# The Effectiveness of RNNs, LSTMs and GRUs in Music Generation

Sean Ross Faughey

Submitted in partial fulfilment of
the requirements of Edinburgh Napier University
for the Degree of
BSc Computing Science

School of Computing

March 2019

# Authorship Declaration

I, Sean Ross Faughey, confirm that this dissertation and the work presented in it are my own achievement.

Where I have consulted the published work of others this is always clearly attributed;

Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work;

I have acknowledged all main sources of help;

If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself;

I have read and understand the penalties associated with Academic Misconduct.

I also confirm that I have obtained **informed consent** from all people I have involved in the work in this dissertation following the School's ethical guidelines

Signed: Sean Ross Faughey

Date: 27/03/2019

Matriculation no: 40054753

# General Data Protection Regulation Declaration

Under the General Data Protection Regulation (GDPR) (EU) 2016/679, the University cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

Please sign your name below *one* of the options below to state your preference.

The University may make this dissertation, with indicative grade, available to others.

The University may make this dissertation available to others, but the grade may not be disclosed.

Sean Ross Faughey

The University may not make this dissertation available to others.

# Abstract

Using artificial intelligence to create music and other forms of art is becoming an important topic in computer science, although remaining less popular than sequence to sequence translation such as converting a piece of text from one language to another. The aims of this research are to generate novel and realistic pieces of music, the purpose of which is to create a further understanding in the technologies involved. These technologies aid musicians who may be struggling to create something novel, and professionals who require novel pieces of music and it is too expensive to hire a musician and too difficult to create themselves given their scope. For example, independent game developers or movie directors who need music for in-game, cutscenes, movie scenes, trailers etc. and do not have a high budget or the ability to create music.

Recurrent neural networks have been researched thoroughly and are a reasonably effective method of creating music artificially. However, the music created has the problem that whilst a short section of the music will sound good and the notes will match, there is a distinct lack of overall structure to the music and overall it is not a convincing piece. This is due to the ‘vanishing gradient problem’ which is a major problem with many feed-forward networks and also a problem with RNNs.

More recently research has gone into understanding the benefits of LSTM, and even later GRU networks which are a type of recurrent neural network where the vanishing gradient problem is not a problem. This dissertation aims to present a prototype of a RNN trained to generate music which can then be used to compare the effectiveness of both a LSTM network and a GRU network, both trained to generate novel pieces of music with a coherent structure which are audibly pleasing after the presentation of both a technical and literature review highlighting research in the field currently.

# Contents

# List of Tables

:

# Acknowledgements

I would like to thank my supervisor Dr Simon Wells for his continued support and guidance throughout the creation of my dissertation.

Secondly, without the patience, love and support from my better half Caitlin I would never have made it to the end of this project, and I cannot begin to express my gratitude. I will always be grateful for what you have done for me over the past three years of my degree.

# 1 Introduction

## 1.1 Background

Long short-term memory (LSTM) (Sepp Hochreiter, 1997) has become a staple of deep learning in the past few years. After research showed that it can provide faster, better generalisations than a standard recurrent neural network (RNN) over temporally distant events (Douglas Eck, 2002) it swiftly became a hot topic for many researchers. It was clear how this could be a benefit to the progress of creating music using machine learning and recently LSTM and other more recent techniques (Verbeurgt K., 2004) are being used to create music algorithmically for those who wanted it. Some examples of this are Aiva by Aiva Technologies which creates emotional soundtracks, Amper by Amper Music who create music of any type given some adjustable parameters to the user and WaveNet by DeepMind who generally use it as a method to generate speech to mimic the human voice effectively but also use it as a method of mimicking music created by humans.

Historically, rule-based and feed forward networks were used in an attempt to enforce the rules of a particular musical style onto an agent which the agent would follow, with the next note chosen based on weights created from the previous node. (Todd, 1989) was one of the first to effectively evaluate this method showing that feed forward networks were able to effectively create transitions between notes and chords. However, these types of networks are not effective at creating music which sounds good over a longer period of time. This is due to the fact that the network has no knowledge of what has come before the note it has just played and therefore has no context as to what the note was played. Therefore, it has no knowledge of what point in a song it is in resulting in no structure, it has no knowledge of what key it is in as there are many notes which overlap different keys resulting in dissonant sequences of notes unless this has been hard coded into the program.

Shortly after the exploration of feed forward networks in algorithmic music composition was the discovery that recurrent neural networks were a more effective method. The main difference between how recurrent neural networks operate is that the output of one layer is fed into a hidden layer before being fed back into itself for evaluation. This provides some context as to the previous note played which in turn provided a better overall sounding result with less dissonant sounds which was also explored by Todd and further researched by (Mozer, 1994). Mozer said ‘While the local contours made sense, the pieces were not musically coherent, lacking thematic structure and having minimal phrase structure and rhythmic organisation.’ The results of these explorations demonstrated that RNNs could be effectively utilised to create convincing sequences of notes and chords however do not demonstrate overall structure. Despite these results with regards to music, this was a major step forward for research in recurrent neural nets and deep learning in general.

A major reason for the lack of structure is the vanishing gradient problem. Gradient descent is the algorithm used to find the local minimum in a search landscape. By finding the gradient of a point in the landscape the algorithm decides which is the most effective next move to make that would be closest to the local minimum. It was said that ‘Recurrent networks are very powerful in their ability to represent context, often outperforming static networks. However… gradient descent of an error criterion may be inadequate to train them for tasks involving long-term dependencies’ (Yoshua Bengio, 1994).This means that for tasks where there are larger temporal dependencies the vanishing gradient problem causes the network to forget too much to create an effective result. It is clear how this problem relates to the task of composing realistic pieces of music as there can be large temporally distant events resulting in an overall structureless composition.

(Douglas Eck, 2002) demonstrated the effectiveness of the LSTM technique in the algorithmic composition of music. A major difference in the way a normal RNN works and how the LSTM variant works is that instead of passing the output to the next layer it passes a hidden score which is a measure of how far the current node is away from the goal. This allows the LSTM to hold information from a longer temporal span than standard RNN techniques. It is said that 'LSTM also solved complex, artificial long time lag tasks that have never been solved by previous recurrent

network algorithms' (Hochreiter, 2001). It is clear from these results that LSTM provides a much more effective method of producing structured compositions over a longer period.

Gated Recurrent Units (Cho, 2014) are a modern approach to dealing with predicting sequences using neural networks. An adaption to LSTMs, they offer a similar performance over sequences, with the same benefit of not suffering from the problem of vanishing gradient descent, whilst being simpler and producing faster training times.

## 1.2 Aims and Objectives

The initial aim of this project is to provide an overview on the tools and methods at the foreground of machine learning today, exploring the effectiveness of each and discussing their relevance to the topic of generative learning and temporally distant events. To accomplish this an extensive literature and technology review is presented using reputable sources of research in the space.

The main topic of this honours project is to provide further insight into the effectiveness of Long Short-Term Memory in the context of creating realistic, correct and audibly pleasing musical compositions. A variation of LSTM networks called Gated Recurrent Units are also explored. This is to be contrasted with the effectiveness of vanilla Recurrent Neural Networks in the same task. To achieve this, an implementation of a Long Short-Term Memory neural network is presented, and an evaluation of the results is made.

To further break down the aims and objectives of this project, they can be described as the following:

- Aims:
    - Explore the methods which have been used in machine learning to learn patterns devolved from sequences of data, and predict the next output given those sequences
    - Explore and evaluate the tools which are available given our scope and limitations which will aid in our implementation of prototype models
    - Evaluate the effectiveness of RNNs, LSTMs and GRUs in learning sequences
- Objectives
    - Define what constitutes as a successful result, and how to measure how successful this result is
    - Create three prototype networks which will learn from a dataset and output music
    - Evaluate the effectiveness of these networks by using metrics by using metrics such as accuracy and loss
    - A manual evaluation of the music generated with respect to our definition of music, and how the music fits some basic musical constructs such as scales

## 1.3 Scope and Limitations

The scope of this honours project is to show how Long Short-Term Memory and Gated Recurrent Units can be used to create effective pieces of music by training the neural network on pieces of music both composed and played by humans, with an emphasis on the longer temporal structure of the music made possible by the use of an LSTM or GRU. Neural networks can take a long time to train to an optimal amount, in many cases up to 100 hours of consistent training. This is not possible in the scope of an honours project where neither the time nor the hardware is available. Therefore, the project aims only to provide insight into how effective the LSTM network is at producing these pieces of music and the impact of using realistic pieces of music and does not promise that the results are a perfect representation of this. Where the dissertation considers the usage of the GPU to perform calculations which

would enable training to be completed faster and therefore offer more potential epochs or deeper networks, the hardware was note available due to test this as the available GPUs did not support the required amount of CUDA computation capability.

## 1.4 Dissertation Structure

1. **Introduction:** Provides a background to the project and an overview. Sets out its aims and objectives. Also describes the scope and limitations, and the structure of the dissertation.
2. **Literature Review:** This section begins with an overview of what the literature review will contain, before moving on to describe different ideas that are important to understand. The ideas include different learning methods, classifiers, model types, neural networks, how they work and the problems that they create such as the vanishing gradient problem, and solutions to these problems.
3. **Technical Review:** A thorough and technical discussion involving software, tools and languages which may or may not be of use for completing the project, including the decision making process behind choosing which were used.
4. **Problem Domain Analysis:**
5. **Implementation:** A thorough explanation and discussion into how the project was created, results were captured and the rationale behind any decisions that were made.
6. **Evaluation:** A comparison of results between the RNN, LSTM and GRU, and a discussion on why they occurred.
7. **Conclusions and Future Work:** A summary of the work created and the results, a critical analysis, a personal reflection and the potential for future work or improvements.

# 2 Literature Review

This section of the report aims to use existing literature and technology in the subject field to create a further understanding of previous developments in the field, and of the field itself. By increasing the understanding we have of the past works and their findings, a good basis is found for the further research to be built upon. Understanding previous work and learning from their results and mistakes is critical to making the correct decisions in further research towards machine learning and its benefits in the space of music generation.

This literature review consists of multiple sub-sections which create a full picture of the previous works. It begins by discussing the types of classifiers which have been used in similar projects and the pros and cons of each type. Then it discusses different type of neural networks and the benefits that they provide. To provide further insight into the creation of these networks, we then consider the problems which afflict these approaches and the solutions which have been provided. After that a technical review is conducted discussing the relevant technologies which have been used previously in relevant studies.

## 2.1 Supervised Learning

Supervised learning is the act of training a model to give an output given a certain input by learning what properties are important to consider when classifying and increasing the weights of those properties. This is done by generating the output of the model over one iteration of training, calculating the distance from the expected output and adjusting the weights appropriately. Therefore, the learning is supervised as it has knowledge of the expected outcome and can learn from its mistakes to reach a local minimum. Supervised learning is suited to problems where the domain is known, the data is labelled, and the goal of the training is classification. For example, supervised learning is good at learning whether an image is of a cat or a dog, or whether something is true or false. (Brownlee, Supervised and Unsupervised Machine Learning Algorithms, 2016)

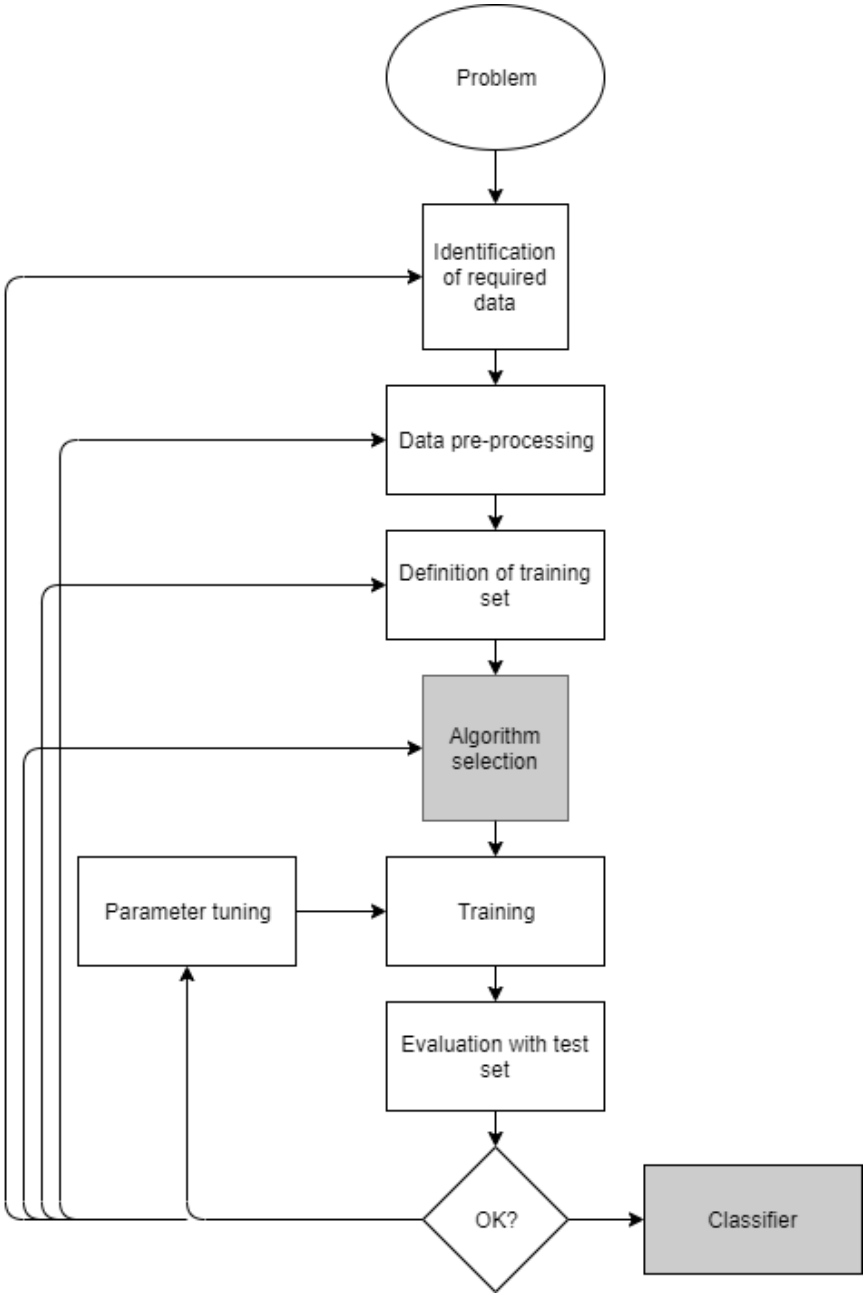



**Figure 1: Supervised Learning Process (Kotsiantis et al., 2007) – This process can be iterated over any number of times until an acceptable accuracy level has been reached**

## 2.2 Unsupervised Learning

Unsupervised learning responds well to training models to predict continuous sets of data. It does not have access to the entire data domain, instead learning patterns between the different parts of data. Commonalities are found between the different pieces of data and the models react to each new piece of data fed into them. Unsupervised learning does not require the data to be labelled, classified or categorised. For example, given a dataset of different weather patterns unsupervised

learning could be used to predict a given days weather based on the weather the previous day, or over a sequence of previous data points. This lends itself to the problem of music generation as we aim to predict continuous data points from a sequence or sequences of previous data. (Brownlee, Supervised and Unsupervised Machine Learning Algorithms, 2016)

## 2.3 Reinforcement Learning

Reinforcement learning is a machine learning technique that creates a trial and error behaviour from an agent, which receives either positive or negative feedback as a penalty or a reward (Simonini, 2018). The agent's goal is to create a function which will maximise its reward. This creates a difference between unsupervised learning where the goal is to find patterns in the data, as here the agent is attempting to maximise its total reward. This makes reinforcement learning techniques particularly suited towards scenarios where the agent has to make the decision between long term and short term rewards. For example, if attempting to teach an agent to play a game of chess it may learn that it may not beneficial to take the enemy queen piece if it leaves the king vulnerable to be taken from future moves.

## 2.4 Classification

Statistical classification is the problem of identifying to which set or subset of categories a particular input belongs. This is done by comparing the input needing to be classified to what has been learned using the supervised learning technique with regards to the features of the input. With regards to our previous weather example, we may have learned that if the average temperature of a day is >20C, and there was <10mm of total rain then the day could be classified as sunny. In this example the temperature and total rain are the features which were considered. If instead an unsupervised method was preferred, there would be no classification of sunny as a result of there being no previous dataset or classifications to learn from. Instead, we would find that days of similar weather patterns would be grouped together without the classifications.

## 2.5 Regression

Where classification aims to identify different clusters or sets of data types, regression analysis aims to estimate how related different data points are to one another. (Ramcharan, 2006) This lends itself towards prediction and forecasting as

the likeliness of any future feature occurring given the previous feature or features is learned as a probability. With regards to our problem this is the most fitting as we are aiming to predict one feature which is the pitch of a note given the previous pitches.

## 2.6 Models

### 2.6.1 Discriminative

Discriminative models aim to solve either the classification or regression problems (Joshi, 2018). They do this by learning the properties of the dataset and what classification they fit into. Then after being given a piece of data to classify they compare each of these properties until a probability of a certain classification is given. Due to the nature of discriminative models they are generally only suited towards supervised learning problems as they require to train off the expected output. More formally put classification algorithms aim to estimate the posterior probability p(y|x) where x is an input and y would be the classification of that input.

### 2.6.2 Generative

Generative models learn to solve a different type of problem to discriminative where they aim to learn how to create data rather than simply classifying them (Joshi, 2018). It does this by learning the features of data and the probability of something containing those features. When asked to complete its predictions it uses these features to create something ‘novel’. It does this by estimating the joint probability p(x, y) which means that x is of a certain classification and that y is a part of x.

### 2.6.3 Feedforward and Convolutional Networks

Feedforward neural networks or multi-layer perceptrons (MLPs) are the typical network which most people think of when envisioning a neural network. The goal of a feedforward network is to approximate a function given an input to create an output. For example, given a function y = f(x) the input x is mapped to a category y in a categorisation problem. Over time the network learns the best approximation for the function f to create y. (Brownlee, 2016) Feedforward networks do as the name suggests and feeds the information forward through the network of neurons (computations) with no recurrent layers.

Convolutional neural networks (CNNs) are similar to feedforward networks in that they have no recurrent layers and strictly send information forward through the network. They have found particular success in image analysis and classification problems due to their strong ability to identify patterns in data and learn from them. What differentiates a CNN from a standard MLP are the convolutional layers which perform convolution operations allowing for the detection of patterns via ‘filters’ (Karpathy A. T.-F., 2014). A typical pattern for a CNN to discover in image analysis would be the edge of an object in an image. Here matrices are defined and applied over each pixel in an image where a specific matrix corresponds to a specific aspect of an image, for example a vertical straight line. Over time, these matrices will identify where in an image their respective attributes are found. With regards to music CNNs have found much success in genre classification and instrument recognition, there has not been much success in music generation. This is due to the lack of the ability to ‘remember’ any information that has happened in the past, such as the previous note, to aid in the next prediction of notes.

### 2.6.4 Recurrent Neural Networks

Recurrent neural networks are a type of neural network where nodes are connected to form a network along a temporal space. RNNs exhibit the ability to use the memory of each node called a cell to ‘remember’ data. This makes RNNs especially suited to modelling data where the temporal factor is of importance such as with sequences of data like words or music. RNNs have found success in speech recognition and recognising hand written words because of this. They work by having the form of a chain of repeating modules, such as a single tanh layer. Simpler RNNs have encountered the problem of vanishing gradients. A second problem when using RNNs is that they can only look at the recent past to influence the weights, and some important information may be lost if it is too temporally distant.

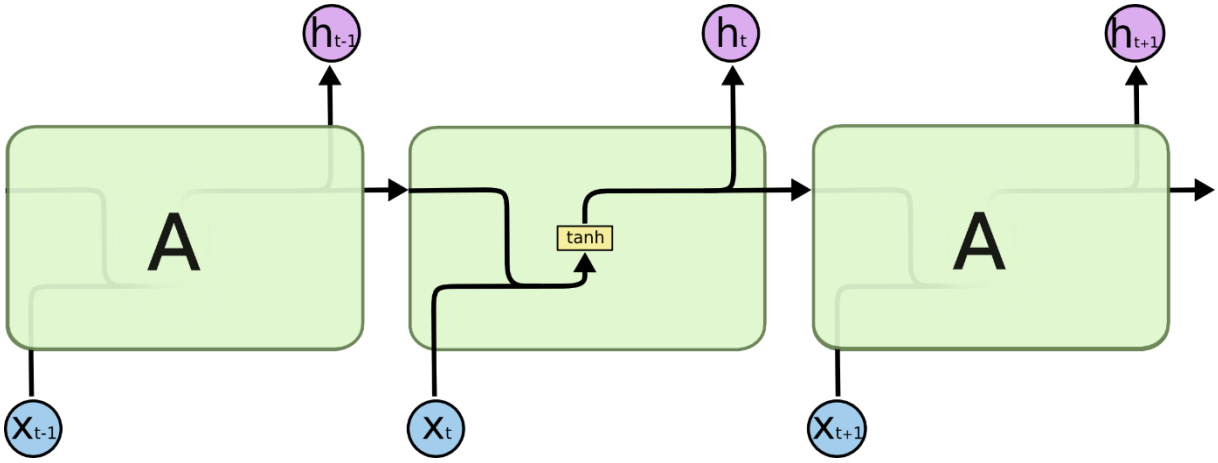



**Figure 2: A simplified view into the workings of an RNN. (Olah, 2015) X<t> = input, H<t> = hidden state, t = timestep. It can be seen how each layers hidden state is retained and influences the next layer**

### 2.6.5 Stochastic Gradient Descent

The purpose of gradient descent is to find the minimum of a loss landscape (and in some cases the maxima). This is the indicator of how close to being correct a neural network is in all proposed cases. In gradient descent, the exact downward direction is calculated for every step. The difference between these two steps is called the negative gradient of the cost function (J. Kiefer., 1952). This means that each of the changes made to every weight and bias of every node is averaged to give the desired change. This is computationally slow resulting in longer training times and shallower networks. In response stochastic gradient descent was created. Here, data is divided into small batches and each step is instead computed with regards to a batch. By repeatedly applying gradient descent to each of the batches eventually the local minimum is converged upon. This causes some steps to be taken in the wrong direction, however it provides much faster computation and arrives at the local minimum faster by taking larger steps in the correct direction.

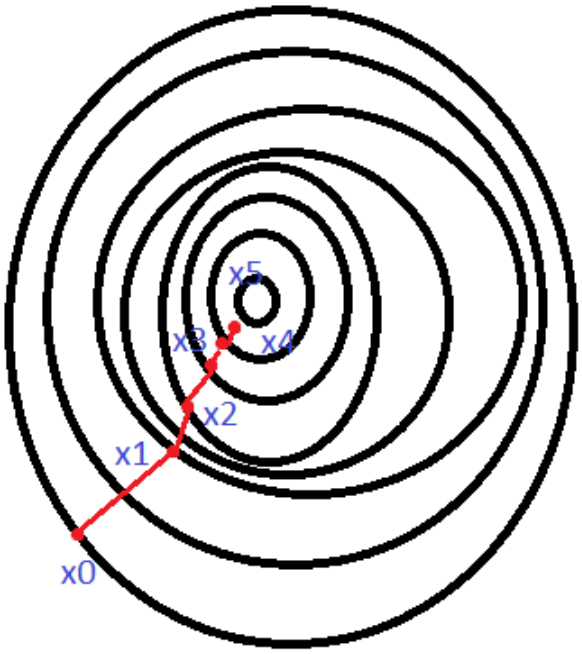



**Figure 3: Stochastic gradient descent visualisation**

### 2.6.6 Back Propagation Through Time

Back Propagation Through Time (backpropagation or BPTT) is one of the core algorithms to how neural networks learn (Hecht-Nielsen, 1992). Backpropagation is the method of calculating the negative gradient of the cost function, which tells you how to change all of the weights and biases of each of the connections in order to efficiently decrease the cost. This occurs for each of the output neurons until a ‘list’ of changes to weights and biases is created to be applied to the layer t-1. Once this is done, this process can be recursively applied to the weights and biases that determine the values of the layer t-1, which would be layer t-2 and so on moving backwards through the network. Therefore, the gradient of the loss of any given weight is going to be the product of derivatives that are part of later layers of the network.

### 2.6.7 Vanishing and Exploding Gradients

Gradients refer to the change of the loss function with respect to the weights of the network which is calculated using backpropagation which are used by stochastic gradient descent to update the weights of the network. Over time this change of weight can become vanishingly small to the point where a network cannot further train due to the change in weights in earlier layers of the network becoming tiny. This can cause problems with training long term dependencies, where the temporal distance of two dependent inputs are not seen or learned (Yoshua Bengio, 1994). This can happen due to the way backpropagation is calculated using the product of derivatives. The earlier in the network a weight is more terms are needed in the product to get the gradient of the loss for this weight. If these terms or at least some of them are less than 1, we can deduce that the gradient is going to be even smaller. To compound on this issue, the product of these terms is usually multiplied to a learning rate which usually resides in the range of 0.1 and 0.001 resulting in a yet smaller gradient. This number is then subtracted from the weight providing the value of the updated weight. The eventual result is an almost non-existent change to the weight, which means that it is not learning.

The exploding gradient problem occurs in a very similar fashion, except it happens when some of the derivatives are larger than 1. If a lot of terms larger than 1 are multiplied together, it is possible to get a term which is a lot greater than 1 or ‘exploding’. This time instead of the weight being barely moved, it is moved an incredible amount and never gets closer to the minimum instead getting further away.

### 2.6.8 Long Short-Term Memory

While exploding gradients are a problem easily solved by using a technique known as gradient clipping, Long Short-Term Memory networks (LSTMs) are an attempt at solving the problem of vanishing gradients that simpler RNNs present, and the problem of forgetting information related over larger temporal distances (Sepp Hochreiter, 1997).

The important difference between LSTMs and simple RNNs with regards to vanishing gradients is that LSTMs implement a ‘relevancy’ calculation. Simply put, this means that if the state at t-1 did not contribute much to the state at time t then during backpropagation the gradients flowing into S<t-1> will vanish and will be hardly considered in the product of derivatives. This type of vanishing is considered acceptable because if it did not contribute to the forward propagation then it should not be considered in backpropagation either.

Similar to RNNs, LSTMs also have a chain-like structure however with the repeating module having a very different structure. Whereas the simple RNN consists of a single tanh layer, a LSTM cell consists of three ‘gates’: input, forget and output. The forget gate decides on what information is worth remembering and what information is irrelevant computed using a sigmoid layer. This is represented by a number between 0 and 1 which represent how much of the input to remember where 0 represents remember nothing, and 1 represents remember everything.

The next step is to decide on what information we are going to store. This is computed using the input gate which is a sigmoid layer which decides on what values to update. Then, a tanh layer computes a set of candidate values to be used to add to the state. Now that the updates have been decided on, the next step is to add the updates to the cell. We proceed by forgetting the information that had been allocated to be forgotten, before adding the candidate values.

Finally, the output gate is used to decide on what exactly is going to be output. This is done by running the data through a sigmoid layer to decide what parts of the cell state we are going to output before running it through a tanh layer to force the data to be between -1 and 1. Then, it is multiplied by the results of the sigmoid layer so that only the data we decided on outputting are output. LSTMs are extremely powerful at discovering patterns and connections in sequences and are the main focus of this dissertation.

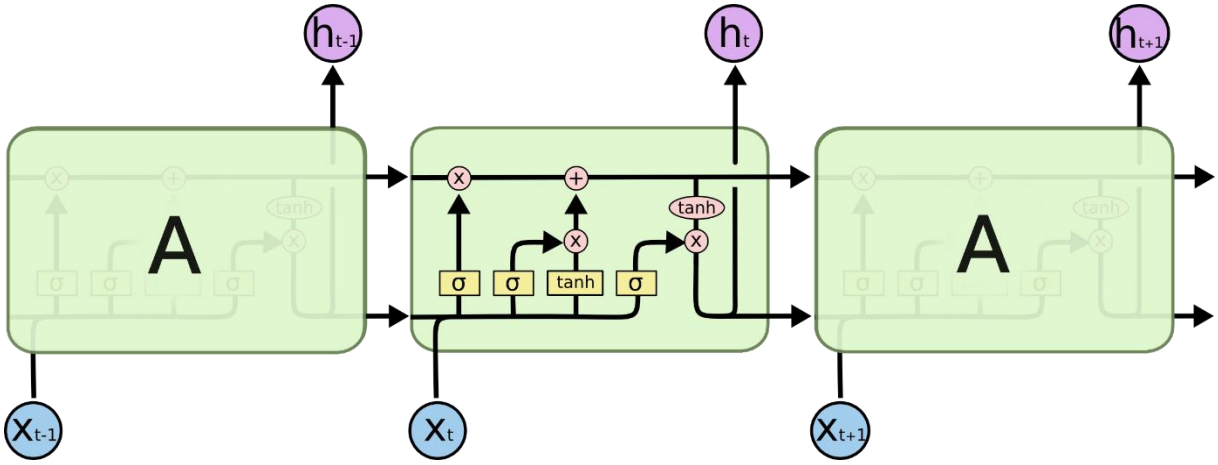



**Figure 4: A visualisation of the insides of a repeating LSTM module. (Olah, 2015) – X<t> = input, h<t>=hidden state, t = timestep. It is clear how the LSTM is able to remember long sequences by the arrow at the top, which passes through each node with only 2 functions applied.**

### 2.6.9 Gated Recurrent Units

Gated recurrent units (GRUs) are a variation of LSTMs which are simpler and thus require less time to compute however overall, they operate very similarly. Some important differences between GRUs and LSTMs are as follows:

- In GRUs, C<t> (memory cell) = a<t> (output activation value). This is not the case in LSTMs where they are calculated separately.
- In GRUs there are 2 gates, the input gate which is used to calculate the candidate, and the update gate which determines how much past information to pass along to the future.

These features decrease the number of computations and the complexity of GRUs and decrease the required computation time. Due to the simplicity of GRUs compared to LSTMs they can be used to create similar results in a shorter time span (Chung, 2014). This can allow for deeper models to be created or layers with larger amounts of hidden units without extremely large computation times. However, the simplicity of the model does come at the cost of losing some of the power of LSTMs and their flexibilities. Yet they retain the ability to remember information over a large temporal span and as such are a good candidate for examination given the aims of the project.

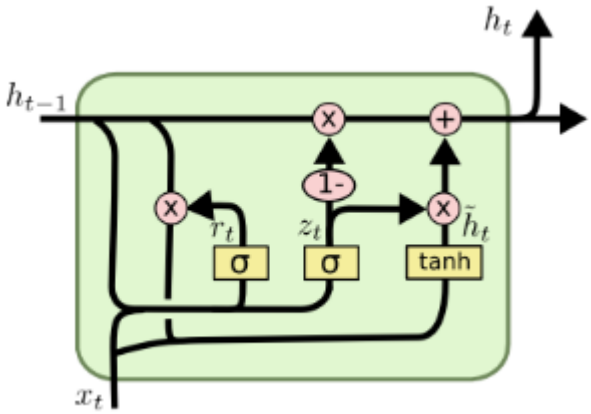



**Figure 5: A view into a single GRU cell. These cells can be linked together much like cells from an RNN, and the GRU's LSTM counterpart. (Olah, 2015)**

## 2.7 Underfitting

Underfitting is when the algorithm, for example a linear regression model, doesn't fit the training data very well and thus isn't able to learn the patterns underlying the data (Cai, 2014). Underfitting causes the inability to classify correctly or the inability to predict future values. This can be caused by applying an incorrect algorithm for the type of problem to be approached, by having an inadequate number of hidden units inside layers, too small datasets and number of epochs and by potentially having an inadequate dataset. This can also happen by having a high bias, meaning that the algorithm has a high preconception that data is going to take a certain format and is stubborn to learn otherwise despite it being presented in the data resulting in a poor fit to the data.

## 2.8 Overfitting

Overfitting or high variance occurs when an algorithm fits itself to a training dataset too strictly, causing it to lose its ability to generalise and perform badly on validation. This can happen by having too much 'noise' compared to the 'signal' (Silver, 2012). Noise is the irrelevant data, randomness and anomalies within a dataset which may cause the model to fit to data which for the vast majority of the time has little to no impact on the results. The signal is the actual pattern within the data that we want the model to actually learn. Therefore, by having too much randomness or noise within a dataset an algorithm will be forced to fit to these anomalies rather than the general pattern within the dataset.

This happens mostly with complex, deep models rather than smaller models as they have the ability to memorise the noise rather than being forced to generalise towards the signal. Therefore, a model has to be complex enough to be able to learn the function necessary yet not overly complex where it simply memorises the training data and performs badly on unseen data. There is no rule by which to follow to create the perfect algorithm for any particular problem and can only be done by trial and error methods. Given the time available for this dissertation this will prove difficult, and instead previous examples of similar experiments will be looked towards for help deciding on the initial configuration of models to provide a platform to then work from.

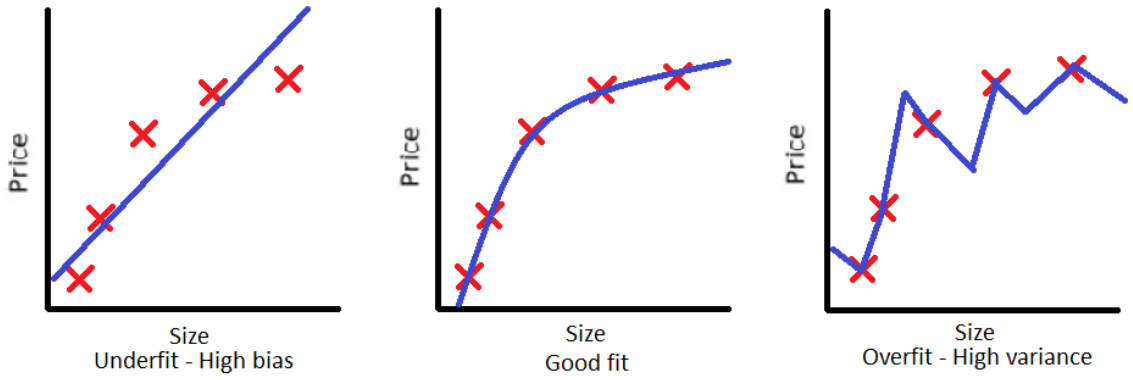



**Figure 6: Underfitting vs a perfect fit vs overfitting (fictional house price vs size data) – the graph on the left presents a stubbornness to learn, and if it was to predict prices for higher sized houses it would fail. The graph on the right adheres to the data too strictly, and fails to see the signal from the noise, while the middle graph presents a good fit and generalises well.**

In figure 6, the graphs demonstrate the three different types of fit with respect to a fictional data set of increasing house prices given the house size. The data (red crosses, excluding noise) can be seen to be plateauing towards the end of the graphs, yet underfitting would show them increasing linearly. The second graph, showing a good fit, while not being 100% correct all the time generalises well and follows the patterns correctly. The third graph demonstrating overfitting is too sensitive to noise and whilst being 100% accurate does not generalise well and is unlikely to perform well on unseen data.

### 2.8.1 Bias-Variance Trade-off

As previously discussed, bias is the stubbornness of a model with regards to how much it is willing to learn from the data given to it compared to its preconceptions of what the data should look like. Variance however refers to the sensitivity of the model to randomness, anomalies, irrelevant data or ‘noise’ inside the data causing it to overfit. To deal with the problem of high variance, simpler algorithms are used which allow the model to generalise a lot better and overfit a lot less. However, to deal with the issue of high bias we must do the opposite where we increase the complexity of the model to allow for algorithms which can actually learn the signals provided in the dataset. Therefore, it is important to strike a good balance between simplicity and complexity so that a ‘sweet spot’ of just enough of both, not too much of either is reached. (Fortmann-Roe, 2012)

### 2.8.2 Detection

The first step to dealing with overfitting and underfitting is to detect it. This can be a relatively difficult thing to do if we have complex signals or large data sets. One of the key problems within overfitting and underfitting is that we don’t know exactly how well our model will perform until it has been tested. Considering the relatively long training times of some models, you don’t want to have to wait until the end to find out it learns nothing, or simply memorises the training data.

#### 2.8.2.1 Validation Split

Validation is a core method of testing whether our model is overfitting or underfitting which can be performed during training. A section of the dataset is put aside and not trained on so that it can be preserved as unseen data. A normal amount of data to be preserved for the validation set is anywhere between 33% to 10% of the data. Once the model has finished an epoch of training, it then attempts to predict the validation set. If the model is performing relatively well in training yet the results of the validation are poor then it can be seen that the model is overfitting. Yet, if both training and validation accuracies are poor then the model can be seen to be underfitting. Specifically, if the training and validation accuracies are not improving then the model is not learning implying that it is not complex enough whilst if the

accuracies are improving and not plateauing then the model has not been trained for enough epochs to fully learn the signals.

### 2.8.3 Overfit Prevention

#### 2.8.3.1 Cross-validation

Similar to a standard validation split, cross validation uses the initial dataset to create batches of validation data which is unseen and can be used to test the effectiveness of the model on unseen data. One key difference between cross-validation and a standard validation split is that mini-batches which are different for every epoch of training *within* the training data are used rather than the same set for every epoch at the end (Pedregosa, 2011). The standard process for cross-validation is to use K-Fold Cross Validation where the dataset is partitioned into k subsets (folds). The algorithm is then trained iteratively on k-1 folds until reaching the end where fold k is used as the test set.

#### 2.8.3.2 Process Mining

One common technique to avoid the problem of underfitting is to start with a model which will overfit on a very small subset of the data (Van der Aalst, 2010). This will provide a starting point to build upon. Once it is found that the model is capable of overfitting on a very small amount of data, the dataset is increased. As the dataset increases it can be found that it will begin to underfit which can be described as ‘underfitting to prevent overfitting’. Once this point has been reached, the model complexity can be increased until it begins to overfit again. This process can be repeated repeatedly until the desired results are reached. This also provides

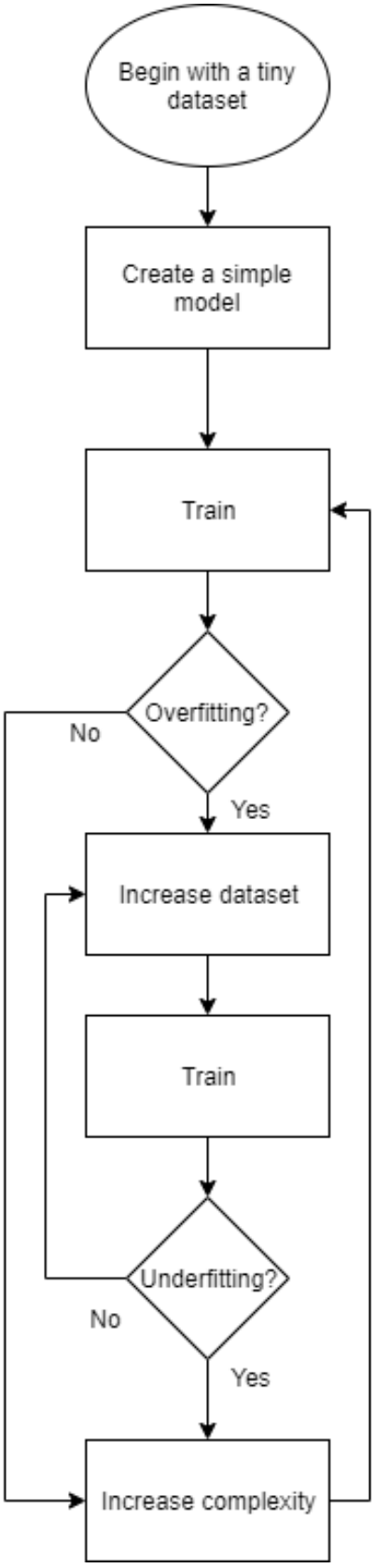



**Figure 7: Steps of process mining**

us with the Occam's Razor test (Hawkins, 2004) where two models, one relatively simple and one relatively complex, provide similar results then the simpler model should be chosen. This will provide with an easier understanding of the model and much faster training times which could potentially allow for more training, resulting in better validation results.

#### 2.8.3.3 Feature removal

A simple technique for avoiding overfitting is to ensure that all features in the dataset are absolutely necessary and make sense to be there given what is being predicted (Shaikh, 2018). With regards to music, an example of an unnecessary feature would be the instrument for every note when trying to predict the next note. These features cause the algorithm to attempt to learn patterns which are not necessarily there, causing overfitting to the training data and will give poor results in the validation tests.

#### 2.8.3.4 Regularization

Regularization refers to a set of techniques which can be used to avoid overfitting which is preferred in classical machine learning. Regularization aims to artificially simplify models to force generalisation rather than allow the model to overfit. This is done by applying penalties to layers which is incorporated into the loss function.

#### 2.8.3.5 Dropout

A very simple yet highly effective method to avoid overfitting is dropout. During the forward or backpropagation of a particular pass a percentage (usually small) of neurons are ignored, meaning they do not contribute to the calculation of stochastic gradient descent for that iteration, as incoming and outgoing edges to the dropped node are also removed.

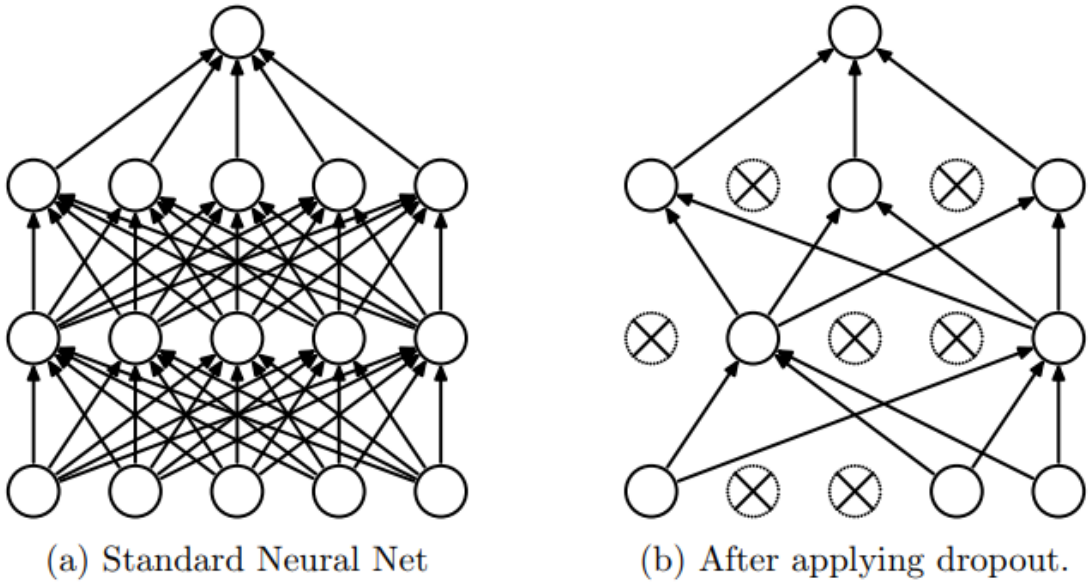



**Figure 8: A visualisation of an MLP before and after applying dropout. (Nitish Srivastava, 2014) – By dropping different nodes every generation a better generalisation is possible.**

## 2.9 Music Theory

For a proper evaluation of the quality of any artificially generated music, there must be some understanding of the concepts and constructs that exist in music. This allows for a comparison between the music generated and what is considered to be music according to our definition. If considering the entirety of music theory, we would struggle to create a full definition even in a life time of work and so this section and the further evaluation focus only on a few aspects: scales and harmonies, structures and rhythms. Even after this minimising of scope, we must further minimise due to the number of different scales, harmonies and rhythms which exist in music. Thus, we must limit ourselves to ones commonly found in western music which are likely to be found in the dataset (as they are the only ones that the algorithm can learn from, and there is simply too much data otherwise (Zeitler, 2005)). We cannot comb through the dataset and examine every scale, harmony, structure and rhythm used as there is simply too much data and must instead generalise. This section offers a general explanation of what scales and rhythms are, what they offer to a human composer, what they mean to the algorithm attempting to learn and use them, and how they may be used in evaluating results.

### 2.9.1 Scales and Keys

Scales are an organised sequence of notes that assuming they are used correctly produce sounds which fit together well. Keys refer to where on the instrument the scale is played, or more accurately, which section of the scale is being played. Scales do not need to be played in order, and keys do not have to be adhered to strictly.

They offer a song a sense of ‘home’ in that they do not vary overly wildly. If we only used the chromatic scale, which consists of every single note in the western system, there would music would seem to be directionless. (McGowan, 2015) This is not to say that they necessarily are pleasant, but that they do not sound dissonant. Dissonance occurs when a note or chord is played which does not fit the scale being used at a time. While dissonance occurs frequently in modern music and if used correctly can have effects such as building tension, this is overly complex for our models to learn and shall be disregarded for this project.

Human composers can use scales to give their music a home and a direction, so that tensions can both be built and released (another concept which shall not be covered here). Certain scales are considered to be particularly effective at conveying different emotions. Generally, minor scales convey a sense of tragedy or sadness, whilst a major scale offers happiness. This allows the user to portray an emotion through music without the necessity of words, or to accompany (or contrast with) the message of the lyrics.

For a machine learning algorithm, scales present a set of patterns which can be learned. For example, if given a song which is played in the key of C the algorithm may identify that it follows the pattern of C, F then G. This is a very common progression called I-IV-V, as it consists of the 1$^{st}$, 4$^{th}$ and 5$^{th}$ notes of the scale. Given another sequence of data in the key of D minor, the algorithm may find that the song repeatedly uses the phrase D, G and A. This is also a I-IV-V. Over time, the algorithm may learn the scales, many different chord progressions and the how to apply them to each other eventually resulting in music.

This provides us with a method of evaluation. It provides us with something to compare the music to, as if a sequence fits one of the many common progressions then we can say that the algorithm has learned this and is able to apply it. We can also identify that the music produced fits one of the many scales and has been applied to a chord progression which results in pleasant sounding music.

### 2.9.2 Rhythm, Duration and Tempo

Simply put, rhythm refers to the recurring intervals between sounds in a song. These intervals may be shorter or longer, and may change regularly throughout a single piece of music. Rhythm is often notated on sheets of music in the form of time signatures and tempo. A time signature refers to how many notes are played in a single bar, for example 4/4, where tempo refers to how quickly these notes are played, for example 120 beats per minute. With regularly varying time signatures between bars the time signature of the piece is created by adding together the numerators of the time signatures, while keeping the denominator consistent, before reducing the fraction into its smallest form. For example, if a song consisted of the time signatures 5/8 and 7/8, the song would be in 13/8 time. Duration refers to how long a note is played for and is measured according to the beats. For example, if a note lasts for the duration of a beat, then its duration is 64, and is scaled down or up accordingly.

This allows a human composer to create music with interesting intervals that may further the composer's ability to portray something towards the listener. For example, the use of varying time signatures such as 5/8 and 7/8 in a single piece, paired with a relatively high tempo, may create a sense of chaos to the music despite in actuality it being organised and thus pleasant to listen to.

For our algorithm, these patterns are readable and reproducible. The algorithm could learn that a tempo within a certain range is acceptable. It may also learn the correlation between beats, durations and time signatures and be able to reproduce interesting rhythms. When paired with the learning of keys and scales, the algorithm may be able to produce interesting melodies.

For the purpose of evaluating music, rhythm duration and tempo offer us the ability to look at how the algorithm has chosen to use its notes and its effects on the music overall. This could either be that it has no idea what its doing, where notes are scattered, there is no coherent timings of intervals or adherence to the beats. There may be an adherence to the beats, however simply by using only single notes per beat, indicating that it has not actually learned anything, or notes and durations may be scattered all over the place with no structure at all.

# 3 Technical Review

This technical review covers the technologies used to create the experiment. The program was written in Python-Keras, with a Tensorflow back-end. Several libraries such as Keras, Tensorflow and Music21 were vital in the development of the project. This chapter discusses each of the candidate languages and tools with respect to performance, features, usability and compatibility.

## 3.1 Languages

### 3.1.1 Python

Python is a favourite amongst data scientists and developers in general with many strengths which lend it to the task of machine learning due to its flexibility and ease of learning. There are many libraries available for free specialised to machine learning. Some of these include:

- **NumPy:** Contains many powerful data structures and functions great for scientific computing purposes
- **SciPy:** Uses NumPy for specific mathematical functions designed towards solving tasks common in scientific programming (including machine learning) such as linear algebra and integration
- **Sci-kit Learn:** Provides learning algorithms with a focus on ease of use, clarity and performance
- **PyBrain:** An entry level machine learning library which offers powerful algorithms for machine learning tasks
- **Tensorflow:** An open source machine learning library with the ability for large-scale multi-layered neural networks
- **Keras:** A high-level API for neural networks capable of running on top of TensorFlow, CNTK or the recently deprecated Theano
- **PyTorch:** A relatively new framework for machine learning with excellent debugging capabilities
- **Gluon:** Another relatively new machine learning framework
- **Pandas:** A data manipulation and analysis library
- **Matplotlib:** A data visualisation tool for python which is capable of producing publication quality graphs

- **Seaborn:** A data visualisation tool based on matplotlib which provides an API for producing publication quality graphs
- **Jupyter Notebook:** A powerful tool for interactively developing and presenting projects

These libraries and tools are just a fraction of quality open source tools that are available to every Python user offering power and flexibility. Python also offers the ability to integrate easily with languages such as Java and C however given the scope of our project this is not a major factor when considered which language to use. A more important consideration could be the speed of development allowed by using Python due to its readable syntax which I am familiar with and is easy to learn. One major downfall is that Python is a relatively slow language due to its high level and so experimentation and fiddling with hyper parameters can take a long time. Overall Python is a very friendly language to use with a rather large amount of flexibility, however this comes at the downfall of having relatively poor performance. Due to the number of libraries available, a highly active machine learning community which use these frameworks and the ease of use of the language, it was chosen as the main candidate for development.

### 3.1.2 R

R is an open source language for statistical computing with the ability to produce publication quality graphs. Much like Python, R also has many machine learning libraries which can be utilised to create machine learning projects including interfaces for both Keras and Tensorflow. Whilst it is not a major concern for this honours project, R does not integrate too well into other environments which could hinder the deployment of a machine learning project written in R to users. R's syntax is fairly unique and can be initially difficult to learn. With regards to our project this could be a major downfall for R as I have zero knowledge of the syntax compared to its competitors thus increasing the potential development time. R is a relatively fast language for machine learning purposes as intensive computational operations are written in C which is extremely fast. Given that we will potentially be operating over large data sets with large numbers of hidden units this could speed training time by multiple factors whereas to do the same thing in Python loops would be required, slowing down performance considerably.

### 3.1.3 Java

Java is a very robust language and can be used for many tasks including machine learning. Weka and Deeplearning4j are incredibly powerful tools for machine learning. Java is a very familiar language for me and would require little to no time taken to learn the language resulting in a very fast development time. However, whilst being familiar with the language at the initial stages of the project machine learning was unfamiliar and Java's code can be relatively long and unclear compared to Python. Java also offers excellent performance especially on larger scale systems which would result in very low training times.

## 3.2 Tools

### 3.2.1 Music21

Music21 is a Python library for importing music providing tools for analysis. It provides the means to parse a midi file and represent it in a way which is comprehensible to a computer without having to manually or automatically enter the data into a Excel file as commonly done with studies similar to this. Music21 is clearly and comprehensibly documented and therefore is easy to use. Music21 offers the ability to read and write all features of a midi file such as different instrument types, notes, pitches, chords rests and durations. Music21 also offers the ability to write a score for visualisation of inputs or outputs. One issue with Music21 is that it is only compatible with Python 3.6.x whereas current stable versions of Python are at 3.7.x.

### 3.2.2 Deeplearning4j

Deeplearning4j (DL4J) is a machine learning library, which offers a powerful and simple method of implementing machine learning algorithms such as neural networks using Java. While retaining a level of readability and power due to the level of expertise required when working with Java it is less popular than its Python counterparts.

### 3.2.3 PyTorch

PyTorch is a machine learning library compatible with our required 3.6 version of Python which offers a lower level approach for mathematically inclined users. This allows users to create their own custom layers which they can manipulate to

experiment with. This level of complexity sacrifices the verbosity, and whilst it is very clearly documented and has a reasonably large community of people to aid in learning and solving any problems which occur with it, it can be rather difficult for a new user looking to get started quickly to read.

### 3.2.4 TensorFlow

Tensorflow is an open source machine learning library popular for implementing machine learning algorithms, particularly with neural networks. Tensorflow is extremely popular and one of the reasons for this is due to its versatility of what it can be ran on as there is local, cloud and even support for mobile devices. There is also GPU support for certain operations such as basic LSTM cells using CUDA (CuDNNLSTM). GPUs are very fast at performing calculation such as linear algebra, which makes them very useful for improving the speeds of Tensorflow's calculations. However, at the moment there has been no implementation of CuDNNLSTM which supports regulizers such as dropout. With regards to performance, Tensorflow is very fast as it is written in C++ whilst being accessible via Python. Tensorflow also offers the use of a visualisation tool called Tensorboard, allowing the visualisation of training parameters, metrics and many other statistics which may otherwise be difficult to understand. Similar to PyTorch however it is lacks readability and is difficult to learn in a short period of time.

### 3.2.5 Keras

Keras is an interface that allows for the harnessing of machine learning toolkits such as Tensorflow, Microsoft's CNTK and Theano as previously discussed. Without sacrificing too much performance which would be provided by any of these options, Keras provides a readability and an ease of understanding that they otherwise lack, even more so than DL4J. This causes a short development time and time to learn.

| Library | Tutorials | High/low level | Training time | Java/Python |
|---|---|---|---|---|
| DL4J | Yes | High | Low | Java |
| PyTorch | Yes | Low | Medium | Python |
| Tensorflow | Yes | Low | Medium | Python |
| Keras | Yes | High | High* | Python |

**Table 1: A comparison of machine learning libraries, visualised**

## 3.3 Conclusion

Overall, Python was decided to be the best choice of language due to its capabilities in terms of its ease of learning and readability and choice of many quality libraries and tools. Whilst Java provided a potentially more powerful language with regards to performance, the ease of use and easiness to read is surpassed by Keras' and given the time constraints this is considered to be the most important feature, and so was the way I decided to move forward with Keras and Python. PyTorch was rejected for mainly the same reason as DL4J as it was considered to be an unnecessary level of complexity given our goals, as was Tensorflow.

# 4 Problem Domain Analysis

This section of the dissertation covers some problems in defining exactly what we are trying to achieve in a quantifiable way, and the solutions to these problems.

## 4.1 Defining Music

The attempt at defining music is an attempt at quantifying what components are necessary for something we hear to be defined as music. Whilst deciding whether something is music is something which humans are naturally very good at, we have a hard time defining exactly what music is or what it is made up of. There have been many attempts at doing so, yet there is ongoing debate. The Oxford English Dictionary defines music as ‘the art of combining vocal or instrumental sounds (or both) to produce beauty of form, harmony, and expression of emotion’ (Allen, 1992), however there are genres which lie outside of these categories by using techniques such as distortion and randomness or lack of form.

Often defining something as music would be incorrect to one listener, or to some listener at one point in time, yet as it is listened to more and music advances to become liked and thus defined as music. All of these aspects when it comes to defining exactly what music is and when we have reached it cause problems for the project of creating it artificially.

To be able to decide when we have reached an acceptable result and not just produced noise, we must reach some definition of music albeit a potentially arbitrary one. Given the difficulties and challenges already posed in this project and given its scope, a simple definition of music must be reached whilst allowing for there to be some sort of complexity involved. There have been few successful attempts at defining music by philosophers or musicologists. (Marcus T. Pearce, 2007) used a panel of expert judges to grade pieces of artificially generated chorales. ‘Judges were advised that their answers should reflect such factors as conformity to important stylistic features, tonal organisation, melodic shape and interval structure; and melodic form’. This would be an unreasonable criteria to judge this project on as the hardware available and the scale of the project do not allow it. However, it does provide some examples of criterion that we can use.

Here, we define our rules:

1. There must be a collection of sounds – that is that there must be multiple notes, and that there must be a reasonable number of unique notes for the length of the piece. Without this, it would simply be the repetition of a single note.
2. There must be some degree of adherence to scales and modes – that is whilst there may be mistakes or lack of structure to a piece, sequences must have some tonal organisation. Without this, it would simply be noise.
3. There must be a degree of structure – for example, 100 notes played at one time would not be music and would be incomprehensible noise. Also, there should be something which constitutes as a melody, rather than just a collection of sounds.

## 4.2 Comparing Degrees of Success

While there are a number of ways to clearly define whether a model is creating new music, or simply repeating the songs it has been trained on, or whether it has learned nothing, it is unclear how to measure the degree of success of a model. Consider, one model produces a song which is listenable but rather boring and fits our rules, when the second model 90% of the time produces rubbish that doesn't fit our rules, yet 10% of the time produces something interesting, novel and fits our rules. It would be difficult to consider which one would be the winner.

Given the scope of the project, it would be fair to say that either of these scenarios would be considered a success, yet we must find a way to compare the two to some degree. Here, we must use a combination of what is already measurable, namely the loss, accuracy and also the potential for further training. If a model has reached its plateau, then it is unlikely to learn much more given many more epochs. However, if the model seems to have more room for learning as its accuracy and loss are still changing then it can be considered more successful. When paired with our ideas of success, we can look at how often good music occurs, how much of it is produced and how much more potential for learning the model has and measure its overall effectiveness.

# 5 Implementation

This section covers the steps taken to implement each of the models including the pre-processing of the data set, the steps taken to identify the best potential level of complexity of the networks for our purposes and the method of prediction.

## 5.1 Data Pre-processing

### 5.1.1 The Original Dataset

When choosing a dataset for a machine learning task, it must be ensured that the dataset is a good one. The ‘garbage in, garbage out’ principle applies here, meaning that if we feed in poor, incorrect or badly managed data into even the perfectly designed models we will yield poor results. To find a good dataset for music generation we must consider the format, the source and the quantity. Inspired by the paper ‘Enabling Factorized Piano Music Modeling and Generation with the MAESTRO Dataset’ by Douglas Eck. Et al the MAESTRO dataset was chosen after a review of the dataset to ensure that it fulfilled all of the project’s requirements.

#### 5.1.1.1 Availability

The first step of locating a good dataset is locating and examining it. This should be done relatively quickly and legally. Therefore, an open licence and reliable download location is a must. After coming across the MAESTRO dataset, it was found that it was available under a Creative Commons Attribution Non-Commercial Share-Alike license meaning that it was fair to download, use and share as desired. The dataset was made available to download from Magenta, who are a part of Google, and so the data could be downloaded in relative safety. After these considerations, the dataset was downloaded for investigation.

#### 5.1.1.2 Midi as a Format

As discussed in the technical review, it was decided due to the available features and ease of use that Python was going to be used with the Music21 library. This library takes midi files and converts them into objects which the computer can then handle. Therefore, we are limited to strictly using midi files for our dataset. This is acceptable however, as this is a widely used format for electronically stored music with a very

strict format that has the ability to contain many different features such as pitch, note, chord, instrument etc.

#### 5.1.1.3 Accuracy

As per the mantra of ‘garbage in, garbage out’ if the model is given a dataset full of musical mistakes and poor composing, then it can be assured that the model will return musical mistakes and poor composing no matter how long it is trained for, or how deep and complex the model is. Therefore, it is imperative that we ensure the accuracy of the music is as high as possible. However, given the size of the dataset and the number of notes, chords and possible relations between each of these it would be physically impossible to sift through it all or even write a script to do so. Instead, it was important to find a source of a dataset which is created by experts. The MAESTRO dataset fulfils this requirement by being created as part of a competition between electronic piano virtuosos, with music from famous composers such as Johann Sebastian Bach and Ludwig Van Beethoven. These files were created automatically using electronic keyboards and thus no transposition error from their playing could occur from human error. It can then be assured that the MAESTRO dataset is of sufficient accuracy for the project.

#### 5.1.1.4 Quantity

When locating a great dataset for machine learning the quantity of data put into the network has a few effects. If the amount of data is too small, then it would be incapable of learning any real patterns of the data and instead would be more likely to overfit. For example, if a dataset consisted of only a single song the algorithm would simply learn how to play that song – or at least the part in the training split. At validation the algorithm would fail to predict any of the patterns possibly even resorting to just playing the most common note from the validation set repeatedly. Therefore, the dataset needed to be large enough for the algorithm to be able to learn the patterns of how notes are related to one another. However, if the dataset is too large then the time taken to train would become too large and the epochs would take too long given our limitations. As the MAESTRO dataset is split into many different songs of reasonable length they can be split into as small or large a dataset as required.

#### 5.1.1.5 Conclusion

Overall, after careful consideration of the factors involved in finding an effective dataset for the project on the MAESTRO dataset it was found to be a great solution.

### 5.1.2 Feature Selection and Removal

An important step in data preparation is deciding on what features are going to be used from the data set. This means considering what information is available from the data, how each potential feature is related to one another and the impact it is going to have with regards to overfitting and underfitting. There must be enough data to be able to form relations, whilst not so many features that relations which are not there are formed, or the algorithm only learns the training data. With regards to the aims of our project, the most important features which could be identified are:

- **Note:** An alphanumeric representation of a musical note. This provides the note e.g. ‘C’ followed by its octave e.g. ‘2’.
- **Chord:** An alphanumeric representation of a chord, which is composed from several notes e.g. the D Major chord would be represented as ['D3', 'F#4', 'A5'].
- **Pitch:** A numeric representation of the degree of highness or lowness of a tone. E.g. A#5 would be represented by the number 82.0.
- **Velocity:** A numeric representation of the hardness of a note, i.e. how hard a note has been struck or played.
- **Duration:** A numeric representation of how many quarter lengths a note is played for. For example, a quarter length would be represented by 1, whereas an eighth note would be represented by 0.5.

After identification of important features for consideration, an evaluation of how these features relate to each other must be conducted to identify what effects each of them may have on a network’s ability to learn and the patterns and rules which will be observed.

- The most fundamental parts of a song, those which a song would not be a song without no matter which definition of music is chosen, are the notes. Without these there would simply be no sound, and thus are a feature which is non-negotiable.

- Given the similarity between a note and a chord with regards to meaning, and the fact that if a note is played at the same time as a chord, this is simply a change of the chord, rather than a separate feature.
- Pitch is a more accurate representation of a note, as a note may lie anywhere within a range of pitches. Therefore, the range of the pitch is directly correlated to the note or chord played. This does not offer much predictive capabilities and in fact would potentially overload the network with meaningless data. Another drawback of including pitch is that it would unnecessarily increase training time. Therefore, pitch was not included as a feature.
- Velocity is an important part of music as it offers another method of expressing mood and can give subjective meaning to a song or part of a song. For example, repeated high velocity notes in a song can aid to the perception of the intended high energy of the song. However, it is unclear as to the relations that velocity may have with the other potential features. It is clear that any note may be played hard or soft, and any attempt to find patterns here could lead to overfitting. Finding a model which could identify patterns in this data is outside of the scope of this dissertation and as such was left out from the features.
- Duration is an important part of music as it provides the song with a rhythm. However, given that any note or chord may be played for any duration it would again be incredibly difficult to find any correlation between these features. However, it may be possible to find a pattern in the durations themselves, for example identifying a triplet which is three notes played inside another note length. Therefore, duration was a feature that was intended for implementation. Unfortunately, this did not become possible as

### 5.1.3 Data Conversion

The initial step in converting the data from a Midi format into a format usable in a neural network is reading the data. This is accomplished by reading each file individually and turning them into score objects. This score object contains information such as the instrument played, and each note played by that instrument. To access information such as notes played by an instrument, scores must then be partitioned by instrument. Then the notes played by the instruments (in this case, a single piano) is iterated over and each note and chord is parsed. If a note is being

parsed, it is simply added to an array in its alphanumeric state. However, if a chord is to be parsed it is passed to the array as a sum of its parts, with the notes which make up the chord separated by a period. After this list of notes and chords is created, it is passed back to be further processed.

Next, a numeric dictionary must be produced from the alphanumeric array so that it may be used later to convert the entire list of notes and chords into a numeric list which may be passed to the neural network, and so that it can return its predictions from the numeric state back into its alphanumeric counterparts.

The next step is to divide the notes up into sequences, with each sequence having one output. The sequences are of length 100, which is a standard starting position for sequence length prior to the hyper parameter tweaking which occurs after the initial stage of implementation. The purpose of these sequences is that for each note that is to be predicted, the algorithm has each note in the sequence to aid in its prediction.

For the list of sequences to be used for an LSTM in Keras, they must be of a defined shape. This is due to Keras and Tensorflow not allowing for the computational graphs shape to be defined dynamically. The implementation of LSTMs provided by Keras expect a 3-dimensional shape where the first dimension is the batch size, the second dimension is the number of time steps (or notes/chords in our case) and the third dimension is the number of features.

### 5.1.4 Normalisation

Normalisation of a dataset has the intention of bringing each of the features of the dataset to the same scale, usually between 0 and 1. After normalisation, we must one hot encode our data. The reason why we must do this is because we must convert our categorical data must be converted to a numeric form as Keras implements algorithms in an efficient way which requires numeric input. We must also one hot encode our data rather than integer encode our data, as integer encoding requires that there is an ordered relationship between a label and its data. However, as chords and notes do not have this, we must instead one hot encode our data to fit Keras implementations of algorithms.

## 5.2 Models

A model takes an input in the form of data which it then performs many mathematical operations on, before using the result to classify or make a prediction on some other input. In Keras, models are formed by stacking layers on top of each other. Before discussing the level of complexity that the models will contain, first there must be some understanding of each of the layers and the jobs that they perform.

### 5.2.1 Layers

#### 5.2.1.1 Dense

A dense or fully-connected layer is a classic layer in a neural network, where each node in the layer receives input from every node in the previous layer.

#### 5.2.1.2 LSTM/GRU/SimpleRNN Layers

Layers which contain either LSTM, GRU or SimpleRNNs are the layers of the model where the majority of the computations are. Each of these layers generally consists of 2 or more other layers, such as tanh. While they are able to be stacked on top of each other, and on top of other types of layers such as dense layers, there are certain parameters which need to be set for this to work. For example, if an LSTM layer feeds into a dense layer its return_sequences parameter must be set to false. If this is not done, then the model will fail to provide the desired results.

#### 5.2.1.3 Activation Layer

The purpose of an activation layer is to act as a middle man between an output and an input from one node to another, or as a final output. The reason this is necessary is that if there were no activation layer, the output of a node would be a function. Without the activation function the model would turn into a linear regression model. Due to the complexity of the relationships involved in music, a simple linear regression model is not suitable to make predictions thus an activation layer is necessary.

### 5.2.2 Model Complexity

An important decision was made for the consistency of results that each of the implementations of RNN, LSTM and GRU networks would have the same level of complexity. This means that they would have the same number of hidden units, with

the same number of dense layers, and the same type of activation function. This would allow for the results of each of the neural networks to be easily compared for their musicality. This is only applicable to our research which is into exactly how effective each of the networks are for creating music where it could theoretically be scaled to much longer training times and much larger amounts of hidden units or layers. In practice, RNNs could be trained for more epochs or with larger numbers of layers or hidden units due to the vastly decreased training time required compared to both GRUs and LSTMs. The same could be said for GRUs when comparing them to LSTMs.

- **Sentence length:** As previously discussed, a sentence length of 100 was the initial starting point. After a few tests on smaller datasets, this sentence length seemed like a reasonable fit as it should provide enough context about the previous notes and chords to give a good amount of data about the next note or chord to be predicted.
- **Number of hidden units:** During the tests on smaller datasets it was discovered that the number of hidden units should roughly correlate to the number of unique notes and chords. However, given that the larger the dataset means that it is more likely a unique note or chord is played, which will increase the total number. This number reached 767 so a hypothetical number of 800 would be considered a good number, but this would increase the time taken to complete an epoch given the current hardware constraints too much. Instead a smaller number of hidden units was reached with the idea that at least some constraints would be realised.

### 5.2.3 Prediction

The first step in generating predictions is to create a starting point for the algorithm to begin predicting from. This was done using a pseudo-random number generator, limiting the range to a number within the dictionary size. Then a similar method was used to create a song length. At this point the algorithm is fed the input which is divided up into sequences just like when the algorithm was being trained, however at this point it is just one sequence of a single note. However, as predictions are made

they are added to the sequence. This happens iteratively until the song length has been reached.

Once this process has completed it is time to refer back to the dictionary and convert our numeric array back into string formats. Once completed, they are required to be converted into a Score object which can be output to a Midi file for playback, or onto a visual score for visualisation. This is done by locating a period in the string which denotes that it is a chord and adding these chords to the piano instrument object. If there is no period found in the string, then it is added to the instrument as a single note. After each note is added, an offset is updated by 0.5. This denotes that each note should be played half a second after the last. This does not represent music but unfortunately it was not possible to get the models to work with multiple features, namely the duration, and so to allow us to listen to the music created we must separate the notes and chords from one another. Without doing so, it would just be a half second of every note in the song playing at one time. This allows us to fit rule 3 of our definition of music.

### 5.2.4 Gradient Clipping

On the first attempt at implementing a GRU algorithm, there were issues present. Huge spikes in loss and dips in accuracy were recorded until the algorithm stopped learning altogether, and the previous weights were lost. These issues were presented initially at epoch 25 before self-correcting. However, at epoch 90 issues increased in regularity until epoch 140 where everything was lost as seen in figure 9. It is unclear what caused this issue, however research into the issue showed that a likely candidate was gradient explosion, where the gradient value grows extremely large causing an exponential growth, an eventually an overflow resulting in a NaN. This was unexpected as over the same dataset with the same parameters neither RNN nor LSTM implementations experienced this. The results from this attempt were similar to that from the RNN attempt where a single note was output consistently, indicating no learning at all.

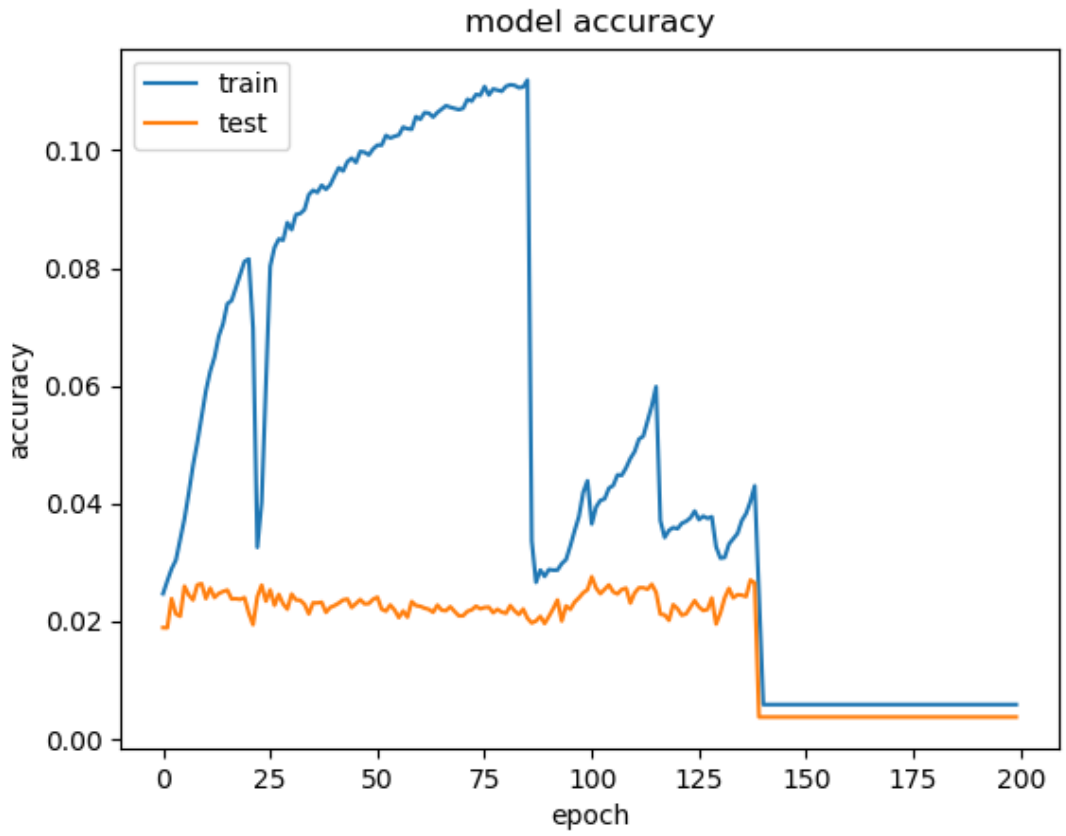



**Figure 9: GRU attempt #1 accuracy – it is clear that the model is experiencing issues throughout and that the weights are lost at epoch 140.**

Fortunately, Keras implements an easy solution to this problem in the form of gradient clipping. Here, a threshold value is set which prevents the gradient from ever increasing past this point. Therefore, the gradient will never reach an overflow. Unexpectedly, the iteration with gradient clipping caused the average time taken per step to increase from 4ms to 6ms. This decrease in performance of 33% greatly hindered the overall time taken to train increasing it to 44 hours from 30 hours.

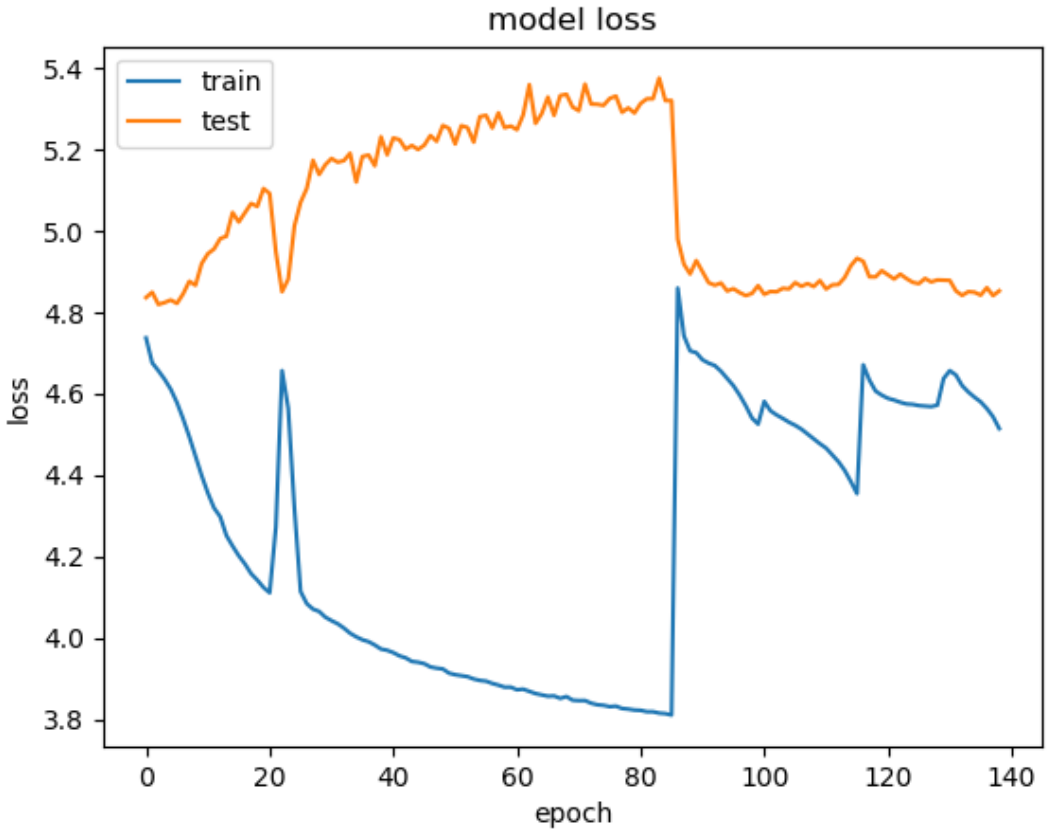



**Figure 10: GRU attempt #1 loss – loss explodes to an overflow, resulting in NaN at 140, and the end of the ability to calculate loss.**

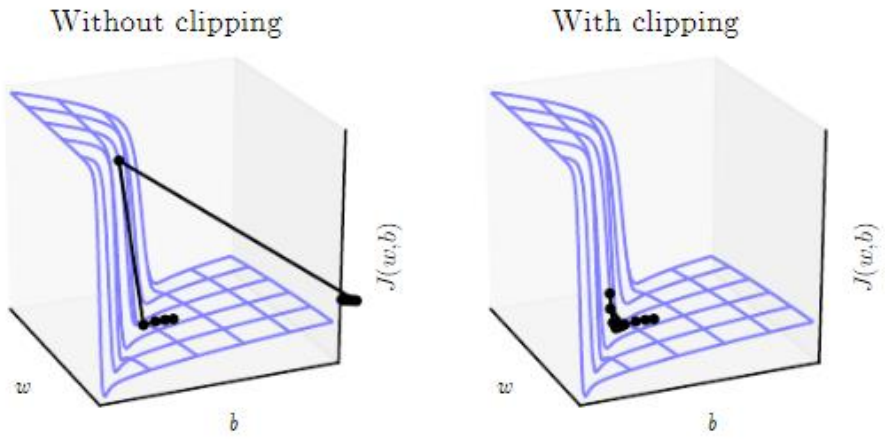



**Figure 11: A visualisation of an exponentially steep gradient, without and with clipping. (Ian Goodfellow, 2016) – It is possible to see how an overflow would occur on the left, and what a reasonable approach towards a steep gradient would look like on the left.**

# 6 Evaluation

The quality of the works produced by the algorithms is difficult to measure due to the unquantifiable nature of music. However, using the definition of music created earlier, the metrics available and some manual exploration of the results we can evaluate them and compare the results of recurrent neural networks, long short-term memory networks and gated recurrent units. Using these methods of evaluation, it has shown that LSTMs outperform both RNNs and GRUs at generating music.

## 6.1 Loss of Rhythm and Tempo

Due to implementation errors, it was not possible to implement the learning of multiple features. Therefore, duration as a feature was lost. This led to us having each note be a quarter note half a second apart. Thus, there is no learned rhythm or tempo. This was unfortunate, massively impacting the quality of the music produced. However, it is still possible to evaluate the effectiveness of the networks to learn scales and its ability to harmonise.

## 6.2 Approach

Each of the results were to be evaluated before being directly compared to one another. The steps taken to do this were:

1. Consider the accuracy and loss of the RNN, and use this as a base line to be beaten by the other models
2. Evaluate the quality of the music produced by the RNN
3. Evaluate the accuracy and loss of the LSTM and GRU
4. Compare these results to the baseline
5. Evaluate the quality of the music produced by the LSTM and GRU
6. Compare the music by these models to that produced by the RNN

## 6.3 Recurrent Neural Networks

The songs generated from RNNs were largely disappointing, even given the initial hypothesis that they would be outperformed by LSTMs and GRUs. The algorithm could not learn relationships between any of the notes or chords at all, and simply took to the method of guessing the same note every time. This note happened to be C2, which I suspect given its position on a keyboard to be the most commonly played

note on the dataset. In figure 12 it can be seen that the RNN's ability to predict is completely random and does not increase over the 200 epochs at all.

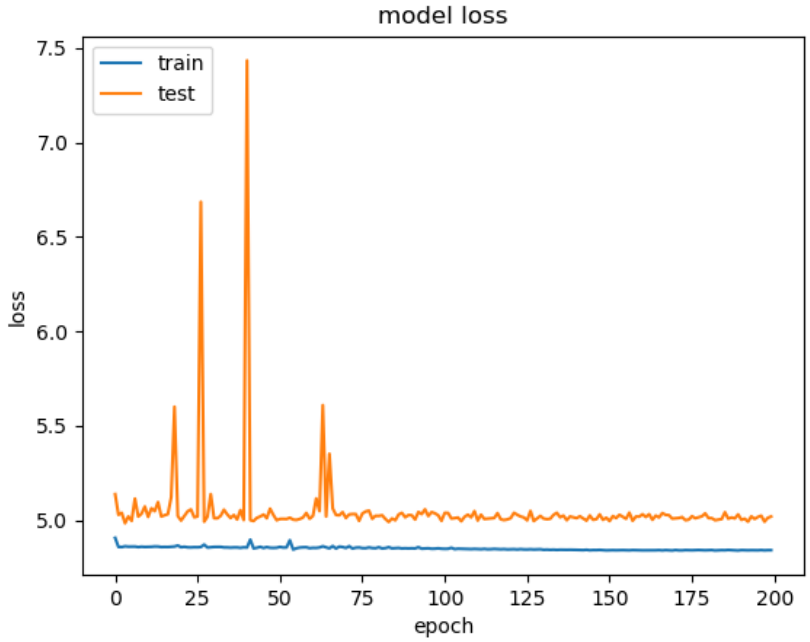



**Figure 12: RNN accuracy – a few spikes where learning is attempted, however they failed**

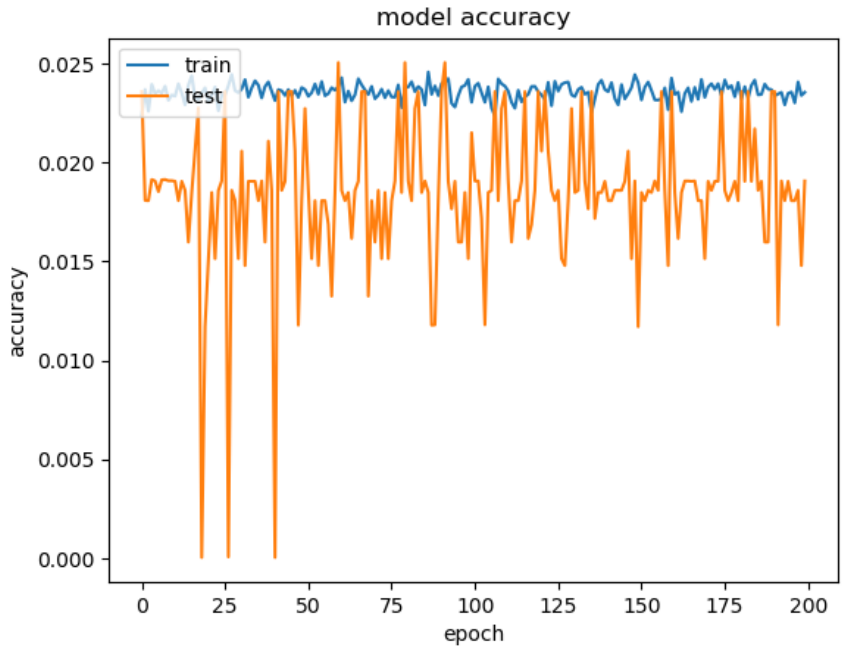



**Figure 13: RNN loss**

Figure 13 presents a visualisation of the loss of the RNN, further confirming that it was unable to learn. Spikes in the data could be interpreted as failed attempts at learning, which made the algorithm less likely to attempt different things and instead learned to predict the same note each time as can be seen in figure 14. This breaks rule 1 of our definition of music, and thus this cannot be considered a successful attempt at creating it.

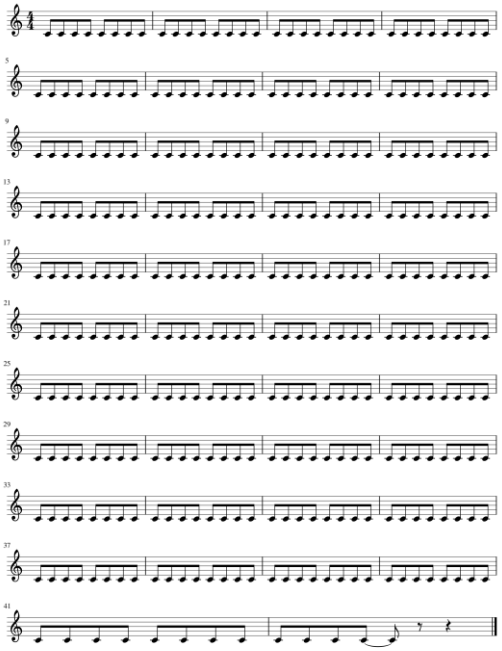


**Figure 14: RNN unable to produce anything similar to music**

## 6.4 Long Short-Term Memory

The results from LSTM networks, whilst slightly disappointing, did yield much better results to the RNNs. Countering the hypothesis there was a distinct lack of overall structure to the song, and the network overall struggled to produce nice sounding music. However, there are periods in each song varying between 3 and 20 seconds of correct music. This implies to me that the network was not complex enough to learn each of the different relationships between notes, such as all keys and the relationships between octaves, yet simpler or more common constraints were beginning to be learned after 40 hours of training.

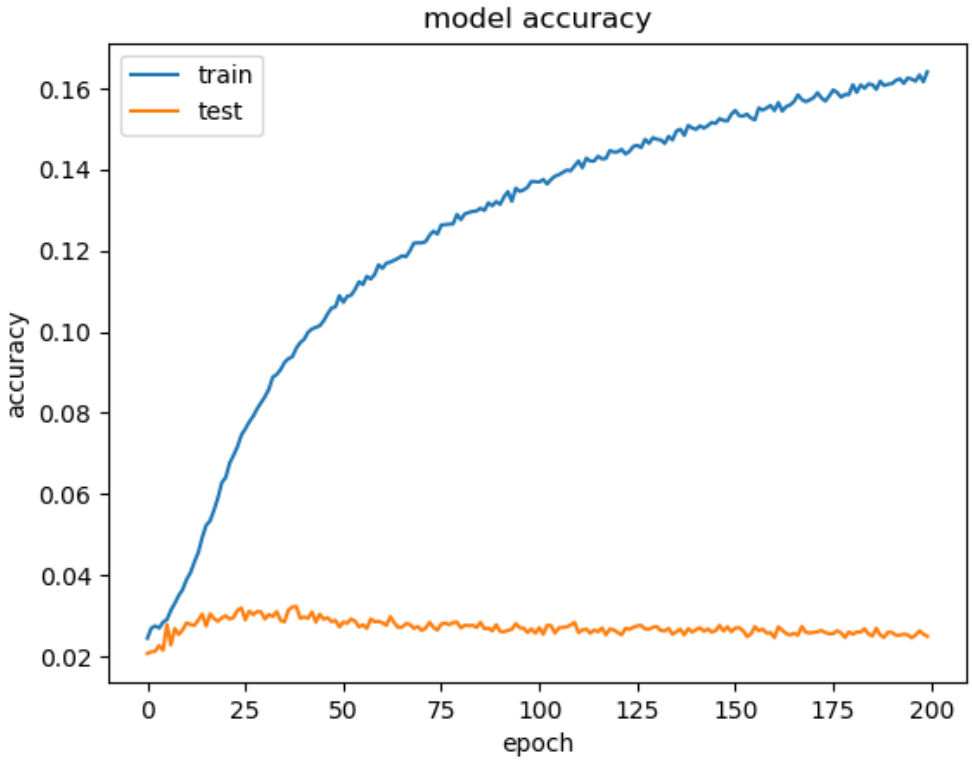



**Figure 15: LSTM accuracy – potential for further training is clear since the training set had not yet reached a plateau**

Contrary to previous experiments using neural networks completed during the learning phase of machine learning and Keras specifically, the accuracy and loss were both abysmally low on both the training and validation sets. While the accuracy did not increase on the validation set, it can be seen to be climbing on the training set in figure 15. Usually this would be a distinct sign of overfitting. This is compounded when looking at the loss, where the training loss is dropping and the validation loss is increasing in figure 16. Given the incredibly large number of potential notes and chords, and the number of reasonable choices for the next note and chord, it is not unsurprising to see such a low amount of accuracy on both model and accuracy. Therefore, despite having such low accuracy this is not necessarily a sign of

underfitting and instead points to the difficulty of the problem with a small amount of overfitting.

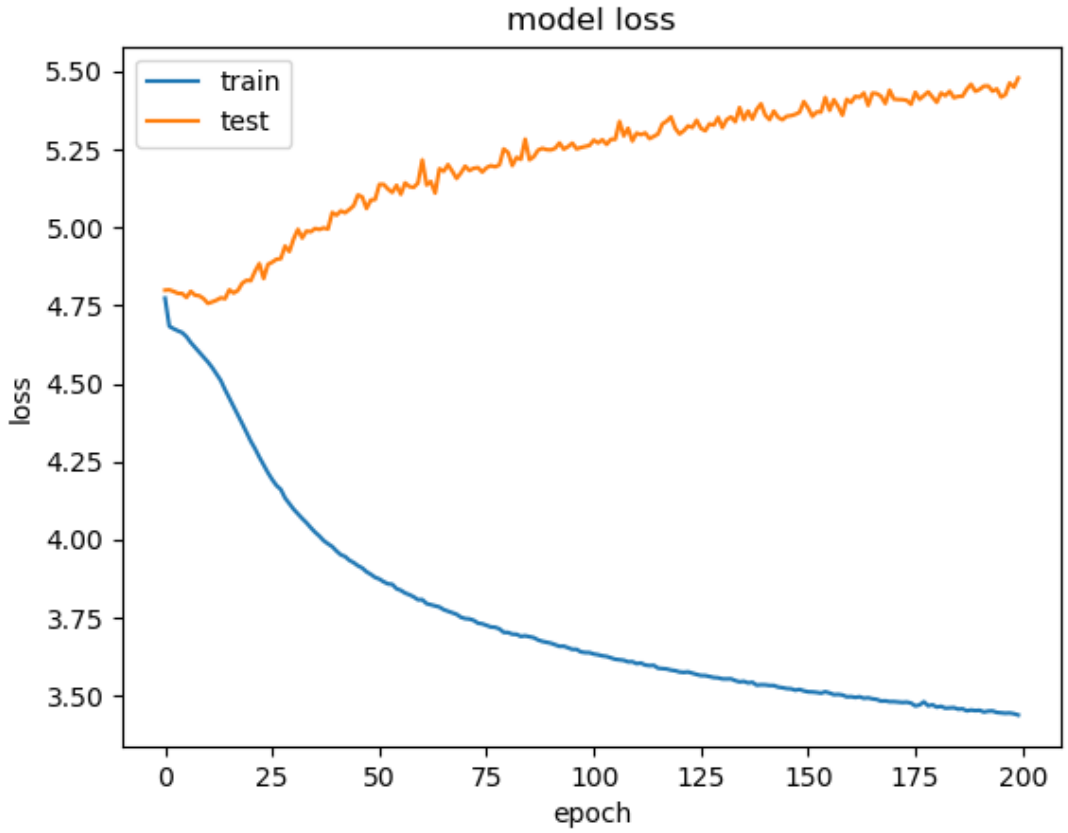



**Figure 16: LSTM loss**

However, considering that the songs produced were certainly not the same as, or overly similar, to any of the songs in the dataset, there is argument for loss and accuracy not being of particular importance here especially when considering the model would need to be more complex before any real overfitting would occur. Secondly, there is argument for a small amount of overfitting in music generation as it would allow for the reproduction of common chord progressions and note combinations. When compared to how humans create music, it could be argued that to some degree we 'overfit'. This is shown by the most common major chord progression in pop music, C–G–Am–F or even the twelve-bar blues. Each genre is likely to have a widely used chord progression, with it being unlikely that classical is any different (especially since early composers are likely to be the founder of many of them). Thus, I expect that a small amount of overfitting is actually a benefit when it comes to training networks for music.

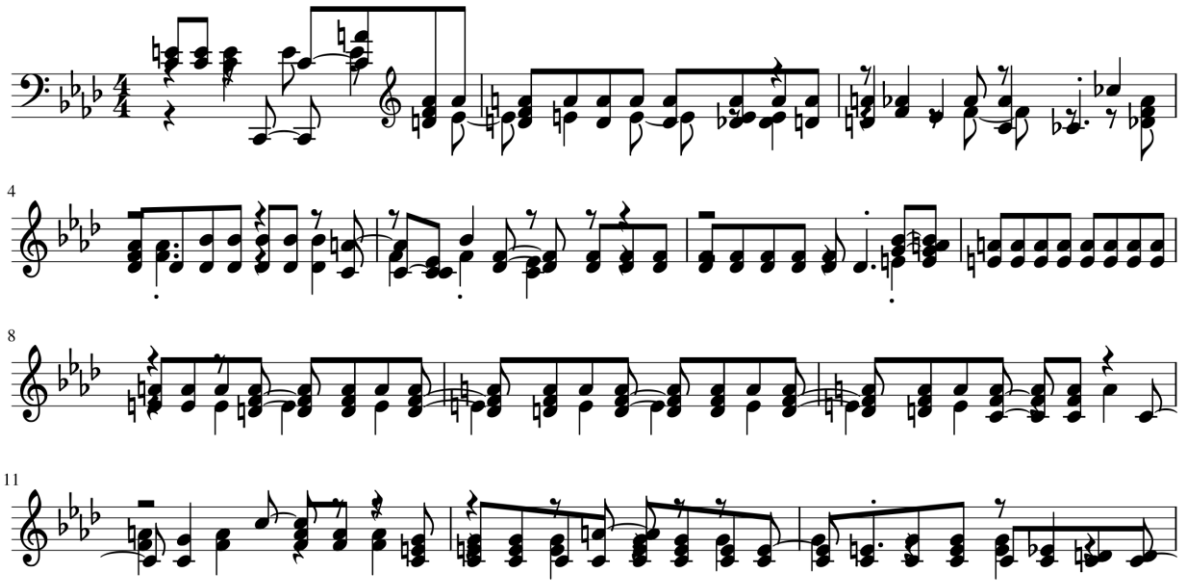


**Figure 17: Score created from LSTM generated music**

The score above demonstrates the LSTM's ability to learn and generate music, and that there is still a lot of potential for learning. The score begins with a nice little melody before beginning to repeat. However, at bar 8 begins a genuinely good piece of music with structure and some pleasing, interesting sounds. This section consists of the chords Dm, A, Fmaj13, F and F6 alongside the notes A, C and G. This is consistent with the C dorian scale, which the notes fit, and the chords harmonise well with whilst being commonly used in western music. Eventually, this piece begins to repeat itself however these results indicate that with further training LSTMs could be

a very effective method of artificially generating music. Other pieces provided examples of music with structure over a relatively large temporal space, reaching 33 timesteps with only a few off notes.

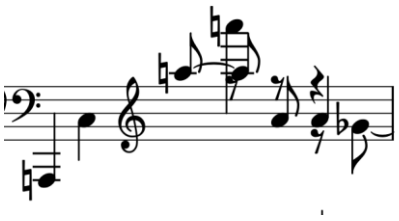


**Figure 18: Structured music - rising and falling pitches**

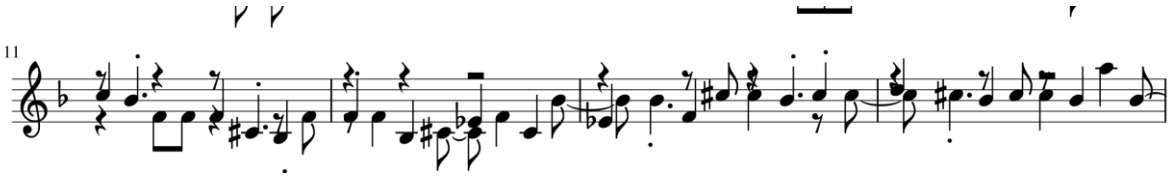


**Figure 19: Longer temporal structure (33 timesteps)**

## 6.5 Gated Recurrent Units

The results produced from Gated Recurrent Units were extremely disappointing. The problems exhibited by the solution of exploding gradients for the GRU caused the training time to be increased by 33%, yet the predictive capabilities were non-existent. This was unexpected, as the training accuracy had been increasing, and the loss decreasing, at the rates predicted which were comparable to the LSTM.

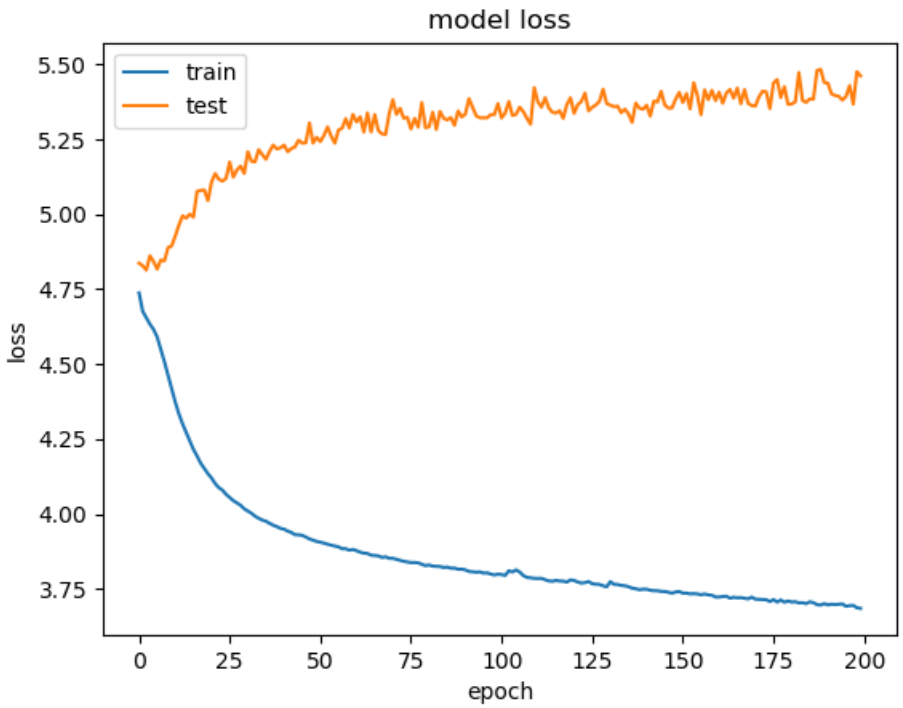



**Figure 20: GRU loss – similar to the LSTM results**

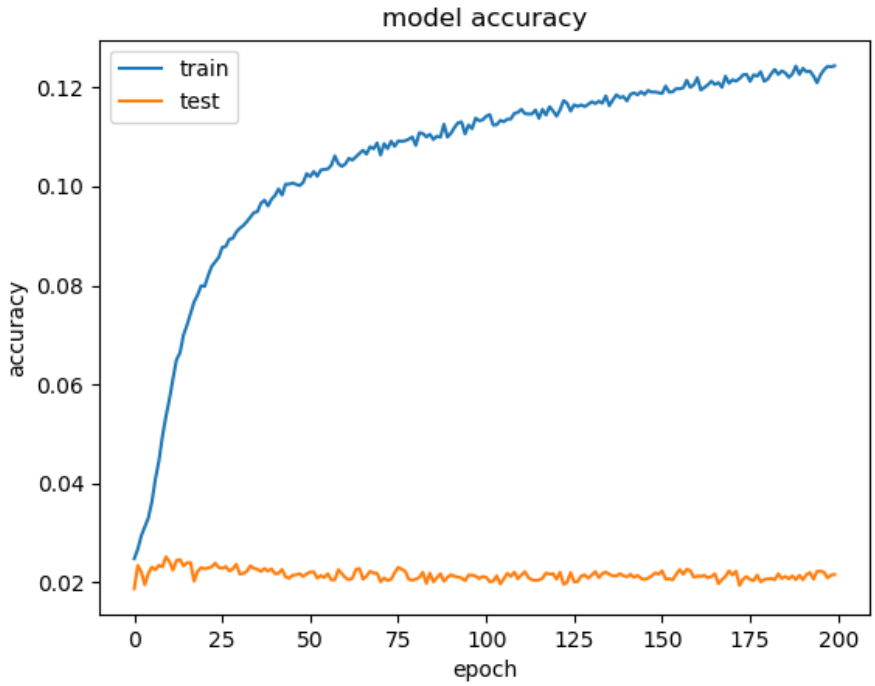



**Figure 21: GRU accuracy – also similar to the LSTM results although with a slightly lower accuracy**

As can be seen in figure 22 the accuracy improved at a comparable rate to the LSTM. This was expected and overall a positive result. The loss produces similar conclusions, where both implementations' loss levels after 200 epochs are at roughly 5.5. I am unsure to what caused the GRU to perform so poorly as all the tests indicate that this should not be the case.

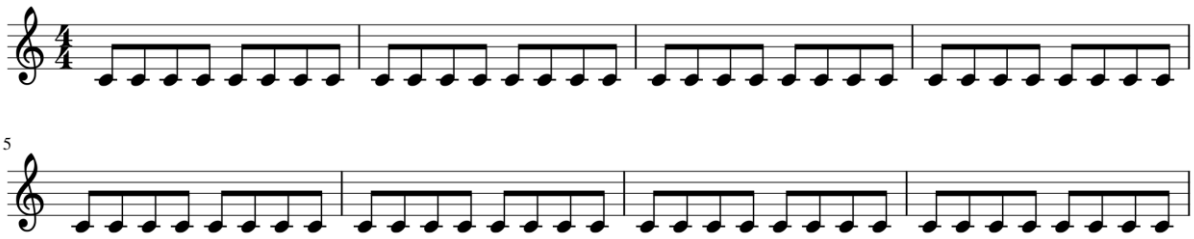


**Figure 22: GRU unable to produce any music**

# 7 Conclusions and Future Work

This project aimed to compare and contrast the benefits and overall effectiveness of three different types of neural networks, RNNs, LSTMs and GRUs in artificially generating music. To achieve this, a foundation to build upon was created by reviewing past works with particular attention made to their successes and failures, and the problems they faced and solutions for those problems. After this, it was important to figure out what technologies and tools were at my disposal which would make the task of solving my own questions as straight forward as possible, and then use those tools and the information gained to implement the algorithms. After this, an analysis was conducted which showed that LSTMs were indeed the best choice for artificially generating music as both GRUs and RNNs simply failed to learn any patterns.

## 7.1 Summary

The aims and expectations of this project were met, although to varying degrees. It was possible to use machine learning techniques to create music, however even despite the previous discussion is remains difficult to measure the effectiveness of any success.

While the initial hypothesis was that RNNs would be the worst performing of all of the implementations, it was not expected for them to completely fail. No learning was produced at all, and as such they produced nothing which fit our definition of music.

The area where this project excelled was in the LSTM implementation. The results whilst being slightly overfit at times were overall of a high standard given the features provided. It was clear that music could be generated using these given more hardware and time.

Due to unconfirmed reasons the initial GRU implementation completely failed, causing further research and delay late into the project. Even after a potential issue, exploding gradients, and a solution for this problem was found the next implementations results were unsatisfactory and no improvement upon the results from the RNN implementation.

The implementation struggled to deal with multiple features which would have potentially improved the results incredibly, adding many different aspects of complexity, likability and novelty to the music generated. One of the main aspects of choosing the MAESTRO dataset was that it was recorded from virtuosos playing their instruments and gave the computer the ability to learn velocity and duration from humans yet this was unfortunately not implemented.

The project managed to meet all of its aims and objectives at least to some degree:

- Thorough research was conducted on past attempts at artificially generating music, highlighting important breakthroughs in the field and identifying the problems and solutions they provided with regards to machine learning using sequential models over large temporal events
- The project successfully explored each of the models in the original plan, although the success of these explorations varied

## 7.2 Critical Analysis

From the perspective of a researcher, Keras is not necessarily the best tools to use. Whilst it was chosen to allow for a quick development time and the ease of getting to know machine learning and what is going on, it lacks tunability. That is, Keras is an API which covers up a lot of the minute details which may be important. For example, with a lower level library it may have been possible to identify exactly what was causing the GRU to break and produce results that contradict the current ones. Therefore, while Keras was an acceptable choice given the aims of the project in reality a lower level approach may have yielded better results.

Given that the only quantifiable results this project produced was in the form of loss and accuracy graphs, the only evidence available points towards GRUs and LSTMs overfitting on the dataset. Overfitting on a dataset is almost always a bad thing (although as previously discussed a tiny amount is acceptable) as it leads to an inability to generalise and thus when it comes to prediction will simply give back the dataset. It is difficult to distinguish whether our dataset has overfit, and the positive results are also exactly the same as some data, as there are so many sequences and no form of automation available to do so. However, this does not affect the aims

and objectives of comparing the models as the results of the LSTM, overfit or not, are still superior to either RNN or GRU.

## 7.3 Personal Analysis

Overall, I would consider the project to have been a success. All of the aims and objectives were attempted and yielded some results. While the GRU failed to perform to the expectations and even the RNN underperformed, I feel the decisions made in the implementation of the project were the correct ones to make at the time when considering the scope. As a prototype getting the results from the LSTM implementation which demonstrated an ability to create music using its long memory that was actually at times decent, I feel that it was a success.

An aspect of the project which could have definitely been improved upon is the time management. The project timeline was not adhered to strictly enough due to other commitments both at home, work and in university such as deadlines for coursework. While this is unfortunate, this is something that should have been planned for and I put this down to a lack of foresight. Relatedly, the main project idea kept changing and the scope increasing, leading to further work which there was not time allocated for. Another reason for time management being a point of weakness was burnout and overall lack of enthusiasm towards the project and university in general. As time progressed, the subject changed, other things got in the way I was feeling disillusioned towards the honours project and university in general with a feeling of just wanting out. This led to a certain degree of apathy towards doing the work which was planned over the Christmas period and into February causing a lot of work needing to be caught up on in very little time. After a meeting with Simon where he talked about me not being the only one, a final definition of the project finally being made and beginning to feel a bit better about the course in general in the second trimester, work began to progress much faster leading to an increased enthusiasm. From that point on, a very strict schedule was adhered to and overall the project management improved tenfold.

One setback during the course of the project which is still left partially undealt with is the GRU failing on first iteration. Since the GRU failed at the end of roughly 30 hours of training completely removing all potential for results this was time wasted very

close towards the end of the project, causing overall panic. My solution to this was a reasonable decision at the time, which essentially resulted in fixing a symptom of the problem since the cause of it is still unverified.

## 7.4 Future Works

This project produced good results for the LSTM iteration, however the GRU and RNNs both spectacularly underperformed. It would be good to run the experiments again without the gradient clipping on the GRU, and be able to tweak the networks until they eventually produced results. This would allow for the verification of the cause of the GRU failure, for the problem to be addressed, and for more accurate results and conclusions to be made.

To create more interesting music, it is important to be able to learn more features than just the notes and chords which are in the dataset. The main features left not investigated are the duration and velocity of notes. By creating a network which is able to predict these we begin to create real pieces of music. This could be done by creating a single implementation which deals with multiple features or creating multiple implementations which deal with the features separately, before bringing the results together for prediction. This is especially possible as the notes and chords themselves are not a good indicator for duration or velocity, and vice versa.

This project focused purely on a single genre, classical music, which was composed and played by virtuosos. It would be interesting to see how the project would have dealt with other genres, and potentially a mix of genres. This could include implementations which would allow for multiple instruments to be played, for example, all the instruments in a jazz band.

Most importantly, it would be important to confirm these results by doing further experiments on more capable hardware. This would allow for further process mining to increase the size of the dataset resulting in more learning potential and less chance of overfitting, with increasing the complexity of the models to allow for the potential for further learning. This is especially important since the amount of hidden units per layer did not match fully our number of unique notes and chords, which is a reason all models struggled to learn to some degree.

# 8 References


Allen, R. (1992). *The Concise Oxford Dictionary.* Oxford: Clarendon Press.

Brownlee, J. (2016, March 17). *Crash Course on Multi-Layer Perceptron Neural Networks*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/neural-networks-crash-course/

Brownlee, J. (2016, March 16). *Supervised and Unsupervised Machine Learning Algorithms*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/

Cai, E. (2014, March 19). *Machine LEarning Lesson of the Day - Overfitting and Underfitting*. Retrieved from The Chemical Statistician: https://chemicalstatistician.wordpress.com/2014/03/19/machine-learning-lesson-of-the-day-overfitting-and-underfitting/

Cho, K. V. (2014). *Learning phrase representations using RNN encoder-decoder for statistical machine translation.* Retrieved from arXiv preprint arXiv:1406.1078

Chung, J. G. (2014). *Empirical evaluation of gated recurrent neural networks on sequence modelling.* arXiv preprint arXiv:1412.3555.

Douglas Eck, J. S. (2002). *A First Look at Music Composition using LSTM Recurrent Neural Networks.* Tucino: Instituto Dalle Molle di studi sull' intelligenza artificiale.

Fortmann-Roe, S. (2012). *Understanding the Bias Variance Tradeoff.*

Hawkins, D. M. (2004). The Problem of Overfitting. *J. Chem. Inf. Comput. Sci.*, 1-12.

Hecht-Nielsen, R. (1992). *Neural Networks for Perception.* Academic Press.

Hochreiter, S. B. (2001). Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies.

Ian Goodfellow, Y. B. (2016). *Deep Learning.* MIT Press.

J. Kiefer., J. W. (1952). Stochastic Estimation of the Maximum of a Regression Function. *The Annals of Mathematical Statistics*, 462-466.

Joshi, P. M. (2018, August 30). *Generative vs Discriminative Models*. Retrieved from Medium: https://medium.com/@mlengineer/generative-and-discriminative-models-af5637a66a3

Karpathy, A. (2015, May 21). *The Unreasonable Effectiveness of Recurrent Neural Networks*. Retrieved from Andrej Karpathy Blog: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Karpathy, A. T.-F. (2014). CS231n Convolutional Neural Networks for Visual Recognition. *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 1725-1732. Retrieved from CS231n.

Marcus T. Pearce, G. A. (2007). *Evaluating Cognitive Models of Musical Composition.* London: Computational Creativity.

McGowan, D. (2015, September 11). *Scales! Wait. Come back.* Retrieved from Just Enough Theory: https://dalemcgowan.com/scales-wait-come-back/

Mozer, M. C. (1994). Neural network composition by prediction: Exploring the benefits of psychophysical constraints and multiscale processing. *Cognitive Science, vol. 6*, 247-280.

Nicolas Boulanger-Lewandowski, Y. B. (2012). *Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription.* Montréal: Dept. IRO, Université de Montréal.

Nitish Srivastava, G. H. (2014). *Dropout: A Simple Way to Prevent Neural Networks from Overfitting.* Toronto: University of Toronto.

Olah, C. (2015, August 27). *Understanding LSTM Networks*. Retrieved from Colah's Blog: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Pedregosa, F. V. (2011). Scikit-learn: Machine learning in Python. *Journal of machine learning research*, 2825-2830.

Ramcharan, R. (2006). Regressions: Why Are Economists Obessessed with Them? *Finance and Development, Volume 43*.

Sepp Hochreiter, J. S. (1997). Long Short-Term Memory. *Neural Computation*, 1735-1780.

Shaikh, R. (2018, October 28). *Feature Selection Techniques in Machine Learning with Python*. Retrieved from Towards Data Science: https://towardsdatascience.com/feature-selection-techniques-in-machine-learning-with-python-f24e7da3f36e

Silver, N. (2012). *The Signal and the Noise.* Penguin Group.

Simonini, T. (2018, March 31). *An introduction to Reinforcement Learning*. Retrieved from Free Code Camp: https://medium.freecodecamp.org/an-introduction-to-reinforcement-learning-4339519de419

Sutskever, I. V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 3104, 3122.

Todd, J. J. (1989). Modeling the Perception of Tonal Structure with Neural Nets. *Computer Music Journal*, 44-53.

Van der Aalst, W. M. (2010). *Process mining: a two-step approach to balance between underfitting and overfitting.* Software & Systems Modeling.

Verbeurgt K., F. M. (2004). *A Hybrid Neural-Markov Approach for Learning to Compose Music by Example.* Berlin: Springer.

Yoshua Bengio, P. S. (1994). Learning Long-Term Dependencies with Gradient Descent is Difficult. *IEEE Transactions on Neural Networks, vol. 5*, 157-166.

Zeitler, W. (2005). *All the Scales*. Retrieved from All the Scales: http://allthescales.org/

# Appendix 1 Project Overview

## Initial Project Overview

## SOC10101 Honours Project (40 Credits)

### Title of Project: Analysing and Creating Music with Intelligent Agents

### Overview of Project Content and Milestones

### The Main Deliverable(s):

An agent-based software application which will generate and play a musical score. The musical score will be entirely generated by the AI.

A report containing:

- Background reading and other work
- Evaluation techniques
- Approach, design and implementation discussion
- Self-evaluation
- References and appendices

### The Target Audience for the Deliverable(s):

- Artificial intelligence researchers with an interest in intelligent agents
- Composers/Musicians
- Those without musical knowledge who require original music (e.g. indie game developers)

### The Work to be Undertaken:

- Investigate research which has previously been conducted in this field or similar fields, identifying useful findings or where work could have been improved/furthered
- Investigate technologies which already exist and may be used in either research or in the application itself
- Specification:
    - Multi intelligent agents working together to create a musical composition in a specific genre which can be read or heard
    - The ability to save and load a composition generated

- Design
- Building
- Implementation
- Survey
    - Ease of use
    - "Likeability" of compositions
    - Usefulness
    - Comparison to previously existing software
- Analysis
- Evaluation
    - Does the application create an original piece of music?
    - How well does the application perform (speed, memory etc)?
    - How "good" is the music?
    - How "creative" is the music?
    - Are there ways to improve the intelligent agents?
    - Other potential future works
    - How have I done compared to the original scope?
- Testing
    - Unit testing
    - What methods can be used to test the success of creating music

**Additional Information / Knowledge Required:**

- Understanding of "rules" in music
- Intelligent agents, neural networks

**Information Sources that Provide a Context for the Project:**

- Prior art

    - Magenta by TensorFlow – "a research project exploring the role of machine learning in the process of creating art and music" - https://magenta.tensorflow.org/

- References


    - Fox, R. and Khan, A., 2013, January. Artificial intelligence approaches to music composition. In Proceedings of the 2013 International Conference on Artificial Intelligence (Vol. 2, pp. 575-581).

    - Goebl, W. and Widmer, G., 2009. On the use of computational methods for expressive music performance. Modern Methods for Musicology: Prospects, Proposals, and Realities, pp.93-113.


## The Importance of the Project:

- Composers and musicians will be able to use this as an aid – for example they may find a nice rhythm section in a generated composition which would fit in nicely into a new song, or they could use a generated piece as a backing track

- Researchers in creative uses of AI who need a replacement for human creativity

## The Key Challenge(s) to be Overcome:

- Music is largely subjective and as such it can be difficult to find ways to measure the success of a generated piece. Effort must be taken to devise a method of testing which is as objective as possible.

- Having AI which generates a piece which is pleasant to listen to is a very complex task and figuring out how to do this will be a major challenge.

# Appendix 2 Diary Sheets (or other project management evidence)

**EDINBURGH NAPIER UNIVERSITY**

**SCHOOL OF COMPUTING**

**PROJECT DIARY**

**Student: Sean Faughey** **Supervisor: Simon Wells**

**Date: 07/02/19** **Last diary date: 31/01/19**

**Objectives:**

Discuss progress from last week
Discuss concern over depth of knowledge required to be presented w.r.t. math in RNN
How much detail needed of frameworks/apis
Any knowledge simon might have of frameworks/apis/best languages

**Progress:**

Researched into the ideas presented but there were issues
Decided on RNN as the way forward
Discussed the beginning of prototype

**Supervisor's Comments:**

Happy progress is being made – need to make more though
APIs can be used but they must be discussed
Tensorflow, keras, Scikit-learn mentioned
Emphasis on usability as need to get up and running fast
Since project has changed need to ensure that there is some question to answer – not just a SE project since I'm doing CS

**EDINBURGH NAPIER UNIVERSITY**

**SCHOOL OF COMPUTING**

**PROJECT DIARY**

**Student: Sean Faughey** **Supervisor: Simon Wells**

**Date: 31/01/19** **Last diary date: 12/12/18**

**Objectives:**

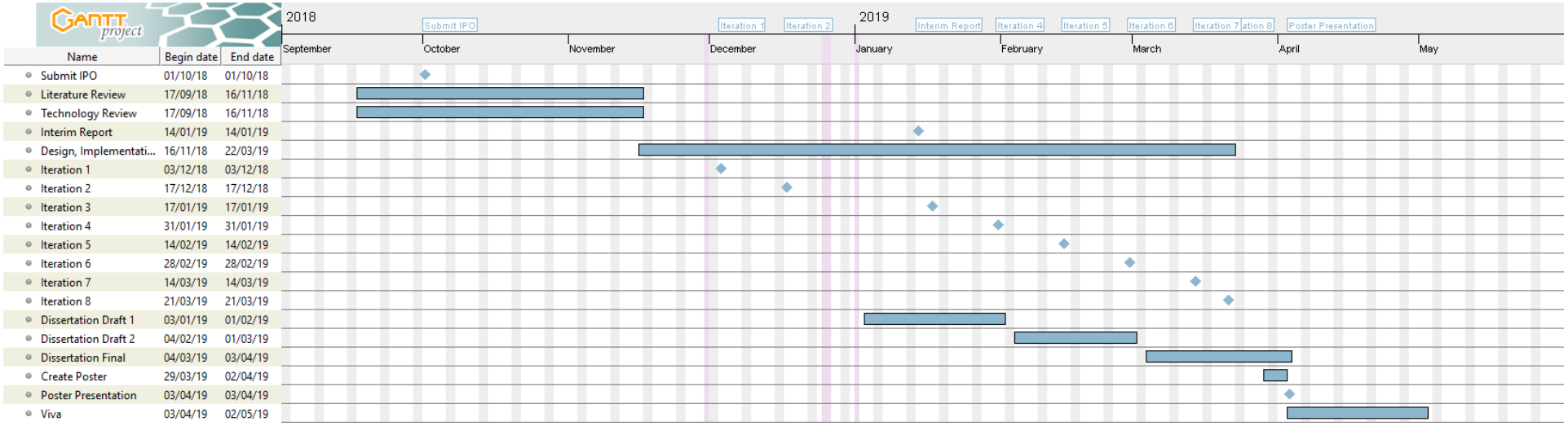
Falling behind – how to progress
Expanding scope worries
Sources for inspiration

**Progress:**

Due to the exam period, coursework, Christmas and other commitments not much happened since previous meeting

**Supervisor's Comments:**

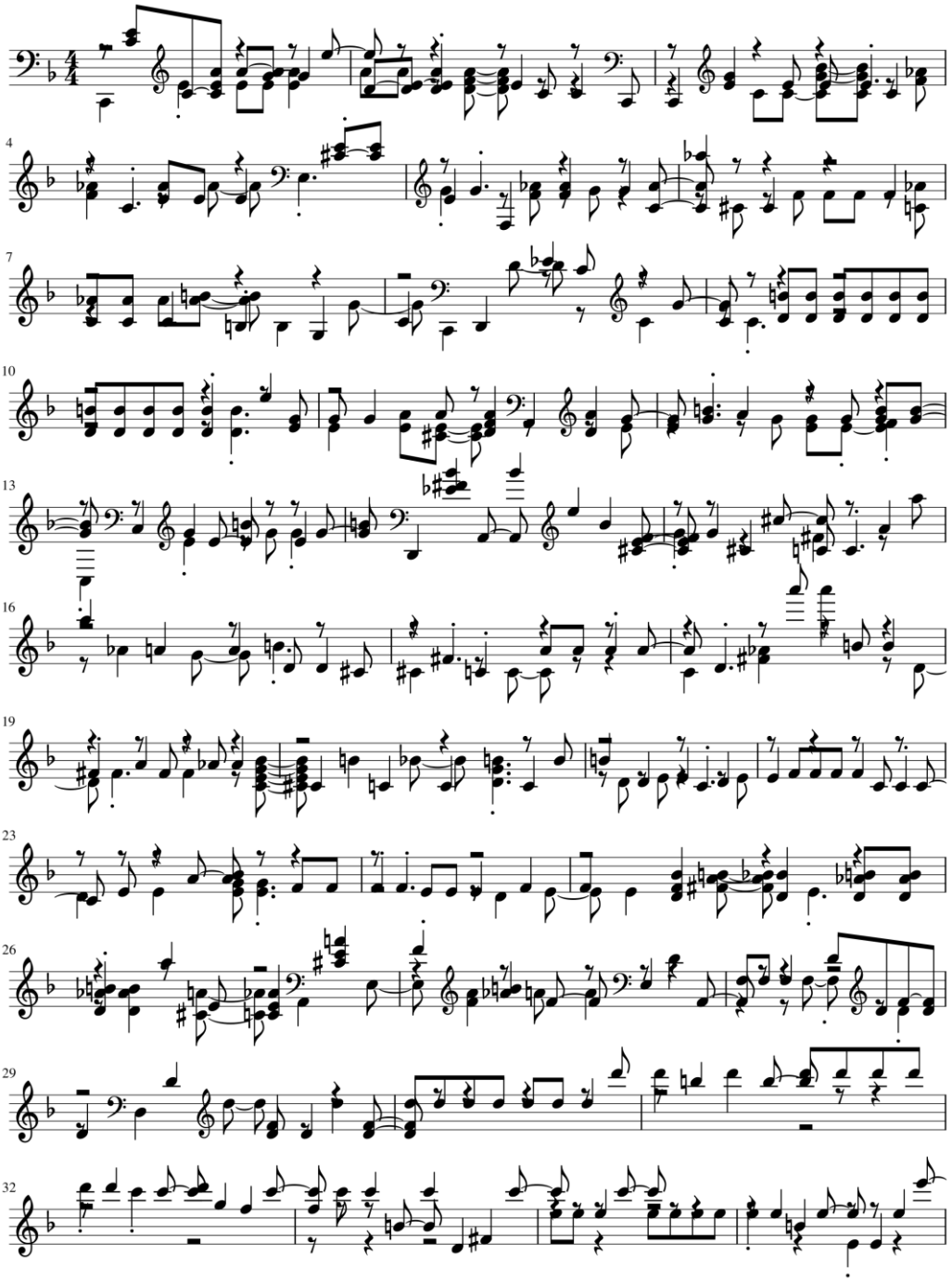
Simon explains I'm not alone
Solutions offered to minimise scope – need to check his github for ideas
Check music generated for games and other sources
Re-investigate multi-agent solution
Meet weekly


GANTT project
2018
2019
Submit IPO
Iteration 1
Iteration 2
Interim Report
Iteration 4
Iteration 5
Iteration 6
Iteration 7
ation 8
Poster Presentation
September
October
November
December
January
February
March
April
May
Name
Begin date
End date
Submit IPO 01/10/18 01/10/18
Literature Review 17/09/18 16/11/18
Technology Review 17/09/18 16/11/18
Interim Report 14/01/19 14/01/19
Design, Implementati... 16/11/18 22/03/19
Iteration 1 03/12/18 03/12/18
Iteration 2 17/12/18 17/12/18
Iteration 3 17/01/19 17/01/19
Iteration 4 31/01/19 31/01/19
Iteration 5 14/02/19 14/02/19
Iteration 6 28/02/19 28/02/19
Iteration 7 14/03/19 14/03/19
Iteration 8 21/03/19 21/03/19
Dissertation Draft 1 03/01/19 01/02/19
Dissertation Draft 2 04/02/19 01/03/19
Dissertation Final 04/03/19 03/04/19
Create Poster 29/03/19 02/04/19
Poster Presentation 03/04/19 03/04/19
Viva 03/04/19 02/05/19

# Appendix 3 Music examples from RNN, LSTM and GRU

RNN:

LSTM:



GRU: