DEPARTMENT OF COMPUTER SCIENCE

Neural Translation of Musical Style

Iman Malik

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Engineering in the Faculty of Engineering.

June 5, 2017

Declaration

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of MEng in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Iman Malik, June 5, 2017

Contents

1 Introduction **1**

1.1 Musical Notation 1

1.2 What is Style? 2

1.3 Visual Style 2

1.4 Related Literature 3

1.5 Project Aims 4

1.6 Overview of Thesis Structure 4

2 Musical Background **5**

2.1 Music Theory 5

2.2 Sheetmusic 5

2.3 MIDI 6

3 Technical Background **9**

3.1 Introduction to Machine Learning 9

3.2 Supervised Learning 9

3.3 Gradient Descent 10

3.4 Introduction to Artificial Neural Networks 11

3.5 Feedforward Neural Networks 14

4 Methodology **19**

4.1 Recurrent Neural Networks 19

4.2 Long Short-Term Memory Networks 22

4.3 GenreNet 25

4.4 StyleNet 26

5 Implementation **29**

5.1 Data Preprocessing 29

5.2 MIDI Encoding Scheme 31

5.3 Training Experiments 33

6 Evaluation **39**

6.1 Overview of Experiments 39

6.2 "Identify the Human" Test 39

6.3 "Identify the Style" Test 44

6.4 Final "Identify the Human" Test 45

7 Conclusion **47**

7.1 Summary of Contributions 47

7.2 Future Work 47

A Additional Material **55**

List of Figures

2.1 A staff with notes. 6

2.2 Range of Dynamics. 6

3.1 The Neuron. 12

3.2 Sigmoid, tanh, and ReLu. 13

3.3 A Feedforward Neural Network. 14

4.1 An unfolded RNN. 19

4.2 A Bi-directional RNN. 21

4.3 The solid line shows a trajectory that results in exploding gradients. 22

4.4 A LSTM cell. 23

4.5 GenreNet 25

4.6 Bromley et al's Siamese Neural Network Architecture [8] 27

4.7 StyleNet with two GenreNet units 27

5.1 Histograms of velocity range across MIDI. 30

5.2 Input and Output Matrix 32

5.3 True and predicted velocity matrix for dropout p = $[0.5, 0.8]$. Track: YOO01.mid 36

5.4 True and predicted velocity matrix for different encoding schemes. Track: chpn-p20.mid 36

5.5 Experimental runs. 38

6.1 "Identify the Human" survey example question. 40

6.2 "Identify the Style" survey example question. 40

6.3 "Identify the Human in a long performance" survey example question. 40

6.4 “Identify the Human” survey results 42
6.5 “Identify the Human” survey Q3 velocity matrices. 43
6.6 “Identify the Human” survey Q15 velocity matrices. 43
6.7 “Identify the Style” survey results 44
6.8 Distribution of responses for the final “Identify the Human” experiment 45
6.9 Final “Identify the Human” survey results 46

A.1 Training snapshot of StyleNet’s predictions. 56
A.2 Poster. 57

Abstract

Can a machine learn to play sheet music? This thesis investigates whether it is possible for a suitable computational model to learn musical style and successfully perform using sheet music. Music captures several aspects of a musician's style. Musical style can be observed in the unique dynamics of their performances and categorising genre. Style is difficult to define but there is a perceivable relationship between dynamics and style. This thesis investigates whether it is possible for a machine to learn musical style through the dynamics of music. Great advancements have been made in music generation using machine learning [47, 10, 52]. However, the focus of previous research has not been on capturing style.

To capture musical style through dynamics, a new architecture called StyleNet is designed. The designed architecture is capable of synthesising the dynamics of digital sheet music. The Piano dataset is created for the purposes of learning style. The designed model is trained on the Piano dataset which contains Jazz and Classical piano solo MIDIs. Different configurations and training techniques are experimented with. The model's generated performances are then assessed by a musical Turing test. The model's ability to perform in different styles is also evaluated. The research concludes that StyleNet's musical performances successfully pass the musical Turing test. This opens many doors for using such a model for assisting the creative process in the music industry.

To summarise, my main contributions and achievements in this project can be listed as follows:

- I designed Stylenet which is a neural network architecture capable of synthesising the dynamics of sheet music.
- I implemented StyleNet using the Tensorflow library with a total of 1000 lines in Python.
- I implemented a batching system to efficiently train the StyleNet model with a total of 500 lines in Python.
- I designed the data representation format for StyleNet.
- I implemented a data preprocessing pipeline for MIDI files with a total of 2000 lines in Python.
- I experimented with different StyleNet configurations and designs.
- I created the Piano dataset which contains a total of 649 Jazz and Classical piano solo MIDIs.
- I successfully trained a StyleNet model which passed the musical Turing test by producing performances that are indistinguishable from that of a human.

Supporting Technologies

- The Tensorflow library was used to implement the designed StyleNet architecture [30].
- Parts of the mido library were used to parse MIDI [29].
- Logic Pro X was used to visualise the generated MIDI performances[56].
- A simple MIDI-parsing script was extended to adapt its functionality to the Piano dataset [4].
- Parts of the pretty-midi library were used to modify MIDI instrument tracks[37].
- The HPC Zoo was used to train the StyleNet model. The Titan X, and GTX 1080 Ti were primarily used for training [62].

Acknowledgements

First and foremost, I would like to express my sincerest gratitude to my supervisor Dr Carl Henrik Ek who has supported me throughout my thesis. His motivation, enthusiasm, and immense knowledge helped me immensely in the time of research and writing of this thesis. Our weekly project meetings were always incredibly inspirational and enjoyable. I could not have imagined having a better supervisor and mentor.

Also I would like to thank my best friend, James Sewart, for his insightful comments and continuous encouragement. The discussions over our daily lunch breaks provided support and laughs throughout the duration of my research.

Last but not the least, I would like to thank my parents for their love, motivation and support.

Chapter 1

Introduction

Music is mysterious. Anthropologists have shown that every record of human culture has some aspect of music involved yet the exact evolutionary role of music is shrouded in mystery. Scholars theorise and state that music must have emerged as an evolutionary aid [46, 20]. One theory proposes that music may have arisen from mothers putting their children to sleep [12]. Some propose that the function of music was to provide social cement for group action [38, 46, 25]. War songs, national anthems, lullabies are all examples of this.

Music also appears to be hardwired into us since birth. A study showed that babies would dance to all types of music and suggested that humans have a predisposition for rhythmic movement in response to music[60]. Recent research has uncovered that the human brain picks up the beat of music automatically[7].

Music is fundamentally a sequence of notes. Humans construct long sequences of notes which are then labelled as music. These sequences of notes are usually articulated through instruments. The audiological form of these notes is then listened to in the form of music. These songs can convey an emotional and psychological experience for the listener [44, 32]. If one were to reverse the notes in the song, it would most likely not be called music anymore. This highlights the complex structure of music.

1.1 Musical Notation

Languages contain structure. Humans write to communicate their thoughts through language. Musical ideas need to be communicated too. Since prehistoric times, humans have been using musical notation to represent their aural perception. From clay engravings to western musical notation, each notation differs significantly from the others. Musical notation has gone through a series of incremental innovations to satisfy the evolving musical demands of each era [50].

The earliest records of musical notation date back to 2000 BC [54, 55]. In Sumer which is now known as modern day Iraq, archaeologists uncovered clay tablets that displayed a musical notation in one of the oldest systems of writing. More recent examples are the Baroque and Classical period. Interestingly, Baroque notation provides less encoded information than Classical [50]. Often the dynamics of the music are not captured. This missing property leaves the score open

to interpretation. In other words, two people could have very different ways of playing a Baroque piece. One could say the same for all genres as the understanding of the tempi and dynamics of a musical piece are relative to the performer's interpretation.

1.2 What is Style?

Style is difficult to define. One can observe performers displaying their individualistic interpretations of music at live concerts. Listeners subconsciously process the perceivable features of these performances. A possible breakdown of these features is dynamics and tempi. This can also be understood as the variations in loudness and timing. Another possible breakdown is the melody and harmonics. These features allow us to label the music in certain ways. Through listening to these features, one is usually able to assign a descriptive label to a performance such as a specific composer or genre. A listener familiar with music can easily tell if a song is Jazz or Classical through these features.

It can be assumed that a set of these features can represent audible "style". These features differentiate artists from each other. They also differentiate genres. This can be observed using the example of Jazz and Classical music. The style of Jazz is different from Classical. Thus any artist that plays Jazz music will have a different style to that of a Classical artist.

1.3 Visual Style

The concept of visual "style" can also be observed in paintings through visual features. Painters express individualism through these features. The style of Picasso is noticeably different from Van Gough. Gatys et al. [13] successfully utilised machine learning to teach a machine the style associated with a certain artist. The results of this were that the artist's style was transferrable onto a photograph. Similarly, Zhu et al. [61] were successfully able to convert a picture of oranges to a picture of apples and vice versa. This is analogous to learning the visual style of different fruit and applying a certain style onto a photo. There have been other several successful attempts at transferring style between photographs [31].

Machine learning has been extremely successful in the domain of images. Such algorithms allow us to teach machines without explicitly programming them. The use of machine learning in the field of Artificial Intelligence (AI) has lead to machines reaching human parity in tasks such as speech recognition [57]. For image classification, AI has surpassed human accuracy [26]. These tasks all display the exceptional ability of machine learning algorithms.

In the past, the field of AI made several promises but failed to meet expectations. This period is called the "AI winter". During this time AI research was partially stalled. The start of the winter is assumed to be 1987 [51]. This was mostly due to the difficulty in training AI. An example of this are the expert systems that were explicitly programmed rule-based systems. These were extremely hard to maintain which lead to their downfall. Now, AI research has slowly come back and is producing impressive results [57]. However, these successes have come into fruition with the knowledge of how to train AI as it can be quite complex. Another important factor in the resurgence of AI has been the availability of large amounts data over the internet and increased computational power through GPU parallelisation.

1.4 Related Literature

Previous studies have focused on generating music rather than style [47, 19, 10, 52]. The CONCERT model was designed to compose simple melodies [47]. However, the limitations of this generative model were that it could not capture the global structure of music. The generated music was said to lack "global coherence". This is problematic as music has long-range dependencies. Based on the CONCERT model, Eck and Schmidhuber [10] tackled this problem by building a model that could learn long-range dependencies. A recent project by Google called Magenta showcased an AI-duet game which allows the player to compose a melody with the trained model [34]. These projects showcase the exceptional power of machine learning in the temporal domain of music.

However, these musical models are trying to learn the syntax of music. Musical sequences contain many dependencies. To predict how a song changes at a given time, it depends on what came before it. The beginning of the song may influence the end of the song. This means one could say there are long-range dependencies alongside short-range dependencies which adds further complexity to the problem. It is also extremely complex to determine what makes a sequence of notes into "music". One can only determine whether a sequence of notes is music through human confirmation.

This leads to say that much research is needed in understanding the complex structure of music. However, the stylistic properties of the music are not focused on by these generative models. These generative models are essentially producing digital sheet music. One could analyse this and say that the digital sheet music is open to interpretation. Similar to visual style, this opens a door for applying style to music.

In the domain of musical style transfer, a project attempted at transferring style between two songs in waveform format [4]. The resulting waveform had a considerable amount of noise, but if one disregards the noise, there was a meaningful stylised signal. It also took an exceptional time to train the model on 10 seconds of audio.

Audio poses several problems in the domain of music. For the task of extracting note pitch, a Fourier transform is first utilised to detect the base frequency, which is then mapped to a specific pitch [39]. Note times would have to be detected with a beat detection algorithm. This process is quite convoluted and time-consuming. There exists a format type called Musical Instrument Digital Interface (MIDI) which simplifies this process. It is the most commonly used format when composing music on a computer. It encodes basic musical properties such as pitch, timing and velocity.

A project called 'Bachbot' focused on generating and harmonising Bach-styled chorales using MIDI. MIDI files were converted into a matrix representation before their use [28]. The generative model successfully passed a musical Turing test. The generated chorales were indistinguishable from Bach's real chorales to 71.6% of "educated listeners".

However 'Bachbot' focuses on generating melodic components. Audible style is defined as the chorales of the song. This definition of style means that the model is still required to learn the structure of polyphonic music and then, generate possible chorales. Another feature that could be defined as style is the change of dynamics over time. How loud does a performer play a sequence of notes given its sheet music? With such a definition, the model would not need to learn to generate "pleasant" music but only the correlation of notes and their amplitude depending on their context. This can be extended to hypothesise different dynamics under different genres.

There exists a lot of music data in the form of MIDI. MIDI files are usually labelled with the composer or genre. These files capture the style of the performers as they are live recordings. This mostly applies to Jazz and Classical due to the fact that MIDI controllers are mostly pianos. These two genres tend to feature piano performances. Additionally, dynamics are captured in MIDI as a parameter called note velocity. This provides us the opportunity to use the state-of-the-art in machine learning to predict sheet music dynamics through note velocities. Can a machine learn style through dynamics and perform on any given sheet music like a human? If so, is it also possible for the model to perform sheet music in different styles? This is analogous to Mozart and Jimmy Hendrix playing sheet music that they have never seen before.

1.5 Project Aims

The high-level objective of this project is to investigate whether a model can learn musical style through dynamics and play sheet music like a human? If so, is it possible for the model to play performances in different styles such as Classical and Jazz? More specifically, the primary goals of this project are:

1. Research literature on music generation and identify the state of the art.
2. Design and implement a model for applying style to digital sheet music.
3. Design a data representation for training a model.
4. Create a valid dataset for extracting note velocity.
5. Perform an experiment on the model's musical performances through human evaluation with a musical Turing test[2].
6. Perform an experiment to evaluate whether the model can perform in different styles.

1.6 Overview of Thesis Structure

The musical theory required to understand this thesis is presented in Chapter 2. Chapter 3 covers the necessary technical background for this project. Chapter 4 discusses the steps taken to design a model that can learn style. Chapter 5 describes the pipeline of execution throughout the project to implement the model with its required dataset. This chapter also covers the experiments performed to efficiently train the model. Chapter 6 discusses the experiments conducted to assess the performance of the model. Chapter 7 lists the results and achievements as a conclusion.

Chapter 2

Musical Background

Music has its unique syntax. It has underlying rules which give it its structure. As this thesis is working primarily with music, one must understand its properties to make well-informed decisions on the tools that will be used. This chapter will provide a high-level overview of musical concepts required to understand this thesis. First, the basics of music theory will be covered. Then high-level details of the MIDI file format will be presented.

2.1 Music Theory

2.1.1 Pitch

Music can be represented by a sequence of notes. However, a note has many properties associated with it in its contextual song. Firstly, a note has a pitch [14]. Pitch is the aural property of a note that can be described using a frequency. Notes are labelled using 8 characters ranging from "A" to "G" to indicate their pitch class. Pitch classes are spaced by a defined frequency interval. The definition of an interval is the frequency between two pitches. The interval between one pitch and double or half its frequency can be defined as an octave. Notes separated by this interval belong to the same pitch class.

2.1.2 Beat

A beat is the basic unit of time for a song. A steady pulse of beats provides rhythm to a song. Tempo is the speed at which beats are played relative to time, and is measured in Beats Per Minute(BPM).

2.2 Sheetmusic

2.2.1 Notation

Notes are represented by their vertical height in sheet music with black dots with tails. The grid-like structure that holds the notes is called a staff as shown in 2.1. There are horizontal

lines that hold the notes. The note's vertical position corresponds to its pitch. The white spaces between the lines indicate the white keys on a piano. There are also vertical lines. These lines divide the staff into measures or bars. A measure will contain time signature to indicate its musical structure. A time signature describes the rhythmic properties of a bar [14]. The numerator describes the number of beats in a bar, and the denominator indicates what a beat is. A time signature of 4/4 or common time defines a bar to be of four 1/4 note beats.

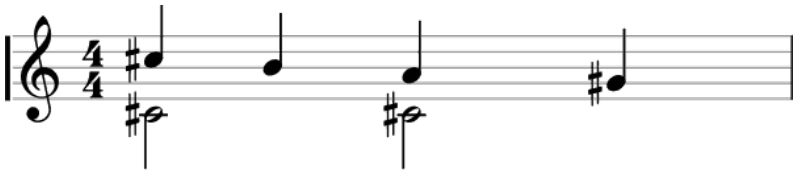


Figure 2.1: A staff with notes.

2.2.2 Dynamics

Dynamics describes how the amplitude of a song varies over time. This is notated in classical musical notation by Italian phrases which indicate relative loudness. Piano or "p" means "soft" and forte or "f" means "loud". These letters can be concatenated together to indicate further detail about the amplitude. The scale of dynamics can be seen in Figure 2.2.

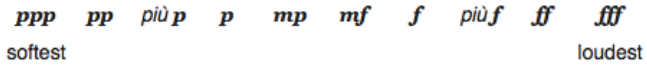


Figure 2.2: Range of Dynamics.

One may also see abbreviations such as "sf". This stands for "Sforzando" which indicates to the performer that they should suddenly emphasise a note. Musical notation can get increasingly complex with abbreviations that describe the gradual change in dynamics. "Cresendo" indicates that the score should gradually become louder whereas "Decrescendo" indicates the opposite. There are increasingly complex terms that describe the dynamics with additional details. An example of this is "mezzo-forte piano" which means "quite loud and then immediately soft". This highlights the verbosity of musical notation.

2.3 MIDI

2.3.1 Overview

How does one represent sheet music digitally? The most commonly used formats represent music in its waveform format such as WAV. It is difficult to extract important musical features from the waveform as mentioned earlier. This motivates the use of the MIDI format. It is a format used across electronic devices to represent music. It is the primary format used by musicians to create music through a digital audio workstation. To understand the MIDI format, the events types and their details will be covered which is then followed by an overview of different MIDI formats.

2.3.2 Events

There are three main types of events that can occur: MIDI, System Exclusive (SysEx) and Meta events. For the purpose of this thesis, only MIDI events will be discussed. MIDI events contain musical properties of notes encoded as numbers. A subset of these events encode musical properties such as the "note on" and "note off" events. These events carry three main musical properties: pitch number, velocity and time.

Note events are responsible for sending a command to play or stop a specific note. These events carry the pitch of the note which is represented on a scale from 0 to 128. C3 is represented by the pitch number 48 and C4 in the next octave is represented by 60. Additionally, the dynamics are also encoded using a parameter called "velocity" which is analogous to volume. An interesting aspect of MIDI is that the timing of notes is encoded using delta times. This means that every event stores a starting time which is the differential time between the previous note event and the current absolute time. If a note event is of type "off", it signals the end of the note being played. On the other hand, when it is of type "on", it signals a command to play the specified note.

2.3.3 Formats

There are three main MIDI formats in use today. These are format 0, 1 and 2. Each format specifies how the event sequences are handled within the file. Before going through the understanding of the formats, one must learn what a MIDI track is. A track is a sequence of time-ordered events. In format 0, all note event reside within one track. This would mean all instrument tracks reside in one stream of events. In format 1, there are multiple tracks where each instrument has its own track. When playing a format 1 MIDI, all tracks are synchronously played with the same tempo and time signature. This is where format 2 differs. It allows the simultaneous playback of several contained asynchronous tracks. These structures allows us to easily extract musical information from the file. Another advantage of using MIDI formats is that there are a large number of files available for download. This allows us to explore data-driven approaches. The next chapter will explain the technical details of such an approach.

Chapter 3

Technical Background

Music can be thought of as a language that is understood across human culture [43]. However, when it comes down to formalising the structure of music, there is no exact answer. For this reason, it is extremely difficult to explicitly program a machine to learn how to play sheet music due to the nature of the problem. When choosing a suitable approach, one must take into consideration the amount of data available for the task at hand. Fortunately, data is not a problem here as there are many MIDI files available for download. A certain type of machine learning model called the Artificial Neural Network has gained popularity in recent years. The artificial neural network was introduced in the year 1943 and is modelled after the biological neuron [35]. They have sparked in popularity in recent years due to large amounts of data available and GPU parallelisation. These models are producing extremely positive results in a variety of research areas [13, 22]. This motivates the use of artificial neural networks for learning style. First, an introduction of machine learning is provided. This is followed by the theory behind artificial neural networks.

3.1 Introduction to Machine Learning

Machine learning algorithms allow machines to learn from and analyse data without requiring the explicit programming of a human. These algorithms are used vastly in making important predictions about the future such as weather forecasting. Firstly, an introduction to supervised learning is given. Then a specific supervised learning approach called Linear Regression will be explained. Lastly, variants of an optimisation technique called gradient descent will be explored.

3.2 Supervised Learning

A teacher uses their knowledge to teach and correct a student's mistakes. This is analogous to teaching a machine learning algorithm using a labelled dataset. When the algorithm makes a prediction on an example, its accuracy can be calculated as the corresponding answer to the example is known. Formally, supervised learning is a type of learning where an algorithm tries to learn the mapping between the input, x and the output, y. There are two main types of supervised learning algorithms: regression and classification.

3.2.1 Classification

Classification is a variant of supervised learning that models the mapping of an input, x to a discrete dependent variable y which represent a category of some type.

3.2.2 Linear Regression

Linear Regression is a type of supervised learning approach that models the relationship of x and y where x is an independent variable and y is a continuous dependent variable. x may be continuous or binary. The general linear equation is used estimate model parameters θ from the data:

$$y_i = \theta_0 + \sum \theta_i x_i \tag{3.1}$$

3.2.3 Least-Squares Regression

The most common parameter estimation method for fitting a regression line is the Least Squares Estimation. This method tries to minimise the sum of squared residuals. A residual is defined as the difference between the actual value of the dependent variable, and the value predicted by the model. Mean Squared Error is an estimator used in regression tasks to evaluate the accuracy of a model. It is the mean of the sum of squared residuals over a training set.

$$RSS = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{3.2}$$

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{3.3}$$

3.3 Gradient Descent

The optimisation technique called gradient descent is a widely used in the field of machine learning [45]. It allows us to minimise a differentiable loss function such as MSE to improve a model. However, these optimisation techniques are often used without their understanding. This section will explain different variants of gradient descent and their implications on learning.

3.3.1 Batch Gradient Descent

When trying to minimise a function, $J(\theta)$, the parameters of a model θ must be adjusted. As our function is differentiable, this can be done by calculating the gradient of $J(\theta)$ with respect to θ. The weights are then adjusted using a learning rate η. For batch gradient descent, the gradients for the whole training are calculated and θ is updated once. It is guaranteed to converge to a global minimum if our loss is a convex function, and a local minimum if non-convex. The update equation is as follows:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta) \tag{3.4}$$

The practical implications of this process are that updates happen after a long period of time and thus training cannot be done online. For larger training sets, training may be extremely difficult due to limitations on memory [45]. Several redundant calculations are performed before every update. The next subsection explains another variant of gradient descent which reduces the implication of the problems faced by batch gradient descent.

3.3.2 Stochastic Gradient Descent

Stochastic gradient descent is another variant of gradient descent which focuses on training examples rather than the whole training set. The gradients are calculated for every training example $(x^{(i)}; y^{(i)})$ where $i = 0, ..., n$ and n is the size of the training set. After the gradients have been calculated for an example, an update is performed. The update equation is as follows:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)}) \tag{3.5}$$

This is beneficial if one requires their training to be online [45]. The implications on memory are quite advantageous if the training set is large. However, due to the high variance in updates, the trajectory of descent can be quite noisy. This can be beneficial as the noisy trajectory will overshoot the local minimum and continue descent into a valley which may lead to a better local minimum. If one slowly decreases the learning rate of stochastic gradient descent, then it is guaranteed to converge to the local minimum like batch gradient descent [45].

3.3.3 Mini-batch Gradient Descent

Mini-batch gradient descent attempts at achieving a balance between batch and stochastic gradient descent by reducing the variance in the updates, and reducing memory usage of larger datasets. As the name states, mini-batches of size n are used for training. The update equation is as follows:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)}) \tag{3.6}$$

3.3.4 Hyperparameter Optimisation

There exist a number of gradient descent variations that introduce different hyperparameters in an attempt to further improve convergence [5]. Gradient descent algorithms may include an additional hyperparameter called momentum [45]. Momentum tries to reduce the noisy trajectory taken during descent. Per-parameter adaptive learning rate methods are usually incorporated as well. The mathematical details of different gradient descent variations fall outside the scope of this thesis.

3.4 Introduction to Artificial Neural Networks

Now with an understanding of machine learning, the underlying details of artificial neural networks can be explained. First, the artificial neuron will be introduced followed by a brief discussion about their activation functions.

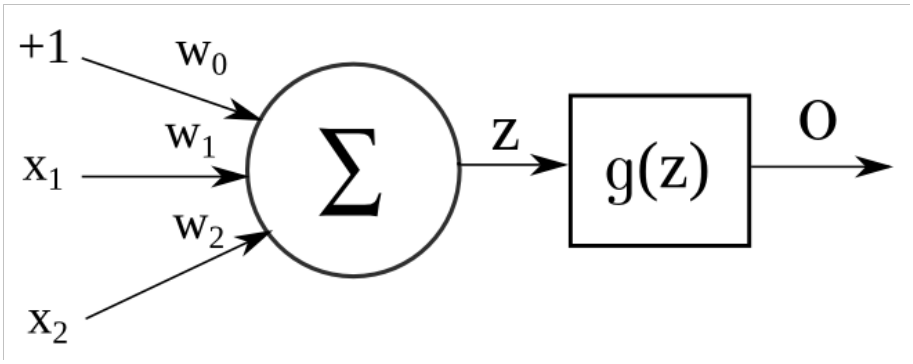


Figure 3.1: The Neuron.

3.4.1 The Neuron

A neuron is a basic unit of computation [36]. They are nodes which are connected by directed edges in a graph or network. These directed edges are called weights and biologically represent synapses. Each node contains an activation function, g, which produces the neuron's output.

The inputs, x_i, where $i = 1, 2, ..., n$, are multiplied with their corresponding weight w_i. There is also a bias unit, b. This value of this unit is always 1 and can also be written as w_0. The bias unit exists to add flexibility to the neuron's output by shifting the activation function g to the left or right. The net input, z, can be calculated as follows:

$$z = \sum_{i=1}^{n} w_i x_i + b \tag{3.7}$$

The output of a neuron, o, is calculated by applying the activation function g on the net input z :

$$o = g(z) \tag{3.8}$$

The following subsection will discuss the most widely used activation functions and their advantages and disadvantages.

3.4.2 Activation Functions

There is a variety of choice when choosing an activation function for a neuron. Non-linear activation functions give neurons their ability to learn non-linear patterns. The most commonly used activation functions are the *sigmoid*, *tanh* and the more recent, Rectified Linear unit or *ReLu*.

Sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-z}} \tag{3.9}$$

The *sigmoid* function takes any real-valued number and places it in the range of 0 and 1. This means extremely small numbers are turned into 0 and large numbers are turned into 1. This property can lead to a undesirable property called "saturation" as extremely small or large numbers will always be represented by 0 or 1.

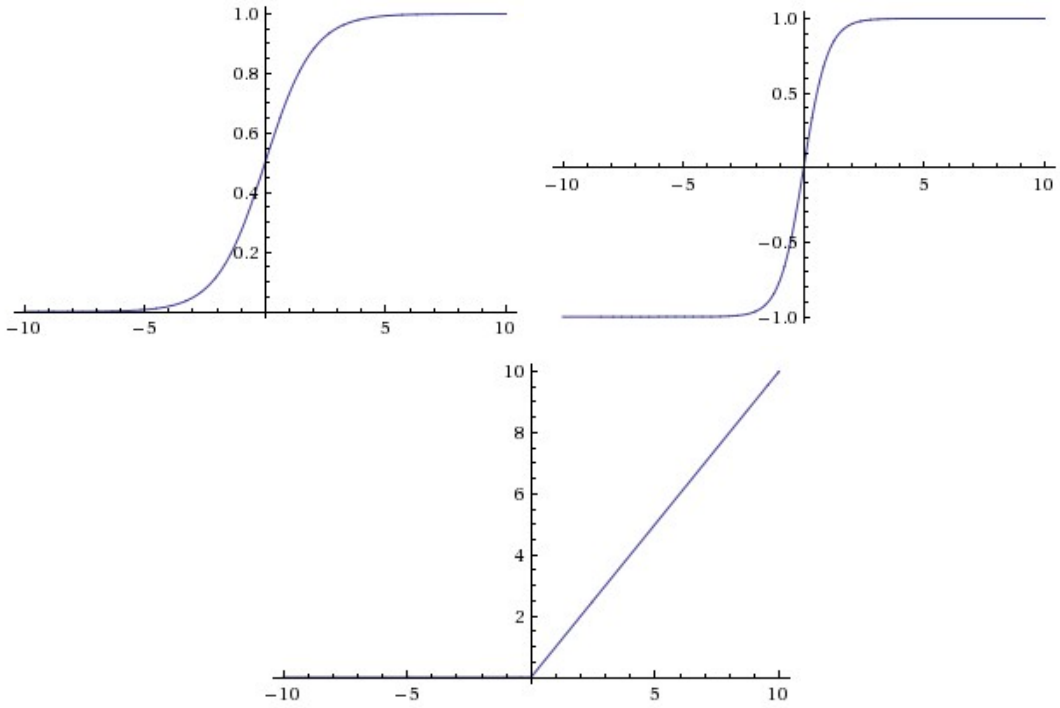


Figure 3.2: Sigmoid, tanh, and ReLu.

Tanh

$$tanh(x) = 2\sigma(2x) - 1 \tag{3.10}$$

The *tanh* function takes any real valued number and places it into the range of -1 and 1. Similar to the sigmoid, it is also verdict to saturation. However, their output is zero-centered.

ReLu

$$f(x) = max(0, x) \tag{3.11}$$

The ReLu takes any input and thresholds it at 0. In comparison to the sigmoid and tanh, it is relatively computationally inexpensive. They have recently gained popularity due to their fast performance in computer vision [24].

3.4.3 Overview of ANNs

An ANN is typically defined by three types of parameters:

- The topology between the different layers of neurons.
- The weights between the neurons.
- The activation function.

The next section will explain the feedforward neural network architecture.

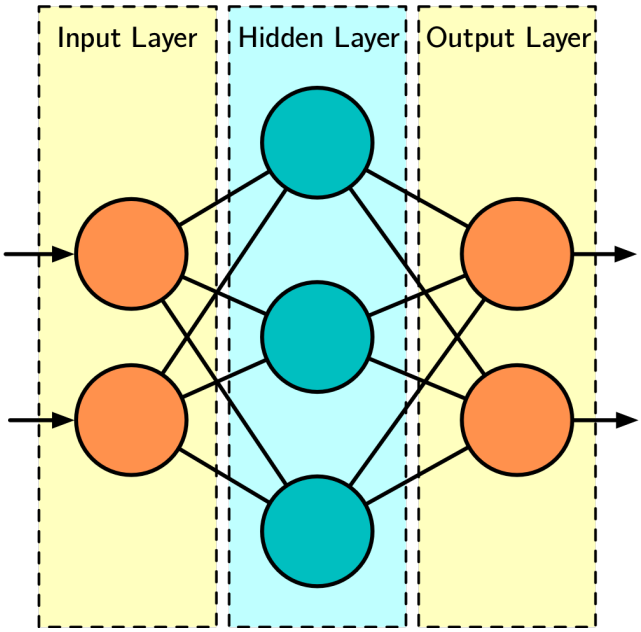


Figure 3.3: A Feedforward Neural Network.

3.5 Feedforward Neural Networks

The Feedforward Neural Network (FNN) is the simplest and most widely used ANN architecture [21]. In this architecture, neurons are arranged in layers. The first layer receives the input and is called the input layer. The last layer produces the output, and is called the output layer. There may be any number of layers between the input and output layer. These layers are called hidden layers. Every neuron in one layer is connected to every neuron in the next layer. However, there are no connections between neurons in the same layer. This can be seen in Figure 3.3. To train these networks, there are two mains steps: forward propagation and backpropagation. First, the details of forward propagation will be explained followed by the derivation of the backpropagation algorithm.

3.5.1 Forward Propagation

Forward propagation describes how data enters through the input layer, travels through the hidden layers and finally, exits through an output produced by the output layer. Firstly the output of the input layer l_0 is initialised with the input x.

$$o_0 = x \tag{3.12}$$

Equations 3.7 and 3.8 can be extended to every hidden layer. The net inputs and outputs are calculated in the order l_1 to l_K where K denotes the output layer.

For k from 1 to K:

$$z_k = g_k(W_k o_{k-1} + b_k) \tag{3.13a}$$

$$o_k = g(z_k(x)) \tag{3.13b}$$

This completes the forward pass.

3.5.2 Error and Loss

In linear regression, the loss function can be defined as the mean squared error (MSE) for X which is a set of input and output pairs, $X = \{(x_1, y_1)...(x_N, y_N)\}$, N is the number of training examples and h is the output of the network and is parameterised by $\theta = \{W, b\}$

$$E(X) = \frac{1}{2N}\sum_{i=1}^{N}(h_\theta(x_i) - y_i)^2 \tag{3.14}$$

During learning, θ is updated to minimise the specified error or loss function. This can be done minimising loss with respect to θ. Gradient descent can be used to iteratively adjusts θ relative to the loss with a learning rate of α using the following equations:

$$\Delta w_{ij}^k = -\alpha\frac{\delta E}{\delta w_{ij}^k} \tag{3.15a}$$

$$\Delta b_i^k = -\alpha\frac{\delta E}{\delta b_i^k} \tag{3.15b}$$

The partial derivatives on the right-hand side can be calculated using backpropagation. To do so, two assumptions must hold of a valid loss function. The first requirement is that the loss function must be generalisable to all the training inputs [27]. This means it can be written as an average $E(X) = \frac{1}{n}\sum_x E_x$ over error functions E_x, for individual training examples, x. This property is critical when learning with batched inputs. Secondly, backpropagation requires our loss function to be differentiable: it needs to be written as a function of the network's outputs. This allows us to perform backpropagation which will be explained in the following subsection.

3.5.3 Backpropagation

Before working the understanding of backpropagation, the loss function will be defined as the quadratic loss function where x^K is the output vector and y is the ground truth vector:

$$E = \frac{1}{2}\|y - x^K\|^2 \tag{3.16}$$

For simplicity, the inputs weights for all nodes at kth layer will be denoted by a vector W_k. Now the derivation of backpropagation will be shown. For this example, a three layered FFN ($K = 2$) will be used. This means the FFN has one hidden layer. To adjust our weights, the error relative to the weights needs to be calculated. Since the error is known at layer K, the chain rule can be used to calculate the error with respect to W_K where $K = 2$. The $\frac{1}{2}$ in the error function simplifies the expression when deriving the error with respect to W_2 using the chain rule:

$$\begin{aligned}\frac{\partial E}{\partial W_2} &= \frac{\partial E}{\partial x_2}\frac{\partial x_2}{\partial W_2} \\ &= (x_2 - t)\frac{\partial x_2}{\partial W_2}\end{aligned} \tag{3.17}$$

Recalling that $g_2(z_2) = x_2$, and $z_2 = W_2 x_1$, the partial derivative δx_2 can be expanded further:

$$\begin{aligned}\frac{\partial E}{\partial W_2} &= (x_2 - t)\frac{\partial x_2}{\partial W_2} \\ &= [(x_2 - t) \circ g_2'(W_2 x_1)]\frac{\partial W_2 x_1}{\partial W_2} \\ &= [(x_2 - t) \circ g_2'(W_2 x_1)]x_1^T \end{aligned} \tag{3.18}$$

The error layer k can be defined with respect to its input as:

$$\delta_k \equiv \frac{\partial E}{\partial z_k}. \tag{3.19}$$

Observing Equation 3.18, the error at layer 2 is the left-hand side of the equation:

$$\begin{aligned}&\text{Let } \delta_2 = (x_2 - t) \circ g_2'(W_2 x_1) \\ &\frac{\partial E}{\partial W_2} = \delta_2 x_1^T \end{aligned} \tag{3.20}$$

Recalling that the input activation for the bias unit is always 1, the error with respect to the bias b can be calculated as just δ_2:

$$\frac{\partial E}{\partial b_2} = \delta_2 \tag{3.21}$$

Now that the δ_2 has been evaluated, the chain rule can used then be used to calculate δ_1:

$$\begin{aligned}\frac{\partial E}{\partial W_1} &= (x_2 - t)\frac{\partial x_2}{\partial W_1} \\ &= [(x_2 - t) \circ g_2'(W_2 x_1)]\frac{\partial (W_2 x_1)}{\partial W_1} \\ &= \delta_2 \frac{\partial (W_2 x_1)}{\partial W_1} \\ &= W_2^T \delta_2 \frac{\partial x_1}{\partial W_1} \\ &= [W_2^T \delta_2 \circ g_1'(W_1 x_0)]\frac{\partial W_1 x_0}{\partial W_1} \\ &= \delta_1 x_0^T \end{aligned} \tag{3.22}$$

As before, the error with respect to the bias units b can be calculated as δ_1:

$$\frac{\partial E}{\partial b_2} = \delta_2 \tag{3.23}$$

This completes a backward pass for this network. Observing Equation 3.20 and 3.22, the back-progagation algorithm can be generalised to two cases for δ : the final layer and any hidden layer.

For $k = K$:

$$\delta_K = \frac{\partial E}{\partial o^K} \circ g_K'(z_K). \tag{3.24}$$

For k from $K-1, ..., 1$:

$$\delta_k = ((W_{k+1})^T \delta_k k + 1) \circ g'_k(z_k), \tag{3.25}$$

Once the δ has been calculated using the applicable equation, the error with respect to the weights and bias can be calculated as follows:

$$\frac{\partial E}{\partial b_k} = \delta_k \tag{3.26}$$

$$\frac{\partial E}{\partial W_k} = \delta_k o_{k-1} \tag{3.27}$$

3.5.4 The Backpropagation Algorithm

These equations can all be summarised in the backpropagation algorithm as follows:

1. Set the corresponding activations for the input layer with $o_0 = x$.
2. Forward propogate for each $k = 1, \ldots, K$ compute $o_k = W_k o_{k-1} + b_k$.
3. Calculate the error $\delta_K = \nabla_o E \circ g'_k(z_K)$.
4. Backpropagate the error for each $k = K-1, ..., 1 : \delta_k = ((W_{k+1})^T \delta_{k+1}) \circ g'_k(z_k)$.
5. Output the gradients for W and b with $\frac{\partial E}{\partial W_k} = \delta_k o_{k-1}$ and $\frac{\partial E}{\partial b_k} = \delta_k$.
6. Perform gradient descent for each $k = K-1, \ldots, 1$ update the W_k and b_k using rules 3.15a and 3.15b with a learning rate of α.

Now, the neccessary technical background has been covered. This leads us back to the aim of this thesis which is to investigate whether it is possible to teach a machine musical style. The next chapter will explore and discuss the choices made with their specifics to tackle the problem at hand.

Chapter 4

Methodology

4.1 Recurrent Neural Networks

How does one deal with sequences, or time? To determine which notes will be articulated at any given time depends on what was articulated in the past or even future. Musicians write music taking the global structure of a song into consideration. This leads to say that music has a complex macro-harmonic structure with many long-term dependencies.

A simple FNN's ability is limited by their assumption that the inputs and outputs are independent of each other. They also lack memory; they do not remember what the previous or future inputs are. This limits the ability of the network to learn about the context of its computation. This motivates the use of Recurrent Neural Networks (RNN) which will allow us to capture musical structure. RNNs have been applied successfully to a range of problems [15, 17, 28].

First, the architecture of the RNN will be discussed and then a special variant of backpropagation called "Backpropagation Through Time" will be introduced. Next, a specific RNN architecture called the Long Short Term Memory Network will be explained. Finally, the design of the musical model is introduced and discussed.

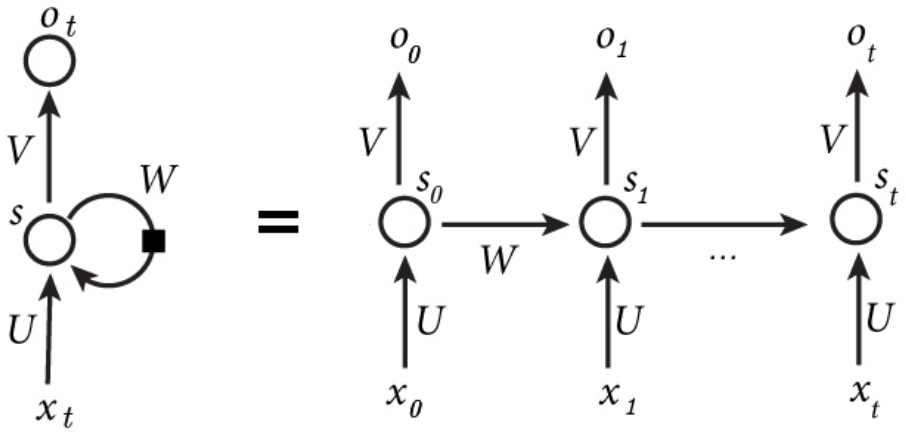


Figure 4.1: An unfolded RNN.

4.1.1 Architecture

Unlike FNNs, RNNs have a feedback loop and feed their previous state s_t into their computation to calculate their output o_t[6]. This state allows the networks to remember what it processed in the previous timesteps. The RNN has three main parameters U, V, W, where U and V are weights corresponding to the input x_t and output o_t. The architecture introduces a new weight called W. This is called the recurrent weight. This weight is responsible for determining how much of the previous state will be introduced into its computation. The recurrent weight is shared across all timesteps. This is advantageous as it greatly reduces the number of parameters the network has to learn. The details of forward propagation in RNNs is almost identical to FNNs. This will be discussed in the following subsection.

4.1.2 Forward Propagation

There are two main parts to RNN forward propagation: state update and output.

State Update

The first step of forward propagation is to update the current state of the network. In order to do this, the product of the input x_t and weight U is calculated which then added to the product of the previous state s_{t-1} with weight W. This computes the current state s_t of the RNN:

$$s_t = d(Ux_t + Ws_{t-1}) \tag{4.1}$$

Output

The product of the previously updated state, s_t, and weight V computes the output o_t:

$$o_t = Vs_t \tag{4.2}$$

This completes the forward pass.

4.1.3 Backpropagation Through Time

After completing the forward pass, the loss of the predicted output against the true output is calculated. This step is identical to the FNN. Now, this leads to performing backpropagation through time. This is done by unrolling the network by a specified number of timesteps. Figure 4.1 shows an RNN unfolded over time. This unrolled RNN is essentially a FNN, and thus backpropagation can be performed as described earlier. However, the difference is that the unrolled RNN shares the same weights across its layers, T. So for calculating the error relative to weight w_k, BPTT calculated the sum of the gradients obtained for w_k in equivalent layers.

$$\frac{\partial E}{\partial w_k} = \sum_{t=1}^{T} \delta_k^t o_{k-1}^t \tag{4.3}$$

4.1.4 Bi-directional RNNs

The vanilla RNN only reads inputs in order; it does not have access or any information regarding the upcoming time-steps. However, the human eye possesses the ability to look ahead of a bar in sheet music. This motivates the Bi-directional RNN [48]. This architecture is composed of two RNN layers. The first layer is called the forward layer. It processes the input sequence in its original or chronological order. The second layer is called the backward layer. This layer processes the sequence in reverse. There is no connection between the two layers. The individual outputs are concatenated to produce the final output.

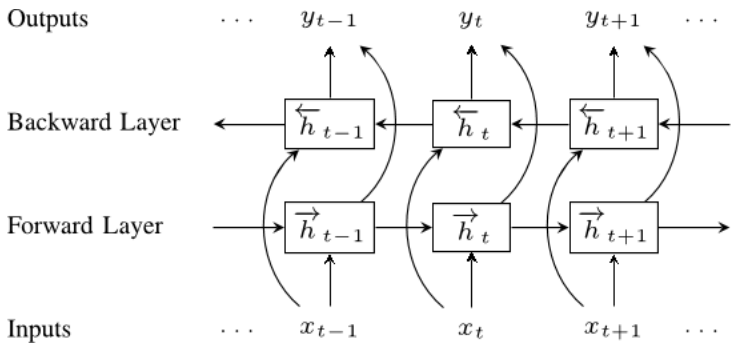


Figure 4.2: A Bi-directional RNN.

4.1.5 Vanishing Gradients

In the domain of music, short musical pieces exist, and there are also extremely long pieces. This leads to say music is dynamic in length. Generally speaking, backpropagation provides us with the ability to iteratively improve a neural model's accuracy by adjusting its weights. However, there are some issues that arise when using it on deep neural networks, especially when working with long sequences such as music [53, 40].

Deep neural networks suffer from the problem called "vanishing gradients", and it can severely diminish the ability for a model to learn. Backpropagation calculates the error with respect to the weights in the network. This is followed by the error propagation through the network using the chain rule. This is repeatedly done over a chain of multiplication to calculate the gradients for the whole network. This lead to problems when this error is propagated down several layers. The output of g' may be small, < 1, then the continuous multiplication this output will lead to smaller values [40]. This may lead to extremely small gradients further down the network. This phenomenon is called the "vanishing gradients" problem. The implication for many-layered networks is that earlier layers take longer to train than layers closer to the output layer.

This issue also impacts RNNs as backpropagation through time also suffers from this problem. When processing long sequences, RNNs are unrolled over a large number of time-steps for the duration of the musical piece. This is analogous to an extremely deep network. As mentioned earlier, the RNN shares a recurrent weight between its timesteps. When this weight is less than 1, then after a certain number of timesteps, the gradients diminish [41]. The implications of this are that they are not able to retain state for long sequences.

4.1.6 Exploding Gradients

If the weights are poorly initialised or become large during training then repeated multiplication may lead to increasingly larger gradients. This phenomenon is called the "explosive gradients" problem. In a RNN, a recurrent weight of > 1 will progressively send larger gradients down the network through time [40]. The implications of this can be disastrous during training. This can be understood through a geometrical understanding as follows.

When training, the surface of the error function may be a smooth convex function, or it may have several local minima. Extremely large gradients may arise during backpropagation during gradient descent. A "zig-zagged" or noisy trajectory is usually taken when descending the surface of the error function when performing stochastic gradient descent. There may exist directions denoted by vector d that may be followed by the gradients during descent which will eventually result in exploding gradients. If the trajectory taken by stochastic gradient descent follows this vector, then it will arrive at a high curvature wall in the error's surface. This means that when stochastic gradient descent calculates its next descent step at this point, it is forced to jump perpendicular to the walls and may land very far in the valley or possibly outside it [41]. This may slow down learning considerably and may even disrupt it.

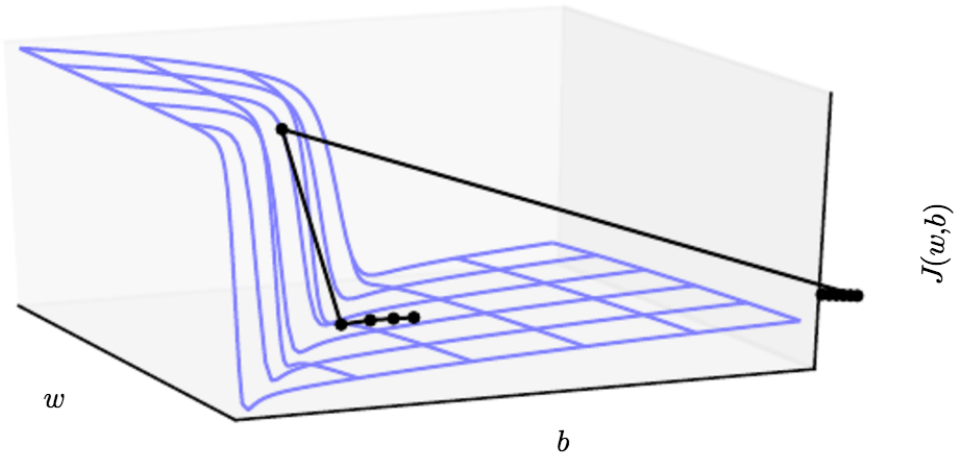


Figure 4.3: The solid line shows a trajectory that results in exploding gradients.

4.2 Long Short-Term Memory Networks

As mentioned earlier, the RNN suffers from the vanishing gradients problem which means they cannot retain state over a long period of time. This would be problematic when long-term dependencies or context needs to be captured in a musical piece. This motivates a special type of recurrent neural network called the Long Short-Term Memory Network (LSTM) which was specifically designed by Hochreiter and Urgen Schmidhuber [18] to avoid these issues.

The RNN has only a simple tanh function within its cell which decides how much weight of the current state of the cell should be considered for the next timestep [6]. Due to its simple design, it can only retain this state for short-range dependencies due to the vanishing gradients problem. Timesteps in the future will be less sensitive to earlier timesteps. It would be advantageous if the RNN could control when to lose or keep its state. This is where the LSTM excels. First, the architectural components of the LSTM block will be explained ,and then the intuition behind how they work will be covered.

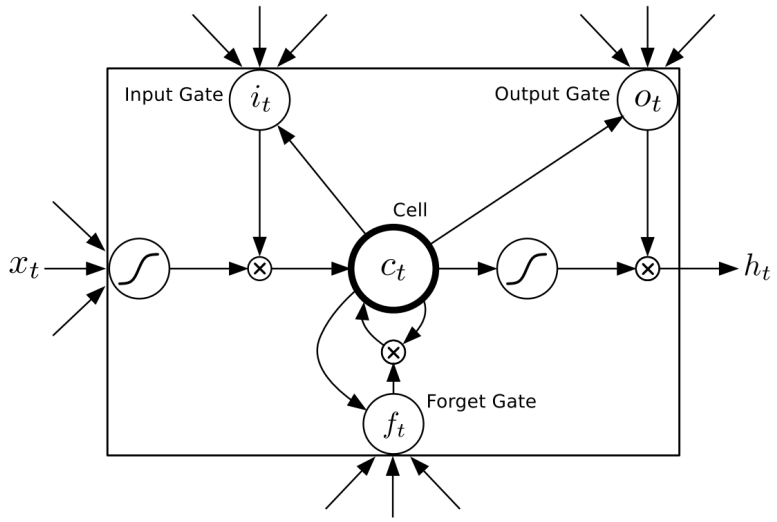


Figure 4.4: A LSTM cell.

4.2.1 Architecture

The architecture of the LSTM adds additional complexity to the vanilla RNN design by adding specialised gates[18]. There are five main architectural components of the LSTM block:

- Input gate i
- Output gate o
- Forget gate f
- Cell c
- Hidden state output h

Figure 4.4 shows the architecture of the LSTM cell. The five main components will be explained in the context of the LSTM's three main steps. These steps will be explained in chronological order.

Forget

Recalling that LSTM can choose to lose or keep its state, it requires memory. Memory or state at timestep t is held in the cell C_t. The forget gate decides which values in the state should be forgotten or kept at every timestep. Its inputs are the previous output, h_{t-1}, and the current input and x_t. With these two inputs, it uses the sigmoid function to output f_t. f_t contains values between a 0 or 1 highlighting which parts of the previous cell state c_{t-1} should be forgotten or kept when updating c_t.

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{4.4}$$

Now the c_t is ready to be updated. This is explained in the next step.

Cell Update

This has two main substeps. First, the input gate selects which sections of the cell state c_t will be updated. The inputs to this gate are x_t and $h_t - 1$. By utilising the sigmoid function, it outputs i_t which contains numbers between 0 and 1 to highlight which sections require updating in the new cell state.

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{4.5}$$

Secondly, a tanh layer computes possible values denotes by $\hat{c}$ for the new cell state. c_t is then computed as follows:

$$\hat{c_t} = \tanh(W_c[h_{t-1}, x_t] + b_c) \tag{4.6}$$

The cell state can be updated with the previously computed $\hat{c_t}$ and i_t as follows:

$$c_t = f_t \circ c_{t-1} + i_t \circ \hat{c_t} \tag{4.7}$$

This concludes the update step. Next, the output step will be explained.

Output

Finally, the block is ready to produce an output h_t for the current timestep. There are two main substeps to compute the output h_t. First, the LSTM selectively filters which values should be passed into the output. This is done by passing x_t and h_{t-1} into the sigmoid function to calculate o_t:

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{4.8}$$

Next, the cell state c_t needs to be transformed before the selective output. This is done by passing c_t into a tanh layer to squash it between $[-1, 1]$ so the Hadamard product with o_t results in the selective output vector.

$$h_t = o_t \circ tanh(c_t) \tag{4.9}$$

4.2.2 Constant Error Carousel

The gating mechanism mentioned above in the LSTM tackles the vanishing gradients problem [18]. This mechanism allows an LSTM cell to withhold cell state for long period of times. The weight of the recurrent connection in the LSTM cell is the activation of the forget gate. When the activation of the forget gate is one, this means that the LSTM has a bias of remembering its state and not forgetting. This state can be constantly "carousel"-ed with its recurrent connection over timesteps and then can be used as part of the output after several timesteps as required.

$$\frac{\partial c(t)}{\partial (t-k)} = 1 \tag{4.10}$$

This is beneficial during backpropagation. A recurrent weight of < 1 leads to vanishing gradients over time which is seen in RNNs. With a recurrent weight of one as seen in Equation 4.10 there will be no chains of multiplications, and this allows the LSTM to withhold the error and learn long-term dependencies.

4.3 GenreNet

With the motivation mentioned above, the intuition behind the initial design of the network can be explained. To learn style, one needs to first focus on a subset of the problem. In the domain of music, there exist several musical styles which are categorised by their parent genre. A genre is a label that encompasses music of a similar "flavour" or style. As stated in the introduction, the goal is to capture style through the variation of dynamics in music. Sheet music for a classical song would be played with a different range of dynamics than another genre. The following section explains the architectural decisions for creating a model to learn the style associated with a genre. This model will be referred to as GenreNet. The goal of GenreNet is to learn how to play sheet music by learning genre-specific style. The goal is to predict the dynamics of any given sheet music input.

4.3.1 Architecture

The model consists of two main layers as seen in Figure 4.5 :

- The Bi-Directional LSTM layers
- The linear layer

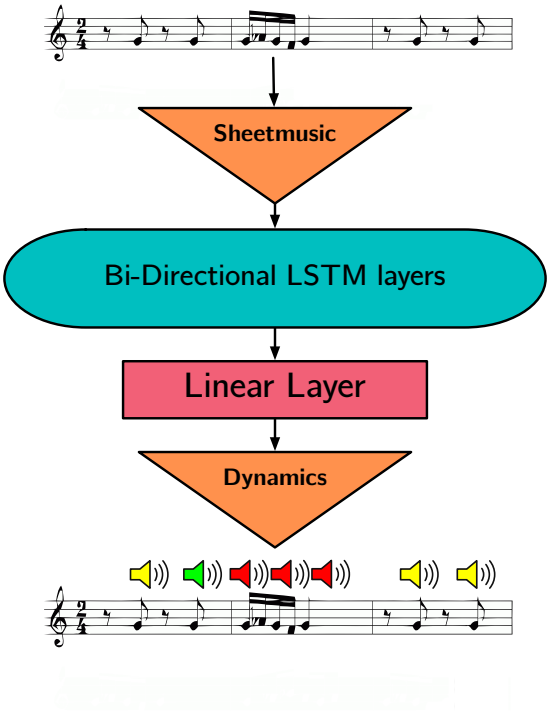


Figure 4.5: GenreNet

The Bi-Directional LSTM layers

The complex long-term dependencies in music need to be captured which motivates using LSTMs. The Bi-directional architectural choice is based on the real task of reading sheet music. Humans can use their sight to skim across sheet music and glance at upcoming notes in the score. They can use this visual "look ahead" to modify their performance. To translate this scenario onto an architecture, the analogous layout would be to use a bi-directional layer with LSTM to give us this foresight. The LSTMs provide memory for remembering dependencies, and the bi-directional architecture allows the model to take the future into consideration. To increase the expressive power of the model, these layers can be stacked which means one layer's output feeds into the other layer's input.

The linear layers

How does one best capture the scale of the output? The model is trying to learn the relationship between sheet music and the dynamics. Dynamics lie on a continuous scale of loudness. The output of LSTMs usually lies between $[-1, 1]$ due to their activation function. To scale these numbers to represent a larger range, a linear layer can be used. A linear layer performs a linear transformation on its input. This transformation is called the identity activation function where z is the weighted sum of its inputs.

$$g(z) = z = \mathbf{w}^T x \tag{4.11}$$

4.4 StyleNet

GenreNet's architecture breaks down the learning problem into two main components. The first is learning the dependencies and underlying pattern in sheet music. The second is to capture the scale of the dynamics. However as stated in the introduction, the goal of this research investigates whether it is possible for a machine to learn to play sheet music like a human. Learning to play sheet music in a specific genre's style by learning the dynamics is what GenreNet is designed to accomplish. However, humans can play music in a variety of styles. Sheet music can be interpreted, and an intended style can be injected into the song. This motivates the design of StyleNet, the rendition model. The purpose of StyleNet is to learn to predict different dynamics across genres when given sheet music.

In the field of computer vision, Bromley et al. [8] introduced a neural network architecture called the Siamese Neural network. This architecture consists of identical subnetworks which share parameters. The purpose of this architecture is to learn the similarity or relationship between two inputs. Bromley et al. [8] used this network to identify forged signatures. A signature and the same signature with a time delay is input to the Siamese network. The shared layer calculates the distance between the subnetwork outputs and compares it against a stored distance value. Signatures were marked as forged if the distance values differed significantly. Another usage is seen in natural language process where Yin et al. [58] used a Siamese network architecture to model a pair of sentences.

Taking inspiration from this architecture, one can create a similar architecture but for music. Siamese networks find and output the similarity between the inputs. However, in this case,

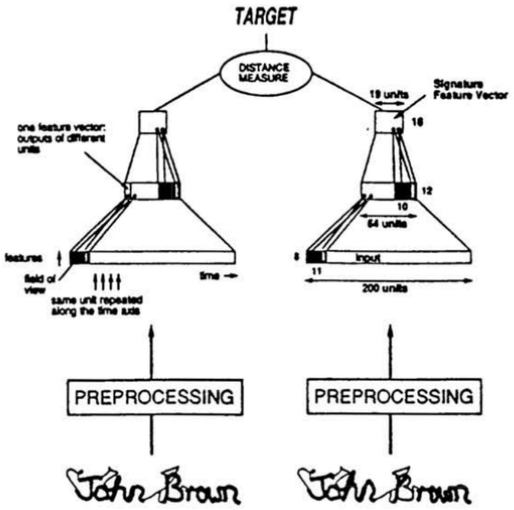


Figure 4.6: Bromley et al's Siamese Neural Network Architecture [8]

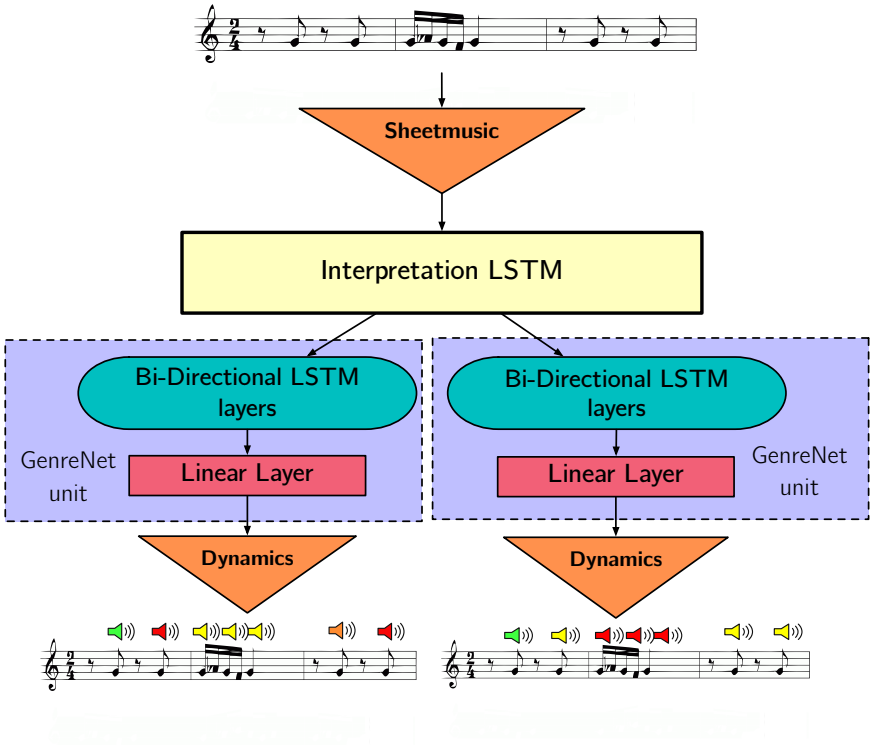


Figure 4.7: StyleNet with two GenreNet units

the similar feature is known. This feature is the sheet music. The task at hand is to convert our similar feature into specific representations of that feature, which is creating stylised sheet music. This can be thought of as reversing the Siamese network where the input is now the similar feature. However, StyleNet is designed for multitask learning. It is learning different styles. This is similar to how Luong et al. [33] trained a neural model to translate English into a range of languages. StyleNet should translate sheet music into different styles. It should produce n outputs where n is the number of styles. Now the architecture design for StyleNet will be discussed.

4.4.1 Architecture

The StyleNet architecture has two main components as seen in Figure 4.7:

- The Interpretation layer
- The GenreNet Units

4.4.2 Interpretation Layer

For the model to "interpret" the sheet music, there is an Interpretation layer. This is the shared layer across GenreNet units which are the subnetworks of this architecture. The Interpretation layer can process sheet music and convert it into a representation that describes its own representation of the sheet music. This is analogous to a human's interpretation of a score. This layer is shared across all of the GenreNet units. This results in reducing the number of parameters the network needs to learn. This ultimately leads to needing less data to train our model on which is always advantageous.

4.4.3 GenreNet Unit

These subnetworks are attached to the interpretation layer. Each GenreNet unit allows the model to learn a specific style. There is one stylised output for every GenreNet unit attached. Multiple GenreNet units can be plugged into the Interpretation layer which allows the model to learn multiple styles. The benefit of having a shared interpretation Layer is that by sharing parameters across several GenreNets This ultimately leads to needing less data to train our model on which is always advantageous.

Chapter 5

Implementation

One of the most difficult tasks in training neural networks is obtaining the necessary data and representing it in a way for the network to process. Neural networks need data to represented in a numerical form which also captures the required information. As mentioned before, large amounts of data now easily available on the internet and this is one of the driving factors that has resulted in many several successful endeavours with neural networks[26]. Now that the StyleNet architecture has been designed, the training data needs to be obtained. The goal is to create a dataset from which StyleNet can learn Classical and Jazz style. In this section, the steps taken to create the Piano dataset are explained. The Piano dataset only contains Piano MIDIs within the Classical and Jazz genre. All MIDIs are in 44 time and format 0. Both genres have 349 MIDIs which creates a total of 698. The dataset will be available as complementary material. This section discusses the data representation chosen for the model. A data representation is designed for the sheet music input and our dynamics output.

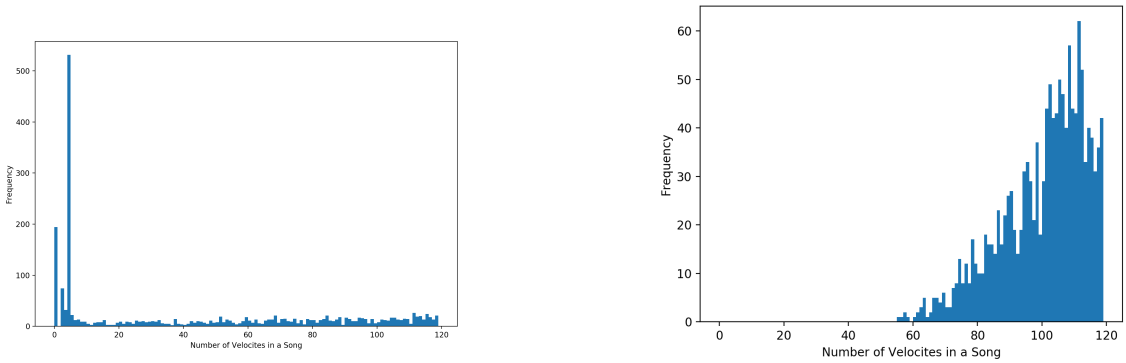
5.1 Data Preprocessing

5.1.1 MIDI

As mentioned in the introduction, retrieving note properties from a musical waveform is difficult. MP3 and WAV are the most commonly used audio formats for music, but they represent music in its waveform. This observation motivates the use of MIDI. There are numerous MIDI files available on the internet. Additionally, these files are mostly free to use. There also exist a variety of genres such as Jazz, Classical, Rock and so on. All of these music files contain a variety of instrument with their digital sheet music. Some MIDI files have instrument solos whereas some have several instrument tracks which can be seen in orchestral tracks. Firstly, several Jazz and Classical MIDI files were downloaded from a variety of websites. One source of classical MIDIs was a yearly piano competition which has uploaded MIDI recordings of its participants called the Yamaha Piano e-Competition[1].

5.1.2 Isolating Genre

Since we are working within the limitations of MIDI, most human-performed recordings are of the piano and drums. This is because MIDI controllers do not exist for the other instruments



(a) All downloaded MIDI

(b) Performance MIDIs

Figure 5.1: Histograms of velocity range across MIDI.

such as the guitar. To best capture style, it would be most useful to choose genres which contain MIDI controller recordings. The piano plays a dominant role in both Jazz and Classical, and thus I decide to focus on these two genres. Labelled MIDI are available for download for both genres. Next, the steps taken to isolate the piano tracks are discussed.

5.1.3 Isolating Piano

Across Jazz and Classical MIDIs, there are several instruments. Each instrument is represented by its own independent note track. Only piano MIDI tracks will be used. The motivation behind this is because piano tracks carry style. Also, learning the dynamics for a range of instruments would increase the complexity of the model. To isolate the piano tracks, one must preprocess the MIDI tracks. Most MIDI files are in format 1 which is a multi-instrument track. There also exist many MIDIs containing separated tracks for the left and the right hand of the pianist. The best-suited format for our problem is format 0 as it allows us to focus on one track. All downloaded MIDI were converted and merged into format 0 MIDI with a single piano track.

5.1.4 Capturing Velocity

Some MIDI files do not capture the velocity dynamics and sound monotonous. This means that the tracks were most likely created in software and not by recording an instrument. Many tracks only contain one global velocity. This can be seen in Figure 5.1a. This is noticeable in the large quantity of MIDIs between 0 and 10 on velocity histogram plot of the downloaded tracks. To eliminate problematic MIDI, it would be best to compare the range of velocities in human recordings. In order to do so, a histogram of the Yamaha competition MIDIs was plotted as seen in Figure 5.1b. Most performances have at least 45 velocities or more. However, most of these performances are quite lengthy whereas there are shorter tracks in the downloaded set. Thus a minimum threshold of at least 20 different velocities was chosen for the dataset.

5.1.5 Quantisation

Now the process of quantitation will be explained briefly. The sheet music matrix needs to represent the MIDI structure. MIDI only contains note messages with their delta times. Thus the delta times are converted into absolute time. Time is on a continuous scale. However, our matrix is discrete. The next step is to bin our note times by aligning them to the nearest sample step. This allows us to capture the notes and represent them in a matrix form. The implication of this is that the notes will lose their exact timing. However, if the note times are not binned, then the notes may not be captured in the matrix representation. To prevent this from happening, time signature was constrained to 4/4time across all the music files and choose a sampling interval of a 1/16th note. All the notes values are binned to the nearest 1/16th note. Now the sampling intervals align with note times which allows us to capture as much information as possible. Other time signatures could be used but 4/4 time is the most common across piano music. 1/16th resolution was chosen as our time signature is 4/4. A finer resolution would result in extremely large input files which would exponentially increase training times.

5.2 MIDI Encoding Scheme

This section will describe the steps taken to design input and output format to represent our MIDIs. Data needs to be in a numeric form for it to be accessible by a neural network. Note velocities, start times and end times is the only required information for the model. First, the input format, the digital sheet music, for the model is explained. Then the design for output format which contains the corresponding velocities is discussed.

5.2.1 Sheetmusic Input Representation

Isolating important features is the first step to designing an input format. The model needs to know what notes are being played at a given time-step. This is analogous to saying a note has a state at any time step. A note can have three states: note is on, note is off, or note is sustained from the previous time-step. One could only have "note on" and "note off" states like Eck and Schmidhuber [11]. However, this would increase the complexity of the learning task as the model would need to learn the difference between a sustained note and a new note being played. Thus this extra piece of information is encoded into the representation.

Using a binary vector, "note on" is encoded as $[1, 1]$, "note sustained" as $[0, 1]$ and "note off" as $[0, 0]$. The first bit represents whether the note was played in that timestep or not and the second represents if the note was held or not. One could also decide on representing the input format as a single number. [0],[0.5], or [1]. However, this representation was not chosen as the activations for the network would be a fraction of 1. This representation would try and encode a categorical state on a continuous scale. The model would have to precisely learn that these small fractional numbers represent categories. Thus a binary vector was chosen to allow strong activation inputs into the network which should make learning from the representation easier.

Next, the note pitch needs to be encoded. At one time-step, any possible note pitch could be played. Recalling that MIDI encodes pitch as a number ranging from 0 - 128, a matrix with the first dimension representing MIDI pitch number is created. The second dimension represents a quantised timestep or a 1/16 note. Due to the note state being a binary vector, the pitch dimension will be twice the size. A pitch dimension of $88 * 2 = 176$ was chosen where 88 is the

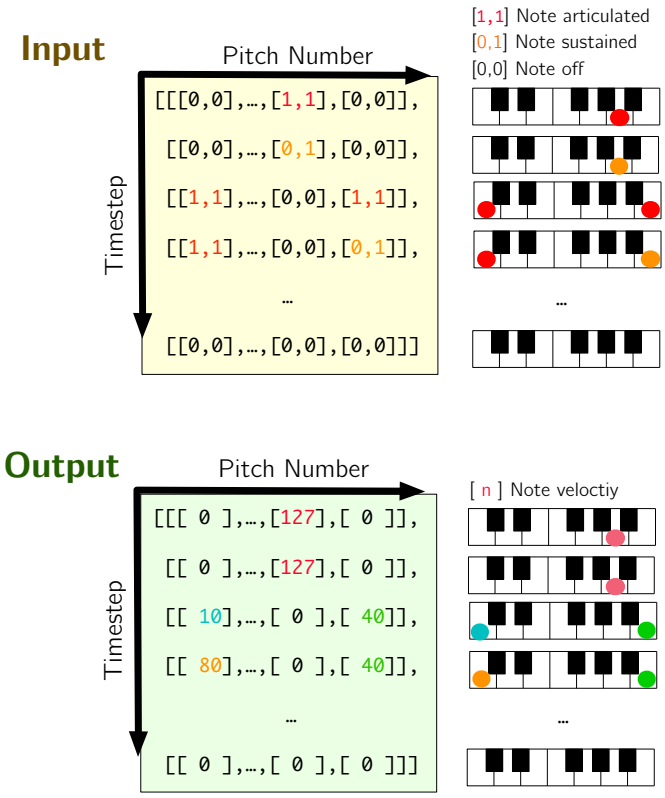


Figure 5.2: Input and Output Matrix

range of note pitches, and 2 is the note state vector. The reasoning behind this is that most pianos only have 88 keys. Thus a pitch axis of size 128 would have pitches that would never be played. For this reason, only pitches from note 21 to 109 are captured. This leads to a pitch axis of length 176. This is because most MIDI pianos are 88 notes wide and for each note, there is a binary vector of length 2. Finally, the number of timesteps for each track is then truncated to the nearest power of 2 to remove trailing zeros and keep track lengths fairly consistent.

5.2.2 Velocity Output Representation

Similar to our sheet music matrix above, the columns of our matrix represents pitch and the rows represent timestep. The difference, in this case, is that pitch dimension is only 88 notes wide. The note velocity is encoded into its corresponding [pitch, timestep] index. To make the learning process easier, it is best to reduce the scale of the data. This helps the network as it does not have to learn the scale itself. This can be done by dividing the velocities by the max velocity, 127. This ensures that all the velocities are between 0 and 1.

Additionally, the created velocity matrix can be decoded back into a quantised MIDI file. This is done by reversing the encoding process. Every note message in the MIDI is binned to an absolute time. This absolute time is used with the note pitch as an index into the velocity matrix. The new velocity is encoded into the MIDI. After this process is performed for every note message in the MIDI, the resulting stylised MIDI is saved.

5.2.3 Summary of data representation

A few observations can be made about the proposed encoding scheme:

- It encodes the note state as a binary vector.
- It represents MIDI using a dense matrix with unarticulated notes encoded.
- Sustained and articulated notes are defined as separate states, unlike, prior works [11].

5.3 Training Experiments

As explained in the introduction, training neural networks requires a strong understanding of their underlying theory [40]. There are several hyperparameters present which can produce suboptimal results if not chosen with correctly [23, 16]. For the training of StyleNet, the piano dataset is used. The goal of StyleNet is to learn Jazz and Classical Style. This section will explain the setup and series of experiments done to justify the final hyperparameters for StyleNet with their intuition.

5.3.1 Setup

Layers

The input interpretation layer is set to be 176 nodes wide and only one layer deep. There are two GenreNet units: one for Jazz and one for Classical. Each GenreNet is three layers deep.

Mini-batches

For training, the model was trained on alternating mini-batches of Classical and Jazz songs. Mini-batches are created based on their genre. A mini-batch size of 4 is chosen. Training is most successful when there is a large variation in the information carried by the data it learns from [27]. For this reason, the dataset is shuffled to encourage different combinations of songs in every mini-batch. This also reduces the chance of a batch of outliers from impacting training negatively.

Learning Rate

Choosing a suitable learning rate for the model can be complex. A small learning rate would result in the model taking an extremely long amount of time to train. On the other hand, a high learning rate can result in gradient descent performing big updates which can cause the algorithm to diverge which can lead to a disastrous training run. After a series of runs, a learning rate of 1e$-$3 was chosen.

Optimiser

The Adam optimiser was chosen. Adam performs stochastic optimisation with an adaptive learning rate and momentum [23].

Training and Validation

Most songs are perceived differently, and it is usually difficult to say two songs sound the same. This observation leads to say that each song example contains meaningful information for the model to train on. This motivates our training set and validation set to be 95% and 5% respectively. This equates to 95% Classical Songs and Jazz songs each for training, and 5% for validation.

Loss function

StyleNet outputs a velocity matrix for each genre through its GenreNet unit. This is a one-to-many setting. Given a sheet music input for a specific genre, it predicts a velocity matrix through a GenreNet unit for the corresponding genre. This is a regression learning problem. Thus a meaningful metric to measure the performance of the model would be the mean squared error between the true and predicted velocity matrix. The genre of the input mini-batch determines which GenreNet's prediction is used. However one must remember that since our matrices are mostly zeroes, the errors calculated will be on an extremely small scale.

5.3.2 Experiments

Truncated Backpropagation Through Time

Backpropagation through time is a very computationally expensive process [27, 15], The time to backpropagate an error for all of the timesteps in a sheet music matrix takes a remarkably long time. The average length of a sheet music matrix is 1500 timesteps. Backpropagation is truncated to 200 timesteps to reduce training time. This limits our model to learn dependencies within a 200 timestep window. However, this improved training time significantly. Convergence time was reduced from 36 hours to around 12 hours with truncation.

Dropout

A common problem faced during training especially with small amounts of data is overfitting. This means the model memorises the dataset and does not generalise well to new data. This motivates the use of dropout. Dropout is a regularisation technique that tries to combat this issue [59, 49]. When training, a neuron in the network may "turned off" with a probability of p. This means that the weights of the "off" neurons are not updated. This is because the neuron has no effect on the loss function and the gradient computed during backpropagation is 0. The result is that only a subset of the network is updated. Such updates allows the weights of neurons to become independent of other neurons. It can also be thought of as an ensemble of models. Bagging is a type of ensemble learning where each model us trained on a subsample

of the training data and thus learns only a subset of the feature space. Dropout can be thought of as an ensemble of different networks trained at each training step on a single sample.

As mentioned by Srivastava et al. [49], a dropout $p = 0.5$ should produce optimal results. However, in practice it is usually not the case. A dropout of p = $[0.5, 0.8]$ was experimented with using a learning rate of 1e-3. A dropout of p = 0.5 means only half the model is only updated for any mini-batch. If the dataset was extremely large, then a dropout of p = 0.5 could produce results without overfitting similar to the experiments in the paper. However, the dataset is small, with matrices that contain mostly zeroes. There are also certain notes in the sheet music matrix which are less commonly played. Thus losing half of a network at a given time could mean that the network would not see enough examples to learn less common patterns. This would result in a model that cannot learn the underlying patterns in the data.

In Figure 5.5b, for the p = 0.5 model the loss converges early at 3.5e−3 as compared to p = 0.8. This can also be visualised by plotting the true velocity matrix against the predictions. This is shown in Figure 5.3. It can be seen that the underfitted model has not learned to predict velocities correctly. It is only producing a small range of numbers which are at a much smaller scale than the true velocities. The $p = 0.8$ model outperforms the underfitted model and converges at $1.0e-3$ and produces predictions which are visibly similar to the true velocity matrix. This observation motivates choosing a dropout value of 0.8.

Gradient Explosion

LSTM networks are vulnerable to having their gradients explode during training. A model with no dropout (p = 1.0) was trained with a learning rate of 1e−3. This resulted in the model diverging. This can be seen in Figure 5.5a. This is most likely due to gradients exploding during the training run which caused network weights to blow up. A commonly used technique to combat explosive gradients is called "clipping gradients by norm" [42]. This method introduces an additional hyperparameter called g. When the norm of a calculated gradients is greater than g, then the gradient is scaled relative to g. This parameter is set to 10.

Encoding Scheme

As mentioned in subsection 5.2.1, the chosen encoding scheme is a binary vector representing the note's state. This encoding was chosen after experimenting with a discrete representation. When training with a discrete representation, the resulting velocity prediction matrix had extremely high velocities as seen in Figure 5.5c. It appears that the network's weights have become excessively large. This result could be due to three states being encoding into one input signal. The distance between each state is small and it makes it difficult for the neuron to separate the underlying states in the signal. Thus it can be speculated that when training on this discrete encoding, an extremely small learning rate would be needed for the model to understand the representation. Otherwise, the weights of the model could become excessively large and create unusually high outputs.

5.3.3 Batching Method

In the field of machine translation, Dong et al. [9] used alternating mini-batches of different language pairs when translating English into other languages. This inspires the batching method

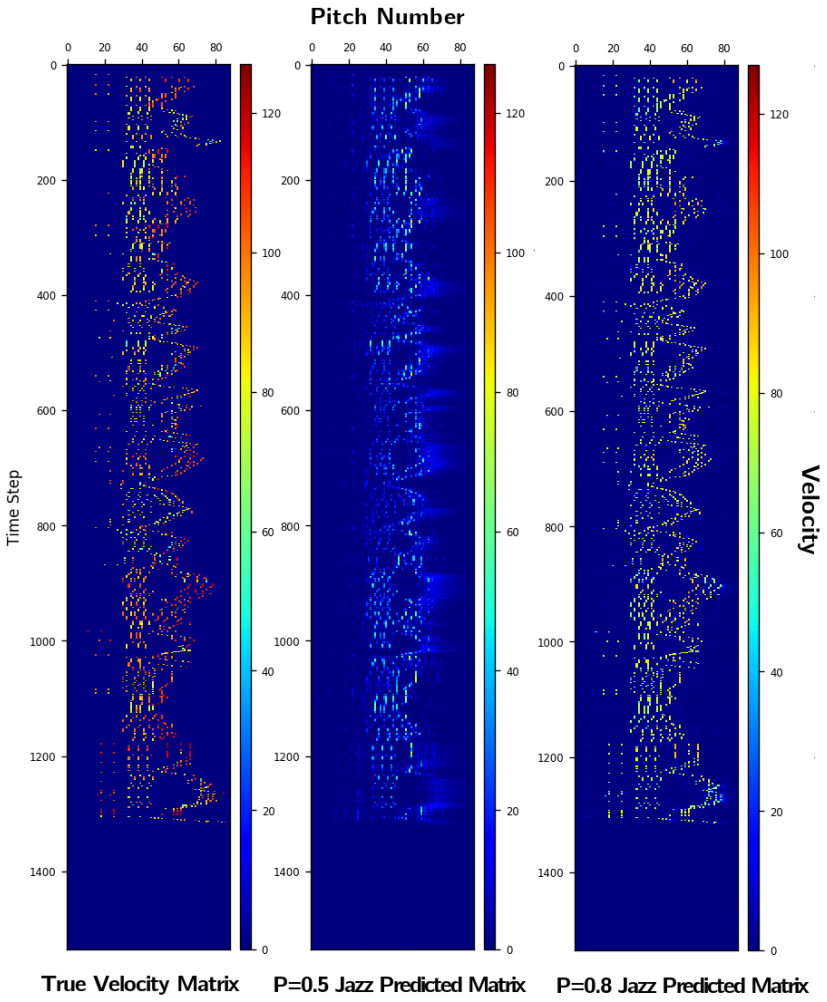


Figure 5.3: True and predicted velocity matrix for dropout $p = [0.5, 0.8]$. Track: YOO01.mid

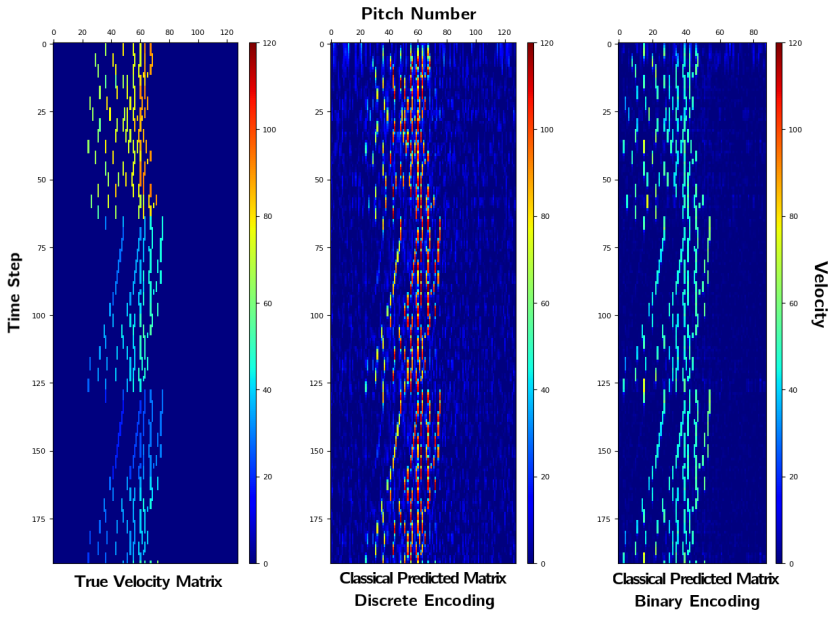


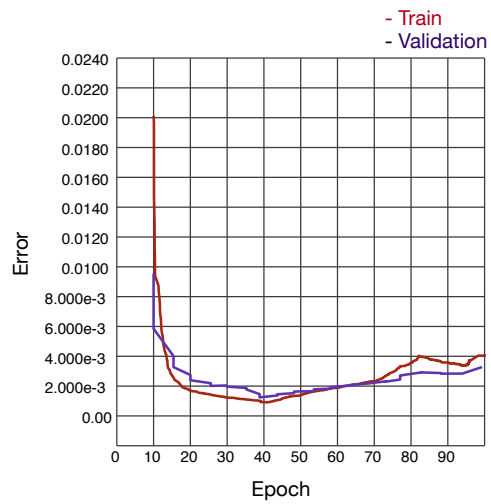
Figure 5.4: True and predicted velocity matrix for different encoding schemes. Track: chpn-p20.mid

for StyleNet. In StyleNet's case, this means one could alternative between mini-batches of Jazz and Classical. Another option would be to train StyleNet on every Jazz mini-batch and then proceed to Classical for every epoch. Both training methods were experimented with as shown in Figure 5.5d.

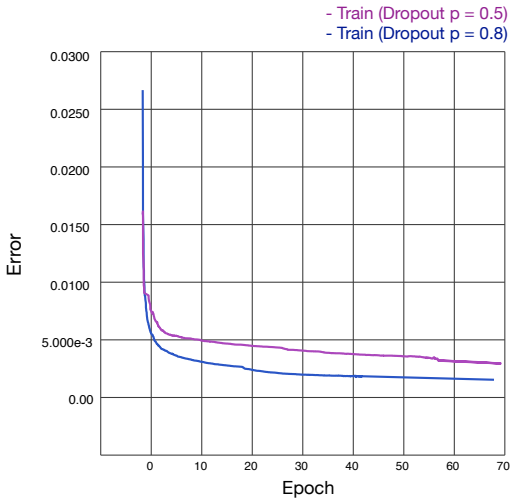
Alternating batch genre converged to a smaller error, 1.1e−4, than bulk genre-batching, 3.2e−3. This is most likely due to sharing the interpretation layer. For bulk genre-training, the calculated weights of the interpretation layer are first updated to decrease the loss for the Classical GenreNet subunit. This is done for every Classical mini-batch. Thus it becomes biased towards Classical music. Then when switching to the next genre, Jazz, the interpretation layer's weights are updated to reduce the loss for the Jazz subunit. However, for training with alternating mini-batches, the shared interpretation layer does not become overly biased towards a single genre. This means that the shared interpretation layer is learning something generic for both genres which could help improve the learning process. The large see-sawing effect by the bulk genre-batching method is mostly likely why it underperformed.

5.3.4 Final Model

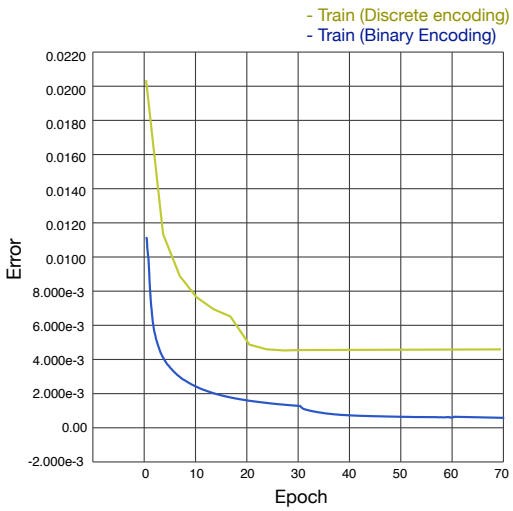
Now the setup and results for the final model as can be listed. A StyleNet model has been successfully trained for the Jazz and Classical music. A dropout of p = 0.8 was applied, and gradients were clipped by norm where g = 10 with a learning rate of 1e−3. The model was training for a total of 160 epochs. The final and validation loss was 7.0e−4 and 1.1e−3 respectively.



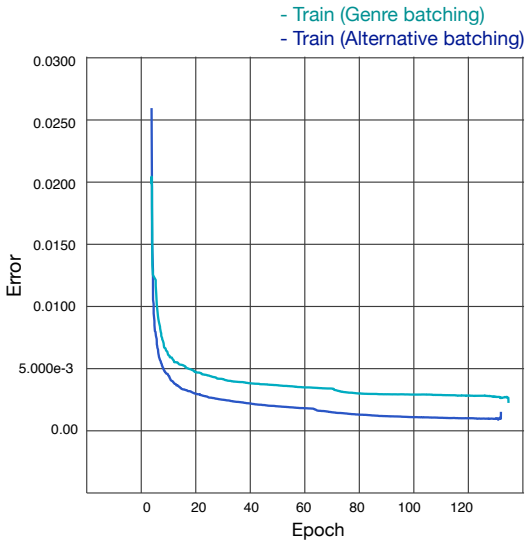
(a) Exploding gradients experiment.



(b) Dropout experiment.



(c) Encoding experiment.



(d) Training method experiment.

Figure 5.5: Experimental runs.

Chapter 6

Evaluation

How does one evaluate a musical performance? Music only holds meaning through the confirmation of a human. The loss metrics of a model do not display its ability to perform convincing music. The decreasing loss shows us that the model is trying to understand the problem numerically. However what one wants is to minimise the "perceptual" loss. Thus it can be quite challenging when trying to evaluating a model in the field of music. This chapter discusses the three experiments taken to assess the performance of the model.

6.1 Overview of Experiments

As mentioned in the introduction, the goal of this thesis was to investigate whether a machine could play sheet music like a human. The second objective was to investigate whether the machine could play sheet music in different styles. Alan Turing's Turing test will be taken as inspiration for the evaluation [2]. The Turing test tests whether a machine can exhibit an intelligent behaviour which is indistinguishable from that of a human. If the model passes a musical Turing test then that concludes that it is possible for a machine to play sheet music like a human. Additionally, the second objective can be tested by asking a human whether they can correctly identify the style of the generated song.

Three experiments are conducted. "Identify the Human" is a musical Turing test. This was performed twice. First on short and then on long audio clips. The other experiment, "Identify the Style" investigates whether the model has learned style. An overview of the surveys can be seen in Table 6.1. The following section will explain the first musical Turing test experiment, "Identify the Human" on short audio clips, followed by "Identify the Style" and lastly, the final "Identify the Human" experiment on a long audio clip. The validation set was used to generate performances for the experiment. This was done to ensure that the model only produced performances on sheet music it had not seen before.

6.2 "Identify the Human" Test

The goal of this test is to evaluate whether the model can produce a musical performance that is indistinguishable to that of a human. This section covers the details of the experiments and its results.

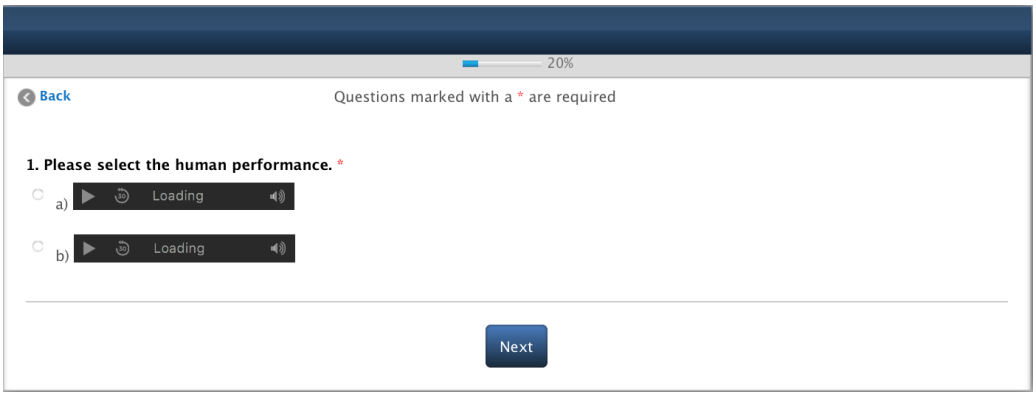


Figure 6.1: "Identify the Human" survey example question.

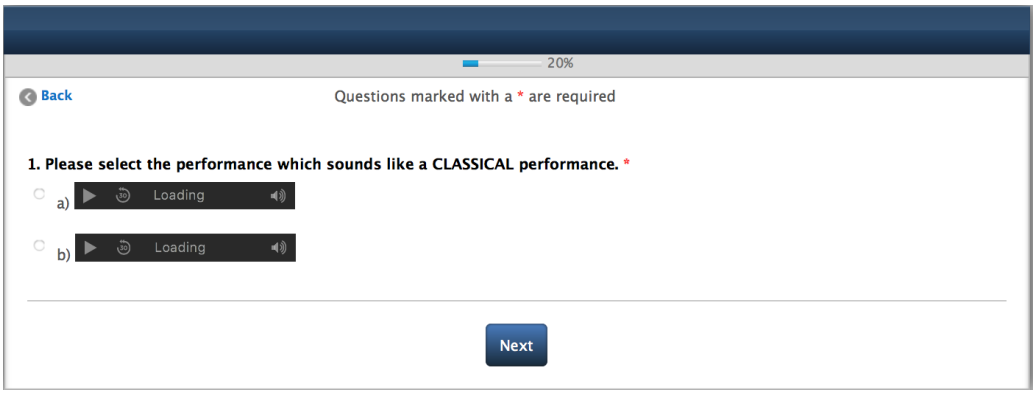


Figure 6.2: "Identify the Style" survey example question.

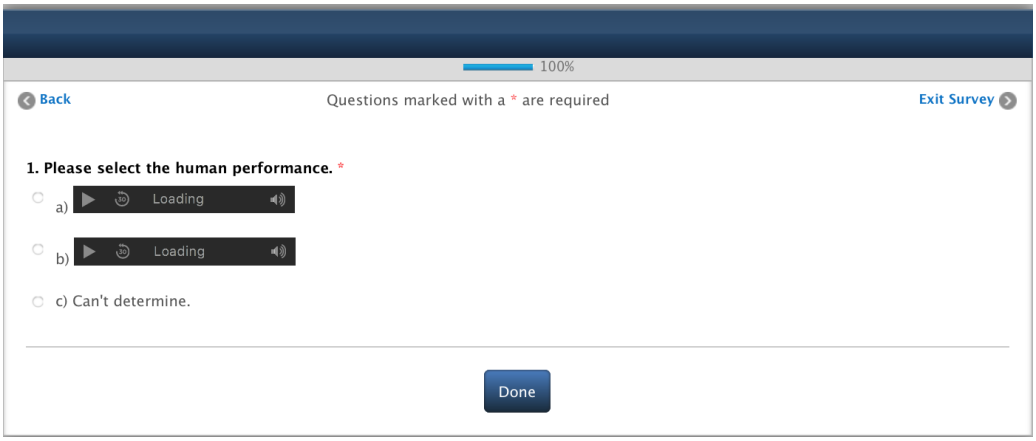


Figure 6.3: "Identify the Human in a long performance" survey example question.

Survey Name	Questions	Respondents
Identify the Human Survey 1	9	30
Identify the Human Survey 2	9	20
Identify the Style Survey 1	9	22
Identify the Style Survey 2	9	20
Identify the Human in a long performance	1	99

Table 6.1: Survey details

6.2.1 Test design

The "Identify the Human" survey was set up in two parts with 9 questions each. For each question, participants are shown two 10 second clips of the same performance as shown in Figure 6.1. One performance is generated and the other is an actual human performance. Participants need to identify the human performance. The ordering of the generated and human tracks was randomised to reduce bias towards a particular answer.

6.2.2 Test results

Figure 6.4 shows the average number of correct answers per question. Combining both surveys, an average of 53% from the participant pool could highlight the human performance. There is no known benchmark to compare it against as this model is the first of its kind. Thus the baseline for this experiment is a random guess between the two available options as seen in Figure 6.1. This reveals that on average, 3% from the participant pool could perform better than random guessing. This is a surprisingly low number and concludes that the model passed the Turing test.

6.2.3 Discussion

Most songs are unique in their musical properties. To analyse the survey results requires the scope of evaluation to become subjective to the question in focus. It is difficult to form generic conclusions about the model when each song is a unique case. One must remember that biases may occur due to the small participant pool. There was no "Cannot Tell" option in the survey. However, it would still be valuable to better understand the cases where the model underperforms and passes as a human performer. The subsection will analyse two questions: Q3 and Q15.

Survey Q3

One can see that that 30% of participants answered with the correct answer. This means 70% chose the generated performance over the human one. This result means the model's performance sounded less synthetic than the real performance. The velocity matrix plot of the two performances can be seen in Figure 6.5. In the human performance, there is a large gradient of velocities at the 68th timestep around key $F3$. The yellow area indicates an area of high velocity. However, in the generated performance, the $F3$ key's velocity increases gradually. The human performance may have surprised the participants with its broad range of dynamics whereas the generated track was more "subtle". One can speculate and say that if the clips were longer then significant changes in velocities may not bias results towards a certain answer.

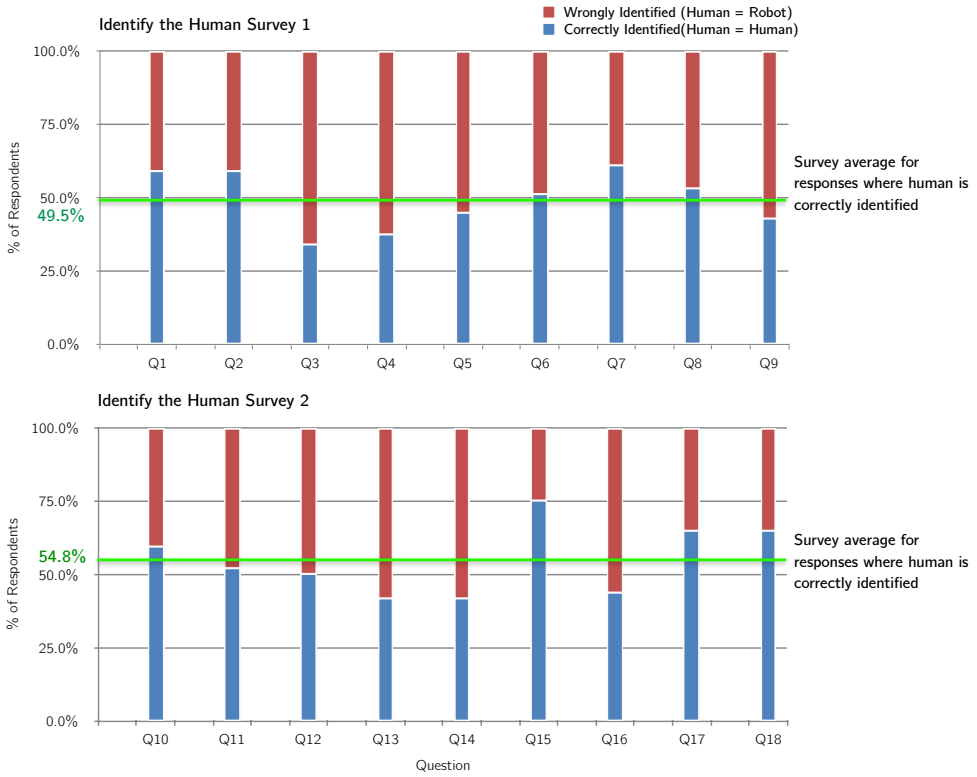
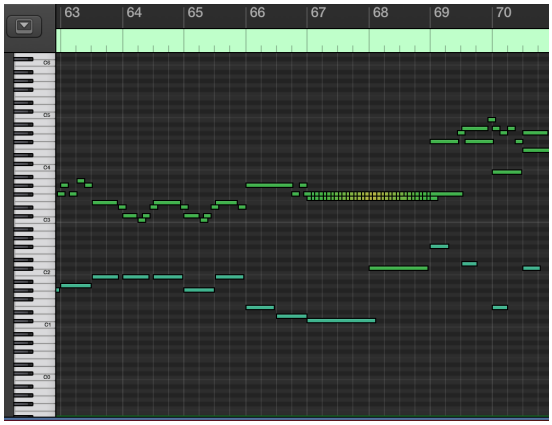


Figure 6.4: "Identify the Human" survey results

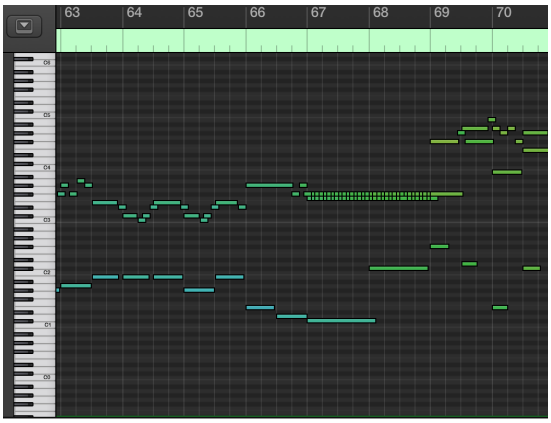
Survey Q15

Another interesting case is Q15. 75% respondents correctly identified the human performance. This means the model's performance is noticeably more synthetic than the real performance. The velocities of the tracks are visualised in Figure 6.6. If one analyses the human performance carefully, there is a larger spectrum of velocities in comparison to the generated performance. There are smaller velocities present. In comparison, the generated performance does not contain these smaller velocities. It can be speculated that the generated performance sounded overly "energetic" to the participant. This may have been the reason why the human performance was chosen. A participant mentioned that when they could not isolate the human performance, they chose the track which sounded softer. Their reasoning was that a human would not write a musical piece that would require so much physical energy. This comment is quite interesting as it means that the model may be able to perform sheet music in a style which is difficult for a human performer. This also means that the lack of a "Cannot Tell" option introduces noise into the survey results.

The next experiment will evaluate the model's performance on its ability to perform sheet music in different styles.

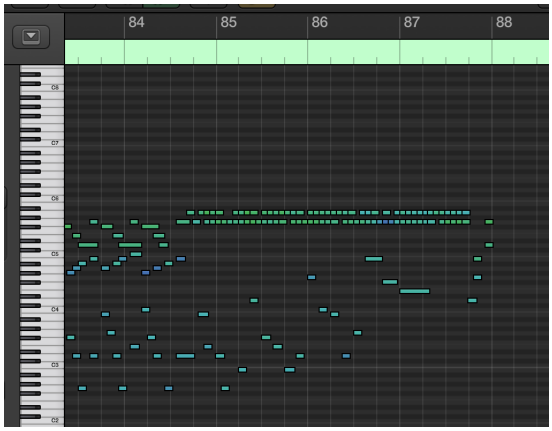


(a) Human performance

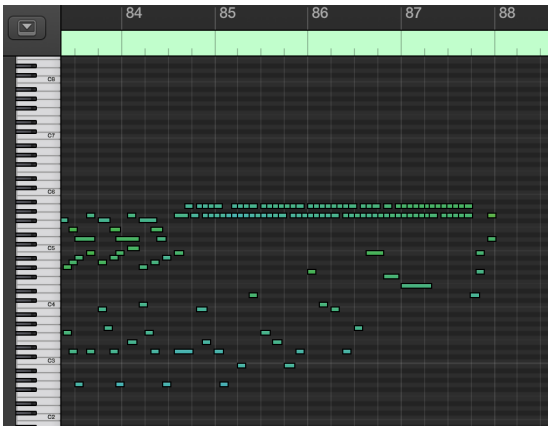


(b) Generated performance

Figure 6.5: "Identify the Human" survey Q3 velocity matrices.



(a) Human performance



(b) Generated performance

Figure 6.6: "Identify the Human" survey Q15 velocity matrices.

6.3 "Identify the Style" Test

With the model successfully producing indistinguishable performance from that of a human, this leads the next investigation into the model's ability to play sheet music in a specific style. This section explains the experiment and provides a discussion of the results.

6.3.1 Test design

The "Classical or Jazz" survey was set up in two parts with 9 questions each. Sheetmusic for a single performance is generated in a Classical and Jazz style. These two stylised tracks are shown to the participants. The task at hand for participants is to correctly identify the style being asked for. The audio setup in section 6.2 was used.

6.3.2 Test results

The results for this experiment can be seen in Figure 6.7. Combining both surveys, an average 47.5% of respondents selected the correct style. Similar to the previous test, the baseline of this test is randomly guessing between both answers. This shows that on average 3.5 of participants performed worse than random guessing. This means they chose the wrong style more often than than a person that just guessed randomly. The analysis of this number shows that the model cannot perform stylised renditions of sheet music.

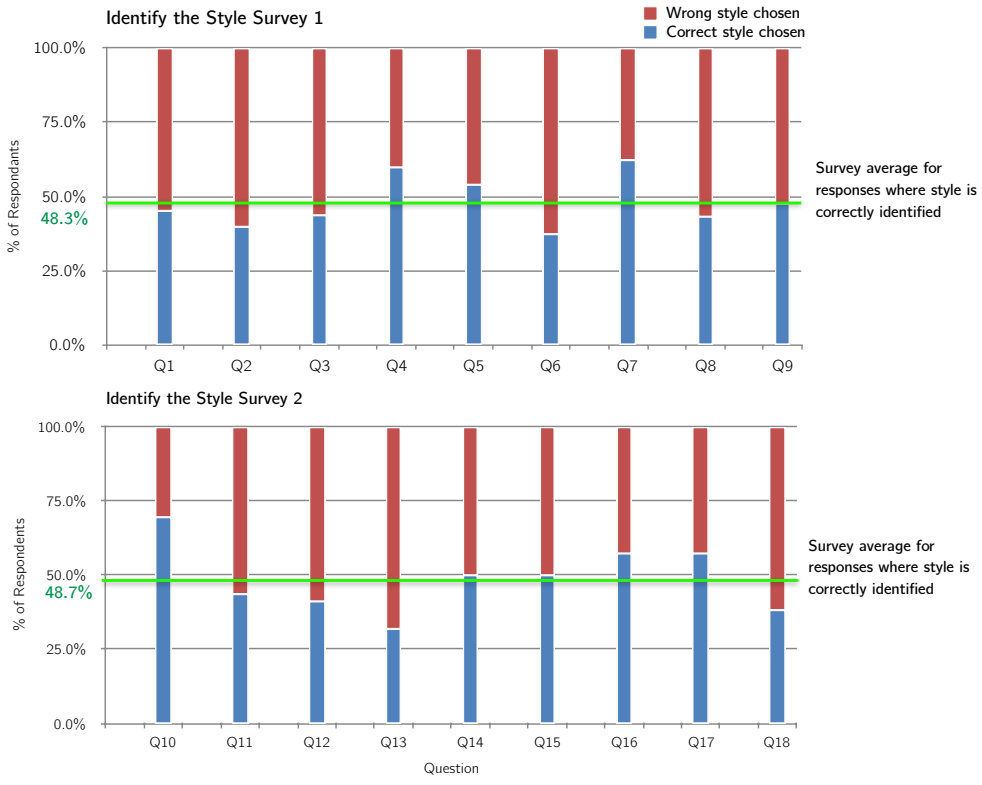
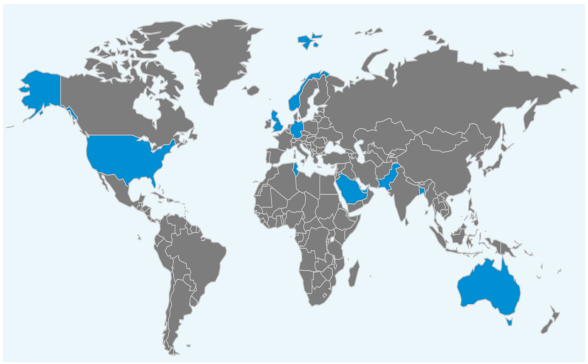


Figure 6.7: "Identify the Style" survey results



Countries	Responses
GB	49.70%
SA	16.17%
AE	13.17%
PK	7.78%
US	4.79%
NO	2.99%
BH	1.20%
DE	1.20%
TN	1.20%
IE	0.60%
BD	0.60%
AU	0.60%

Figure 6.8: Distribution of responses for the final "Identify the Human" experiment

6.3.3 Discussion

The experiment indicates that the model is not producing performances that are distinct enough to be categorised as different genres. How does one define a genre? Everyone has their own internal definition of a genre. It is hard to generalise across individuals and formalise how one labels a composition. Another interesting observation is that in the space of 10 seconds, is it possible for someone to label a song with its corresponding genre? Participants commented by saying they needed more context as 10 seconds was too short for them to identify the style. However, they also mentioned that it was extremely challenging to determine the genre when both tracks featured the same instrument.

This result leads to a few interesting observations. It is known that musicians perform renditions of certain songs. An example of this is Metallica. Metallica is a metal band that has performed their tracks in an Orchestral style [3]. For the style to be easily observable, it is often the case that the performance must over-exaggerate the qualities of that genre such as the instrument set. This could mean that to learn style one also needs to learn the complex interactions between the instruments rather than focusing on a single instrument.

6.4 Final "Identify the Human" Test

As mentioned earlier, some participants mentioned that 10 seconds is not long enough to determine the human performance. It can be hard to assess a short clip without its surrounding music context. This scenario is analogous to taking a sentence out of its contextual paragraph. Thus a more valid Turing test would be to assess the model on a complete performance. This motivates this final Turing test.

6.4.1 Test design

The experiment set-up was identical to the "Identify the Human" test for short audio clips, but the only difference is that participants had to answer one question featuring an extended performance. As mentioned earlier, many participants mentioned that they would select a random answer when they were not sure. Thus a new option called "Cannot Tell" was added

to the survey. An example survey question can be seen in Figure 6.3 The song used for this experiment was "chpn-p25.mid" which is a 2:30 Classical piece called "Etudes Op.25" by Frédéric Chopin.

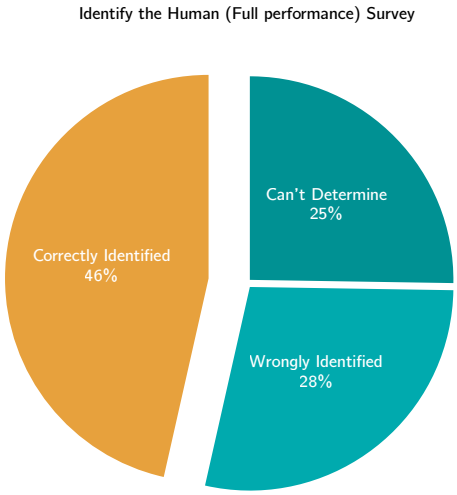


Figure 6.9: Final "Identify the Human" survey results

6.4.2 Test results

The survey was completed by 99 people. Figure 6.9 shows that only 46% participants could identify the human. This shows that humans are not capable of differentiating between synthetic and real music. The distribution of responses can be seen in Figure 6.8 The "Cannot Determine" answer is grouped together with participants incorrectly choosing the generated track as a human. This means 54% percent of the participant pool cannot distinguish the human performance. This is more than half of a larger participant pool in comparison to the previous experiments. This leads to say that the trained StyleNet model has successfully passed the Turing Test and can generate performances that are indistinguishable from that of a human.

6.4.3 Summary of Results

Three experiment have been successfully carried out on the trained StyleNet model. The experiments can be summarised as follows:

1. The first musical Turing test experiment, "Identify the Human", was performed on short audio clips. The results of this experiment concluded that participants could not tell the difference between short generated and real performances.
2. The second experiment "Identify the Style" was held to identify whether StyleNet could perform in different musical styles. Results concluded that participants cannot correctly identify the style of the generated performances. This result leads to say that the model cannot generate noticeably stylised performances.
3. The last experiment "Identify the Human" was performed on a long performance. Similar to the initial test, participants could not tell the different between the two tracks. The results of this experiment strengthen our initial findings. **In conclusion, StyleNet can generate performances that are indistinguishable to that of a human**.

Chapter 7

Conclusion

7.1 Summary of Contributions

This project has completed the goals mentioned in chapter 1:

1. Firstly, a new neural architecture called StyleNet was designed specifically to learn style.

2. A suitable data representation was designed for the StyleNet model. This data representation is analogous to digital sheet music.

3. The piano dataset was created which adheres to certain requirements: all tracks within the dataset are piano solos from Jazz and Classical MIDI.

4. Training experiments were carried out to best train the StyleNet model.

5. StyleNet's performances were evaluated using two musical Turing tests. The model passed both tests. These results conclude that StyleNet can produce human-like performances.

6. An experiment was carried out to determine StyleNet's ability to perform sheet music in a genre-specific style. However, the results showed the model is not able to do so.

7.2 Future Work

7.2.1 Learning Genre-specific Style

With the success of StyleNet genenerating performances for sheet music, the next steps would be to redefine the problem. Currently, the model is learning from piano sheet music. However as speculated in chapter 6, genres usually encompass specific instrument sets. The next steps for this project would be to try and model all of the instruments in a given performance. The would be analogous to having different StyleNets for different instruments. This new model would try and learn the complex interactions between the instruments within the genre. This would be analogous to having a band performing a song using a set of instruments.

7.2.2 Experiment with Truncated Backpropagation

Figuring out which combination of hyperparameters yield optimal results is a challenging task. The evaluated StyleNet model was trained using backpropgation over 200 time steps. This limits the model to learning only dependencies within this window. One could further investigate the perceivable effect different backpropagation time steps have on the model.

7.2.3 Use Case

The trained StyleNet can successfully synthesise the dynamics for sheetmusic. When musicians work on music, they have to encode velocities manually as they create MIDI tracks in their digital audio workstation. This monotonous task is an opportunity for StyleNet. StyleNet should be able to synthesise and inject dynamics into MIDI. This would be extremely beneficial for musicians.

There are several MIDI available that hold one global velocity, or lack a realistic range of velocities. StyleNet could be used to add a range of velocities to such tracks. This opens many doors for using StyleNet to aid the creative process in the field of music.

7.2.4 The Piano Dataset

As highlighted in the introduction, one of the driving forces in the resurgence of neural networks has been the availability of free large datasets. These datasets have promoted a culture of creating challenges and competitions for the public to tackle. Such competitions usually result in significant advancements. This has been especially true for computer vision where datasets such as PascalVOC and ImageNet could be seen as drivers of new novel research. However, this is not the case for music. I have spent a significant amount of work creating and curating the Piano dataset for which I have also produced a testbed survey. One of my next goals is to make this dataset available and create a set of challenges for the public to tackle. Hopefully, the Piano data set can stimulate developments in the area of music generation.

Bibliography

[1] Yamaha piano e-competition. URL http://www.piano-e-competition.com.

[2] M Alan. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950. ISSN 0026-4423. doi: http://dx.doi.org/10.1007/978-1-4020-6710-5_3. URL papers2://publication/uuid/E74CAAC6-F3DD-47E7-AEA6-5FB511730877.

[3] Metallica Album. URL http://ultimateclassicrock.com/metallica-sm/.

[4] Neural Beats. URL https://github.com/snikolov/neural-beats/.

[5] Yoshua Bengio. Practical Recommendations for Gradient-Based Training of Deep Architectures. 2012. URL https://arxiv.org/pdf/1206.5533.pdf.

[6] Colah's Blog. URL http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[7] Fleur Bouwer. *What Do We Need To Hear a Beat?* PhD thesis, 2016. URL http://fleurbouwer.nl/wp-content/uploads/2016/05/FleurBouwer{_}Dissertation{_}Complete.pdf.

[8] Jane Bromley, James W. Bentz, Léon Bottou, Isabelle Guyon, Yann Lecun, Cliff Moore, Eduard Säckinger, and Roopak Shah. Signature Verification Using a Siamese Time Delay Neural Network. *International Journal of Pattern Recognition and Artificial Intelligence*, 07(04):669–688, 1993. ISSN 0218-0014. doi: 10.1142/S0218001493000339.

[9] Daxiang Dong, Hua Wu, Wei He, Dianhai Yu, and Haifeng Wang. Multi-Task Learning for Multiple Language Translation. pages 1723–1732. URL http://www.aclweb.org/anthology/P15-1166.

[10] D. Eck and J. Schmidhuber. Finding temporal structure in music: Blues improvisation with LSTM recurrent networks. In *Neural Networks for Signal Processing - Proceedings of the IEEE Workshop*, volume 2002-Janua, pages 747–756, 2002. ISBN 0780376161. doi: 10.1109/NNSP.2002.1030094.

[11] Douglas Eck and Jürgen Schmidhuber. A First Look at Music Composition using LSTM Recurrent Neural Networks. *Idsia*, 2002. URL http://www.idsia.ch/{~}juergen/blues/IDSIA-07-02.pdf.

[12] Dean Falk. Prelinguistic evolution in early hominins: Whence motherese? *Behavioral and Brain Sciences*, 27(04):491–503, aug 2004. ISSN 0140-525X. doi: 10.1017/S0140525X04000111. URL http://www.journals.cambridge.org/abstract{_}S0140525X04000111.

[13] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A Neural Algorithm of Artistic Style. *arXiv preprint*, pages 3–7, 2015. ISSN 1935-8237. doi: 10.1561/2200000006.

[14] P. Gossett. *A Geometry of Music: Harmony and Counterpoint in the Extended Common Practice*, volume 18. 2012. ISBN 9780199714353. doi: 10.1215/0961754X-1630505.

[15] Alex Graves. Generating Sequences With Recurrent Neural Networks. URL `https://arxiv.org/pdf/1308.0850v5.pdf`.

[16] Klaus Greff, Klaus@idsia Ch, Rupesh Kumar, Srivastava Rupesh@idsia Ch, Jan Koutník, Hkou@idsia Ch, Bas R Steunebrink, Bas@idsia Ch, Urgen Schmidhuber, and Juergen@idsia Ch. LSTM: A Search Space Odyssey. URL `https://arxiv.org/pdf/1503.04069.pdf`.

[17] Karol Gregor, Ivo Danihelka, Alex Graves, and Daan Wierstra. DRAW: A Recurrent Neural Network For Image Generation. *arXiv preprint*, pages 1–16, 2015.

[18] Sepp Hochreiter and J Urgen Schmidhuber. LONG SHORT-TERM MEMORY. *Neural Computation*, 9(8):1735–1780, 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL `http://www7.informatik.tu-muenchen.de/{~}hochreit{%}5Cnhttp://www.idsia.ch/{~}juergen`.

[19] Dominik Hörnel and Wolfram Menzel. Learning Musical Structure and Style with Neural Networks. *Computer Music Journal (CMJ)*, 22(4):44–62, 1998. ISSN 1881-6096. doi: 10.2307/3680893.

[20] David Huron. Science & music: lost in music. *Nature*, 453(7194):456–457, 2008. ISSN 0028-0836. doi: 10.1038/453456a.

[21] Aaron Courville Ian Goodfellow, Yoshua Bengio. Deep Learning Book. *Deep Learning*, 21(1):111–124, 2015. ISSN 1437-7780. doi: 10.1016/B978-0-12-391420-0.09987-X. URL `http://www.deeplearningbook.org/`.

[22] Melvin Johnson, Mike Schuster, Quoc V Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation. URL `https://arxiv.org/pdf/1611.04558.pdf`.

[23] Diederik P Kingma and Jimmy Lei Ba. ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION. URL `https://arxiv.org/pdf/1412.6980v8.pdf`.

[24] B. Wansink K.M. Kniffin, J. Yan and W.D. Schulze. Imagenet classification with deep convolutional neural networks. *Journal of Organizational Behavior*, 2016.

[25] Kevin M. Kniffin, Jubo Yan, Brian Wansink, and William D. Schulze. The sound of cooperation: Musical influences on cooperative behavior, 2016. ISSN 10991379.

[26] Alex Krizhevsky, Ilya Sutskever, and Hinton Geoffrey E. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems 25 (NIPS2012)*, pages 1–9, 2012. ISSN 10495258. doi: 10.1109/5.726791. URL `https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

[27] Yann A. LeCun, L??on Bottou, Genevieve B. Orr, and Klaus Robert M??ller. Efficient backprop. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7700 LECTU:9–48, 2012. ISSN 03029743. doi: 10.1007/978-3-642-35289-8-3.

[28] Feynman Liang and Churchill College. BachBot: Automatic composition in the style of Bach chorales Developing, analyzing, and evaluating a deep LSTM model for musical style.

2016. URL http://www.mlsalt.eng.cam.ac.uk/foswiki/pub/Main/CurrentMPhils/Feynman{_}Liang{_}8224771{_}assignsubmission{_}file{_}LiangFeynmanThesis.pdf.

[29] Mido library, . URL https://mido.readthedocs.io/en/latest/.

[30] Tensorflow library, . URL https://www.tensorflow.org.

[31] Fujun Luan, Sylvain Paris Adobe, Eli Shechtman Adobe, and Kavita Bala. Deep Photo Style Transfer. URL https://arxiv.org/pdf/1703.07511.pdf.

[32] L O Lundqvist, F Carlsson, P Hilmersson, and P N Juslin. Emotional responses to music: experience, expression, and physiology. *Psychology of Music*, 37(1):61–90, 2008. ISSN 0305-7356. doi: 10.1177/0305735607086048.

[33] Minh-Thang Luong, Quoc V Le, Ilya Sutskever, Oriol Vinyals, Lukasz Kaiser, and Google Brain. MULTI-TASK SEQUENCE TO SEQUENCE LEARNING. 2016. URL https://arxiv.org/pdf/1511.06114.pdf.

[34] Google Magenta. URL https://magenta.tensorflow.org/.

[35] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, dec 1943. ISSN 0007-4985. doi: 10.1007/BF02478259. URL http://link.springer.com/10.1007/BF02478259.

[36] Warren S Mcculloch and Walter Pitts. A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY*. *Bulletin of Mothemnticnl Biology*, 52(l2):99–115, 1990. URL http://www.cs.cmu.edu/{~}epxing/Class/10715/reading/McCulloch.and.Pitts.pdf.

[37] Pretty Midi. URL http://craffel.github.io/pretty-midi/.

[38] Steven Mithen, Iain Morley, Alison Wray, Maggie Tallerman, and Clive Gamble. The Singing Neanderthals: The Origins of Music, Language, Mind and Body. *Cambridge Archaeological Journal*, 16(01):97–112, 2006. ISSN 0959-7743. doi: 10.1017/S0959774306000060.

[39] Meinard Müller, Daniel P W Ellis, Anssi Klapuri, and Gaël Richard. Signal Processing for Music Analysis. *IEEE JOURNAL OF SELECTED TOPICS IN SIGNAL PROCESSING*, 0(0), 2011. doi: 10.1109/JSTSP.2011.2112333. URL https://www.ee.columbia.edu/{~}dpwe/pubs/MuEKR11-spmus.pdf.

[40] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *Proceedings of The 30th International Conference on Machine Learning*, (2):1310–1318, 2012. ISSN 1045-9227. doi: 10.1109/72.279181. URL http://jmlr.org/proceedings/papers/v28/pascanu13.pdf.

[41] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *Proceedings of The 30th International Conference on Machine Learning*, 2012. ISSN 1045-9227. doi: 10.1109/72.279181.

[42] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *Proceedings of The 30th International Conference on Machine Learning*, (2):1310–1318, 2012. ISSN 1045-9227. doi: 10.1109/72.279181. URL http://jmlr.org/proceedings/papers/v28/pascanu13.pdf.

[43] Isabelle Peretz. The nature of music from a biological perspective. *Cognition*, 100(1):1–32, 2006. ISSN 00100277. doi: 10.1016/j.cognition.2005.11.004.

[44] Leonid Perlovsky. Musical emotions: Functions, origins, evolution, 2010. ISSN 15710645.

[45] Sebastian Ruder. An overview of gradient descent optimization algorithms. *Web Page*, pages 1–12, 2016. URL `http://arxiv.org/abs/1609.04747`.

[46] Jay Schulkin and Greta B Raglan. The evolution of music and human social capability. *Frontiers in neuroscience*, 8:292, 2014. ISSN 1662-4548. doi: 10.3389/fnins.2014.00292. URL `http://www.ncbi.nlm.nih.gov/pubmed/25278827http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC4166316`.

[47] Walter Schulze and Andries Van Der Merwe. Music generation with Markov models. *IEEE Multimedia*, 18(3):78–85, 2011. ISSN 1070986X. doi: 10.1109/MMUL.2010.44.

[48] M. Schuster and K. K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997. ISSN 1053-587X. doi: 10.1109/78.650093. URL `http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=650093`.

[49] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. URL `http://www.jmlr.org/papers/volume15/srivastava14a.old/source/srivastava14a.pdf`.

[50] Hope R Strayer. From Neumes to Notes: The Evolution of Music Notation. *Musical Offerings*, 4(1), 2013. doi: 10.15385/jmo.2013.4.1.1. URL `http://digitalcommons.cedarville.edu/musicalofferings/vol4/iss1/1`.

[51] J. R. Stuart and N. Peter. *Artificial Intelligence a Modern Approach*, volume 72. 2003. ISBN 0137903952. doi: 10.1017/S0269888900007724. URL `http://web.info.uvt.ro/{~}hpopa/AI/RussellS,NorvigPArtificialIntelligence-AModernApproach(2Ed,Ph,2003)(T)(1112S).pdf`.

[52] Bob L. Sturm, João Felipe Santos, Oded Ben-Tal, and Iryna Korshunova. Music transcription modelling and composition using deep learning. *arXiv cs.SD*, 4:08723, 2016. URL `http://arxiv.org/abs/1604.08723`.

[53] Ilya Sutskever. Training Recurrent Neural Networks. *Ph.D thesis*, 2012.

[54] M. L. WEST. THE BABYLONIAN MUSICAL NOTATION AND THE HURRIAN MELODIC TEXTS. *Music and Letters*, 75(2):161–179, 1994. ISSN 0027-4224. doi: 10.1093/ml/75.2.161. URL `https://academic.oup.com/ml/article-lookup/doi/10.1093/ml/75.2.161`.

[55] David Wulstan. The earliest musical notation. *Music and Letters*, 52(4):365–382, 1971. ISSN 00274224. doi: 10.1093/ml/LII.4.365.

[56] Logic Pro X. URL `https://www.apple.com/uk/logic-pro/`.

[57] W Xiong, J Droppo, X Huang, F Seide, M Seltzer, A Stolcke, D Yu, and G Zweig. ACHIEVING HUMAN PARITY IN CONVERSATIONAL SPEECH RECOGNITION. 2017. URL `https://arxiv.org/pdf/1610.05256.pdf`.

[58] Wenpeng Yin, Hinrich Sc, Bing Xiang, and Bowen Zhou. ABCNN: Attention-Based Convolutional Neural Network for Modeling Sentence Pairs. URL `https://arxiv.org/pdf/1512.05193.pdf`.

[59] Wojciech Zaremba, Ilya Sutskever, Oriol Vinyals, and Google Brain. RECURRENT NEURAL NETWORK REGULARIZATION. 2015. URL `https://arxiv.org/pdf/1409.2329.pdf`.

[60] Marcel Zentner and Tuomas Eerola. Rhythmic engagement with music in infancy. *Proceedings of the National Academy of Sciences of the United States of America*, 107(13):5768–73, mar 2010. ISSN 1091-6490. doi: 10.1073/pnas.1000121107. URL `http://www.ncbi.nlm.nih.gov/pubmed/20231438http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC2851927`.

[61] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. URL `https://arxiv.org/pdf/1703.10593.pdf`.

[62] Bristol HPC Zoo. URL `http://uob-hpc.github.io/zoo/`.

Appendix A

Additional Material

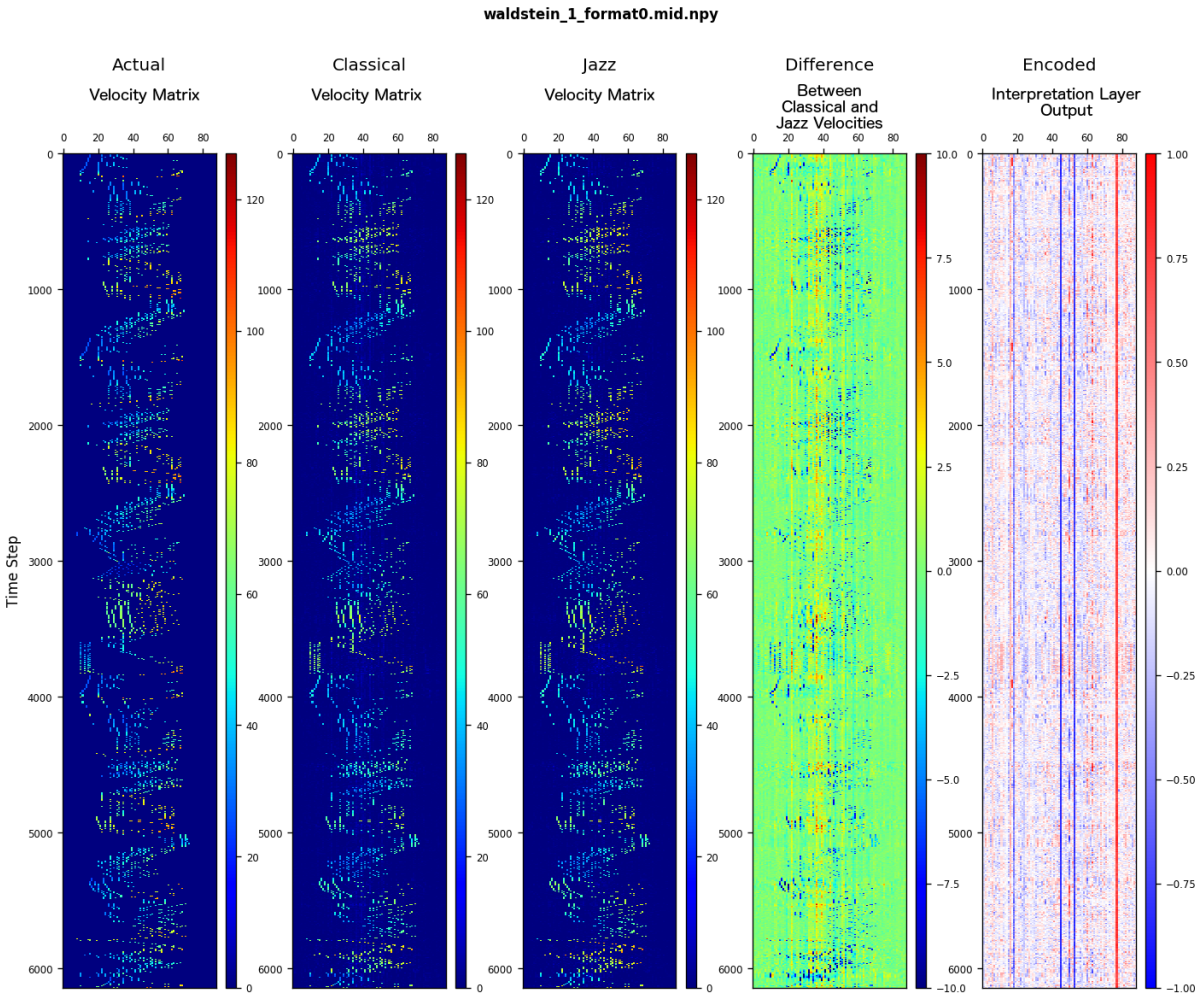


Figure A.1: Training snapshot of StyleNet's predictions.

Neural Translation of Musical Style

Iman Malik, supervised by Dr. Carl Henrik Ek

1. What is Musical Style?

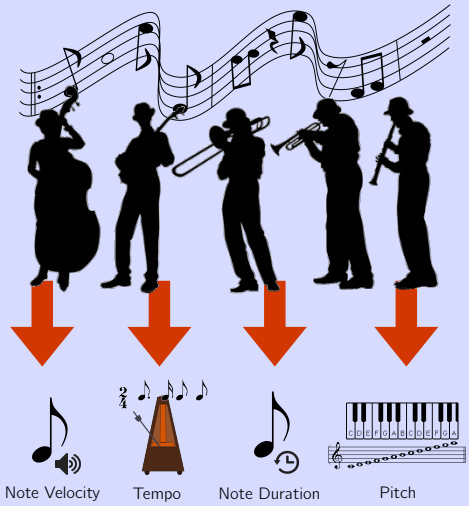


Fig 1. Definitive properties of musical style.

- Machine-generated music still struggles to capture the '**human touch**' of playing music.[1]
- Humans express emotion through **musical style**.
- Different **genres** and **instruments** display definitive styles.
- **Can a machine learn musical style?**

2. Data

- Recordings of songs exist in **MIDI format** which can be obtained easily through the internet.
- MIDI captures the stylistic properties of songs.

Fig 2. Capturing style by playing on a MIDI keyboard.

3. Model

- Music can be thought of as a **sequence of notes.**
- Notes played in the **future** are dependent on the notes played in the **past**.
- **Traditional Neural Networks** assume their inputs are **independent** of each other.
- **Long Short Term Memory** networks are capable of learning these **long term dependencies**.
- The network cells have **feedback loops** and **memory** which gives them the ability to **retain** information over long periods of time.

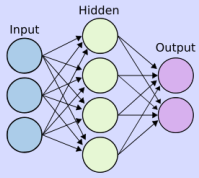


Fig 3. Generic feed-forward neural network architecture.

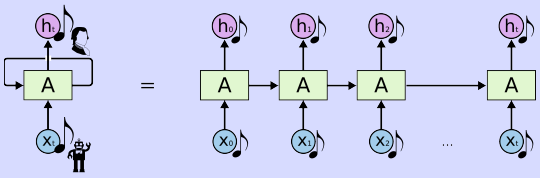


Fig 4. A LSTM network unrolled over time.

4. Outcome

- The **neural model** will be trained on "**robotic**" music and will learn to output **stylised** music.
- First, it will learn to stylise the **note velocities** for a given MIDI.
- The model will then learn about **note durations** to mimic human imperfections.
- Finally, it will learn how style differs between **genres** and **instruments.**

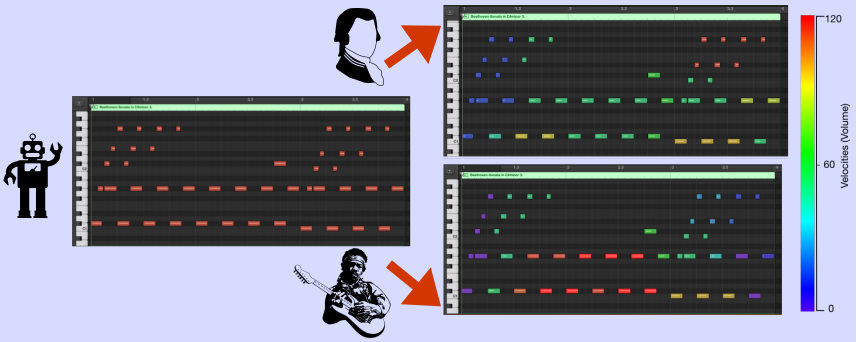


Fig 5. Stylising robotic input to Classical Piano and Rock.

University of BRISTOL

References:
[1] Douglas Eck and Juergen Schmidhuber. 2002. *A First Look at Music Composition Using LSTM Recurrent Neural Networks.* Technical Report. Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale.
[2] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (November 1997), 1735-1780.

Figure A.2: Poster.