

Faculty of Technology, Design and Environment
Oxford Brookes University
School of Engineering Computing and Mathematics

BSc (Single Honours) Degree Project

Programme Name: Computing Project

Module No: U08096

Surname: Garlick

First Name: Samuel

Title: A machine learning approach to multi-instrumental composition

Student No: 16069781

Supervisor: Mr Peter Marshall

2TM Supervisor:

(if applicable)

Date submitted: 20th March 2019

Extract from the Student Conduct Regulations:

2.2.1 Students shall not cheat (i.e. obtain, or attempt to obtain, an unfair academic advantage) in any assessment of their competencies or academic ability or professional skills. They shall comply at all times with the provisions of the Regulations for Students taking Assessments. In particular, they shall not commit collusion, plagiarism, falsification, duplication, submit the work of others as their own or allow another person to undertake an assessment for them.

Explanation of terms used in the Student Conduct Regulations:

Impersonation means taking an assessment on behalf of another student, or allowing another person to take an assessment on your behalf.

Collusion means producing assessed work by working with another person whom you have not been authorised to work with by the Module Leader. This includes, but is not limited to, allowing another student to copy your work.

Falsification means presenting invented data, for example claiming that a program works when it does not, or claiming that it produces results which it actually does not.

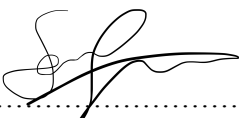
Plagiarism means submitting the work of someone else as if it were your own. If you include material by someone else in your assignment, you must show clearly in the text how much was copied from elsewhere. It is not enough just to list references at the end of your assignment.

Guidance on the correct use of references can be found on www.brookes.ac.uk/services/library, and also in a handout in the Library.

If you do not understand what any of these terms mean, you should ask your Module Leader to clarify them for you. The full regulations may be read in the Library, or accessed on-line at <http://www.brookes.ac.uk/regulations/sturegs.html>

If you do not understand what any of these terms mean, you should ask your Project Supervisor to clarify them for you.

I declare that I have read and understood Regulations 2.2.1 of the Regulations governing Academic Misconduct, and that the work I submit is fully in accordance with them.

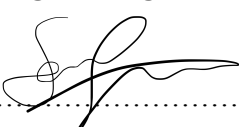
Signature 

Date: 18th March 2019

**REGULATIONS GOVERNING THE DEPOSIT AND USE OF OXFORD BROOKES UNIVERSITY
MODULAR PROGRAMME PROJECTS AND DISSERTATIONS**

Copies of projects/dissertations, submitted in fulfilment of Modular Programme requirements and achieving marks of 60% or above, shall normally be kept by the Library.

I agree that this dissertation may be available for reading and photocopying in accordance with the Regulations governing use of the Library.

Signature 

Date: 18th March 2019

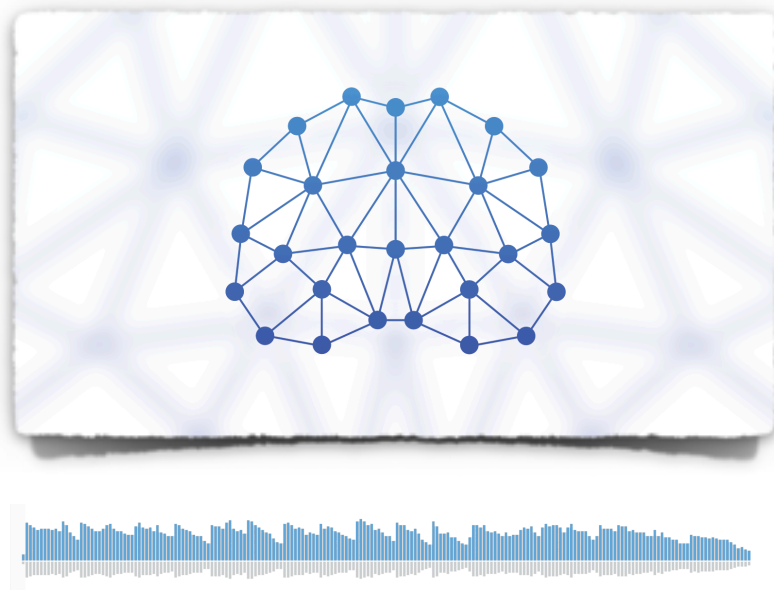
Honours Project Report

A Machine Learning Approach To Multi-Instrumental Composition

A report submitted as part of the requirements for the degree of BSc (Hons) in Computer Science

At

Oxford Brookes University



Sam Garlick 16069781

0. Abstract	5
1. Introduction	5
1.a. Motivation	5
1.b. Objectives	6
1.c. Project Structure	6
1.d. Research Question	6
2. Literature Review	7
2.a. What the papers bring to the field	7
2.b. Negatives & Comparisons	10
2.c. Conclusion	11
3. Methodology	13
4. Data Representation and Feature Engineering	16
4.a. Semi-Tone Representation	16
4.b. Instrument Encoding	16
4.c. Duration Representation	17
4.d. Data Reduction and Normalisation	20
4.e. Conclusion	21
5. Training Data	22
5.a. The Data Set	22
5.b. MIDI Format	23
5.c. Data Pre/Post Processing	23
6. Network Configuration	25
6.a. Network Layout	25
6.b. Optimising	26
6.c. Bottlenecking	26
6.d. Activation	27
7. Music Generation	29
7.a. Data Generation	29
7.b. MIDI to Music Conversion	30
8. Evaluation	31
8.a. Evaluation Issues	31
8.b. Subjective Evaluation	31
8.c. Analytical Results	33
8.d. Evaluation Conclusion	33
8.e. Professional, Ethical, Social, Security and legal issues	34
9. Discussion	36
10. Conclusions and Further Work	36
10.a. Conclusions	36
10.b. Further work	38
10. References	39
11. Appendices	42
11.a. Sample Audio	42
11.b. LSTM Glossary Definition	43
11.c. Detailed Project Proposal	44
11.d. Project Log	51
11.e. Ethics Review Form E1	53
11.f. Source Code	55

0. Abstract

Current machine learning based approaches to music generation lack the ability to encode multiple instruments and non-ambiguous duration. However, non-machine learning based approaches to music generation, such as arpeggiators, cannot generate music naturally enough that people believe it to be composed by a human. The reason for the inability for a neural network to compose multi-instrumental natural music may be due, in part, to the way the music is encoded for the neural network to train upon. This project presents a way to flexibly encode duration and instruments in a song that allows a neural network to learn and compose music for multiple instruments. This project also proposes a method to evaluate the generated music using an imitation game test, known more commonly as a 'Turing test'. Remarkably, the neural network created in this project was able to produce music well enough that, for the majority of songs, people believed the composer of the music to be human.

1. Introduction

1.a. Motivation

Currently machine learning models for composing music are very limited because they lack the ability to compose music comprised of multiple instruments, and secondly, they lack the capacity to clearly determine the duration of a note. This is significant because it limits the potential music machine learning can produce. The inability to compose music for multiple instruments limits machine learning to compose music for only a single instrument, rather than create music for an orchestra. Additionally, without the ability to clearly encode duration of a note has the effect of preventing the network from producing complex melodies with multiple shorter notes throughout a longer base note. This limits the set of melodies the network can compose to simple, robotic sequential series of fixed length notes; thus preventing the network producing natural melodies.

Overall these two factors limit the capabilities of machine learning compositions. They cannot produce symphonies or even music for a rock band, but instead are limited to a single score of robotic melodies. This project aims to eliminate these restricting issues in order to enable computers to generate complex music on command. The style and genre of music that this project can generate is limited by the style and genre of the music the project is trained upon. Theoretically, a neural network could be trained upon any genre allowing the network to generate any style and genre of music. This means companies who have a vested interest in producing a large volume of music, such as Sony Records, will benefit from this technology as then can produce music on command for free. Alternatively, movie

soundtrack composers such as 'Hans Zimmer' (Hans-Zimmer, 2001) or 'John Williams' (John Williams 1995) could train the network on movie sound tracks, or advertising companies such as 'Adam&EveDDB' (Adam&EveDDB 1996) could train the network on advertising jingles. Companies, composers and organisations who might use this technology will benefit because they can generate new music on demand for monetary gain.

1.b. Objectives

Objective 1: Create a Neural Network that can compose a song with different scores for each instrument.

Objective 2: Using the same network as Objective 1, encode each note with a duration to enable the network to learn not just pitch, but also timing.

Objective 3: To create a neural network that can compose music indistinguishable from music composed by a humans.

1.c. Project Structure

1. Review and comparison of current literature and methodologies used for generating music using machine learning.
2. Overview of how the project was formed and completed
3. How training data was represented to allow a neural network to train upon
4. The process of obtaining and converting the training data
5. How the neural network was configured
6. How music was generated
7. Evaluation of the project
8. Discussion of how the neural network may be learning
9. Conclusions and further work

1.d Research Question

Investigating a duration-based approach of time series prediction using machine learning for multi-instrumental music generation.

2. Literature Review

This literature review references four papers related to the field of generating music using machine learning. These four papers each introduce different ideas to the field, ranging from a paper establishing the field to new papers that use modern machine learning techniques, such as optimising LSTM cells using resilient propagation (Liu & Ramakrishnan 2014). The papers will be reviewed and compared with one-another in order to gather information currently established in the field; whilst exploring ideas and issues that may arise using those methods for generating music using machine learning.

2.a. What the papers bring to the field

The father of machine learning composed music, Todd (1989), introduced the concept that a computer could learn to compose music unlike ever before. Todd (1989) described, in detail, how music could be generated using a Recurrent Neural Network (RNN) with one hidden layer, where the outputs of the hidden layer fed back into itself. By using a Recurrent Neural Network, Todd (1989) explained that music could be treated as time-series data, where the music represents a series of sequences of notes. This implication that music could be treated as time-series data implies that the notes playing at time (T) could be used to predict the notes playing at (T+1), thus the notes playing at (T+1) could be used to predict the notes playing at (T+2) up to (T+N). Todd (1989) established that Recurrent Neural Networks were imperative to generating music, as the recurrence created a form of “memory” necessary to generate rhythm and structure required by music, thus laying down the foundations on the field.

Todd (1989) also established different ways to treat music as inputs to a network, hypothesising that each semi-tone can be mapped directly to an input node in the network. Thus, you could represent 12 semi-tones (A - G#) with 12 input nodes and each output note mapping back to each semi-tone. Todd (1989) also hypothesised that outputs of the network could instead represent the change to the notes, for example, (+2) would change a B to a C#. As Todd (1989) discussed, using the aforementioned *transitional* encoding could potentially create the issue that while the network is generating new melodies, the network may accidentally change key, causing the melody to de-tune itself.

Todd (1989) proposed two ways to encode duration: firstly, to have a separate pool of units to encode duration; alternatively, to remove duration entirely and instead keep all notes at a fixed length. Eck & Schmidhuber (2002) proposed ending notes with a zero to signify the note has finished. Eck & Schmidhuber (2002) disputes Todd's (1989) idea of using a separate pool to encode a note starting, hypothesising it may not scale well. However, the idea that Eck & Schmidhuber (2002) proposed may also create issues because Eck & Schmidhuber (2002), like Todd (1989), represented notes switched off with a zero. This means a string of sequential notes each lasting one time interval, should be encoded as a string of ones (1, 1, 1) but the notes are required to have their last interval encoded as a 0, thus the string of ones becomes (0, 0, 0), which implies all the notes are off.

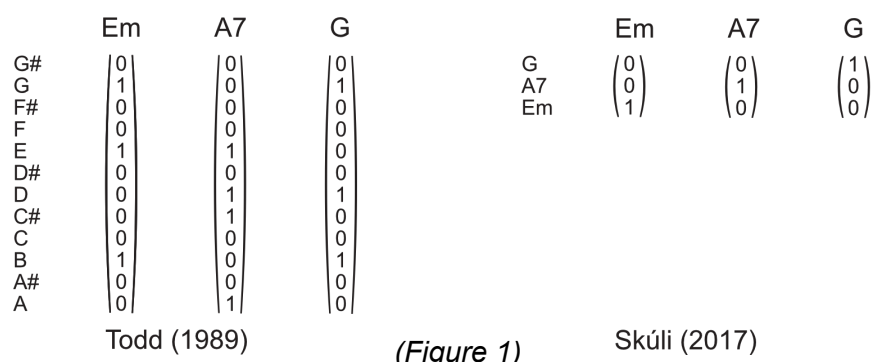
Eck & Schmidhuber (2002) added Long Short Term Memory (LSTM) cells to the field as they observed that standard RNNs lacked the ability to learn global structure. This resulted in RNNs limited to producing small melodies, as they were only able to predict notes for time (T) using (T-1) as inputs. However, LSTM cells were able to predict notes for (T) using (T-1), (T-2) ... (T-N). The introduction of LSTM cells provided networks the ability to learn global structure necessary to enable networks to produce full songs rather than small repetitive melodies crucial to music generation.

Eck & Schmidhuber (2002) represented data similar to Todd (1989), marking notes on with a one and off with a zero. Thus, to introduce chords meant activating multiple semi-tones to build a chord. Eck & Schmidhuber's (2002) addition of LSTM cells were able to rebuild the chords their network trained on in addition to learning melodies. This meant that, when creating new melodies, their network could create music with chords, providing more flexibility over the music the network could train on and create.

Importantly, the addition of LSTM cells prevented the vanishing and exploding gradient problem (Hochreiter et al. 2001) relevant to Recurrent Neural Networks. The vanishing and exploding gradient problem is an issue, such that as weights get recursively iterated over weights larger than one explode and weights smaller than one vanish. Liu & Ramakrishnan (2014) introduced training the

network using Resilient Propagation (RProp) to train their LSTM network, instead of using Back Propagation Through Time (BPTT), which was commonly used to train LSTM networks. Liu & Ramakrishnan (2014) concluded that using RProp to train their network expedited the learning rate, faster than BPTT.

Liu & Ramakrishnan (2014) did not represent duration in their data, however, they did comment that duration could be represented using methods that, as previously stated, Todd (1989) & Eck & Schmidhuber (2002) proposed. As Liu & Ramakrishnan (2014) did not encode duration in their data, their network would generate music with a fixed duration, which has the effect of creating robotic, unnatural music. Skúli (2017) represented their data similar to Liu & Ramakrishnan (2014), such that duration had not been encoded, but instead all notes were quantised to a fixed length. Skúli (2017) represented their data differently to all previous methods, such that rather than mapping semi-tones to an input in the network, all the semi-tones and chords were one-hot encoded. One-hot encoding is the process of encoding categorical data, for example “cat”, “dog”, “bunny”, “fish” to (0001), (0100), (0100) and (1000) respectively where each column in the encoding uniquely represents a category. For example, the chords: Em, A7, G, could be encoded in a three-dimensional vector, each dimension representing a different chord. Thus, the output of the network could be categorised into one of the aforementioned chords. This is different to the method first outlined in Todd (1989), that may need a larger vector to store all the notes required to create the chords, as shown in ‘figure 1’. However, using Todd’s (1989) method allows you to reuse inputs required for different chords. Skúli’s (2017) representation limited the network, as unknown chords cannot be trained in the network, thus new chords cannot be composed. However, Todd’s (1989) encoding allows for new chords that have not been directly one-hot encoded, as the chords can be built from the individual notes.



2.b. Negatives & Comparisons

Todd (1989) proposed several possible ways to configure the network and treat music as inputs to a network, exploring each idea, discussing why it may be advantageous or not to use when generating music via machine learning. This is good because it allows you to understand why it works, or how it could be improved. For example, as mentioned in '2.a. What the papers bring to the field', Todd (1989) proposed encoding duration using a separate pool of dimensions to encode when a note starts. Todd (1989) thoroughly explained how this could work, however, Eck & Schmidhuber (2002) disputed this approach without justification commenting, only, 'it may not scale well'. Eck & Schmidhuber (2002) proceeded to propose their idea without thoroughly explaining how it could work.

Skúli (2017) did not reference other literature, this may have limited their work as they may not have known or thought to encode the inputs by mapping semi-tones to dimensions in the input vector, alternatively they may have been trying something new. As mentioned in '2.a. What the papers bring to the field' this limits the network because new chords will need new dimensions in the input vector, whereas the method outlined in Todd (1989) allows you to reuse dimensions to create new chords. This issue may have been resolved had Skúli (2017) previously established literature.

No paper evaluated the quality of music generated, however, Liu & Ramakrishnan (2014) did evaluate the accuracy of their network as a means to empirically compare Back Propagation Through Time against Resilient Propagation. Nevertheless, they did not evaluate the generated music; commenting, "these evaluation metrics (*do*) not necessarily correspond to human's perception of music". Eck & Schmidhuber (2002) also commented "(it) is difficult to evaluate the performance objectively". No paper qualitatively evaluated the generated music due to music being very subjective. Although music cannot be qualitatively measured, other assessments could be used, for example, a Turing test style evaluation, known as the "Imitation Game" (Gunderson, 1964). This could be done by having a collection human composed music and machine learning composed music, then randomly selecting a random ratio of human/machine learning compositions then asking a human to attempt to guess what composed the music.

2.c. Conclusion

One core element to include in the project is the use of LSTM cells, which were introduced to the field by Eck & Schmidhuber (Eck & Schmidhuber 2002). LSTM cells have the ability to prevent the vanishing gradient problem (Hochreiter et al. 2001) and importantly, learn the global structure of music not just local structure. LSTM cells are therefore crucial to the success of the project. Liu & Ramakrishnan (2014) introduced the use of Resilient Propagation to expedite learning rates as a means to optimise their model. For this reason, the model in this project is optimised using Resilient Propagation as, although not imperative to the success of the project, it will increase the number of epochs achievable in the limited time frame for the project. Consequently, allowing for more experimentation to achieve better results.

Most papers mapped semi-tones to the different dimensions in the input vector, only disputed by Skúli (2017), who instead encoded specific semi-tones and chords as different dimensions using one-hot encoding. In turn meaning new chords would need to be re-encoded. Todd's proposed method (Todd 1989) provides greater expandability, as new chords could be created by re-using different dimensions rather than re-encoding the inputs. The method proposed in Todd (1989) for encoding semi-tones is used in this project due to its flexibility and expandability.

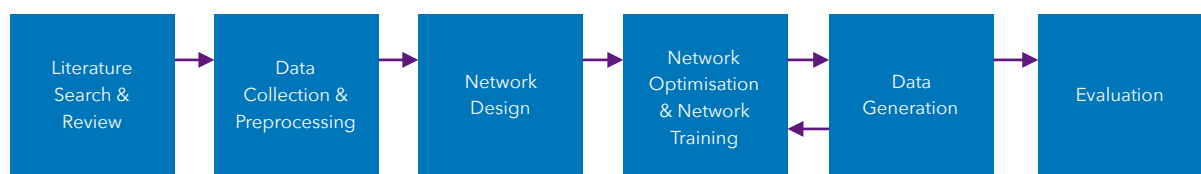
Both methods proposed by Todd (1989) and Eck & Schmidhuber (2002) have potential flaws. Firstly, the method proposed by Todd (1989) uses a separate pool to encode when a note starts. However, this may be ambiguous as the duration pool does not specify which semi-tone to refer to. This means the pool will restart all notes currently playing, preventing complex melodies such as multiple notes with different durations simultaneously playing. This restriction limits the music the network can compose. Secondly, as mentioned previously, ending notes in a 0 to denote the ending may limit a string of notes lasting one time interval. These notes would be represented as a string of zeros, which could be interpreted as the notes being off. This project uses a modification of Todd's method (Todd 1989), more information can be found in section '4. Data Representation and Feature Engineering'.

As mentioned in '2.b. Negatives & Comparisons', qualitatively measuring music is not feasible in the scope of this project as music is subjectively perceived meaning you'd need to objectify people's opinions. Therefore, the main objectives of the project are to generate music using machine learning that, ideally, should be indistinguishable to human composed music. As mentioned before, this could be examined using an "Imitation Game" style test (Gunderson 1964). The success of this test could be based upon how poorly the subjects are able to distinguish whether the composer of the music presented was a human or a computer. Other approaches could be taken to evaluate the neural network. For example, Liu & Ramakrishnan (2014) evaluated their model based upon its accuracy value to compare different optimisation techniques. Whilst this approach is applicable for comparing different optimisation methods, it is not appropriate for evaluating the quality of the generated music. More information can be found in section '8. Evaluation'.

In conclusion, the methods for encoding duration outlined by Todd (1989) and Eck & Schmidhuber (2002) could be improved. The idea proposed by Todd (1989) does not allow for more complex music, likewise the method proposed by Eck & Schmidhuber (2002) may limit songs comprised of smaller notes. Additionally, no paper commented on the use of, or ability, for a neural network to learn and compose songs containing music for multiple instruments, thus limiting them from composing a symphony. These issues will be addressed in this project.

3. Methodology

The methodology used for this project closely resembles an Agile development methodology. The project was broken down into the steps required to complete the project; these outlined steps are, as follows, Literature Search, Literature Review, Data Collection, Data pre-processing, Network Design, Network Training, Network Optimisation, Data Generation and Evaluation, as depicted in 'figure 2'. Each step of the project was treated as an Agile sprint (Beck et al., 2001). Another possible methodology would be the Waterfall Methodology (Powell-Morse 2016) however, Agile was favoured due to its flexibility. By using an Agile development process allowed for flexibility, as time scales and sprints could easily be adapted if sprints weren't completed on time or an unforeseen error occurred.



(Figure 2)

The first sprint was to collect then review literature previously written regarding using neural networks as a means to generate music. Other methods for computer generated music do exist, such as arpeggiators (Attack Magazine 2013). However, this project uses neural networks due to its ability to learn from the data rather than manually configuring the arpeggiators. As mentioned in sections '2. Literature Review', all literature was related to generating music using neural networks. The literature was systematically reviewed based upon the contributions each paper brought to the field. Each addition was compared and reviewed to assess if this project would benefit from the contribution. Contributions outlines by review includes the addition of LSTM cells and a basis for data representation.

The next step of the project was to collect and pre-process data the network would train on. As the objective of the project is to enable a neural network to compose multi-instrumental music, the network would need several multi-instrumental compositions to train on in order to learn how to compose music. The first step was to find previously composed MIDI that contained music containing different instruments. MIDI is file format for storing structural data about a song which can be played in a type of program called a Digital Audio Workstation. This is

different to a standard audio file which will instead store audio as waveforms. Digital Audio Workstations synthesise the instruments in a MIDI file and play the structural data as an audible song. It is due to this musical structuring in the MIDI file format that lends itself well towards machine learning, as theoretically, a neural network can learn these structures. This is explained in more detail in section '5 Training Data'. MIDI cannot be directly fed into a neural network; the data need to be reformatted to allow the network to train on the data. The next sprint involved pre-processing the data so it adhered to the requirements necessary to learn multiple instruments. Then creating a program that could take a MIDI file and reformat it into the data representation outlined in section '4. Data Representation and Feature Engineering' to allow the network to learn features such as semi-tones, duration and instruments. Each feature is not explicitly taught to the network but instead inferred from the data.

The neural network in this project was designed and built using Keras, a high-level neural network API for Google's Tensor-flow (TensorFlow, 2015). Using Keras in this project allowed for more experimentation as more focus could be applied to details, such as the data representation and testing different network optimisations rather than writing the lower level maths required for an LSTM network. Overall this meant there was more time devoted to improving the networks ability to compose music.

The next step after optimising and training the network is to generate new data, more information can be found in section '7. Music Generation'. The data generated by the network will be in the same format used to train the network, as this is the format the network will have learnt. Therefore, once the network has generated music, the composed data will need reformatting back into the original MIDI format as a means to represent the data as music. This is explained in more detail in '5.c. Data Pre/Post Processing'. Once the music is in this format, it can then be used in a Digital Audio Workstation program to synthesise instruments to play the MIDI file.

The final sprint for the project was Evaluation. This sprint is comprised of gathering data as a means to evaluate neural networks as an approach to multi-instrumental composition. As mentioned in section '2. Literature Review', using

accuracy as a measurement is insufficient for evaluating a neural networks ability to compose music. A higher accuracy implies the network has not learnt how to compose music, but instead how to memorise a sequence of notes, therefore the accuracy of the network was not used as a means to evaluate the network's ability to compose music. Instead, the project will be evaluated using an "Imitation Game" (Gunderson, 1964). A test subject will be presented with a mixture of human composed songs and machine composed songs, and must choose the songs they believe to be composed by a human and the songs they believe to be composed by a computer. More details can be found in section '8. Evaluation'.

4. Data Representation and Feature Engineering

In order to achieve the objectives outlined in the introduction, the data the network trains on needs to encode specific features to enable a network to learn how to compose music. These three features are semi-tones, note durations and the instruments different notes are assigned to. If the network can successfully learn these features, it should be able to generate data with these features interoperable as music

4.a. Semi-Tone Representation

The input music for this project can be represented as a time-series matrix, each column being a vector representing an individual time slice of the original music (as represented in figure 3). Each time slice needs to encode, for each instrument, whether or not a note for a particular semi-tone is active and the duration for that note. This project uses a similar notation to represent whether or not a note is active using the method proposed in Todd (1989), where each dimension in a time vector is mapped to a specific semi-tone. Active notes can be represented with a 1, and inactive notes can be represented with a 0. The project will use Todd's method to represent semi-tones, because different chords can be represented by reusing different dimensions, allowing for more expandability and a greater range of notes using a fixed set of dimensions from the input vector. For this project, the semi-tones will range from A1 - G#6 to fully encompass all semi-tones used in the dataset.

Semi-Tone Representation

G#	0	0	0	0
G	1	0	0	1
F#	0	0	0	0
F	0	0	0	0
E	0	1	1	0
D#	0	0	0	0
D	1	0	0	1
C#	0	0	0	0
C	0	1	0	0
B	1	1	1	1
A#	0	0	0	0
A	0	0	1	0
	T	T+1	T+2	T+N

(Figure 3)

4.b. Instrument Encoding

There are a few possible ways to allow a neural network to compose music for multiple instruments. One strategy is to train a network to compose music for a single instrument, then generate multiple melodies from that network for each instrument. However, there is no guarantee that the melody produced for one instrument will flow with a melody for another. This means each instrument may

play very different melodies and, probably, out of time with each other creating very dissonant music. Alternatively, the network could train on the music for each instrument at the same time by designating specific dimensions in the input vector for each different instrument (as depicted in figure 4). This should enable the network to compose music with more harmony, as all the music for each instrument will be trained on and generated at the same time. Thus, all instruments should play time with each other composing melodies that flow well together. This method for encoding instruments into separate pools has the disadvantage of exploding the input space as the size of the input vector is directly proportional to the

number of instruments encoded, thus creating an issue of dimensionality (Trunk 1979). To combat this issue, only semi-tones C1 - C5 were encoded as this was the minimum range that the dataset existed within (4.d. Data Reduction and Normalisation).

Instrument Encoding

Ins. 1	G#	0	0	0
	A	0	0	0
Ins. 2	G#	0	1	1
	A	1	0	0
Ins. 3	G#	0	0	0
	A	1	1	1
Ins. 4	G#	0	0	0
	A	0	0	0
		T	T+1	T+N

(Figure 4)

4.c. Duration Representation

Representing duration within each input vector could be achieved in a multitude of ways. As mentioned in the literature review, Todd (1989) proposed encoding duration using an additional dimension in the vector to encode when a new note starts, this however, would not allow for more complex melodies where a note continuously plays while new notes start. Eck & Schmidhuber proposed, in Eck & Schmidhuber (2002), to represent a note's duration by terminating notes with a 0. Therefore, the length of a note is determined by how many time slices a succession of 1s last for. However, as mentioned in section '2. Literature Review', this method may lead to ambiguity when there is a succession of 1 length notes. This is because a succession of notes each lasting one time frame would need to read a 1 to encode the semi-tone as active and at the same time read a 0 to encode the note in the previous time frame has ended. This conflict prevents a succession of notes being represented in the data, and therefore restricts the

capabilities of the network. Due to both of these methods restricting the capabilities of the network it is important to theorise new methods to remedy these restrictions. Three methods were developed in this project 'Discrete', 'Static' and 'Active-Start'.

The method, 'discrete', could work such that the input vector is a collection of discrete duration vectors where each duration vector encodes the duration of a specific semi-tone. Each dimension in the duration vector maps to a specific duration of time steps that semi-tones play through. For example, the first, second and third dimension of the duration vector could be mapped to represent 1, 2 and 3 time steps respectively (as shown in figure 5). Therefore, if a note lasts 3 time steps, the third dimension could be represented by a 1 whilst the other dimensions are 0. The benefit of this is that duration in the vector is clearly laid out providing a structure that

Discrete Encoding

G#	1	0	0	0	0
	2	0	0	0	0
	3	1	0	0	1
G	1	1	1	1	1
	2	0	0	0	0
	3	0	0	0	0
...					
A#	1	0	0	0	0
	2	1	0	0	0
	3	0	0	1	0
A	1	1	0	0	1
	2	0	1	0	0
	3	0	0	1	0
		T	T+1	T+2	T+N

(Figure 5)

a neural network should easily learn from. However, this raises some issues. Firstly, encoding the input data like this will cause the vectors to explode in size because each semi-tone requires a sub-vector to encode all n unique durations. Additionally, it limits all notes in the training data, as the generated notes have to fall into any of the n durations. This also has the issue that a note will only be represented in the data when the note starts, thus the following time frames will not indicate that a note is active which means active notes may get misrepresented as inactive. This issue of ambiguous notes may hinder the networks ability to learn sequences, as a '0' can represent both a semi-tone being active but not starting and a semi-tone being inactive. These active semi-tones misrepresented by a 0 may hinder the network's ability to learn duration and timing.

Additionally, another method to encode duration, 'static', could encode duration such that when a note starts playing, instead of representing the note with a one or zero, the note could be represented with an integer denoting how many time slices the note plays through (depicted in figure 6). For example, a melody containing a note lasting 3 time slices, 2 time slices then 1 time slice could be represented as (3,0,0,2,0,1). This removed the additional dimensions required in the 'Discrete' encoding. However, it suffers the same issue that the network may struggle to learn when a semi-tone is active, as a '0' can represent both semi-tones being active but not starting, and a semi-tone being inactive.

Static Encoding

G#	0	0	0	0	0
G	3	0	0	1	2
F#	0	0	0	0	0
...					
B	1	1	1	0	1
A#	0	0	0	0	0
A	0	0	1	0	0
	T	T+1	T+2	T+3	T+N

Lastly, a method 'Active-Start', a variation on Todd's method in Todd (1989), could work, such that each semi-tone is represented by a sub-vector of 2 dimensions. The first dimension encodes whether a semi-tone is currently active, denoted by a one, or inactive, denoted by a zero. The second dimension encodes when to start a note. This is different to the method outlined in Todd (1989), as each semi-tone has its own 'pool' to denote when a note starts (as shown in figure 7). The benefits of this method are firstly, unlike the methods 'static' and 'discrete', the network can easily know the active semi-tones playing in a time slice, enabling the network to learn the melodies much easier. Secondly, unlike 'discrete', notes can last any length of time. Lastly, unlike 'static' and 'decay', the inputs and outputs can all be represent using discrete binary values making it easier for the network to learn.

Active-Start Encoding

G#	A	1	1	0	0
	S	1	0	0	1
G	A	1	1	1	1
	S	0	0	1	0
F#	A	0	0	0	1
	S	0	0	0	0
...					
B	A	0	0	0	1
	S	0	0	0	0
A#	A	0	0	1	1
	S	0	0	1	0
A	A	1	1	0	1
	S	0	0	1	0
		T	T+1	T+2	T+N

(Figure 7)

In conclusion, the best of the aforementioned methods to encode duration is 'Active-Start'. 'Active-Start' has the disadvantage of doubling the size of the input vector, due to each input vector requiring $2n$ dimensions, where n is the number of semi-tones to encode. However, it provides an unrestricted and representative structure to encode music. Doubling the required dimensions has the issue of dimensionality (Trunk 1979), which may reduce the rate the network can optimise itself. Despite this issue, the 'Active-Start' method will be used in this project, as the issue of dimensionality has a minimal effect on this project because there are only four instruments encoded in the data. However, it is important to note that a model with more instruments may be strongly affected or potentially not optimise at all. This method has the benefit of encoding a structure that enables the neural network to easily learn and generate music because instruments are mapped to specific pools, and semi-tones are mapped to specific dimensions of the input vector. Additionally, duration in structure is easily interoperable, as active notes are always represented by a '1' and inactive notes are always represented by a '0'. By encoding duration for each semi-tone, it is possible to now have more complex melodies, such as a series of shorter notes with one long note playing in the background of the song.

4.d. Data Reduction and Normalisation

The neural network for this project requires a large number of input dimensions. This project uses four instruments per song to evaluate if a Neural Network can compose multi-instrumental music. Four instruments were chosen due to a quartet being a prominent style in classical music. Each instrument requires encoding 5 octaves of 12 semi-tones, this is because in the data used for this project all notes exist in the octaves between the semi-tones A1 - G#5. Therefore, in order to remove redundant data, only the octaves C₁ - C₅ (inclusive) will be encoded. This does however limit the network to only learning, and therefore composing notes between these ranges. Additionally, as stated in section '4.c Duration Representation', duration for the project is encoded using the 'Active-Start' this requires each semi-tone to use 2 dimensions, one to encode if the semi-tone is active and one to encode when the note begins.

The input vector will require 480 dimensions (4 instruments * 5 octaves * 12 semitones * 2 dimensions = 480 dimensions) to fully encode all this data. Whilst

LSTM cells are capable of handling this many inputs, it is important to remove redundant data where possible (Trunk 1979). However, removing dimensions from the network would mean restricting the possible semi-tones that the network can learn and compose. This may strongly limit the networks ability to learn certain songs that rely on a semi-tone or prevent the songs that network can compose. For this reason the only octaves encoded in the data will be $C_1 - C_5$ (inclusive) as opposed to $C_{-2} - C_8$ because, as mentioned before, this is the minimum range that encompasses all of the semi-tones in the data set. By restricting the network to learn only the notes existing in the range $C_2 - C_5$ will cause all output data to also exist in this range. This restricts the possible notes a network can generate. For other data sets it may might worth changing these ranges to allow the network to learn a larger range of semi-tones. The data is already normalised as all the data is encoded using only ones and zeros.

4.e. Conclusion

In conclusion, the data will be encoded into a matrix such that each column of data represents a time frame in the music, and each instrument has their own pool of dimensions in the input vector. Each pool consisting of the 5 octaves that the music dataset spans over. Each octave containing the 12 semi-tones, and each semi-tone containing the 2 dimensions required by the 'Active-Start' method. This will work such that when a semi-tone is active, the 'Active' dimension in the vector for the corresponding semi-tones will contain a one, and if the note begins in the same time step, its corresponding 'Start' dimension will also be active. This creates a matrix for each song in the data set that allows the Neural Network to train from, that should enable the network to learn structure, instruments, semi-tones, timing and transitions from the data.

5. Training Data

5.a. The Data Set

The training data for this project is, of course, music; more specifically, classical music. Classical orchestral music was chosen due to the heavily studied, underlying mathematical structure (Director 2005) prominent in classical music, which a neural network should find easier to train on. Additionally, an orchestra, by definition, provides multiple instruments worth of data which a network can train on to achieve an objective for this project by generating multi-instrumental compositions. The dataset for this project was found at (Grossman 1997). This data set was chosen due to the large amount of data provided and because the file format provided was MIDI. An example of the songs acquired from the data set can be found in appendix '11.a.1 Subjective Evaluation Songs' (human songs 1 to 5).

To evaluate if a neural network can learn to compose symphonies, it is important to outline the instruments the network will train on and compose with. As mentioned in section '4. Data Representation', there are four instruments encoded into the data, due to quartets being a prominent style in classical music. Once the data has been generated and re-encoded into MIDI, the newly created MIDI file can run through a Digital Audio Workstation application to synthesise the instruments for the song. The output music in this project uses four synthesisers for the four instruments (1 per instrument) with the following names 'Flutes', 'String Ensemble', 'Full Brass', 'Solo Cello'. These four instruments were used in order to synthesise the sound of an orchestra by representing three major instrument families, woodwinds, strings and brass. However, the string ensemble did not contain the deeper string instruments such as a double bass or cello, for this reason the fourth instrument in the quartet was chosen to be a cello, thus allowing for deeper, richer, more enhanced audio. More information on this is available in section '7 Music Generation'). The data set initially had instruments that differed to the four outlined instruments previously chosen. To rectify this issue the data was curated, by removing instruments that didn't fit into the four categories, then removing duplicate instruments.

5.b. MIDI Format

The MIDI file format lends itself well to this project, as it stores the structural data about a song that the neural network can learn. Unlike standard audio format, such as MP3 or WAV, the MIDI format, “Musical Instrument Digital Interface” does not store any actual audio. Instead, MIDI format stores only structural data about the music. MIDI format stores information such as when a semi-tone turns on, when a semi-tone turns off and the track the semi-tone plays on. Because the MIDI format contains no actual audio, it requires a 'Digital Audio Workstation' program to decode the MIDI data and synthesise the instruments.

MIDI data is especially useful for this project as it provides the structure of a song which can be formatted into data that a neural network can train upon. This is different to a normal audio file, which instead stores data as wave-forms, these waveforms lack the structure required for this project. Whilst a neural network has the potential to learn the wave forms in a standard audio file, achieved in (Oord et.al. 2016), it does not provide the flexibility a MIDI file does. Training on the MIDI format essentially allows the neural network to compose a skeleton, allowing one to customise the rest of the song such as the instruments used.

5.c. Data Pre/Post Processing

The initial data had a few issues. Firstly, the songs were in different keys; this is an issue as it may make the data more difficult to train on because the neural network may struggle to find a base key to work from. To solve this, each semi-tone was transposed to the key of B. This is because B was the most common key amongst the data. This does mean that each song composed will be in the key of B, however this is not an issue because musical scale has a logarithmic ratio between notes. The wave length of each semi-tone can be denoted as $W_k \approx W_{k-1} * 1.0595$ (How Music Works, 1997). This means that a song can be transposed up or down and each semi-tone will stay in tune relative to one another. Whilst this ratio between notes allows music to be transposed between keys, this project is only evaluated in the key of B. This is due to a lack of available time in the fixed time span available for this project, however it is important to note that changing keys will give a very different impression to the music and may change how well people evaluate competency of the neural network (8. Evaluation).

Secondly, the songs had different timings, meaning songs played at different beats per minute, but due to the way MIDI stores its data, this is not an issue. MIDI does not store timing as a fixed length of time, but instead stored by an amount of ticks. This means the length of a song playing at 128bpm will remain structurally the same as a song playing at 180bpm. Lastly, the length of the songs are different meaning longer songs may have a greater influence on the network whilst training. To rectify this, smaller songs were duplicated to ensure the songs had a more consistent influence over the network, this is known as random over sampling. Another method could have been Random Under Sampling, however this method would not have been applicable to the project, as removing sequences may remove important structures in the data that the network may require. As mentioned previously, by adding multiple copies of shorter songs to the data set will mean the songs have a more consistent influence over the network. Alternatively, each song could be segmented into the different sections to create a data set of all the different verses from each song. This method will mean that each segment has a more consistent effect on the network compared to the whole song with, for example, 5 verses and 1 chorus. However, this may create a dataset that is unrepresentative on the original music because the network may learn to treat different segments equally, despite a segment appearing far less in the original data. Therefore, the former method will be used in this project because this approach creates a more consistent and representative dataset of the original data.

In order to convert the MIDI data to input data for the network, a processing tool was created. Events in a MIDI file are stored in chronological order, meaning a note turning on or off are not necessarily stored next to each other. The tool would read in events from the MIDI file, store the events in a buffer and once a note's on event and corresponding off event are found, the note would be placed in the matrix defined in section '4 Data Representation and Feature Engineering'. A similar process would turn the generated data into a MIDI file.

6. Network Configuration

6.a. Network Layout

As Eck & Schmidhuber concluded in Eck & Schmidhuber (2002), LSTM cells greatly improve a neural network's ability to compose music. LSTM cells provide a form of 'memory' necessary for music composition because music, fundamentally, is a time-based process, requiring the knowledge of previously played notes to create rhythm.

LSTM Equations: (Eck & Schmidhuber 2002)

$$\text{Forget Gate: } f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

$$\text{Input Gate: } i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\text{Candidate Gate: } \tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

$$\text{Output Value Function: } o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$\text{Hidden State Function: } h_t = o_t * \tanh(C_t)$$

$$\text{Cell State Function: } C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

(Please refer to Appendix '11.b. LSTM Glossary Definition' for term glossary)

LSTM cells work based upon how data flows through the cell over time by taking in their hidden values from the previous time step and the input to the cell in the current time step. The data first passes through the forget gate function which, essentially, tells the LSTM how much data it should forget from the previous time step, and then to decide what new information is to be stored in the cell state. The data now flows through the input gate and the candidate gate in order to create an intermediary input value and create the candidate value. This input gate value and candidate gate value are combined to create the new state of the LSTM cell using the cell state function. The hidden value and input value for the cell are used to calculate the output value which, in turn, is used to calculate the cell's hidden state by passing the initial input values through the output value function followed by the hidden state function, in order to manage the cell state of the LSTM cell. Essentially, LSTM cells work by first filtering the contents of the cell state to forget what the cell currently knows. Then, the input value and a candidate value are calculated, which are used to update the cell state. Lastly, the output value is calculated, which in turn combines with the cell state to create a new hidden state and output for the LSTM cell. The forget gate allows the LSTM cell's cell state to

learn through time what it should and what it should not learn, thus providing the memory required to avoid the long-term dependency problem.

The network contains 4 LSTM layers followed by 2 dense layers. The 2 dense layers fully connect the outputs from the LSTM cells to the network's outputs $O_k = \sigma(W_k \cdot v_k)$. A dense layer is a layer that contains regular neurons that connect all neurons in one layer to all neurons in the next. Thus, allowing the neural network to optimise itself, by regression, to learn the notes it should compose. Other layouts had been experimented with; however larger network configurations struggled optimising themselves. In most experiments, larger network layouts did not converge at all. It should be noted that Gated Recurrent Units would also be suited for this task, however it was decided to use the tried and tested LSTM cells for this project due to their proven success in the field achieved in Eck & Schmidhuber (2002).

6.b. Optimising

The neural network in this project is optimised using Resilient Propagation due to the success Liu & Ramakrishnan (2014) found when using Resilient Propagation in order to expedite learning rates for an LSTM network. This may, in part, be due to the fact that Resilient Propagation (Riedmiller & Braun 1992) does not take the differential of the gradient into account, but instead just the sign of the differential. This means that, weights will still be updated even if the differential is minutely small, thus resilient propagation can train a deep network even as the differentials diminish whilst propagating through layers during training.

6.c. Bottlenecking

It was found during experimentation on the network configuration for this project that it was beneficial to bottleneck the network to force the network to learn the main structures in the music. The use of a bottleneck was considered in the early stages of the project due to the network often learning dissonant artefacts in the music, and it was theorised this could be prevented by restricting the networks ability to learn. This was achieved by reducing the amount of neurons in the penultimate dense layer. The optimum size of this layer was proportional to the size

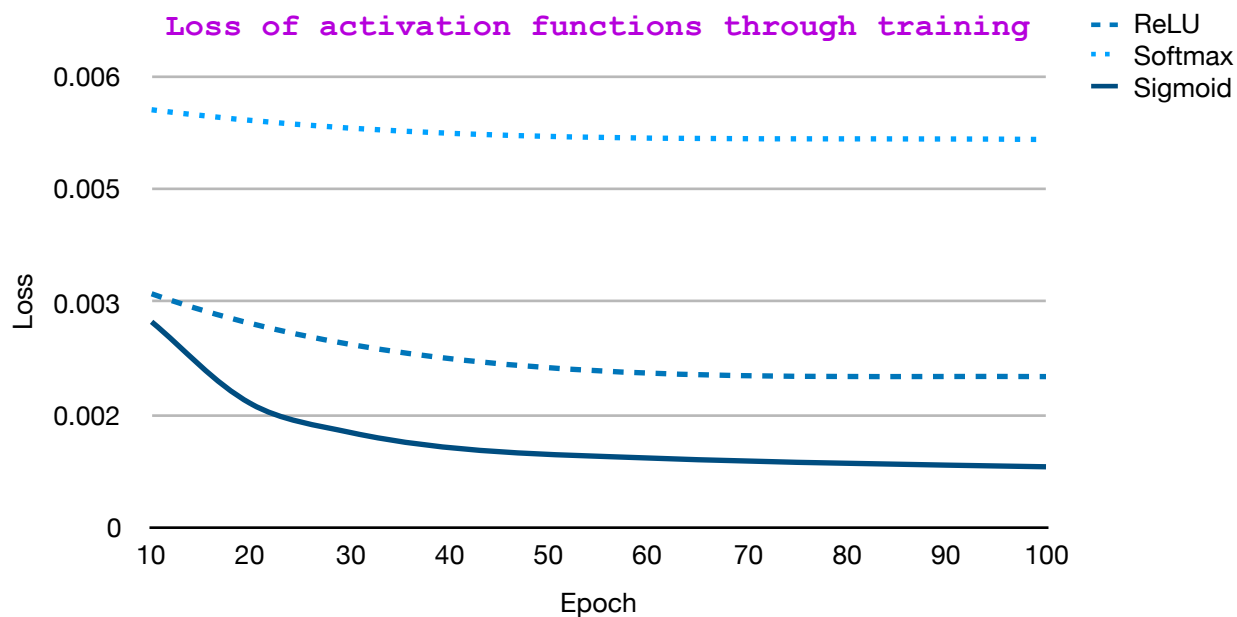
of the dataset, such that a smaller data set required a smaller bottleneck and a larger data set required a larger bottleneck.

The perceived effect of bottlenecking is, as expected, reducing the amount of structure the network can learn. Often when the network was not bottlenecked, the network would learn weird artefacts from the music, such as notes that occurred very little but enough for the network to learn; these artefacts would often cause the generated music to feel unorganised. However, bottlenecking the network would force the network to learn only the most important and common structures and, therefore, prevent the network learning obscure artefacts in the music. Often smaller bottlenecks would produce more repetitive music, additionally if the bottleneck is too small the network would struggle to learn anything at all. Lastly, bottlenecking the network had the effect of expediting the learning rate as there were less weights to optimise, thus the network would converge to an optimum much faster. Comparisons of different bottleneck sizes can be found in appendix '11.a.2 Bottle-neck Comparisons'.

6.d. Activation

LSTM cells are required to use a \tanh function, in order ensure that the inner state values of an LSTM range between the values 1 & -1. However, there was room for experimentation regarding the dense layer activations. The LSTM cell is required to use the \tanh activation function because it has to manage an inner state value which must be able to range between positive and negative values, which is a feature the \tanh function provides. Three different activations were chosen for experimentation, Rectified Linear (ReLU), Sigmoid and Softmax. The different activations functions were compared using their loss metric, a metric calculated using the mean squared error ($\frac{1}{n} \sum_{i=1}^n (Y_i - \tilde{Y}_i)^2$) function. In this instance the

loss metric can be thought of as being a measurement that describes how unsure the network is regarding what notes it should play next. Therefore, the higher the loss metric, the less certain the network is. Whilst this does not directly correspond to how *good* the music is, it does give a measurable value to compare how well the network learns and how confident it is.



(Figure 8)

The results of the test (figure 8) show that the sigmoid activation function $\sigma(x) = \frac{1}{1 + e^{-x}}$ outperformed ReLU and softmax by training much faster and with a lower loss, thus, greater certainty when predicting new notes. Due to these results, the sigmoid activation function is used in the neural network for this project.

7. Music Generation

7.a. Data Generation

As mentioned in '4.a. Semi-Tone Representation', the data for this network is treated as time-series data. Treating music as time-series data means, theoretically, active semi-tones at time T can be predicted using the active semi-tones from time $T-1$. Music can be generated from this principle by using the network to predict active notes, then feeding the newly predicted notes back into the network and repeating.

A seed is required to generate the first time frame in a sequence. This is because the network will use data at time $T-1$ to predict T . Therefore, in order to predict the first time frame, T_0 , the seed T_{-1} is required. Todd (1989) proposed seeding the network using a random time-slice from the dataset as the first inputs to the network. By seeding the network using time slices from the dataset means the network is theoretically more likely to create music that flows from that time-slice and is, therefore, more likely to start re-creating the song that time-slice originates from. An alternative method is seeding the network with random values. This can be done by creating a blank vector and choosing random values for each dimension. A random seed is less likely to re-create a song from the data set, but is instead more likely to create music from the structures and semi-tones the network has learnt.

It was found that during experimentation, the network would struggle to predict data when using a random seed such that, frequently, no music would be generated. Often, the network may require multiple attempts with different random seeds to generate a full song of music. When experimenting on this project it was important that the network was guaranteed to compose music, as simulations were often carried out over night with the results evaluated during the day. If data was not produced, it would take an additional nights worth of experimentation to collect data. This wasted time is an issue as this project has a fixed time scale which means losing a days worth of data may mean compromising other experimentation. Todd's method (Todd 1989) for generating music was used during experimentation as this method was far more reliable, compared to random seeding which was important when results were relied upon. However, during the evaluation stage of

this project random seeding was used for song generation, as the songs would be far less likely to be recreations of the dataset and were therefore more original.

Lastly, each time the network composed a time-frame, the values in each dimension were rounded to the closest whole number. This was done to force the network to continuously generate music. Over time, the values the network would compose would often diminish and songs would often end with a flurry of random dissonant notes. This was solved by rounding the notes, meaning the sequence (0.2, 0.8, 0.85) would be rounded to (0, 1, 1). This would prevent the generated songs from prematurely finishing with a slur of notes, but instead force the network to continuously produce notes until the end of the sequence or the end of the song.

7.b. MIDI to Music Conversion

The network would compose a sequence of vectors to create a song matrix where each vector represents a time frame. The song matrix would get saved to a .CSV file, then at a later date the .CSV would get converted to MIDI. This was achieved by first recreating the matrix, then scanning each column to determine when a note turns on. Once the scanner detects a note turning on, it will be placed into a buffer, thus when the scanner detects the note ending, the note will be removed from the buffer then placed in the within the sequence of notes. This sequence of notes is then encoded into the MIDI file.

These newly generated MIDI files can now be placed into a DAW (Digital Audio work station). For this project the DAW used was Apple's 'Logic Pro X' due to availability to the software. Logic Pro X would take the AI composed MIDI file and turn the structural data into audio by synthesising the instruments. As mentioned in section '5.a. The Data Set', the four instruments synthesised were called 'Flutes', 'String Ensemble', 'Full Brass' and 'Solo Cello'. These four instruments were chosen in order to create an orchestral sound, due to the genre of music the network trained upon. Once the song has been given the aforementioned synthesisers, the song can be exported as a standard audio file. The evolution of compositions through training can be found in appendix '11.a.3 Evolution of Composition Through Training'.

8 Evaluation

8.a. Evaluation Issues

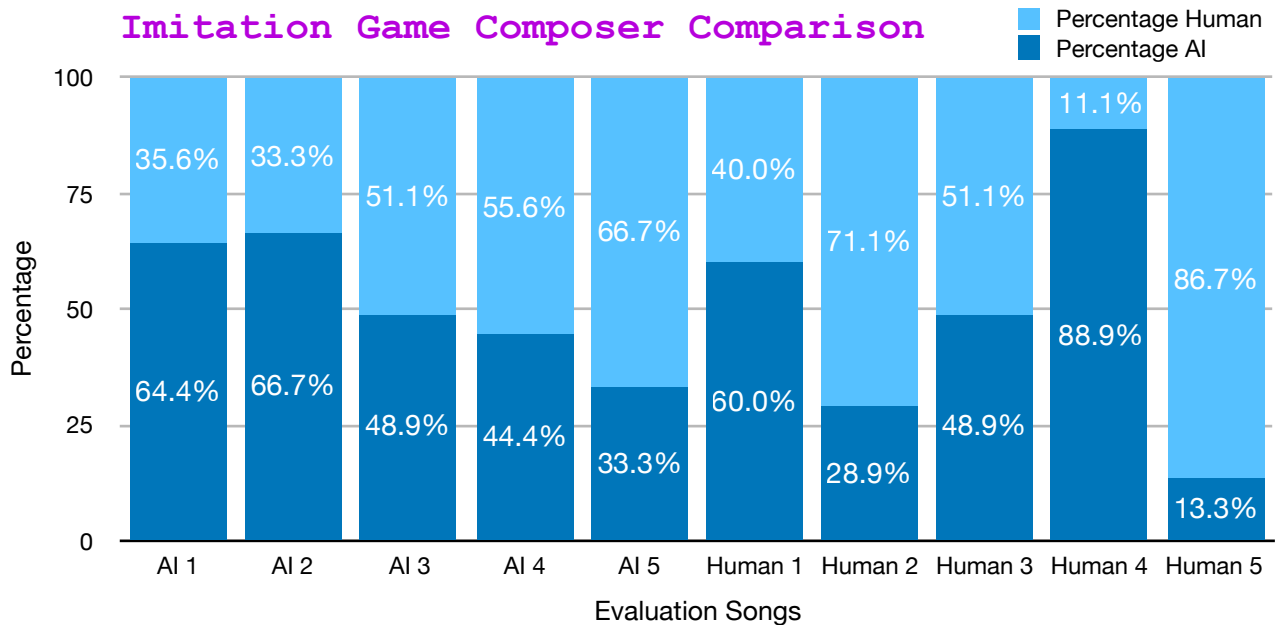
As mentioned in both '2. Literature Review' and '3. Methodology', the accuracy of the network is not a good metric to evaluate the network. Liu & Ramakrishnan (2014) were able to use the accuracy measurement as a means to quantitatively compare different optimisation methods. Subjectively, music is evaluated based upon people's personal preference, therefore people may or may not like the music composed by this project regardless of the accuracy metric of the network. In order to evaluate how well the network can compose music, the network's compositions should be compared to human compositions to evaluate how well the network can compose music compared to a human.

8.b. Subjective Evaluation

In order to evaluate how well the model can compose music like a human, a Turing style test known as an 'Imitation game' (Gunderson 1964) was used. An imitation game does not directly take into consideration how 'good' the music is, but instead how well it can lead a human into believing the composition was written by a human. An Imitation Game, informally known as a 'Turing Test', is a test of a machine's ability to exhibit intelligent behaviour similar to that of a human. Often an Imitation Game is performed such that a human subject is presented with a test object and has to deduce whether they believe object is controlled by a human or a computer. This method for evaluating the model is applicable because if the model can lead the subject into believing the composed music was written by a human, there is an implication that the model can compose music as well as a human. This method provides a means to empirically test how well a neural network can compose music like a human. The more able the project can compose music like a human, the better as it implies the music is more natural; thus, less of a machine.

The imitation game used to evaluate the project's success was performed by initially generating five new songs (Section 7. Music Generation) and then taking five randomly selected songs from the dataset, these songs can be found in appendix '11.a.1 Subjective Evaluation Songs'. The test was presented to 45 subjects who each had to go through all songs displayed in a random order. These test subjects ranged in profession and age in order to gain a wide scope, and

therefore, obtain more generalised results which could be extrapolated to general public. The subjects were given the choice of selecting human or computer for the song presented to them. For each song, these results give a ratio denoting how much people believe a song to be composed by a human or a computer. The success of which is based upon whether the majority of songs composed by this project are incorrectly identified as human, because a result like this implies the project is as good at composing music as a human.



(Figure 9)

The graph above (figure 9) depicts the results of the imitation game, where each column denotes the ratio that each song was voted for. One negative of this evaluation is that test songs may have been unintentionally biased, as subconsciously a lower standard of human songs could have been selected in order to deceive people into believing the human songs were composed by a computer in order gain better results. To prevent this bias, all songs were randomly chosen from a pool of all the songs in the data set.

As depicted in the graph above, only two songs were correctly concluded as being composed by this project (AI 1 & AI 2) and, incorrectly concluded that three songs composed by this project were instead composed by a human (AI 3, AI 4 & AI 5). Therefore, the majority of this project's songs used in the evaluation were able to convince the majority of test subjects that the composer was a human. These results are a success, as overall the network was able to exhibit intelligent

behaviour by composing music that sounded natural enough to convince people that it was a human. The songs used in the subjective evaluation can be found in appendix '11.a.1 Subjective Evaluation Songs'.

8.c. Analytical Results

The music generated by the project was assessed by university students studying music in order to gain an analytical and critical evaluation of the music, and by extent the capabilities of the network. These students were excluded from the subjective evaluation as it felt important to prevent these students from gaining any preconceived bias before an additional evaluation. It was found that the music resembled a 17th century style of music which was believed to be caused by the quartet instruments used in the project. The students pointed out structures in the network such as the 'trills' and the 'polyphonic nature of the music', both commonly present in classical music. The music also contained different responses in the music where the flutes would be the main instrument in the phrase and then in the next phrase the brass would respond to the flutes by becoming the main instrument and playing similar melodies.

The students were shown the project's songs used in section '8.b. Subjective Evaluation', in addition to the results of the imitation game. They theorised that people commonly believed a song to be identified as computer generated by the test subjects when the music displayed more polyphonic melodies. In particular, song 'AI 1' (Appendix '11.a.1 Subjective Evaluation Songs') started with a strong polyphonic and dissonant melody, thus appeared to lack harmony which people may have noticed. Therefore, people may have believed the lack of harmony meant the song couldn't be composed by a human.

8.d. Evaluation Conclusion

In conclusion, this project has been a success as all objectives outlined in section '1.b. Objectives' were met as the project was able to both generate music that encodes duration and pitch of a note for each instrument in a song. Additionally, the project has learnt to compose music so well that, for the majority of the tests, the subjects believed that the project's compositions were composed by

a human and could therefore be argued that the project can compose music that's as natural as music composed by a human.

This project has been an improvement to previous method such as Todd (1989) and Eck & Schmidhuber (2002) as this project was able to compose multi voiced music. Additionally, this project built upon the method for encoding duration outlined in Todd (1989) by providing multiple dimension for each semi-tone in the input vector, thus allowing for more encoded data. This encoded data enclosures attributes about the semi-tones such as whether the semi-tone is active, or whether the note has just begun.

Other products with similar capabilities to this project include arpeggiators (An Introduction To Arpeggiators, 2013). An arpeggiator is a function that automatically steps through a sequence of notes based on an input note. Some benefits of this project, compared to arpeggiator, are that this project can produce full songs rather than the fixed sequence of notes that arpeggiators produce. Additionally, an arpeggiator requires more skill to use, as an arpeggiator has parameters which need to be manually tuned in order to create the desired sequence. However, the disadvantages of this project compared to an arpeggiator are that, firstly, a vast amount of training data is required. Secondly, the network needs to train on that data, which can be very time consuming and may not produce the desired sequence. The implications of this technology are that a less skilled musician can produce more music, albeit less polished, than a skilled musician's finely tuned music.

8.e. Professional, Ethical, Social, Security and legal issues

One prominent issue in music is copyright. The current act is the 'Copyright, Designs and Patents Act 1988' (Copyright Witness Ltd. 2000). This gives the creators of musical works, artistic works or sound recordings the rights to control the ways in which their material may be used. This project requires music to train upon, which may cause a copyright issue as the copyright owner may not give permission for their works to be trained upon. The rights of the law encompass broadcast and public performance, copying, adapting, issuing, renting and lending copies to the public. The copyright holder may feel that their music is getting adapted as the network will learn based upon the training music in order to create

new music. The music in this project was originally composed by Bach, which no longer falls under copyright as the law states that the duration of the copyright last '70 years from the end of the calendar year in which the last remaining author of the work dies'. Bach died in 1750 which is 269 year after the current data (2019) and therefore no longer falls under copyright. However, others using this technology may infringe on the copyright law by using other artists work's.

Another legal issue that may arise is the legal implications of who owns the music generated by this project. The 'Copyright, Designs and Patents Act 1988' (Copyright Witness Ltd. 2000) gives the creators of the music legal protection against copyright, but, who is the creator? From a pedantic standpoint it is the computer that composed the music and should therefore own the legal rights of their music. However, it could be argued that the human that programmed the computer to compose the music should take legal responsibilities as a computer can not be held responsible for copyright, as the computer did not publish the music to the public or profit from the music.

As shown in section '8.b. Subjective Evaluation', human subjects struggled, and in most cases were not able to effectively differentiate human composed music compared to computer composed music. This may cause a social issue that if large scale music companies are able to produce music at a substantial rate then it will be harder for up and coming artists to become well known; as they will have to be exceptionally creative to to differentiate themselves from the computer-generated music. Additionally, there may be a need for less jobs as now music production can be automated which will further increase the skill required to make it in the industry.

9. Discussion

The Analytical Evaluation (8.c. Analytical Results) was assessed by students specifically studying music in order to give credibility to their opinions. These students concluded that the network was able to learn structures in the music such as 'trills', 'responcence' and the use of 'phrases'. Impressively, the network was able to learn these structures without being explicitly told about them, but instead inferred them from the data. Unfortunately, this project has to be completed within a fixed time frame, and could not experiment with different music genres, however, it would be interesting to see what the network can learn from other music styles, datasets and other amounts of instruments.

When listening to the data it was often clear from music produced by the project that the network would often learn specific structures or patterns. This can be heard when listening to the network whilst training (Appendix '11.a.3 Evolution of Composition Through Training'). These structures showed that the network was not necessarily learning to be creative but instead replicate the more common structures found across the data set. This lack of creativity may severely limit the applications of this technology, such that it may not be suitable for music production companies. To allow the network to appear more creative it could be possible to add random elements to the network, however this would need further investigation.

10. Conclusions and Further Work

10.a Conclusions

The network was able to produce the best results when configured in certain ways. The LSTM cells provided a 'memory' which enhanced the network's ability to predict notes through time compared to a standard Recurrent Neural Network (Eck & Schmidhuber 2002). Additionally, the network was shown to train fastest when optimised using resilient propagation (Liu & Ramakrishnan 2014). It was found that bottlenecking the network would generate more 'harmonic' music as the network would be forced to learn the strongest and most common structures from the data (6.c. Bottlenecking) and furthermore the sigmoid activation function was shown to outperform other activations functions when comparing the loss value (6.d. Activation). Finally, the encoded data was in the format outlined in '4. Data Representation and Feature Engineering'. These network configurations, when

trained upon the dataset (5.a. The Data Set), were able to generate music to the extent that, for the majority of test samples, test subjects were not able to differentiate the music composed by this project compared to music composed by Bach (8. Evaluation).

As the evaluation showed (8. Evaluation), the neural network in this project was able to compose multi-instrumental music to the extent that people believed the composer of the music was human. However, there were a few limitations. Firstly, the music produced is not creative; a neural network is a mathematical model where, whilst generating model, all weights are a fixed value taught from the data. Essentially this means that the music produced will only be mimicking songs from the training set instead of creating new music. This may potentially be an issue of not having enough training data or perhaps a limitation of using a mathematical model in an attempt to produce creative arts.

When reflecting on the motivation for this project, it would not be suitable to say that this technology would be applicable to the examples outlined in section '1.a. Motivation', as the music does not exhibit creativity. This technology might instead be more applicable to instances where a variation in generated music is required, such as procedural generation in games (Van Der Linden et al. 2014). This application for this project would be suitable, as a neural network could be trained for a specific scene and then generate suitable music when required, such as a fight scene. This technology benefits this area as it allows for a variation of content in the game's music as new music can be generated when required. This technology could be used save game publisher's money as they do not need to spend time composing all variations of a song used in a game, but instead train one model to generate the music.

In conclusion, the project has been a success as the research question 'Investigating a duration-based approach of time series prediction using machine learning for multi-instrumental music generation.' has been addressed and all objectives were met (1.b. Objectives). This project was able to generate multi-instrumental music allowing for complex durations that, additionally, this project was able to compose multi-instrumental music so remarkably well that in a blind

test people believed, in the majority of tests (8. Evaluation), the composer of the project's music to be human.

10.b. Further work

One important piece of data stored in MIDI is note velocity which is a measurement of how hard a note was pressed; a lower velocity means a softer note and a higher velocity means a harder, louder note. This value was not included in the project in order to apply more focus to other areas, such as duration representation. However, the velocity of a note could be encoded in the data using the 'Active-Start' method outlined in section '4.c. Duration Representation', by providing a third dimension per semi-tone which could encode the velocity of a note.

More work could be emphasised on the structure of a song, such that the dataset encodes what section of a song a note is from; such as, intro, verse or chorus'. This should allow the network to gain a greater sense of semantic information from the music, rather than relying on the network to infer structure from the data. This should create music with much greater structure and clearly defined phrases.

There could be more experimentation with other music genres of music, and the instruments encoded in the data. Additionally, experimentation could be done to encode vocals into the music to give a complete song rather than purely instrumental music, however this may be a very complex task.

10. References

Adam&EveDDB (1996) **Adam&EveDDB** [Online] www.adamandevddb.com
Available at: <http://www.adamandevddb.com>.

[Accessed: 14 March 2019]

Attack Magazine (2013). **An Introduction To Arpeggiators** [Online]
[www.attackmagazine.com](https://www.attackmagazine.com/technique/tutorials/an-introduction-to-arpeggiators/) Available at: <https://www.attackmagazine.com/technique/tutorials/an-introduction-to-arpeggiators/>.

[Accessed: 13 March 2019]

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R. and Kern, J., (2001)
Manifesto for Agile Software Development [Online] [Agilemanifesto.org](http://www.agilemanifesto.org). Available at: <http://www.agilemanifesto.org/>.

[Accessed: 28 February 2019]

Copyright Witness Ltd. (2000) **Fact sheet P-01: UK Copyright Law** [Online]
[www.copyrightservice.co.uk](https://www.copyrightservice.co.uk/copyright/p01_uk_copyright_law) Available at: https://www.copyrightservice.co.uk/copyright/p01_uk_copyright_law.

[Accessed 12 Mar. 2019]

Director, B., (2005). **What Mathematics Can Learn From Classical Music**
[Online] [archive.schillerinstitute.com](https://archive.schillerinstitute.com/fid_91-96/944_math_music.html) Available at: https://archive.schillerinstitute.com/fid_91-96/944_math_music.html

[Accessed: 11 February 2019]

Eck, D., & Schmidhuber, J., (2002) **A First Look at Music Composition using LSTM Recurrent Neural Networks**, Manno Switzerland, *Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale*

Grossman, D. J., (1997). **Dave's J.S. Bach Page - MIDI Files - Orchestral Suites**.
[Online] [www.jsbach.net](http://www.jsbach.net/midi/midi_orchestralsuites.html). Available at: http://www.jsbach.net/midi/midi_orchestralsuites.html [Accessed 20 Oct. 2018].

Gunderson, K., 1964. **The imitation game**. p.234-245. Oxford UK, Mind.

Hochreiter, S., Bengio, Y., Frasconi, P., and Schmidhuber, J., (2001). **Gradient flow in recurrent nets: the difficulty of learning long-term dependencies**. In Kremer, S. C. and Kolen, J. F., editors, *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE Press.

How Music Works (1997). **How Music Works** [Online] Available at: <http://www.howmusicworks.org/110/Sound-and-Music/Chromatic-Scale-Notes>. [Accessed: 15 February 2019]

Liu, I. T., & Ramakrishnan, B., (2014) **Bach In 2014: Music Composition with Recurrent Neural Network**, Pittsburgh USA, arXiv preprint arXiv:1412.3191.

Oord, A.V.D., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K., (2016). **Wavenet: A generative model for raw audio**. London UK, *arXiv preprint arXiv:1609.03499*.

Powell-Morse, A., (2016) **Waterfall Model: What Is It and When Should You Use It?** [Online] airbrake.io Available at: <https://airbrake.io/blog/sdlc/waterfall-model>. [Accessed: 15 March 2019]

Riedmiller, M., and Braun, H., (1992). **RPROP-A fast adaptive learning algorithm**. In *Proc. of ISCIS VII*, Universitat.

Skúli, S., (2017) **How to Generate Music using a LSTM Neural Network in Keras**. [Online] towardsdatascience.com Available at: <https://towardsdatascience.com/how-to-generate-music-using-a-lstm-neural-network-in-keras-68786834d4c5>. [Accessed 15 Oct. 2018]

TensorFlow (2015). **TensorFlow** [Online] www.tensorflow.org Available at: <https://www.tensorflow.org>. [Accessed: 13 March 2019]

Todd, P., (1981) **A Connectionist Approach to Algorithmic Composition**, California USA, *The MIT Press*.

Trunk, G.V., (1979). **A problem of dimensionality: A simple example**. p.306-307. *IEEE*.

Van Der Linden, R., Lopes, R., and Bidarra, R., (2014). **Procedural generation of dungeons**. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(1), pp.78-89.

Williams, J. (1995). **John Williams** [Online] www.johnwilliams.org Available at: <http://www.johnwilliams.org>.
[Accessed: 14 March 2019]

Zimmer, H. (2001). **Hans-Zimmer** [Online] www.hans-zimmer.com Available at: <http://www.hans-zimmer.com>.
[Accessed: 14 March 2019]

11. Appendices

11.a. Sample Audio

11.a.1 Subjective Evaluation Songs

AI1: <https://soundcloud.com/sam-garlick/ai-1?in=sam-garlick/sets/extracts-used-in-subjective-evaluation>
AI2: <https://soundcloud.com/sam-garlick/ai-2?in=sam-garlick/sets/extracts-used-in-subjective-evaluation>
AI3: <https://soundcloud.com/sam-garlick/ai-3?in=sam-garlick/sets/extracts-used-in-subjective-evaluation>
AI4: <https://soundcloud.com/sam-garlick/ai-4?in=sam-garlick/sets/extracts-used-in-subjective-evaluation>
AI5: <https://soundcloud.com/sam-garlick/ai-5?in=sam-garlick/sets/extracts-used-in-subjective-evaluation>
Human1: <https://soundcloud.com/sam-garlick/human-1?in=sam-garlick/sets/extracts-used-in-subjective-evaluation>
Human2: <https://soundcloud.com/sam-garlick/human-2?in=sam-garlick/sets/extracts-used-in-subjective-evaluation>
Human3: <https://soundcloud.com/sam-garlick/human-3?in=sam-garlick/sets/extracts-used-in-subjective-evaluation>
Human4: <https://soundcloud.com/sam-garlick/human-4?in=sam-garlick/sets/extracts-used-in-subjective-evaluation>
Human5: <https://soundcloud.com/sam-garlick/human-5?in=sam-garlick/sets/extracts-used-in-subjective-evaluation>

11.a.2 Bottle-neck Comparisons - Each network trained to 110 epochs

Bottleneck 50: <https://soundcloud.com/sam-garlick/bottleneck-size-50-110-epochs>
Bottleneck 150: <https://soundcloud.com/sam-garlick/bottleneck-size-150-110-epochs>
Bottleneck 250: <https://soundcloud.com/sam-garlick/bottleneck-size-250-110-epochs>
Bottleneck 350: <https://soundcloud.com/sam-garlick/bottleneck-size-350-110-epochs>
Bottleneck 450: <https://soundcloud.com/sam-garlick/bottleneck-size-450-110-epochs>

11.a.3 Evolution of Composition Through Training

10 Epochs: <https://soundcloud.com/sam-garlick/ai-evolution-10-epochs?in=sam-garlick/sets/evolution-of-composition>
20 Epochs: <https://soundcloud.com/sam-garlick/ai-evolution-20-epochs?in=sam-garlick/sets/evolution-of-composition>
30 Epochs: <https://soundcloud.com/sam-garlick/ai-evolution-30-epochs?in=sam-garlick/sets/evolution-of-composition>
40 Epochs: <https://soundcloud.com/sam-garlick/ai-evolution-40-epochs?in=sam-garlick/sets/evolution-of-composition>
50 Epochs: <https://soundcloud.com/sam-garlick/ai-evolution-50-epochs?in=sam-garlick/sets/evolution-of-composition>
60 Epochs: <https://soundcloud.com/sam-garlick/ai-evolution-60-epochs?in=sam-garlick/sets/evolution-of-composition>
70 Epochs: <https://soundcloud.com/sam-garlick/ai-evolution-70-epochs?in=sam-garlick/sets/evolution-of-composition>
80 Epochs: <https://soundcloud.com/sam-garlick/ai-evolution-80-epochs?in=sam-garlick/sets/evolution-of-composition>
90 Epochs: <https://soundcloud.com/sam-garlick/ai-evolution-90-epochs?in=sam-garlick/sets/evolution-of-composition>
100 Epochs: <https://soundcloud.com/sam-garlick/ai-evolution-100-epochs?in=sam-garlick/sets/evolution-of-composition>
110 Epochs: <https://soundcloud.com/sam-garlick/ai-evolution-110-epochs?in=sam-garlick/sets/evolution-of-composition>

11.b. LSTM Glossary Definition

Term	Definition
σ	Sigmoid function
\tanh	Hyperbolic tan function
x_t	Input Values to the current time step
h_t	The hidden state in the current time step
h_{t-1}	The hidden state in the previous time step
f_t	Output value from the forget gate for the current time step
i_t	Output value from the input gate for the current time step
o_t	Output value from the output gate for the current time step
C_t	Output value from the Candidate gate for the current time step
C_{t-1}	Output value from the output gate from the previous time step
\tilde{C}_t	The intermediary candidate value
W_f	The weights for the forget gate
W_i	The weights for the input gate
W_o	The weights for the output gate
W_c	Weights for the candidate gate
b_f	The bias value in the forget gate
b_i	The bias value in the input gate
b_o	The bias value in the output gate
b_c	The bias value in the candidate gate

11.c. Detailed Project Proposal

11.a.1 Defining the project

11.a.1.1 Detailed Research Problem

Can a Neural Network learn and compose a multi-instrumental score of Orchestral Music?

11.a.1.2 Keywords

Music; Composition; Neural Network; Time-Series;

11.a.1.3 Project Title

Multi-Instrumental Composition Generated By A Neural Network.

11.a.1.4 Client, Audience and Motivation

This project is important to anyone working in the music industry, who could use this technology to create music for an artist. For example the musician, the producers, the record labels and to an extent the listener, entities such as Sony Records, Ultra Music, or musicians such as Hans Zimmer. This technology could allow musicians to generate whole songs for their music. This will benefit the Record labels and produces by cutting down time for a musician to make a song, allowing record labels and musicians to sell more songs, in turn producing more money.

11.a.1.5 Project Plan and Risk Management

This will be done by collecting training data, in the form of MIDI files, which will need converting into data to feed into a Time-Series Recurrent Neural Network. Once the network has been trained, it will be tested using the network to output data that will be converted to MIDI files which, then be converted into Audio using software like Logic Pro X. As there is no Empirical way to judge music, the outputted audio will be judged based on how well the music is both structured and rhythmic. As mentioned in 1.1, detailed research problem the music style the project will be trained on will be orchestral, this is due to orchestral being particularly structural and having a dearth of songs dating back centuries for the project to train on.

Both the project plan and the risk management can be found in detail later on in the document, under the Appendices 1 and 2. Each week of the project will be broken down in to essentially sprints, similar to that of Agile sprints, to ensure the project keeps to the outlined plan. All work completed

by the end of the sprint will be backed up to prevent loss of data as explained in Appendix 2, Risk management.

11.a.2.0 Mini Literature Review

11.a.2.1 Abstract

Humans have predominantly been the soul composers of music; although a lot of people choose to make their music using computers, it is still the human composing, not the computer. Many papers have been published where a Neural Network has been able to compose music however, this technology has been used only to compose a single instruments worth of music. This project should change this such that a Neural Network can compose multiple scores of music to be played by different instruments at a time.

11.a.2.2 Initial Literature Review

Todd (1989) was one of the first papers to produce music using machine learning, the paper did so by producing a reasonably simple feed forward neural network with the outputs feeding back into the inputs of the network. The feedback loop was introduced to give what the paper referenced as 'memory', enabling the network to know what it had previously produced enabling it to introduce rhythm. Rhythm is inherently reasonably repetitive meaning it needs this memory loop to enable it to produce music.

This use of a feedback loop restricted the network as it prevented the network from having strong structure. This was solved in Eck & Schmidhuber (2002) which introduced the use of LSTM enabling the network to learn structure as well as rhythm allowing the network to compose songs rather than just melodies. Liu & Ramakrishnan (2014) then build upon this further by introducing resilient propagation rather than back propagation through time to prevent the issues of vanishing gradients allowing the network to train faster.

The paper 1, 2 and 3 all split up music into time frames enabling the network to work like a time-series network where you predict the next time frames values from the previous frames. The 'time frames' are a list of

inputs for the network, each index of the list corresponds to a note on the musical scale, if the input is one the note is being pressed, if the input is 0 the input is off.

These papers very closely map to the this project as it will be building from these papers creating not a single score of music for one instrument but instead multiple scores of music for multiple instruments. One issue that each of these papers talked about but didn't entirely solve was how to tell if a note carries on playing between time frames or if instead there are two separate notes. This is not something this project intends to solve.

The papers differ in a few areas, for example in Todd (1989) the author had transposed all the music to the same key, as with Eck & Schmidhuber (2002). However Liu & Ramakrishnan (2014) did not transpose the music, but did mention why it could be useful. In this project all input MIDI data will be transposed to the same key, as it should help the network train faster and build stronger connections between different keys and key changes which should enable the network to create MIDI data with more rhythm in it's composition. Transposing the training data to one key may create an issue, such that input data in a different key to that of key the network was trained on may produce unexpected results, such as the network forcing the music to the key that it was training in or creating music with a varying key.

Todd (1989) uses feed forward neural network with the output layer feeding back into the inputs to add a level of memory, which by today's standard is reasonably outdated. Eck & Schmidhuber (2002) & Liu & Ramakrishnan (2014) both outdate Todd (1989) using newer machine learning techniques, like deeper networks and LSTM blocks, diminishing the relevance of Todd (1989). However, a lot of the techniques Todd (1989) outlined such as, each key mapping to a specific input to the network rather than the transitions between semitones (+1, +5, -7) mapping to the input, are still relevant. Techniques outlined in Todd (1989) are still relevant and being used by more recent papers, this project will be completed using these techniques.

11.a.2.3 Relevant Professional, Social, Ethical, Security & Legal Issues

The main issues that encompass this project are ethical issues; such that if this project can entirely replicate scores of music for multiple instruments, then this project could completely replace human musicians. What is to stop a musician from using this software to create music and then selling the music as if they wrote it?

Social issues are small, but related to how the previous issue, such that if this project can create music, will someone know whether the music they're listening to be created by humans or machines?

Security issues are essentially non-existent as no personal data of any kind is stored or processed,.

Professional and legal issues are minor; worst case scenario is a person opposed to machine-made music accidentally purchases a machine-made song and wants compensation.

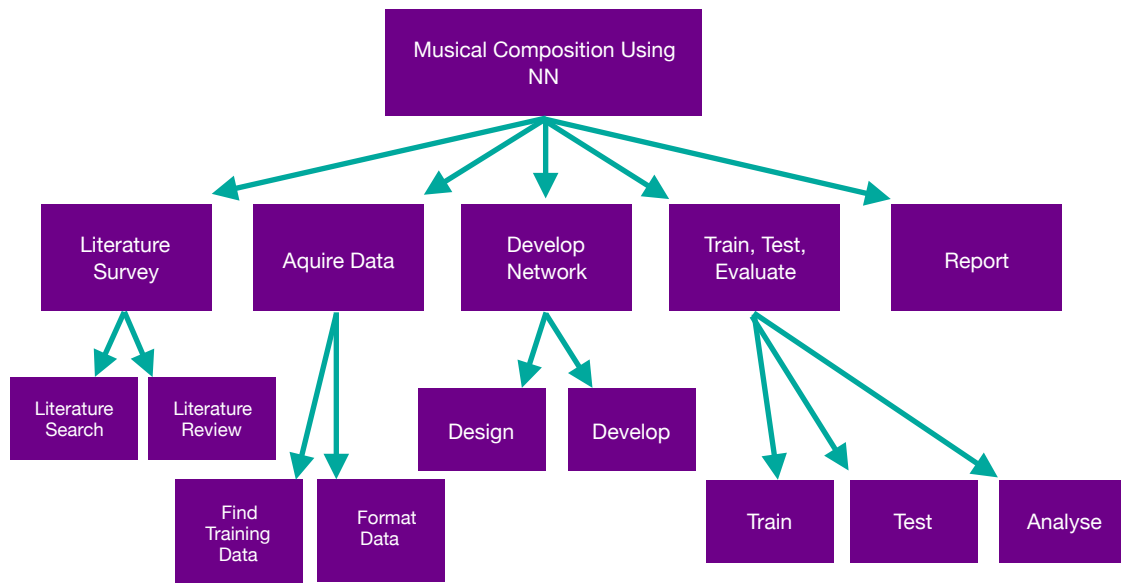
The training data for the project will all have to be sourced from non-copyrighted sources for free MIDI files as to prevent copyright infringement.

11.a.2.4 Bibliography

- [1] A Connectionist Approach to Algorithmic Composition, Peter M Todd, 1989
- [2] A First Look at Music Composition using LSTM Recurrent Neural Networks, Douglas Eck, Jürgen Schmidhuber, 2002
- [3] BACH IN 2014: MUSIC COMPOSITION WITH RECURRENT NEURAL NETWORK, I-Ting Liu, Bhiksha Ramakrishnan, 2014
- [4] Neural Networks for Time Series Processing, Georg Dorffner, 1996

11.a.3. Appendix

11.a.3.1. Work Breakdown



(Figure 10)

11.a.3.2. Time Estimates

Literature Survey (5 Weeks)

- Literature Search (1 Week)
- Literature Review (4 Week)

Aquire Data (3 Weeks)

- Find Training Data (2 Week)
- Format Training Data (1 Week)

Develop Network (4 Weeks)

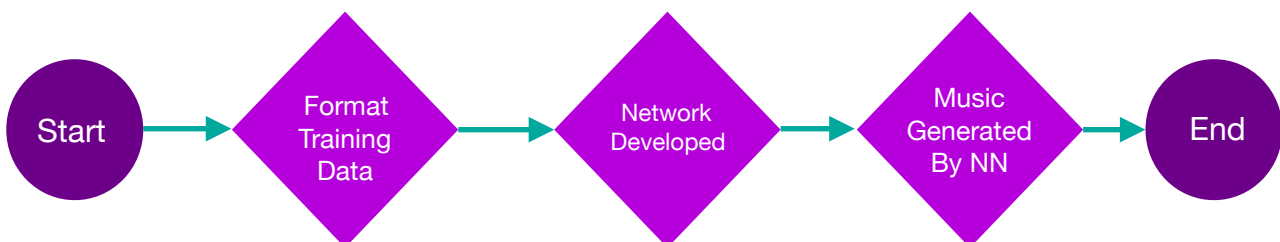
- Design Network (1 Weeks)
- Develop Network (3 Weeks)

Train, Test, Evaluate (3 Weeks)

- Train (1 Weeks)
- Test (1 Weeks)
- Evaluate (1 Weeks)

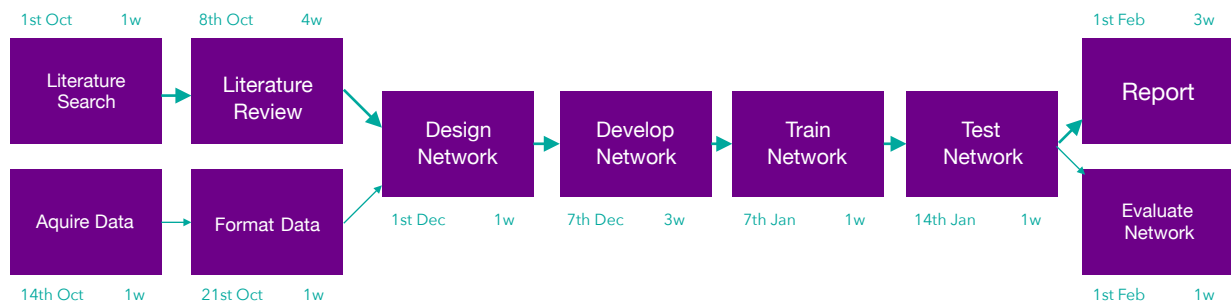
Report (3 Weeks)

11.a.3.3. Milestone Identification



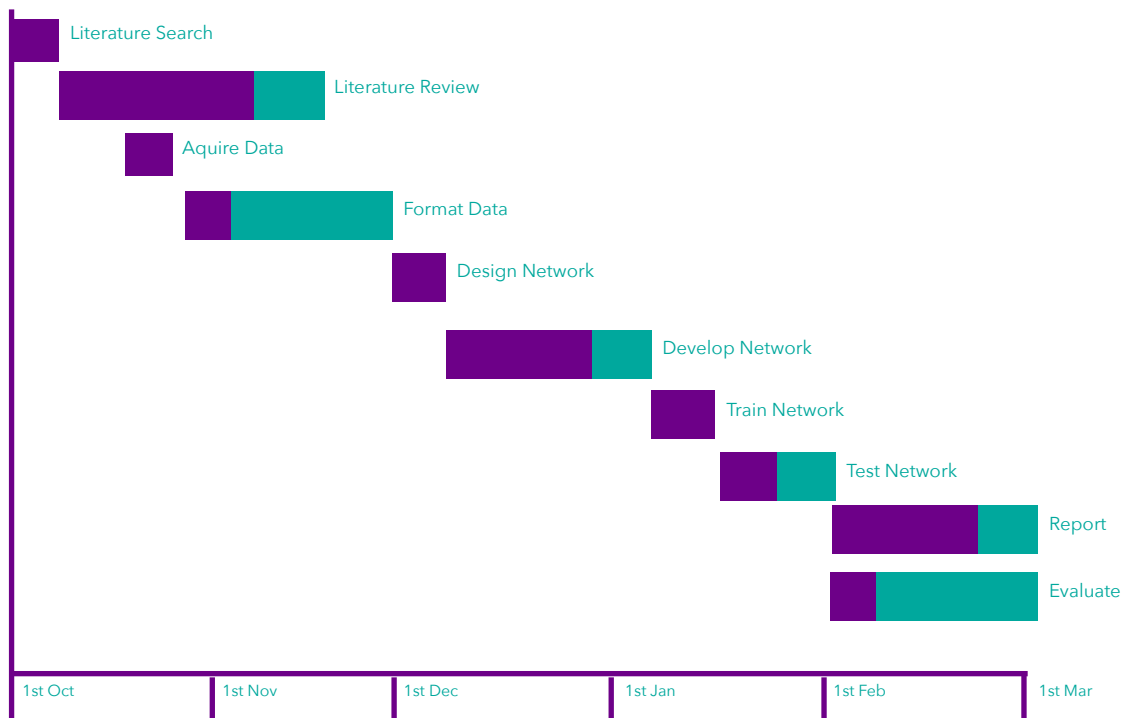
(Figure 11)

11.a.3.4. Activity Sequence



(Figure 12)

11.a.3.5. Scheduling



(Figure 13)

11.a.4. Risk Management

11.a.4.1. Identifying Risks

- 1. Losing Data**, Data acquired to train the network, becomes compromised meaning causing the loss of data meaning training data will need to be re-acquired.
- 2. Formatting Data**, The data acquired for this project will come from various sources and may not all be formatted in the same way, which could cause issues that may cause delays while the data is formatted.
- 3. Project doesn't produce intended results**, If the project doesn't produce intended results, the network may need to be redesigned which will extend the expected completion time.

4. Project takes longer than anticipated, Time estimates may have been underestimated causing more work required to complete the project as the same amount of work will need completing in a smaller time frame.

5. Computer Fails, If the computer that runs the project fails, all project completed on said computer may have to be redone.

6. Illness, Getting ill while working on project may cause delays or cause work to be produced to a lower standard.

11.a.4.2. Assess Impact of Risks

	Negligible	Minor	Moderate	Significant	Severe
Very Likely					
Likely			6		
Possible		4		2	
Unlikely			3	1	
Very Unlikely					5

(Figure 14)

11.a.4.3. Alleviating Critical Risks

After plotting the aforementioned risks there does not appear to be any critical risks.

11.a.4.4. Control Risks

There will be various precautions and contingency plans to prevent risks from affecting the project.

Firstly at the end of each, a back up of all data will be taken, with a time stamp saying when the data was backed-up, thus keeping everything in chronological order so that in an event where backed-up data is required, only the most recent data is used.

Each backup of data taken will be saved across multiple places, these places include the computer the project is being created on as well as Google Drive and

Drop Box, so that in the event of a computer failure data can be recovered. Back-ups will be saved to two separate Cloud services so that in the event that one of the services goes offline, data can be recovered from the other service.

All risks will be monitored weekly to prevent risks getting out of hand, for example if one particular processes takes longer than anticipated, then the project plan may need adjusting.

Formatting data may take longer than expected which is why I've designated a large portion of time for that process.

11.d. Project Log

2018	
22nd Sep. - 27th Sep.	Collection of research
29th Sep. - 3rd Oct.	Literature research
4th Oct	First supervisor meeting - Introduced concept, discussed ideas
4th Oct. - 19th Oct	literature research
12th Oct.	Supervisor meeting - Further discussion of ideas
13th Oct. - 19th Oct	Worked on detailed project proposal
19th Oct	Supervisor meeting - Discussion of training set and feature engineering
20th Oct.	Obtained training data
21st Oct.	Submitted detailed project proposal
22nd Oct. - 25th Oct.	background research
26th Oct.	Supervisor meeting - Discussion of network configuration
27th Oct. - 1st Nov.	background research
2nd Nov.	Supervisor meeting - Further discussion of network configuration
3rd Nov. - 8th Nov.	Development of program to convert MIDI to CSV according to feature representation
9th Nov.	Supervisor meeting - Further discussion of network configuration
10rd Nov. - 15th Nov.	Further development of program to convert MIDI to CSV according to feature representation
16th Nov.	Supervisor meeting - Discussion of machine learning framework
17th Nov. - 3rd Dec	Further research into machine learning frameworks
4th Dec.	Meeting with head of department for advice regarding recurrent network configuration
7th Dec.	Supervisor meeting - Discussion of machine learning framework
8th Dec. - 13th Dec.	<i>Exam Revision</i>
14th Dec.	Last meeting with supervisor for semester
15th Dec.	Exams
16th Dec. - 31st Dec.	Further research
2019	
2nd Jan. - 13th Jan.	Initial Development of network model
14th Jan. - 26th Jan.	Initial literature review and further development
27th Jan. - 7th Feb.	Project experimentation
8th Feb.	Supervisor meeting - discussing work over Christmas break

9th Feb. - 12th Feb.	Project experimentation and initial work on report
13th Feb.	Supervisor meeting - discussing experimentation
14th Feb. - 18th Feb.	Project experimentation whilst working on report
20th Feb.	Supervisor meeting - discussing experimentation
21st Feb. - 26th Feb.	Further experimentation whilst working on report
27th Feb.	Supervisor meeting - discussing further experimentation
28th Feb. - 6th Mar.	Finish experimentation whilst working on report
7th Mar.	Setup subjective evaluation
8th Mar.	Supervisor meeting - report feedback
10th Mar.	Complete subjective evaluation
10th Mar. - 12th Mar.	Working on report
13th Mar.	Supervisor meeting - report feedback
14th Mar	Working on report
15th Mar.	Final supervisor meeting - Final report amendments
16th - 20th Mar.	Finishing report

11.e. Ethics Review Form E1

This form should be completed by the Principal Investigator / Supervisor / Student undertaking a research project which involves human participants. The form will identify whether a more detailed E2 form needs to be submitted to the Faculty Research Ethics Officer.

Before completing this form, please refer to the University **Code of Practice for the Ethical Standards for Research involving Human Participants**, available at <http://www.brookes.ac.uk/Research/Research-ethics/>, and to any guidelines provided by relevant academic or professional associations.

It is the Principal Investigator / Supervisor who is responsible for exercising appropriate professional judgement in this review. Note that all necessary forms should be fully completed and signed before fieldwork commences.

Project Title: Multi-Instrumentalist Composition using Neural Networks

Principal Investigator / Supervisor: Peter Marshall p0068193

Student Investigator: Sam Garlick 16069781

		Yes	No
1.	Does the study involve participants who are unable to give informed consent? (e.g. children, people with learning disabilities, unconscious patients)		✓
2.	If the study will involve participants who are unable to give informed consent (e.g. children under the age of 16, people with learning disabilities), will you be unable to obtain permission from their parents or guardians (as appropriate)?		✓
3.	Will the study require the cooperation of a gatekeeper for initial access to groups or individuals to be recruited? (e.g. students, members of a self-help group, employees of a company, residents of a nursing home)		✓
4.	Are there any problems with the participants' right to remain anonymous, or to have the information they give not identifiable as theirs?		✓
5.	Will it be necessary for the participants to take part in the study without their knowledge/consent at the time? (eg, covert observation of people in non-public places?)		✓
6.	Will the study involve discussion of or responses to questions the participants might find sensitive? (e.g. own drug use, own traumatic experiences)		✓
7.	Are drugs, placebos or other substances (e.g. food substances, vitamins) to be administered to the study participants?		✓
8.	Will blood or tissue samples be obtained from participants?		✓
9.	Is pain or more than mild discomfort likely to result from the study?		✓
10.	Could the study induce psychological stress or anxiety?		✓

11.	Will the study involve prolonged or repetitive testing of participants?		✓
12.	Will financial inducements (other than reasonable expenses and compensation for time) be offered to participants?		✓
13.	Will deception of participants be necessary during the study?		✓
14.	Will the study involve NHS patients, staff, carers or premises?		✓

If you have answered 'no' to all the above questions, send the completed form to your Module Leader and keep the original in case you need to submit it with your work.

If you have answered 'yes' to any of the above questions, you should complete the Form E2 available at <http://www.brookes.ac.uk/Research/Research-ethics/Ethics-review-forms/> and, together with this E1 Form, email it to the Faculty Research Ethics Officer, whose name can be found at <http://www.brookes.ac.uk/Research/Research-ethics/Research-ethics-officers/>

If you answered 'yes' to any of questions 1-13 and 'yes' to question 14, an application must be submitted to the appropriate NHS research ethics committee.

Signed:

Principal Investigator /
Supervisor

Signed:

Student Investigator



Date: 28/09/2018

11.f. Source Code

11.d.1. Neural Network Code

```
#####--project.py--#####
import MidiCSVReader as reader
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, CuDNNLSTM
import random

sequence_length = 100
output_sequence_length = 2000

epochOfEpochs = 20
epochsPerEpochOfEpochs = 10

def saveFile(defaultInputs, fileLocation):
    print("Saving File: " + fileLocation)
    full_prediction_sequences = np.zeros((0, len(data[0])))
    current_prediction_sequences = defaultInputs

    for i in range(0, output_sequence_length):
        reshaped = np.reshape(current_prediction_sequences, (1, sequence_length,
len(data[0])))
        prediction = model.predict(reshaped)
        for p in range(len(prediction[0])):
            if prediction[0][p] < 0.5:
                prediction[0][p] = 0
            else:
                prediction[0][p] = 1

        full_prediction_sequences = np.vstack((full_prediction_sequences,
prediction))
        current_prediction_sequences = np.vstack((current_prediction_sequences,
prediction))

        current_prediction_sequences = np.delete(current_prediction_sequences,
(0), axis=0)

#    print("Saving Data")
    output_csv = open(fileLocation, "w+")
    for i in range(0, full_prediction_sequences.shape[1]):
        line = ""
        for j in range(0, full_prediction_sequences.shape[0]):
            if len(line) > 0:
                line = line + ", "
            line = line + str(full_prediction_sequences[j][i])
        output_csv.write(line + "\n")
    output_csv.close()
    print("Saved File ")

print("Loading Data")
data = reader.loadAllData("C:/Users/Sam/Google Drive/Dissertation/Data/2 Encoded
CSVs/")

print("Creating Sequences")
inputs = []
outputs = []
for i in range(0, len(data) - sequence_length):
    seq_in = data[i : i + sequence_length]
    seq_out = data[i + sequence_length]
    inputs.append(seq_in)
```

```

        outputs.append(seq_out)

print("Format Data For LSTMs")
inputs = np.reshape(inputs, (len(inputs), sequence_length, len(data[0])))

print("Reformatted to: ")
print(inputs.shape, "at", (inputs.size * inputs.itemsize)/10e8, "Gb")

layerSize = 500
print("Building Model")
model = Sequential()

model.add(CuDNNLSTM(450, input_shape=(inputs.shape[1:]), return_sequences=True))
model.add(Dropout(0.4))

model.add(CuDNNLSTM(450, return_sequences=True))
model.add(Dropout(0.4))

model.add(CuDNNLSTM(450, return_sequences=True))
model.add(Dropout(0.4))

#model.add(CuDNNLSTM(layerSize, return_sequences=True))
#model.add(Dropout(0.4))

model.add(CuDNNLSTM(450))
model.add(Dropout(0.4))

model.add(Dense(450))
model.add(Dropout(0.3))

model.add(Dense(len(data[0]), activation='sigmoid'))

print("Compile Model")
model.compile(optimizer='rmsprop',
              loss='mse',
              metrics=['accuracy'])

print(model.summary())

print("Training")
outputLocation = "C:/Users/Sam/Google Drive/Dissertation/Data/3 Maia Outputs/
bottleTest-450-" #.csv

#model.load_weights("full.h5")
#model.load_weights("test.h5") #8
for i in range(0, epochOfEpochs):
    print("loading weights")
    model.fit(np.array(inputs), np.array(outputs), epochs=epochsPerEpochOfEpochs)
    # saveFile(np.random.rand(sequence_length,len(data[i])), outputLocation +
    str(i) + "-A.csv")
    inputData = inputs[random.randint(0,len(inputs)-1)]
    saveFile(inputs[0], outputLocation + str(i) + ".csv")
    print("saving weights")
    model.save_weights("test.h5")

#saveFile(np.random.rand(sequence_length,len(data[0])), outputLocation + str(0) +
"-A.csv")

print("fin")

```



```
#####-MidiCSVReader.py--#####
import numpy as np
import os

def loadCSV(csvFileLocation):
    #load raw CSV
    rawCsvMatrix = []
    for line in open(csvFileLocation):
        values = np.array(line.split(", "))
        #inputs are only 1 & 0 so do not need percision
        floatValues = values.astype(np.int8)
        rawCsvMatrix.append(floatValues)

    #transform
    fullMatrix = []

    for y in range(0, len(rawCsvMatrix[0])):
        column = []
        for x in range(0, len(rawCsvMatrix)):
            column.append(rawCsvMatrix[x][y])

        fullMatrix.append(np.array(column))

    return fullMatrix

def loadAllData(folderPath):
    CSVs = os.listdir(folderPath)

    allData = []
    for csv in CSVs:
        path = folderPath + os.path.basename(csv)
        print("Loading:",path)
        csv = loadCSV(path)

        for i in range(0, len(csv)):
            allData.append(csv[i])
        print(len(csv))

    print(len(allData))
    return allData
```

11.d.2. MIDI Convertor

```

////////-MIDIConvertor.java-////////
package midi.convertor;

import java.io.File;

public class MIDIConvertor {
    public static final int NOTE_ON = 0x90;
    public static final int NOTE_OFF = 0x80;

    public static final int LOWER_NOTE = /*c*/1;
    public static final int UPPER_NOTE = /*c*/6;
    public static final int LOWEST_BOUND = -2; //c-2

    public static final int DEFAULT_VELOCITY = 70;

    public static void main(String[] args) throws Exception {
        //Convert Midi to CSV
        File[] fils = (new File("mid").listFiles());
        for (File f : fils){
            if (f.getName().endsWith(".mid")){
                System.out.println(f.getName());
                new Convertor("mid/"+f.getName());
            }
        }

        //Convert CSV to Midi
        String outputFolder = "";
        for (File f : (new File(outputFolder).listFiles())) {
            if (f.getAbsolutePath().toString().endsWith(".csv")) {
                new EncodeToMIDI(f.getAbsolutePath(), "output-"+f.getName()
+ ".mid");
            }
        }
        //    new EncodeToMIDI(outputFolder+"k3.csv", "output.mid", Format.ActiveOn);

        //Song Matrix should be 576, tracklength;
        /*
        octaves = 6
        tracks = 4
        semitones = 12

        6 * 4 * 12 = 288
        288 * 2 = 576 //(Add the row for if a note is on or off)

        */
    }
}

```

```

//////////EncodeToMIDI.java-//////////
package midi.convertor;

import java.io.BufferedReader;
import java.io.File;
import javax.sound.midi.MidiSystem;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import javax.sound.midi.InvalidMidiDataException;
import javax.sound.midi.MidiEvent;
import javax.sound.midi.Sequence;
import javax.sound.midi.ShortMessage;
import javax.sound.midi.Track;

public class EncodeToMIDI {

    private HashMap<Integer, Boolean> keysActive;

    public EncodeToMIDI(String in, String out) throws FileNotFoundException,
IOException, InvalidMidiDataException, Exception {
        System.out.print(in + " ");
        Sequence seq = new Sequence(Sequence.PPQ, 4);
        ArrayList<ArrayList<Float>> songMatrix = convertFileToMatrix(in);

        int trackSize = songMatrix.size() / 4;
        for (int trackNo = 0; trackNo < 4; trackNo++) {
            Track t = seq.createTrack();
            ArrayList<ArrayList<Float>> keys = getSongMatrixRange(songMatrix,
trackNo * trackSize, (trackNo + 1) * trackSize);
            if (trackNo != 0) {
                System.out.print(", ");
            }
            System.out.print(decodeTrack(t, keys, trackNo));
        }
        System.out.println("");
        MidiSystem.write(seq, 1, new File(out));
    }

    private ArrayList<ArrayList<Float>>
getSongMatrixRange(ArrayList<ArrayList<Float>> arr, int s, int e) {
        ArrayList<ArrayList<Float>> range = new ArrayList<>();
        for (int i = s; i < e; i++) {
            range.add(arr.get(i));
        }
        return range;
    }

    private ArrayList<ArrayList<Float>> convertFileToMatrix(String location)
throws FileNotFoundException, IOException {
        ArrayList<ArrayList<Float>> songMatrix = new ArrayList<>();
        File f = new File(location);
        if (f.exists()){
            BufferedReader br = new BufferedReader(new FileReader(f));
            String line = "";
            while ((line = br.readLine()) != null) {
                if (line.length() > 5) {
                    ArrayList<Float> row = new ArrayList<>();
                    String[] tokens = line.split(",");
                    for (String s : tokens) {
                        s = s.replaceAll(" ", "");
                        row.add(Float.parseFloat(s));
                    }
                    songMatrix.add(row);
                }
            }
        }
        return songMatrix;
    }
}

```

```

        private String decodeTrack(Track t, ArrayList<ArrayList<Float>> notes, int
trackNo) throws InvalidMidiDataException, Exception {
            return decodeActiveOn(t, notes, trackNo);
        }

        private String decodeActiveOn(Track t, ArrayList<ArrayList<Float>> notes, int
trackNo) throws InvalidMidiDataException {
            int notesCount = 0;
            //Refresh HashMap
            keysActive = new HashMap<>();
            for (int i = 0; i < notes.size()/2; i++) {keysActive.put(i,
Boolean.FALSE);}

            for (int seqNo = 0; seqNo < notes.get(0).size(); seqNo++) {
//                System.out.println("Track: " + trackNo + " Sequence#:" + seqNo);
                for (int keyPair = 0; keyPair < notes.size(); keyPair = keyPair + 2)
                {
                    //int seqNo = seqNo;
                    int keyNo = (keyPair / 2);
                    int key = keyNo + 36;
                    boolean keyActive = notes.get(keyPair).get(seqNo) > 0.5;
                    boolean keyStart = notes.get(keyPair + 1).get(seqNo) > 0.5;

                    //Decide whether to turn on or off
                    boolean turnKeyOff = false;
                    boolean turnKeyOn = false;

                    if (!keyActive && !keyStart) {
                        //Both bit off (00)
                        if (keysActive.get(keyNo)) {
                            turnKeyOff = true;
                        }
                    }

                    } else if (keyActive && !keyStart) {
                        //Key Active But No Start (10)
                        if (!keysActive.get(keyNo)) {
                            turnKeyOn = true;
                        }
                    }

                    } else if (keyStart) {
                        //Key not active, but is start (01) ==> (Techically an error
but is same as new key starting)
                        //Or key is started (11)
                        if (keysActive.get(keyNo)) {
                            turnKeyOff = true;
                        }
                        turnKeyOn = true;
                    }
                }

                //Turn Nots on And Off
                if (turnKeyOff) {
                    keysActive.put(keyNo, Boolean.FALSE);
                    ShortMessage off = new ShortMessage();
                    off.setMessage(ShortMessage.NOTE_OFF, 0, key,
MIDIConvertor.DEFAULT_VELOCITY);
                    t.add(new MidiEvent(off, (long) seqNo));
                }
                if (turnKeyOn) {
                    notesCount++;
                    keysActive.put(keyNo, Boolean.TRUE);
                    ShortMessage on = new ShortMessage();
                    on.setMessage(ShortMessage.NOTE_ON, 0, key,
MIDIConvertor.DEFAULT_VELOCITY);
                    t.add(new MidiEvent(on, (long) seqNo));
                }

                //ADD KEY PAIR STUFF HERE MAJIK
            }
        }

```

```

        for (Integer i : keysActive.keySet()) {
            if (keysActive.get(i)){
                keysActive.put(i, Boolean.FALSE);
                ShortMessage off = new ShortMessage();
                off.setMessage(ShortMessage.NOTE_OFF, 0, i+24,
MIDIConvertor.DEFAULT_VELOCITY);
                t.add(new MidiEvent(off, (long) notes.get(0).size()));
            }
        }
//        ShortMessage on = new ShortMessage();
//        on.setMessage(ShortMessage.NOTE_ON, 0, 24, 70);
//
//        ShortMessage off = new ShortMessage();
//        off.setMessage(ShortMessage.NOTE_OFF, 0, 24, 70);
//
//        t.add(new MidiEvent(on, (long) 0));
//        t.add(new MidiEvent(off, (long) 13.5));
        return Integer.toString(notesCount);
    }
}

```

```

//////////-Convertor.java-//////////
package midi.convertor;

import java.io.File;
import javax.sound.midi.MidiEvent;
import javax.sound.midi.MidiMessage;
import javax.sound.midi.MidiSystem;
import javax.sound.midi.Sequence;
import javax.sound.midi.ShortMessage;
import javax.sound.midi.Track;
import static midi.convertor.MIDIConvertor.NOTE_OFF;
import static midi.convertor.MIDIConvertor.NOTE_ON;
import static midi.convertor.MIDIConvertor.LOWER_NOTE;
import static midi.convertor.MIDIConvertor.UPPER_NOTE;

/**
 *
 * @author Sam
 */
public class Convertor {
    public Convertor(String filename) throws Exception{
        SongMatrix sm = new SongMatrix();

        Sequence sequence = MidiSystem.getSequence(new File(filename));
        int trackLength =
convertTickToQuantizedBeatUp(getTrackLength(sequence.getTracks()));

        int trackNumber = 0;
        for (Track track : sequence.getTracks()) {
            if (!isEmpty(track)){
                TrackMatrix trackMatrix = buildTrackMatrix(trackLength);
                parseTrack(track, trackMatrix);
//                System.out.println(trackMatrix.getMatrix().length);
                trackNumber++;
                sm.addTrack(trackMatrix);
            }
        }
        System.out.println(sm.songMatrix[0].length);

        sm.saveCSV("out/"+filename+".csv");
    }

    static void parseTrack(Track track, TrackMatrix trackMatrix){
        for (int i=0; i < track.size(); i++) {
            MidiEvent event = track.get(i);
            parseEvent(event, trackMatrix);
        }
    }

    static void parseEvent(MidiEvent event, TrackMatrix trackMatrix){
        MidiMessage message = event.getMessage();
        if (message instanceof ShortMessage) {
            ShortMessage sm = (ShortMessage) message;
            if (sm.getCommand() == NOTE_ON) {
                int key = sm.getData1();
                trackMatrix.noteOn(key, event.getTick());
            } else if (sm.getCommand() == NOTE_OFF) {
                int key = sm.getData1();
                trackMatrix.noteOff(key, event.getTick());
            }
        }
    }

    static TrackMatrix buildTrackMatrix(int length){
        int arrayHeight = (UPPER_NOTE - LOWER_NOTE) * 12 /*semitones*/;

        TrackMatrix midiArray = new TrackMatrix(length, arrayHeight);
        return midiArray;
    }

    static long getTrackLength(Track[] tracks){
        long trackLength = 0;

```

```

        for (Track track : tracks){
            trackLength = Math.max(trackLength, track.ticks());
        }
        return trackLength;
    }

    public static int convertTickToQuantizedBeatDown(long length){
        float divisor = 60;//120 ticks in beat, 60 gives half beats, 30 gives
quarter
        return (int)Math.floor((float)length/divisor);
    }

    public static int convertTickToQuantizedBeatUp(long length){
        float divisor = 60;//120 ticks in beat, 60 gives half beats, 30 gives
quarter
        return (int)Math.ceil((float)length/divisor);
    }

    private static boolean isEmpty(Track track){
        boolean empty = true;
        for (int i=0; i < track.size(); i++) {
            MidiEvent event = track.get(i);
            MidiMessage message = event.getMessage();
            if (message instanceof ShortMessage) {
                ShortMessage sm = (ShortMessage) message;
                if (sm.getCommand() == NOTE_ON || sm.getCommand() == NOTE_OFF) {
                    empty = false;
                }
            }
        }
        return empty;
    }
}

```

```

////////-SongMatrix.java-////////
package midi.convertor;

import java.io.FileWriter;
import java.io.IOException;
import java.util.Arrays;

public class SongMatrix {

    int[][] songMatrix = null;

    public void addTrack(TrackMatrix matrix){
        if (songMatrix == null){
            songMatrix = matrix.getMatrix();
        }else{
            int[][] newSongMatrix = new int[songMatrix.length +
matrix.getMatrix().length][songMatrix[0].length];
            for (int i = 0; i < songMatrix.length; i++){
                newSongMatrix[i] = songMatrix[i];
            }
            for (int i = 0; i < matrix.getMatrix().length; i++){
                newSongMatrix[songMatrix.length + i] = matrix.getMatrix()[i];
//                System.out.println(songMatrix.length + i);
            }
            songMatrix = newSongMatrix;
        }
    }

    @Override public String toString(){
        String m = "[";
        for (int[] row : songMatrix){
            m += "\n  "+Arrays.toString(row);
        }
        return m + "\n]";
    }

    void saveCSV(String outcsv) throws IOException {
        FileWriter fw = new FileWriter(outcsv);
        fw.write(toCsv());
        fw.close();
    }

    private String toCsv(){
        String csv = "";
        for (int[] row : this.songMatrix){
            if (csv.length()>0){
                csv += "\n";
            }
            String currentRow = "";
            for (int col : row){
                if (currentRow.length() > 0){
                    currentRow += ", ";
                }
                currentRow += col;
            }
            csv += currentRow;
        }
        return csv;
    }
}

```



```

////////-TrackMatrix.java-////////
package midi.convertor;

import java.util.ArrayList;
import java.util.Arrays;

public class TrackMatrix {

    ArrayList<ActiveNote> activeNotes = new ArrayList<>();

    public static final String[] NOTE_NAMES = {"C", "C#", "D", "D#", "E", "F",
"F#", "G", "G#", "A", "A#", "B"};

    private int[][] trackMatrix;

    public TrackMatrix(int width, int height){
        trackMatrix = new int[height * 2][width];
    }

    public void noteOn(int key, long eventTick){
        ActiveNote an = new ActiveNote();
        an.key = key;
        an.startTime = eventTick;
        activeNotes.add(an);
    }

    public void noteOff(int key, long eventTick){
        ActiveNote activatedNote = null;
        for (ActiveNote an : activeNotes){
            if (an.key == key){
                addNote(key, an.startTime, eventTick);
                activatedNote = an;
            }
        }

        if (activatedNote != null){
            this.activeNotes.remove(activatedNote);
        }
    }

    private void addNote(int key, long start, long end){
        int relativeKey = key - (MIDIConvertor.LOWER_NOTE -
MIDIConvertor.LOWEST_BOUND) * 12;
        int startPos = Convertor.convertTickToQuantizedBeatDown(start);
        int endPos = Convertor.convertTickToQuantizedBeatUp(end) - 1;

        int activeKeyPos = relativeKey * 2;
        int activeKeyOnPos = activeKeyPos + 1;

        //        if(endPos<startPos){
        //            endPos++;
        //        }

        this.trackMatrix[activeKeyOnPos][startPos] = 1;
        for (int i = startPos; i <= endPos; i++){
            this.trackMatrix[activeKeyPos][i] = 1;
        }

        //        System.out.println(getNoteNameFromKey(key) + " " + startPos + " - " +
endPos);
    }

    private static String getNoteNameFromKey(int key){
        int octave = (key / 12)-1;
        int note = key % 12;
        String noteName = NOTE_NAMES[note];
        return noteName + octave;
    }

    @Override public String toString(){
        String m = "[";
        int rowNo = 0;

```

```

        for (int[] row : trackMatrix){
            String rowLabel = NOTE_NAMES[(rowNo/2) % 12]; //Note
            rowLabel += (rowNo/24) + MIDIConvertor.LOWER_NOTE; //octave
            rowLabel += rowNo%2==1? "'" : " "; //Active
            while (rowLabel.length() < 4){rowLabel += " ";}

            m += "\n " + rowLabel + " " + Arrays.toString(row);
            rowNo++;
        }
        return m + "\n]";
    }

    int[][] getMatrix() {
        return this.trackMatrix;
    }

    private static class ActiveNote {
        public int key = 0;
        public long startTime = 0;
    }
}

```