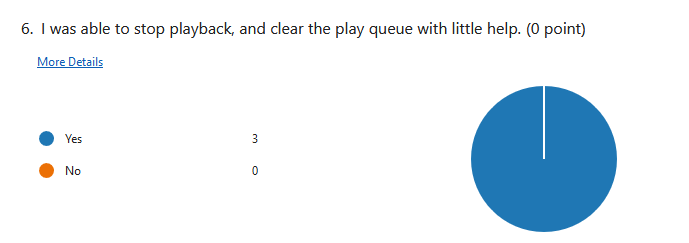
### Section 2: Listen Page

#### Synopsis

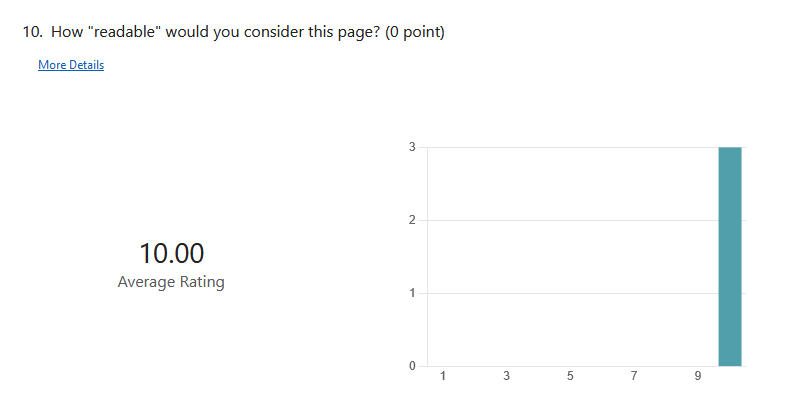
This section asks the user about their experience with the listen page (Objective 4.3). Before answering this section, the users were individually shown the listen page and are asked to: queue sound files to the playlist; stop the playback and clear the queue; play and pause the music; rewind and fast forward and skip and replay.

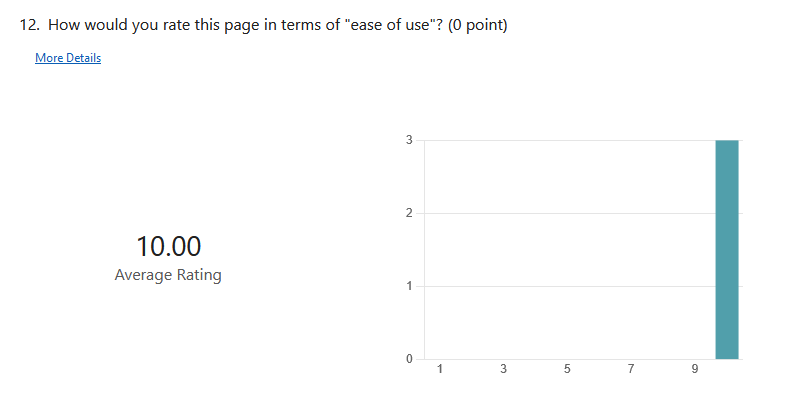
After this, the users were asked to rate the “ease of use” and “readability” of the page out of 10 as well as how their scores could be improved.

#### Responses









#### Evaluation

Again, these responses are not too surprising. This is because this page used icons that would be recognisable to the users since it is very likely that they have used a music player (e.g. Spotify or Apple Music) before.

### Section 3: Training Page

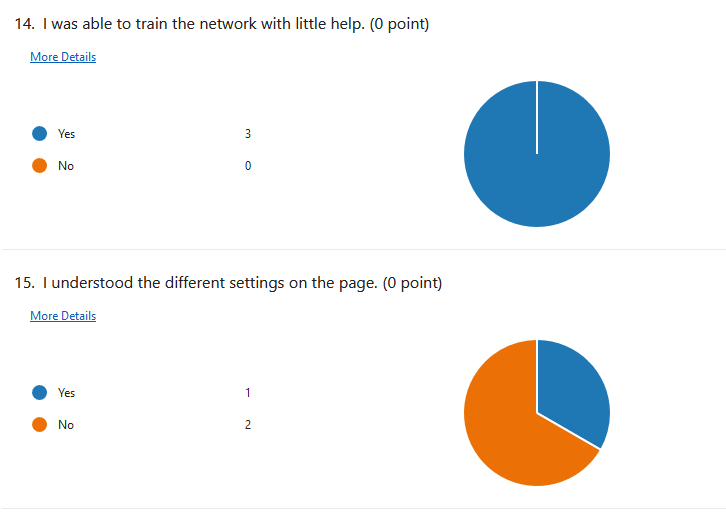
#### Synopsis

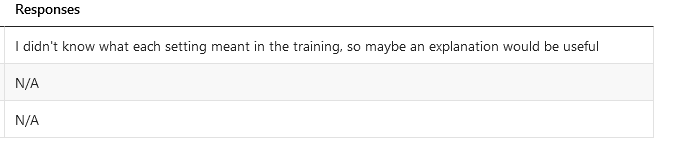
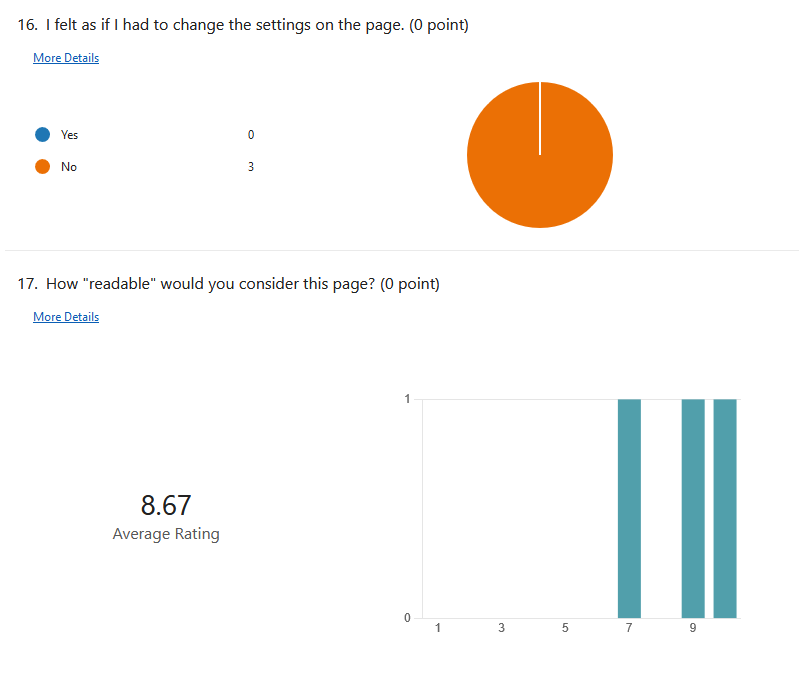
This section asks the user about their experience with the training page (Objectives 2.4, 2.6). Before answering this section, the users were individually shown the training page and were asked: if they were able to train the network; if they understood the different settings displayed to them on this page and if they felt obligated to change them.

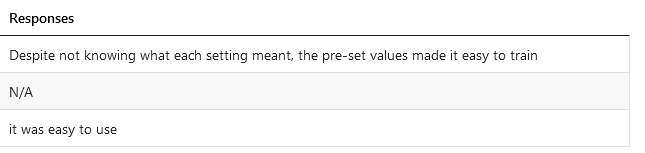
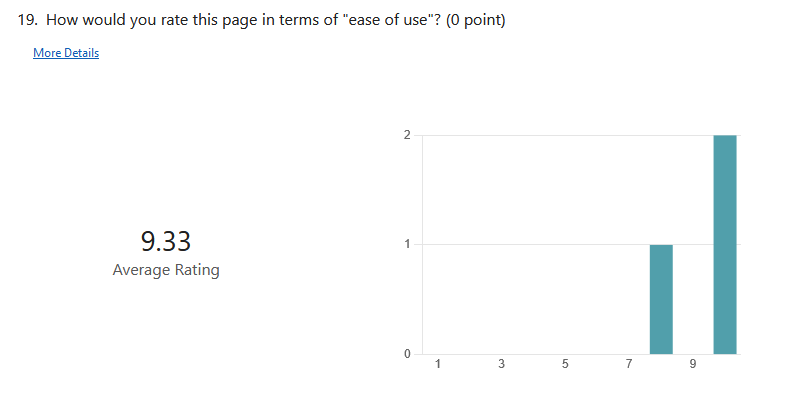
The last question is necessary since when making the questionnaire I doubted the ability of the majority of the users to understand the different settings. However, this doesn’t necessarily matter as long as they do not feel forced into changing any of them since this would not harm the ease of use of the page.

After this, the users were asked to rate the “ease of use” and “readability” of the page out of 10 as well as how their scores could be improved.

#### Responses







#### Evaluation

All of the users managed to use the page with no problems. Although one user did not understand the settings and was their main criticism of this page of the program, this is not a problem because they didn’t feel obliged to change them. Perhaps an explanation would improve the page but since the settings are supposed to be for more “advanced users”, I would say that a dropdown menu to hide them would be better suited.

### Section 4: Generation Page

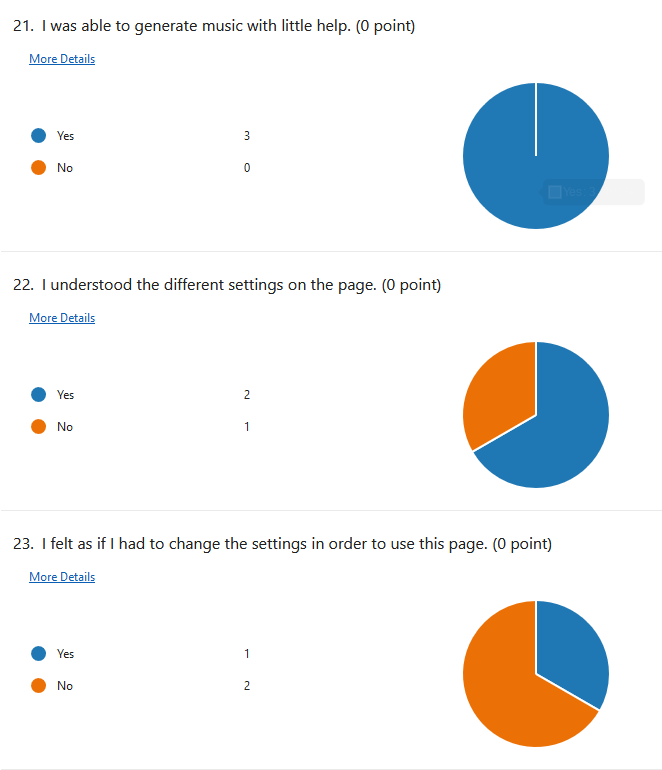
#### Synopsis

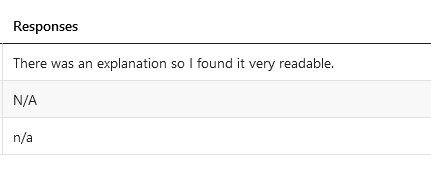
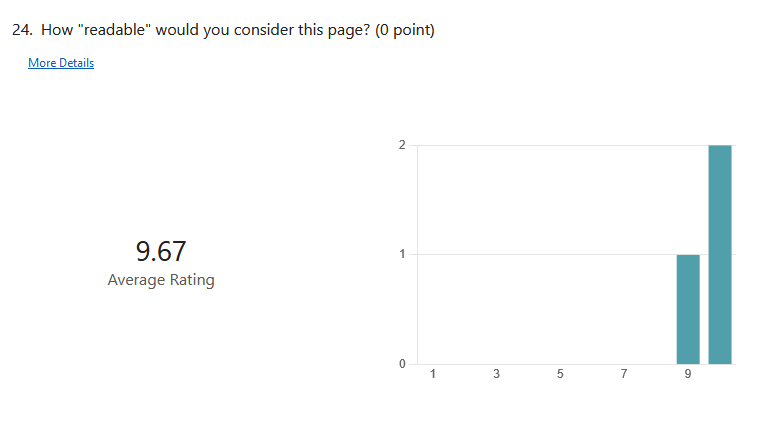
This section asks the user about their experience with the generation page (Objective 4.2). Before answering this section, the users were individually shown the generation page and were asked: if they were able to generate music with the page; if they understood the settings displayed to them on this page and if they felt obligated to change them.

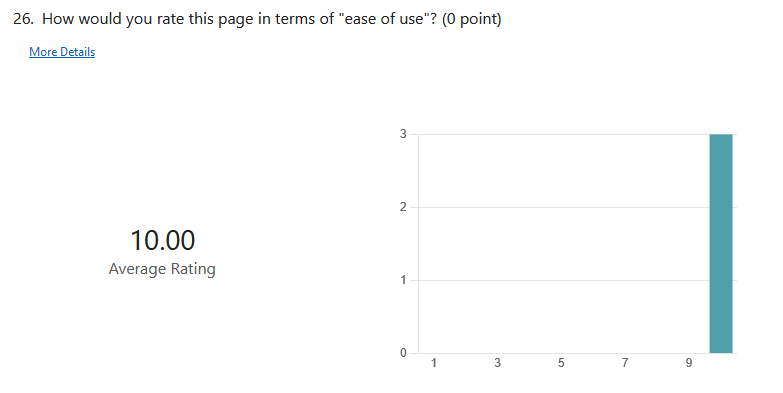
The last question is necessary since when making the questionnaire I doubted the ability of the majority of the users to understand the different settings. However, this doesn’t necessarily matter as long as they do not feel forced into changing any of them since this would not harm the ease of use of the page.

After this, the users were asked to rate the “ease of use” and “readability” of the page out of 10 as well as how their scores could be improved.

#### Responses







#### Evaluation

This page also received a flawless response. I anticipated this would be the case since, as compared to the training page, the settings were much easier for someone who does not have any prior knowledge of machine learning to understand.

### Section 5: Settings Page

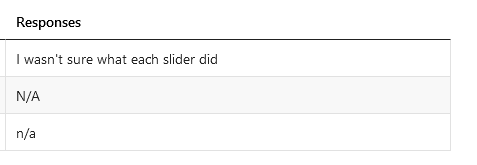
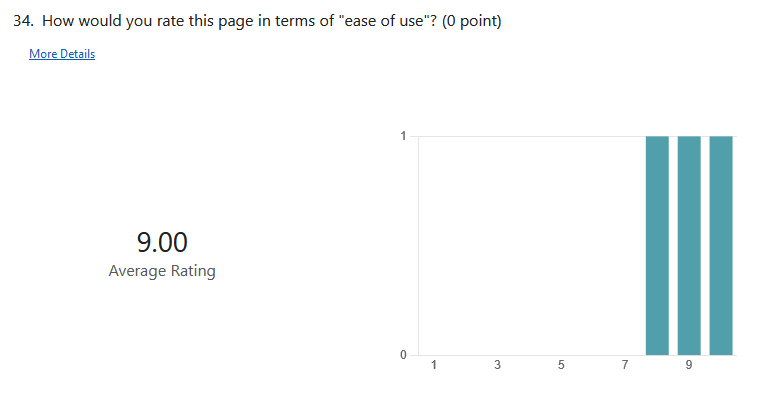
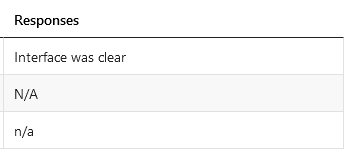
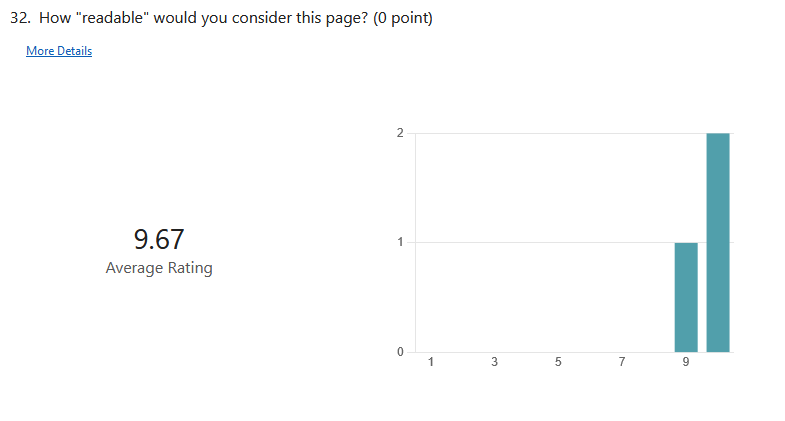
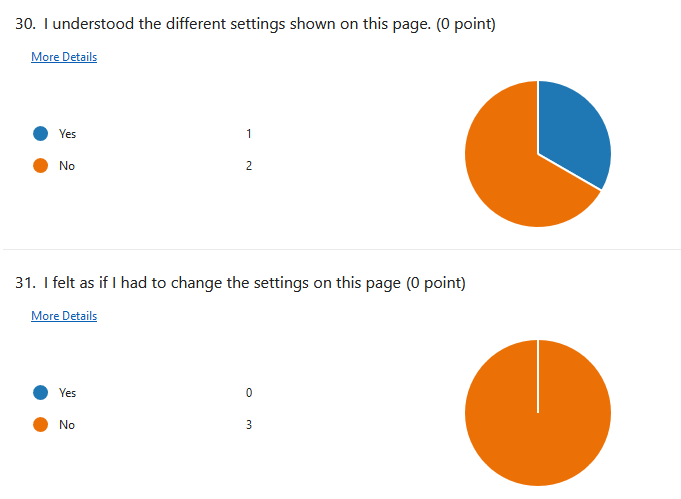
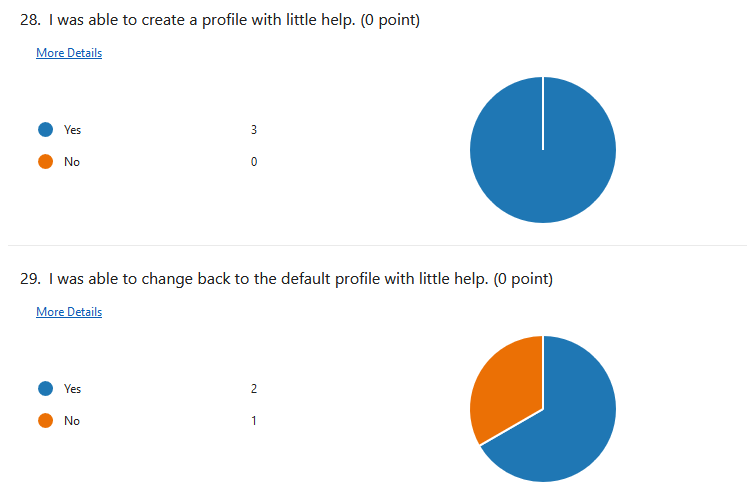
#### Synopsis

This section asks the user about their experience with the settings page (Objective 4.4). Before answering this section, the users were individually shown the settings page and were asked: if they were able to create a new profile, and then switch back to the default profile; if they understood the different settings displayed to them on this page and if they felt obligated to change them.

The last question is necessary since when making the questionnaire I doubted the ability of the majority of the users to understand the different settings. However, this doesn’t necessarily matter as long as they do not feel forced into changing any of them since this would not harm the ease of use of the page.

After this, the users were asked to rate the “ease of use” and “readability” of the page out of 10 as well as how their scores could be improved.

#### Responses



#### Evaluation

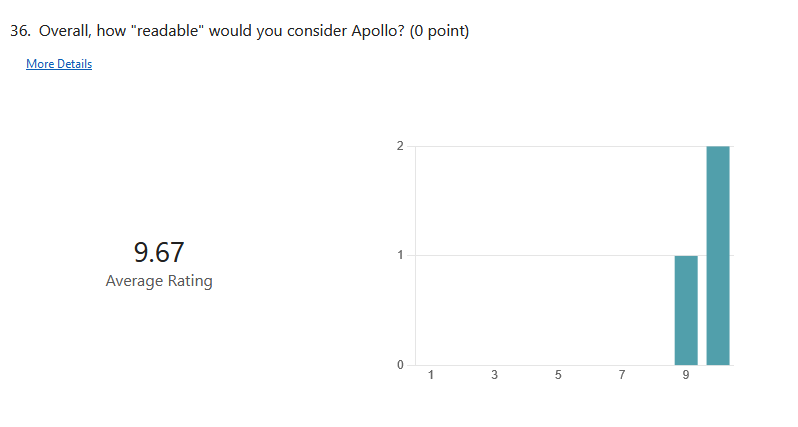
Since the settings were taken directly from the training and generation pages, it does not surprise me that the settings on this page were also deemed as confusing. As I mentioned on the training page section, an explanation would be an easy way to fix this, but I would still say that a dropdown menu to hide the “confusing settings” would be better since if the user requires an explanation, I would argue that it is not something they should change anyway.

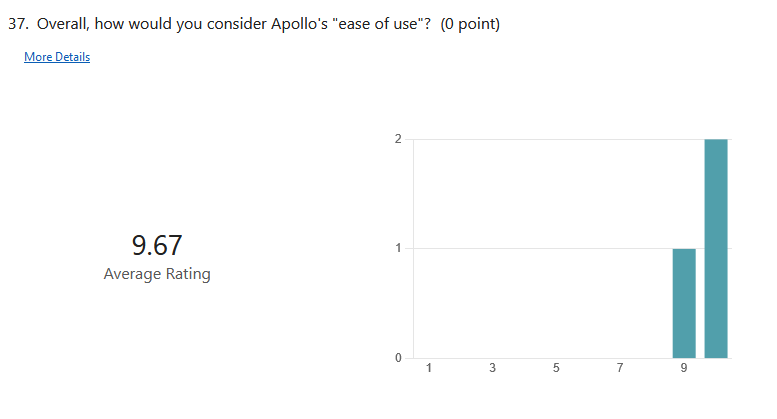
### Section 6: Overall Evaluation

#### Synopsis

This section asked the user for their overall evaluation on Apollo. Specifically, the readability and the ease of use of the program as a whole.

#### Responses





#### Evaluation

Overall, the users seem very happy with Apollo, and so personally, I would say that I successfully fulfilled objective 4.

## Self-Evaluation

### Objective 1

*“To understand neural networks and be able to create a recurrent neural network from scratch”*

I have certainly achieved this objective. This was the first objective I achieved during the technical implementation and my consistent research throughout this project is evidence of this. The components of the neural network were separated to into its own “NeuralNet” class library and the code for the matrix mathematics was separated into its own “MatrixMaths” class library.

However, I would argue that this code is not performant both because of the quality of my code and the fact that C# is simply slow, and so if I were to attempt a project like this again I would either use a more performant language (e.g., C++) to create the network, or use a machine learning API (e.g., Tensorflow). In my opinion, creating the network from scratch rather than using an API was beneficial for my understanding of the topic, but if I were to implement a machine learning algorithm for a business, or in a project where the machine learning algorithm isn’t central, I would most likely use Tensorflow as it abstracts the complex mathematics and would allow me to concentrate on the more important areas of the project.

Furthermore, I would say that my network isn’t amazing at generating music. However, at this moment in time I do not know why that is the case. So further research is required in order to improve the overall performance of the network.

### Objective 2

*“To allow the neural network to save and load its state to avoid it having to be trained every time it is used”*

I have definitely achieved this objective. When training begins, a state file is created just before the training happens and just after. In addition to this a temporary state file is created every epoch in order to prevent the error after training being a NaN. These state files are utilised by the program in order to allow the user to revert the previous set of training they did on the network.

### Objective 3

*“To create a program which can process MIDI files”*

This objective was completely fulfilled by the program. The MidiManager class handles the reading and writing of MIDI files to and from the string notation created. Then this string notation is converted into an array of one hot vectors by the Vocab class which can be inputted into the network as an input.

Also, to avoid having to convert the training data into vectors every time the program is reopened, the TrainingFileManager class writes the array of one hot vectors into a binary file, in order to cut out the time needed to perform the conversion. Also the vocabulary needed by the profile in order to interpret the output is stored in the profile’s schema.json file, cutting out the necessity to read the MIDI files for this purpose after the initial creation of a profile.

However, this is one of the main places where the program is slow. With bigger files, the process of reading and converting the MIDI files into vector form takes a long time. However, this is something that cannot be performed by a library, so the code would have to be optimised instead. For instance, instead of reading the file on two separate occasions (i.e. when the Vocab object is being created, and when the Training Data is being created), the file could be read, and then the string output stored until it is needed for either purpose. The main downside to this approach, however, is that it would make the program consume more memory.

### Objective 4

*“To create an easy-to-use graphical user interface”*

I think that I fully met this objective. The program does not hide any functionality, and any changeable settings are clearly shown to the user. In terms of settings, since the topic focus of the project is complex, there is only so much that can be done in order to simplify the terms into something which every user can understand. For instance, in the user questionnaire, the training page received negative feedback for the settings not being understandable. These settings cannot be easily explained to someone with no prior machine learning experience and so what really matters in this scenario is whether the user felt forced into changing things they did not understand since this could harm their experience with the program.

However, I would say that it is not fully clear if the program is working on something (like training the network) or if there is a bug which is making the UI lag and the program unresponsive. Therefore, if I were to improve the final product with respect to the GUI, I would add a pop up when the program is doing something which the user must wait for rather than after it is completed. This pop up would contain a progress bar with the approximate amount of time until the task is completed, and the program is useable again.

## Final Comments

Overall, I think the project was a success and I would say that I met all my objectives. If I were to continue working on this project, the first thing I would do is make a separate API for the network, which is accessed remotely by the clients running the program. This is because the neural network is CPU and memory intensive, making it unsuitable for many machines. Making the network as an API would mean that the network can run on a specialised machine which is suitable for running the network so that the hardware requirements for the program isn’t so high.

Furthermore, since I used WPF, the program is restricted to Windows only. So, another change I would make is the GUI framework I used. A likely candidate is AvaloniaUI (<https://github.com/AvaloniaUI/Avalonia>) since Avalonia XPF allows for existing WPF apps to run on Linux and Mac operating systems.