Hearing the future: Predicting the next piece of audio

This dissertation is submitted in partial fulfilment of the requirements for the degree of MSc Advanced Computer Science

Yiwei Su

Supervised by Dr. Dave Marshall

School of Computer Science and Informatics

Cardiff university

September 2023

Abstract

Music is a comprehensive art form that integrates emotional expression, information dissemination, and audience engagement. With the rapid rise of digital music platforms and the explosive growth of music content, there is a growing interest in developing intelligent systems with the capability to understand and create music. One highly notable research direction involves predicting the future few seconds of a musical composition based on preceding segments. This technology has vast application prospects, including music recommendation systems, automatic music composition, and improvisational music. However, traditional methods of music prediction exhibit limitations when dealing with nonlinear and complex time-series data. Hence, there is an urgent need for more advanced approaches to enhance the accuracy of music prediction.

To address this challenge, this study introduces Long Short-Term Memory (LSTM), a powerful deep learning model particularly suitable for modeling time-series data. In the context of music prediction, melodies in music often repeat at different positions within a song. LSTM's excellent ability to capture long-term dependencies in music data makes it an ideal choice for addressing music prediction, especially in handling the repetitive melodies that frequently appear in music. This paper provides a detailed explanation of how LSTM works and justifies the rationale for choosing LSTM as the proposed solution.

By training the LSTM model on a diverse dataset comprising over 1000 music compositions, the model learns the fundamental structures and patterns of music sequences. Experimental results demonstrate a notable advancement in music sequence prediction. The model performs well in predicting known music styles and exhibits the ability to make simple melody predictions for unknown music styles, albeit with slightly reduced accuracy. These findings provide strong support for the further development of music generation systems and the enhancement of the intelligence level in music composition, opening new possibilities for future music creation and music technology research.

Acknowledgements

Firstly, I would like to thank my supervisor, Dr Dave Marshall, for his invaluable advice on the writing of my dissertation, which enabled me to complete my dissertation successfully. Secondly, I would like to express my special thanks to my parents for their financial and emotional support on my academic journey. It's their trust that keeps me going.

Table of Contents

1.Introduction

1-1. Motivation and Applications

Music, as a universal art form, possesses a potent capacity for emotional expression, information conveyance, and audience engagement. With the widespread adoption of digital music platforms and the continuous emergence of musical content, there is an increasing need to develop intelligent systems capable of comprehending and generating music. Such systems hold the potential to revolutionize various aspects of the music domain. To illustrate, by predicting the musical direction of a song in the upcoming few seconds, we can enhance the performance of music recommendation systems, enabling personalized transitions between tracks and thereby delivering more captivating music experiences for enthusiasts. In the realm of automated composition, music generation technologies offer novel inspiration to composers, expanding their creative horizons and exploring diverse musical possibilities. Furthermore, music generation technologies can serve as tools for improvisation, serving as robust aides for musicians during spontaneous performances. By analyzing the current musical segment, these systems can generate harmonious melodies, enriching the diversity of musical performances. Hence, a strong interest in the field of music generation is natural, and the development of a system capable of accurately predicting the musical trajectory based on preceding segments of a musical composition holds significant importance. This has the potential to benefit numerous domains, including music recommendation systems, automated composition, and improvisational creativity, with the prospect of catalyzing revolutionary advancements in the music industry.

The prediction of music sequences has been a long-standing research topic among both scholars and music enthusiasts. In fact, there have been various traditional methods for music generation even before the exploration of neural networks for music generation (Yamshchikov & Tikhonov, 2020). These traditional approaches typically rely on rule-based statistical methods to generate or predict musical patterns. However, these methods have certain limitations when it comes to capturing the intricate and nuanced relationships within music. With the recent advancements in machine learning and deep learning techniques, there is now an opportunity to leverage these tools to enhance the accuracy and complexity of music prediction.

Long Short-Term Memory (LSTM) is a form of recurrent neural network (RNN) that has shown exceptional performance in capturing long-term dependencies and sequential patterns in various domains such as natural language processing and speech recognition. LSTM models can remember and utilize information from the past while processing new input, making them particularly suitable for tasks involving sequence prediction. By training LSTM on large datasets of musical compositions, it is possible to enable the models to learn the underlying structures and patterns. LSTM models have the potential to capture the intricate relationships between musical elements such as rhythm, melody, and harmony, and generate coherent and musically relevant continuations.

Another model used in the field of music generation is the Transformer. Transformer is a model architecture based on the attention mechanism, designed to handle a variety of sequence tasks. One of its most prominent features is the incorporation of the attention mechanism, enabling the model to simultaneously consider all positional information within a sequence, unconstrained by local information. This capability empowers the model to capture intricate relationships within sequences, significantly enhancing its ability to comprehend contextual information. Building upon this theoretical framework, Huang et al. (2017) introduced the Transformer into the field of music generation and successfully developed the Music Transformer, a SeqToSeq model specifically designed for the domain of music. By combining the Transformer architecture with music theory, this model can understand and capture the complex relationships between musical notes, rhythms, and harmonies, thereby generating stylistically consistent thematic music successfully. However, it is worth noting that Transformer exhibits high computational resource and large-scale data requirements, along with a relatively complex configuration compared to traditional RNN-based models (Karita et al., 2019). In this research, the priority will be given to the use of neural network models based on traditional RNN.

In summary, the objective of this research is to delve deeply into the feasibility of music prediction. To achieve this goal, I will meticulously select suitable models and validate their effectiveness through a series of rigorous experiments. I aspire that the outcomes of this study will provide robust

support for the development of more intricate and intelligent music generation systems in the future, thereby paving the way for advancements in this field.

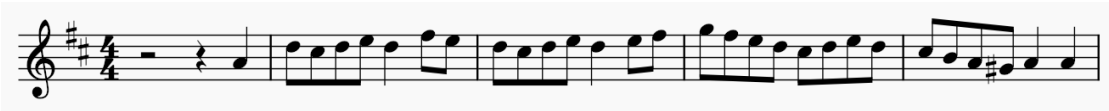
1-2. Roadmap of Dissertation

In this dissertation, the content of each chapter is organized as follows. Chapter Two delves into the background knowledge of music prediction, providing a detailed exploration of the challenges and the approach taken. Chapter Three focuses on the software and libraries employed in the practical implementation, offering readers a comprehensive understanding of the tools and resources necessary for practical application in similar projects. Chapter Four concentrates on the data processing procedures, elaborating on data acquisition and preparation. Chapter Five outlines the construction of neural network models and the selection of parameters. Chapter Six presents the various steps of the experiment in code form for reproducibility. Chapter Seven discusses the model's evaluation methods and assesses the model. Chapter Eight summarizes the experimental results and explores directions for future work. Chapter Nine offers personal reflections on the research journey.

2.Background

2-1. Music Theory

Music is a universal art loved by people all around the world, and its essence lies in the combination of sounds in the dimension of time. Before the widespread application of artificial intelligence, music composition was considered a creative art form mastered only by talented musicians. In recent years, artificial intelligence (AI) technology has made significant advancements, and people have begun to explore the use of AI in music composition. In modern Western music, the primary notation system used to represent music is the staff notation, as shown in the diagram below:



The most essential component of staff notation is the note, which represents different durations of sound. The whole note, half note, quarter note, eighth note, and sixteenth note are the most common types of notes. Rests are used to indicate intervals of silence corresponding to different durations of sound.

Melody involves the temporal sequence of note events, representing at least pitch, onset, and duration (including rests) of a single (monophonic) voice (Rohrmeier & Rebuschat, 2012). Different instruments may use the same pitch and work together to form a composition.

2-2. MIDI

MIDI (Musical Instrument Digital Interface) is a standard established collaboratively by electronic instrument manufacturers, often metaphorically referred to as the "computer-understandable musical score." It aims to facilitate the exchange of information and control signals between computer music programs, synthesizers, and other electronic audio devices, laying a robust technological foundation for the advancement of the electronic music field (Moore, 1988). MIDI files, as the offspring of this standard, bear crucial sound attribute information from music, including played notes, instrument selections, note timings, accompaniments, and more. In essence, the MIDI system constructs a comprehensive framework for composition, orchestration, and electronic simulation performance,

providing robust support for music creation and performance.

MIDI employs instructions such as notes and control parameters to represent music, specifying which note to play and at what volume, among other details. In this study, our primary focus revolves around the manipulation of note data generated by instruments. Therefore, MIDI format is considered an ideal choice for this task. A typical MIDI file typically comprises one or multiple tracks, each presenting musical notes in a staff-like notation. These notes, when combined, form a complete MIDI file, with the data collectively representing various elements and characteristics of the music, including melodies, harmonies, rhythms, and more. A key function of MIDI is to guide digital music processors in simulating sounds. These digital music processors are responsible for playing sounds based on their internal representations of analog instruments (Breve et al., 2021).

In practice, using the MIDI format for music data offers several advantages. Firstly, MIDI files do not contain actual sound waveform data, which results in their typically compact file sizes, allowing them to convey rich musical information with relatively small file sizes. Secondly, MIDI data is highly editable, enabling convenient modifications and adjustments to elements like notes, volumes, and instrumentations, providing musicians and composers with significant flexibility. Additionally, programming languages like Python offer excellent support for MIDI format, facilitating the saving of generated music data as readily accessible MIDI files through data streams. In summary, MIDI provides critical support and assistance in various fields, including machine learning and music composition (Loy, 1985).

2-3. Machine Learning and Neural Network

Machine Learning is a branch of Artificial Intelligence (AI) that enables computers to learn from data and improve performance on specific tasks without explicit programming. It revolves around the concept of algorithms and statistical models that recognize patterns, relationships, and insights in datasets. The process begins with data collection, where relevant information is gathered and transformed into features. These features are then used to train machine learning models to make predictions, classify data, or solve complex problems. Machine learning is typically divided into

three main phases: data preprocessing, model training, and evaluation. Data preprocessing involves converting raw, unstructured data into a format that computers can understand (Zhou et al., 2017). Model training involves selecting appropriate algorithms and tuning their parameters. The final evaluation phase is used to assess whether machine learning is achieving the desired goals. For example, classifier performance evaluation may involve error estimation, dataset selection and performance measurement. After the evaluation, model training can be adjusted based on the evaluation results, such as modifying the selected learning algorithm or adjusting its various parameters (Japkowicz & Shah, 2011).

Neural networks, as a pivotal branch of machine learning, draw inspiration from the principles governing the operation of neurons in the human brain and have found extensive applications in addressing various machine learning and artificial intelligence tasks. Neural networks are comprised of multiple layers, each containing varying numbers of neurons, interconnected through weighted connections. The fundamental building blocks of a neural network, including how they handle input values, compute results, and establish connections among themselves, can be altered, collectively defining what is known as the network architecture. Modifying these aspects can lead to changes in the network's learning behavior, predictive accuracy, and more (Michelucci, 2018).

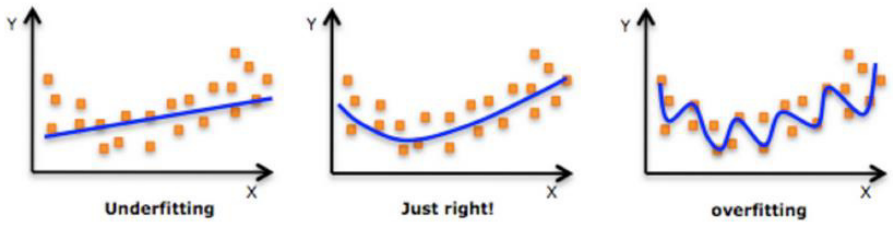
Neural networks offer significant advantages when it comes to handling complex data patterns, particularly in the domain of music data processing. Music data typically exhibits highly intricate nonlinear characteristics, and music prediction tasks demand models capable of capturing temporal information within music. It is precisely due to these characteristics that neural networks have emerged as the ideal choice for music data analysis and modeling. It is worth noting that neural networks employ activation functions to introduce nonlinearity, thereby enabling them to learn and represent complex data patterns. Without activation functions, each layer in a neural network would perform a linear transformation of the input from the preceding layer. Regardless of the network's depth, this would ultimately result in a simple linear combination of inputs. The introduction of activation functions empowers neural networks with the capacity for nonlinear modeling, enabling them to approximate various complex nonlinear functions. This makes neural networks excel in

handling a wide range of nonlinear modeling tasks (Sharma, 2017).

2-4. Underfitting and Overfitting

The concepts of overfitting and underfitting have been introduced early in fundamental classifiers in machine learning, such as linear regression and logistic regression. Overfitting occurs when the model excessively learns from the training data, capturing not only the underlying patterns but also noise and random fluctuations. As a result, this type of model performs excellently on the training data but often fails to generalize well to unseen or new data (Ying, 2019). This leads to a decrease in performance and the creation of overly complex models that may consume unnecessary learning time and computational resources. On the other hand, overfit models tend to perfectly fit every detail of the training data, overlooking the hidden patterns in the data (Wu & Shapiro, 2006). Conversely, underfitting occurs when the model is too simplified and fails to capture the true relationships within the data. It performs poorly on both the training data and unseen data because it oversimplifies the problem. Underfit models cannot grasp the complexity inherent in the data.

To better comprehend these two scenarios, let's consider linear regression as an example. In linear regression, as illustrated in the figure below, the model attempts to fit a straight line to describe the data's relationship. If the model chooses an overly simple linear function, it may fail to capture the true underlying patterns in the data, resulting in underfitting. Conversely, if the model selects a function that is overly complex, it may find unnecessary minor fluctuations in the data, leading to overfitting.



$\theta_0 + \theta_1 x$ $\theta_0 + \theta_1 x + \theta_2 x^2$ $\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

In this study, when using LSTM neural networks to predict music, we also face the challenges of overfitting and underfitting. When our model underfits, it means that it has not effectively captured the core features and complexity of the music in the training dataset. This can lead to overly random and chaotic predictions when trying to forecast actual music. In other words, the model has not learned enough rich musical features and patterns, so its predictions may not accurately reflect the true essence of the music.

Overfitting, on the other hand, is a more challenging issue to address. In contrast to underfitting, when the model overfits, it almost perfectly fits the music information in the training dataset. While this may perform well on the training data, it may lead to a problem: the model's ability to generalize to new music data is limited. In other words, the model may generate predictions highly like the music in the training dataset, and it may even end up mimicking the music from the training data. This is because the model is too focused on specific examples in the training dataset and lacks the ability to learn the broader features and underlying patterns of the music.

2-5. Problem Statement and Approach

This section will delve into the challenges faced in music prediction tasks and perform a comparative analysis of methods used in music prediction tasks. Lastly, I will provide a detailed exposition of the chosen method and elucidate the reasons for this selection.

In this research, the music prediction task confronts two primary challenges. Firstly, the nerual network model necessitates the capability of long-term memory. Music compositions often consist of recurring elements across different time axes. Consequently, in the process of music prediction, the model must possess the capacity to recall previous elements, enabling it to comprehend and forecast future musical trends (Huang et al., 2018). Secondly, it is imperative to devise effective methods for capturing diverse information within the music, with the most crucial elements being musical pitch and note duration. Pitch determines the melody and harmony of the music, while note duration influences the rhythm and timing of musical sequences. Therefore, accurate extraction of these musical features is essential in processing music data to achieve predictions about the future

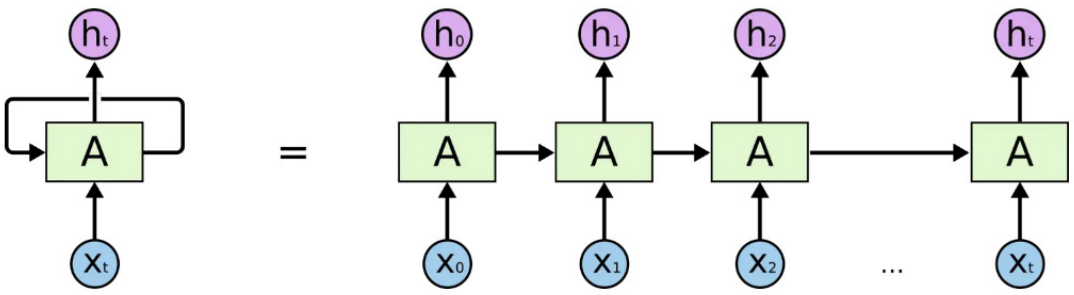
developments in music. This aspect will be elaborated upon further in the subsequent "Data Processing" chapter.

Addressing these challenges is crucial for the success of music prediction tasks. Insufficiencies in long-term memory and information capture may result in inaccurate predictions by the model, consequently affecting the overall coherence of the music. The following sections will introduce two models that have been utilized in music prediction tasks.

2-5-1. RNN

Recurrent Neural Networks (RNN) belong to a class of neural networks that process sequences of data as inputs and produce sequences of data as outputs, operating recursively in the direction of sequence progression. All nodes within RNN, known as recurrent units, are connected in a chain-like manner. What distinguishes RNN is their ability not only to consider the current input but also to maintain a form of "memory" of past information. This is manifested in the network's ability to remember previous information and apply it to the computation of the current output. Specifically, the nodes in the hidden layers are interconnected, and the input to the hidden layers includes not only the output from the input layer but also the output from the hidden layer at the previous time step (Ciaburro & Venkateswaran, 2017).

Consider an example: when comprehending the meaning of a sentence, understanding each word in isolation is insufficient; one needs to process the entire sequence of words connected. The same principle applies to music prediction—if you want to predict the next few seconds of a piece of music, you need knowledge of the entire melody.



As shown in the diagram, a typical RNN network comprises an input (X_t), an output (h_t), and a

neural network unit (A). Unlike regular neural networks, the neural network unit in an RNN is not only connected to inputs and outputs but also forms a loop back to itself. This network structure reveals the essence of RNN: information from the previous time step's network state influences the network state in the next time step. In more detail, this can be seen as decomposing the RNN into multiple independent neural network units. At time 0, the initial input is X_0, the output is h_0, and the network unit's state at time 0 is stored in A. When the next time step arrives, the state of the network unit currently is influenced not only by the input at time 1 (X_1) but also by the state of the unit at time 0. This pattern continues until the end of the time sequence at time t.

In previous research, many scholars have attempted to employ RNN models to learn musical patterns for the purpose of music prediction. For instance, Todd (1989) trained RNN models to predict both the pitch and duration of musical notes, subsequently generating music note by note. However, the implementation of RNN has encountered a series of challenges, with the most prominent ones being the issues of gradient vanishing and gradient explosion. These problems arise during the training of deep RNN, where information fails to effectively propagate across lengthy sequences, consequently constraining the network's capacity to model long-term dependencies effectively (Sherstinsky, 2020). These challenges are especially notable in the field of music generation, particularly when generating longer musical compositions. As the length of the musical piece increases, RNN face challenges concerning their performance and stability. This increase in complexity can lead to instability and inconsistency in the musical structure. Furthermore, for lengthy sequences like music data, RNN encounter the issue of high storage and computational costs, making it difficult to handle large-scale music datasets in practical applications.

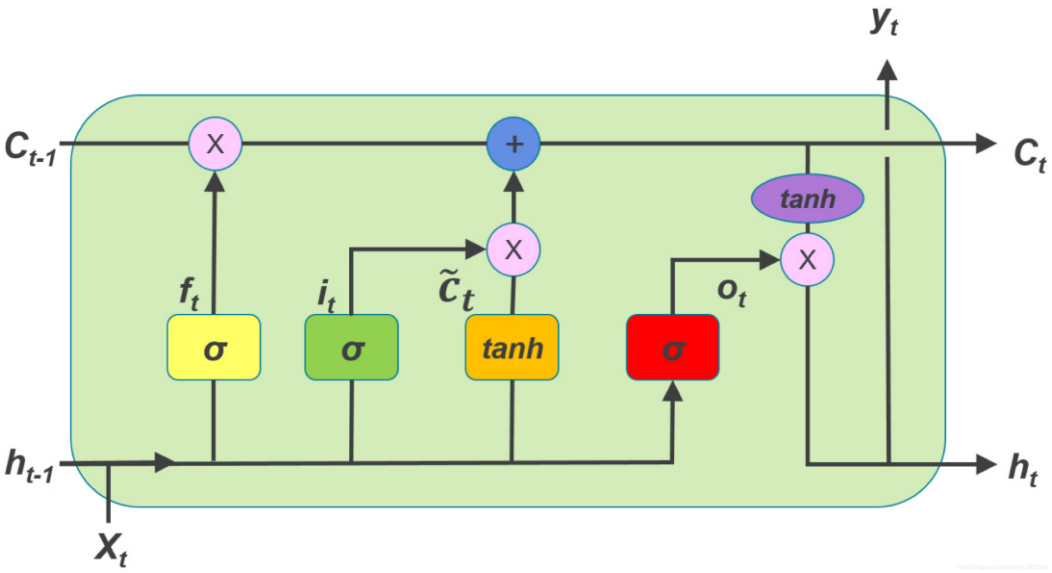
2-5-2. LSTM

Music data often requires the long-term storage of specific information. For example, in a song, a particular melody may persist throughout the entire composition. However, traditional RNN are not the optimal choice in such cases due to their limitations in effectively handling long-term dependencies. LSTM are a specialized variant of RNN designed to handle long-term dependencies in sequential data. Unlike traditional RNN, LSTM automatically retain long-term information internally

without the need for explicit learning (Yang et al., 2018). LSTM achieve this by introducing gate mechanisms, including the forget gate, input gate, and output gate, which efficiently capture and preserve essential information in music sequence data. This design effectively overcomes the vanishing gradient problem often encountered in traditional RNN, significantly enhancing their capability to process long sequences of data.

In applications like music prediction, a LSTM's ability to handle long-term dependencies is particularly crucial. This is because works of the same music style often share certain common elements that need to be retained throughout the music generation process. However, the music data in this study often exhibits vast and intricate characteristics, posing challenges for a traditional RNN that tend to lose essential information when handling complex, long sequential music data. Consequently, the adoption of LSTM networks proves advantageous in better capturing and leveraging the long-term dependencies in music, ultimately leading to improved accuracy and quality in music prediction.

Moreover, when confronted with intricate music prediction tasks, by using the previous layer's hidden state as input for the current LSTM layer, the structure of the LSTM network can be further strengthened through the stacking of multiple layers (Donahue et al., 2015). This structural enhancement enables a LSTM to capture and preserve intricate patterns and long-term memory, thereby further improving its performance in music prediction more effectively.



At the heart of LSTM lies the concept of a "cell state," which functions like a conveyor belt running along the entire sequence. It allows for the long-term storage of information with minimal alteration. The gate mechanisms serve as selective filters for information passage and consist of a sigmoid neural network layer and element-wise multiplication operations. The sigmoid layer outputs values between 0 and 1, determining how much information should pass through.

Specifically, the forget gate plays a pivotal role in deciding which information should be discarded or forgotten from the cell state and is computed as follows:

$$F_t = \delta\big(W_f \cdot [h_t - 1, X_t] + b_f\big)$$

Where δ represents the sigmoid function, W_f is the weight matrix, h_{t-1} denotes the previous time step's hidden state concatenated with the current time step's input X_t, and b_f is the bias term.

The input gate decides which new information should be added to the cell state and consists of two parts, calculated as follows:

$$i_t = \delta(W_i \cdot [h_t - 1, X_t] + b_i)$$

$$c_t = \tanh(W_c \cdot [h_{t-1}, X_t] + b_c)$$

Where W_i, W_c, b_i and b_c are the weight matrices and bias terms, respectively.

The output gate determines how information from the cell state should be passed to the current time step's hidden state and output. It also comprises three parts:

$$o_t = \delta(W_o \cdot [h_{t-1}, X_t] + b_o)$$

$$C_t = f_t \cdot C_{t-1} + i_t \cdot c_t$$

$$h_t = o_t . \tanh(C_t)$$

Where W_o and b_o are the weight matrix and bias term, respectively.

With the continuous advancement of deep learning technology, LSTMs have found widespread application in music prediction tasks. Previous research, such as that by Eck and Schmidhuber (2002), fully leveraged the long-term memory capabilities of a LSTM, successfully learning complex structures in blues music and using them to create creatively unique and coherent musical melodies. Similarly, Colombo et al. (2017) employed a LSTM to generate monophonic melodies in folk music. These studies highlight the substantial potential of a LSTM in the field of music composition,

providing robust support for a deeper understanding and advancement of innovation in the realm of music.

Given the outstanding performance of LSTMs in addressing long-term dependencies and its remarkable capabilities in handling lengthy sequences of musical data, this research will also employ LSTM technology for music prediction.

3. Software and Associated Libraries Used in This Project

3-1. TensorFlow

TensorFlow is a versatile and scalable software library designed for numerical computation using data flow graphs. This library empowers users to efficiently develop, train, and deploy neural networks and various machine learning models for practical applications. TensorFlow's core algorithms are written in C++ and CUDA and have been meticulously optimized. It provides APIs in multiple programming languages, with Python offering the most comprehensive and stable support. TensorFlow usage typically consists of two phases: construction and execution. During the construction phase, TensorFlow functions are used to build the computational graph of the machine learning model. In TensorFlow, all computations are transformed into nodes on a computational graph, with edges between nodes describing dependencies between computations. TensorFlow offers essential building blocks such as fully connected layers, convolutional layers, RNN modules, and nonlinear activation functions. Constructing and adding these modules layer by layer in Python makes TensorFlow convenient to use. Additionally, TensorFlow provides various loss functions like cross-entropy and mean squared error, and adding loss computation operations to the output tensor completes the forward pass of a neural network model (Pang et al., 2019).

3-2. Keras

Keras (Ketkar & Ketkar, 2017) is an advanced neural network API written in Python, serving as an interface for effortlessly creating and training deep learning models. It provides a user-friendly, modular, and extensible framework for constructing various types of neural networks, including feedforward, convolutional, recurrent, and more. Its simplicity and abstraction make it suitable for both beginners and experienced machine learning practitioners. With Keras, users can rapidly prototype and experiment with different neural network architectures, facilitating the development of cutting-edge machine learning solutions for tasks like image recognition, natural language processing, and more. This dissertation primarily employs the Keras API on the TensorFlow platform for constructing deep learning models.

3-3. Music21

Music21 is a powerful Python library designed for computational musicology research. This library employs a fundamental data structure known as "stream" to represent the temporal sequence of music. With Music21, we can extract various musical attributes from MIDI files, including pitch, notes, chords, scales, and tonality. Furthermore, Music21 provides a suite of general-purpose functions for reading, writing, and processing musical scores, along with tools for generating music fragments and chord progressions. These capabilities enable us to effortlessly extract information from MIDI files and save music generated by neural network models as MIDI files.
One of Music21's strengths is its cross-platform compatibility and its ability to handle various music data formats. It seamlessly interfaces with MusicXML, a universal format used by major notation software, facilitating the analysis, construction, and storage of music data. In addition to MusicXML, Music21 supports kern files, MIDI files and other formats, greatly simplifying large-scale corpus analysis (Tymoczko, 2013).

In summary, Music21 offers convenience in music processing, combining adaptability with the robustness of Python. It empowers users to work with music data efficiently and effectively. In this dissertation, we primarily utilized Music21 for preprocessing MIDI-format music data.

3-4. NumPy

NumPy is a fundamental Python library for numerical computation. It supports the creation and manipulation of large, multi-dimensional arrays and matrices, and provides a wide range of mathematical functions for working with these data structures. In the fields of science and data analysis, NumPy is considered an indispensable tool, enabling efficient data processing, mathematical operations, and seamless integration with other data science libraries. One of the key features of NumPy is its N-dimensional array object, ndarray, which is designed to store data in memory, leading to faster batch operations on array elements (Idris, 2015). In this study, we will utilize the NumPy library to convert labels in the training dataset into NumPy arrays, enabling more efficient array operations and numerical computations, such as finding the maximum value in an array or performing dimension expansion.

3-5. Matplotlib

Matplotlib is a powerful Python data visualization library that has become an indispensable tool in the field of Python data science (Bisong, 2019). It is widely used for generating various types of charts and plots, including static, dynamic, and interactive visualizations. Matplotlib provides a rich set of tools and functionalities that enable users to effortlessly create high-quality graphics, such as line plots, scatter plots, and more.

The primary goal of data visualization is to gain deeper insights into data through visual representations (Telea, 2014). In the context of this research, data visualization plays a crucial role in assessing the results of music prediction. Through Matplotlib, we can intuitively display the comparison between the model's predictions and the actual values, which aids in evaluating the accuracy of the model in music prediction tasks. Furthermore, Matplotlib allows us to visualize the model's performance during the training process, including changes in accuracy and loss functions. This visualization provides valuable guidance for optimizing the model, enabling me to gain a better understanding of its performance and make necessary improvements.

4.Data Processing

For a machine learning model to make accurate predictions, a substantial amount of data is required for support. Music21 can read various formats of music data, such as MIDI and Kern, and converting melodies represented in different formats into numerical forms that can be accepted by neural networks for model training. The following steps outline the data processing:

4-1. Dataset Selection

The model struggles to comprehend subtle differences in various music styles. Including songs from different genres as part of the dataset might pose a challenge for the model, as it needs to attempt to identify commonalities among these distinct music styles, which can be quite challenging.

In this study, we chose a dataset comprising music pieces with similar styles, sourced from Kaggle. This dataset consists of over a thousand songs and possesses relatively simple and uniform melodic features. This characteristic makes the dataset highly suitable for model training, as its music features are not extremely broad but rather fall within a certain observable range.

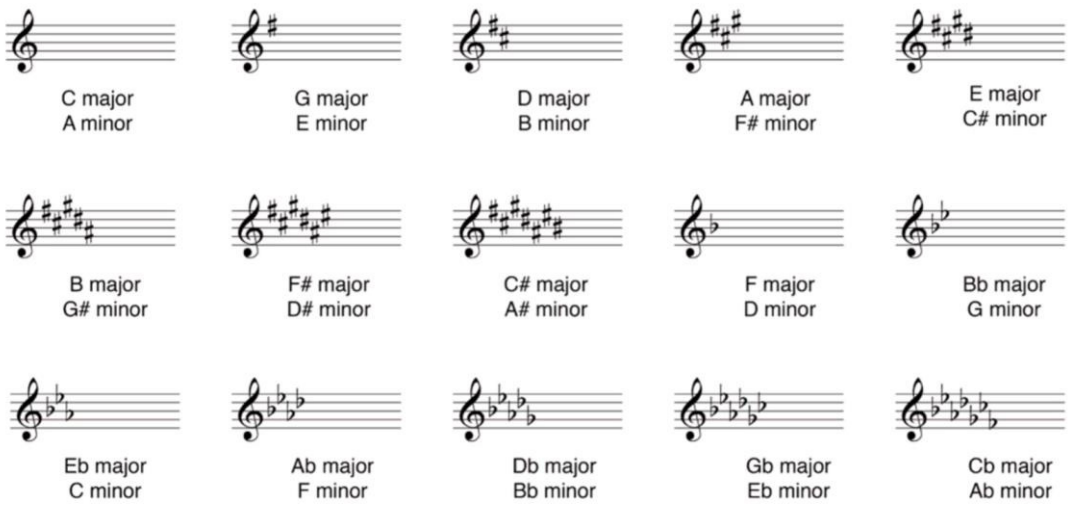
It is worth noting that, when predicting the music of a specific song, training the model on a single song, as opposed to using multiple different songs for training, may yield superior performance. This approach allows the model to focus more on a particular song, thereby gaining a deeper understanding of and making better predictions regarding its musical development and structure. In contrast, training on multiple songs may lead the model to become confused between different songs due to the distinct musical styles, emotions, and elements they encompass. However, employing a single song as a dataset may encounter issues related to insufficient data volume, potentially resulting in overfitting. Therefore, in this research, we have decided to train the model on a dataset containing multiple songs to ensure that it possesses better generalization capabilities across various music styles and elements.

4-2. Extracting Music Information from Files

Music21 provides the ‘music21.converter. parse ()’ function for reading music files in different formats. It returns a Score class instance, which typically includes a collection of instruments that perform the music. Different instruments play different parts within the same Score. Each part can contain musical elements like note, rest, and chord. These elements each contain information about pitch and duration, which need to be extracted during data preprocessing for use by the neural network model.

4-3. Standardizing Key Signatures

In music theory and notation, a key signature is a symbol used to represent the key or tonality of a musical composition. Key signatures are typically displayed in sheet music, appearing immediately after the clef sign at the beginning of a staff. In the Western music system, there are a total of 15 different key signatures:



Each key signature corresponds to both a major and a minor key, in different key signatures, the positions and relationships of individual notes change. Furthermore, it's common for the distribution of key signatures in a dataset to be uneven, with some key signatures appearing more frequently than others. This non-uniform distribution can pose challenges, as it may make it difficult for the neural network to systematically learn the distribution patterns of notes in different key signatures, potentially leading to inaccurate predictions. Therefore, handling key signatures is a crucial step in preprocessing music data.

This task is challenging because it involves processing key signatures for all songs in the dataset. One approach is to determine the key signature used in all songs by analyzing the frequency of occurrence. However, this method is error-prone and carries the risk of contaminating the dataset.

I later discovered that Music21 can read the key signature attributes within measures of a song and modify them. Given this capability, I decided to use Music21 to process key signatures in songs. The approach involves standardizing the key signatures of all songs to either C major or A minor to eliminate the influence of key signatures on the dataset. The process begins by iterating through each song, if a song's key signature is already C major or A minor, it is skipped. If the key signature of the song is not C major or A minor, or if the key signature cannot be determined, it is modified to either C major or A minor. The following pseudocode outlines this step:

```
function detectAndChangeKey(song):
    # Check if it's in C Major or A Minor
    if song.key == "C Major" or song.key == "A Minor":
        # The song is already in C Major or A Minor, no changes needed
    else:
        # Change the song to C Major or A Minor
        transpose(song)
```

4-4. Encoding Symbolic Music

Neural networks require numerical inputs, so I need an encoding method to relate musical notes to numerical values. In this study, I adopted a symbolic music encoding method suitable for neural networks, where I use MIDI pitch values to represent the pitch of notes. This way, each note corresponds to a unique MIDI pitch value.

There are a total of 128 MIDI notes, represented by the numbers 0 to 127, corresponding to the basic pitches in Western music. This allows me to map notes to numbers and use them for training the neural network. The following chart illustrates the MIDI pitch values for the C major scale:

Note Name	Note Symbol (C Major)	MIDI Note Number
C	C4	60
D	D4	62
E	E4	64
F	F4	65
G	G4	67
A	A4	69
B	B4	71
C	C5	72
D	D5	74
E	E5	76
F	F5	77
G	G5	79
A	A5	81
B	B5	83

In this dissertation, I used the music21 library to extract each individual note from every song and mapped them to their corresponding MIDI pitch values. Rests were represented using the lowercase letter "r." When the situation involves chords, it becomes more complex, and I will provide an explanation later. Next, I needed to handle the duration of notes, in most vocal compositions, note values shorter than sixteenth notes are uncommon. Therefore, in this research, I used sixteenth notes as the base unit of duration and represented note durations using "-". Each number or "-" represented the duration of a sixteenth note. For example, "60 - - - r - 65 - 62 -" signifies a sequence of a quarter note C4, an eighth note rest, an eighth note F4, and an eighth note D4. The following pseudocode outlines this step:

```
function encodeSong(song):
    for each event in song:
        If event is a Note:
            symbol = MIDI note number of the note + duration
        ElseIf event is a Rest:
            symbol = "r" + duration
```

After extracting the data representing notes and their durations from the musical composition, we needed to store this information along with their corresponding numerical values. To optimize efficiency during training, I chose to create a list to store the relationships between notes and their corresponding MIDI pitch values. This list of notes and their associated MIDI pitch values would be used during the model's training and later in the music prediction phase, where the model's output in

numerical form would be converted back to MIDI pitch values using this list and then used to generate the music using the music21 library.

4-5. Converting to One-hot Encoding

In the context of the music prediction task presented in this dissertation, the output architecture of the LSTM neural network model is connected to the Softmax activation function. The output of the Softmax activation function is structured to represent a complete probability distribution, which is utilized for the model's classification task. Each output value within a category signifies the predictive probability of the respective category. Consequently, it becomes imperative for the input labels to be presented in the form of a probability distribution, facilitating meaningful comparisons with the model's output.

One-Hot encoding provides an ideal solution for this requirement. It represents the labels of actual samples as a categorical variable with a length equal to the total number of possible elements, where only the variable corresponding to the given element has a value of 1, and all other variable values are set to 0 (Briot, 2021).This encoding method not only aligns seamlessly with the output requirements of the Softmax activation function but also enables us to employ a straightforward formula for computing cross-entropy loss, this approach significantly enhances the efficiency and intuitiveness of model training and performance evaluation. Keras provides the keras.utils.to_categorical function, which seamlessly transforms integer-encoded MIDI note values into a matrix-based one-hot encoding.

4-6. Sampling from Output Vectors

The results obtained through model predictions are in the form of a probability vector, indicating the probability of each category. At this point, it is necessary to employ a sampling method to sample from the probability distribution and select a value as the result. Experimental findings suggest that while training models based on likelihood can yield outstanding performance in language understanding tasks, decoding methods that maximize output probabilities (e.g., greedy sampling) can lead to overly repetitive or unimaginative text generation (Holtzman et al., 2019). Therefore,

introducing a degree of randomness is essential during music sequence prediction to avoid the degradation of generated results.

In this dissertation, we choose to employ temperature sampling as our sampling method. Temperature sampling involves introducing the parameter τ, which alters the probability distribution, followed by random sampling from the modified probability vector. By adjusting the temperature parameter $\tau \in [0, \infty)$, we aim to strike a balance between the authenticity and diversity of the generated results. Specifically, when $\tau < 1$, it biases the distribution towards high-probability events, effectively amplifying the significance of larger probabilities.

4-7. Some Problems in Data Processing

4-7-1. Handling Chords

A chord refers to a group of three or more notes stacked vertically at intervals of thirds or non-thirds, forming sound with specific pitch relationships (Hewitt, 2008). During the preprocessing of musical note data, encounters with chords are quite common. As chords typically consist of three or more notes, handling chords is inherently more complex compared to dealing with individual notes (Oore et al., 2018). When employing the previously mentioned encoding method for notes and their durations to process chords, we designate the root note of a chord as the first note, followed by encoding the remaining notes in sequential order. For instance, the C Major Triad, composed of C, E, and G notes, might be encoded as "60 64 67 -" using the encoding method described earlier for a quarter note. However, this encoding approach is evidently an erroneous data preprocessing method as it distorts the original intent of the musical composition. This erroneous encoding approach may lead to significant errors in music prediction by neural network models. Hence, the encoding method mentioned earlier may not be suitable for encoding chords composed of multiple notes.

I have devised three approaches to address this issue. The first approach involves traversing all the notes within a chord and randomly selecting one note to represent the chord's sound. However, this method is susceptible to issues when dealing with chords that span a wide range, as the randomly chosen to note may inadequately represent the chord, leading to significant fluctuations in the predicted musical pitch by the neural network model due to erroneous notes.

The second approach aims to mitigate the issue by selecting the central note within the chord to serve as its representative. While this approach partially addresses the challenge posed by large-span harmonic notes, it still carries limitations, including the potential for data contamination within the dataset.

Ultimately, the third approach was adopted, involving the extraction of multiple notes from a chord, and encoding them as a tuple to represent the chord. For instance, a C Major Triad comprising the notes C, E, and G is encoded as the tuple (60, 64, 67). This approach retains the chord's intrinsic meaning while minimizing information loss.

4-7-2. Handling Multi-Track Music

In multi-track music, each instrument or voice has its own staff in the musical notation, representing its independent notes and pitches. This approach allows different instruments or voices to play simultaneously, creating rich musical textures. When dealing with single-track music, data preprocessing usually involves encoding a single voice, but in the case of multi-track music, there arises a challenge. Multi-track compositions often include various instruments playing in harmony, and each instrument has its own separate staff. Representing an entire piece of music using just one staff is inadequate, as it fails to capture the full musical complexity and leads to dataset contamination, resulting in significant errors during model training and music prediction.

To address this challenge, it becomes necessary to preprocess all tracks of a music piece individually, converting each track into a numerical format suitable for neural network models, using the encoding method discussed earlier. However, each track may have distinct musical characteristics. For instance, a piano track may consist of single notes as well as chords, whereas a percussion track may predominantly feature consecutive single notes. If different tracks are fed into the same neural network for training, it may struggle to systematically learn the underlying patterns in a music piece, leading to significant fluctuations or inaccuracies in generating music predictions.

To tackle this issue, I developed a function that, if a music piece contains multiple tracks, iterates through all the tracks. It encodes the notes of each instrument track and feeds them into separate neural network models. For instance, it extracts all piano tracks from the dataset and uses them as input to an independent neural network model. After training individual models for each track, it independently predicts the music for each track. Finally, all tracks are combined to produce a complete musical composition. This approach allows each neural network model to receive data from the same type of instrument, enhancing the accuracy of music prediction. However, it does place higher demands on the dataset, as different music pieces may use varying instruments, and the musical styles across tracks can differ significantly.

5. Model Description

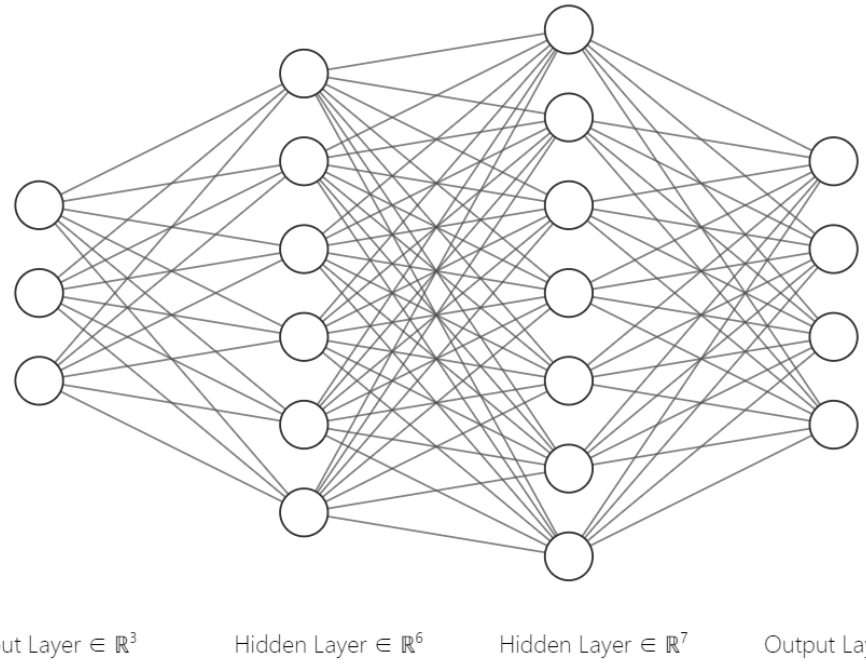
5-1. Model Architecture

To achieve accurate predictions of music trends, it is essential to select and construct an appropriate neural network model that maximally captures the temporal relationships and patterns within the music data. During the model selection process, attention will be focused on the following key factors:

- Neural Network Architecture Design: The architecture of the neural network plays a pivotal role in model performance. Two critical factors within this are the network's depth (number of layers) and width (number of neurons in each layer). Deeper networks often possess stronger representational capabilities but are susceptible to overfitting. In this study, careful consideration of the dataset's complexity will be undertaken to determine the optimal network architecture.
- Layer Selection: The choice of various neural network layers is equally vital. This includes decisions regarding the use of LSTM layers, GRU layers, or Dense layers, as well as defining the input and output of each layer. These choices will directly impact the model's performance and capabilities.
- Overfitting Mitigation Strategies: Attention will be devoted to addressing overfitting during the model construction process. In this dissertation, the inclusion of Dropout layers will be employed to prevent model overfitting.

The basic neural network model consists of an input layer, hidden layers, and an output layer. The number of neurons in each layer can be adjusted based on the specific requirements of the task. Alterations in the number of layers and nodes can lead to different outcomes. The depth of a neural network is a primary factor influencing its expressive capacity (Huang et al., 2016). Deeper neural networks are capable of fitting more complex data. However, due to the presence of overfitting, there is not a strictly positive correlation between the depth of the network, the number of neurons, and prediction accuracy. To obtain an appropriate model, it is necessary to arrive at conclusions through repeated experiments.

Below is a schematic diagram of a simple neural network comprising a 3-node input layer, two hidden layers, and a 4-node output layer:



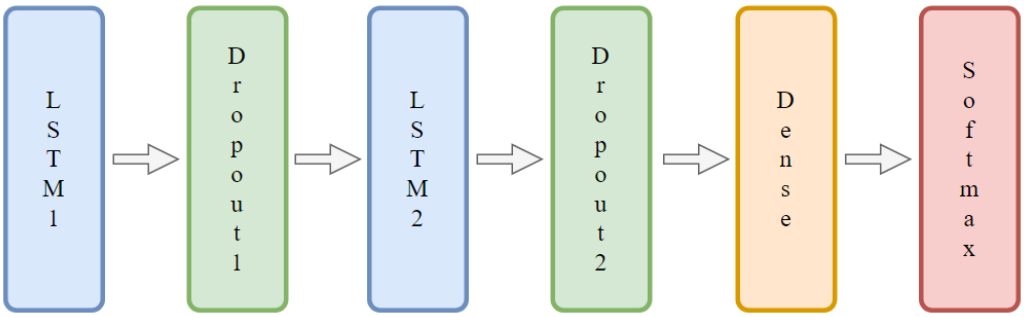
The research conducted in this dissertation pertains to the prediction of music temporal sequences. RNN and their variants, particularly LSTM networks, have emerged as cutting-edge models for handling sequential tasks (Vaswani et al., 2017). The selection of the LSTM model for music prediction tasks is contingent on its outstanding performance in dealing with complex music sequences.

Firstly, music compositions typically exhibit a multi-level structure, encompassing themes, harmonies, melodies, and recurring patterns. The long-term memory mechanism within LSTM networks enables them to capture these long-range dependencies in music sequences. Furthermore, their gate mechanisms effectively mitigate the vanishing gradient problem, aiding in understanding the overall structure of music compositions and thereby enhancing accuracy in predicting future notes.

Secondly, temporal relationships are crucial in music, such as the durations between notes, the way chords are played, and the coordination between instruments. LSTM networks are sensitive to

temporal relationships, making them well-suited for handling temporal information within music sequences.

In summary, employing an LSTM model for music prediction is an excellent choice. The LSTM model constructed in this dissertation consists of two LSTM layers, two Dropout layers, one Dense layer, and a Softmax activation function, as illustrated in the diagram below:



In this model, the LSTM layer is characterized by three parameters: input_shape, which is used to specify the length and features of the input sequence. In this dissertation, the model takes one-hot encoding as input. Units specify the number of neurons in a single layer, and return_sequences is set to True to return the entire sequence rather than just the output of the last time step. The Dropout layer (Srivastava et al., 2014) is a commonly employed regularization technique in neural networks. It mitigates overfitting risk by concealing the output of a fraction of neurons during model training. In this model, the Dropout layer parameter is set to 0.4 to mask 40% of the neurons. The final layer of this model consists of a fully connected layer, with the 'units' parameter set to the number of distinct musical notes at the input layer. Its purpose is to map the output from the LSTM layers to the ultimate musical note predictions.

5-2. Parameter Selection

Parameter selection is a crucial factor influencing the predictive capability of the model. The following are several key parameters:

- Learning Rate: An appropriate learning rate controls the step size of model parameter updates at each iteration, ensuring a smooth and efficient training process. Some optimizers feature adaptive learning rates.

- Activation Function: Activation functions introduce nonlinearity into neural networks, enabling them to learn and represent complex nonlinear relationships. Generally, Softmax is a common choice.
- Loss Function: The loss function is used to measure the model's performance by quantifying the gap between predicted values and actual values.
- Optimizer: The optimizer determines the strategy for updating model parameters, directly impacting the speed and stability of the training process. In this study, Adam is a well-suited choice.
- Batch Size: An appropriate batch size not only affects the training speed but also relates to model performance. The selection of batch size should consider factors such as computational resources and the distribution of training data to achieve the best training results.

5-2-1. Activation Function

In this study, the Softmax activation function was chosen for the output layer to determine the most probable answer for classification problems. It takes the output from the Dense layer and maps it to a set of probability values, representing the prediction probabilities for each note. The note with the highest probability is then selected as the output. The formula is as follows:

$$P_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_i}}$$

$$y = \arg\max P_i$$

5-2-2. Loss Function

The loss function serves as the objective function for model optimization, guiding the direction of model parameter updates during training (Hennig & Mahmut, 2007). Sparse categorical cross-entropy loss function was utilized in this study. This variant of the cross-entropy loss function considers sparsity between categories when calculating the loss. Traditional cross-entropy loss treats all categories equally, while sparse categorical cross-entropy applies weights to sparse categories, suitable for cases where labels are integer-encoded. The loss function is computed as follows:

$$L = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

Where y_i represents the true labels, and $\hat{y}_i$ represents the predicted labels.

5-2-3. Optimizer

The Adam optimizer (Kingma & Ba, 2014) was employed in this research, which is an efficient gradient descent algorithm. Adam combines the advantages of AdaGrad, suitable for sparse gradients, and RMSProp, effective for online learning. Adam is characterized by low memory usage and adaptive learning rates. It incorporates the concept of momentum, accelerating convergence by maintaining first-order moment estimates (mean) and second-order moment estimates (square of the mean) of gradients. The momentum term helps in traversing flat regions in the parameter space, reducing the risk of getting trapped in local minima. Adam's default parameters generally perform well across various deep learning tasks, simplifying hyperparameter tuning.

This study also considered using the RMSProp optimizer with time-varying adaptive learning rates. It shares similarities with the AdaGrad optimizer but replaces the sum of square gradients with moving averages of square gradients, reducing gradient scaling. Although Adam and RMSProp often exhibit similar performance (Haji & Adnan, 2021), Adam was ultimately chosen as the optimizer for the music prediction task after experimental comparisons.

5-2-4. Batch Size

In this study, due to the large volume of the music dataset, it is necessary to train the model by feeding the data in batches to improve training efficiency and reduce computational costs. To enhance the model's generalization ability, it is advisable to limit the ratio of batch size to learning rate as much as possible (He et al., 2019). Therefore, in this research, the batch size during model training is set to 64, meaning that 64 samples are specified for updating parameters each time the model is trained.

6.Implementation

In this chapter, we will delve into the specific implementation details of music prediction, including the introduction of relevant libraries, data preprocessing and preparation, the construction and training of neural network models, and the saving of prediction results. We will gradually introduce the necessary code and methods to ensure that the entire implementation process is clear and understandable and can be easily reproduced when needed.

6-1. Importing Libraries and Data Reading

In this study, we utilized NumPy for data format operations and numerical computations, the matplotlib library for data visualization, the music21 library as a tool for processing music data, and the pickle library for file read and write operations. The following code shows the importation of these libraries:

```
import pickle
import numpy as np
import music21 as m21
import matplotlib.pyplot as plt
```

With the functions provided by the music21 library, we can easily read music data from a folder and transform it into instances of the Score class. The following code serves as an example of reading the dataset:

```
song = m21.converter.parse(file)
```

6-2. Data Analysis and Data Processing

After obtaining information from the music dataset, it was essential to conduct data analysis and processing. Initially, we needed to identify the different tracks and instruments used in the dataset:

```
parts = m21.instrument.partitionByInstrument(song)
for i in range(len(parts.parts)):
    instrument_name = parts.parts[i].getInstrument().instrumentName
```

Before encoding the MIDI pitches of the music work, it was necessary to parse the tonality information of the music work and transpose the music objects relative to the key of C to facilitate further music analysis and processing, as shown below:

```
for element in score.recurse():
    # find the key
    if isinstance(element, m21.key.Key):
        # find the tonic
        if element.mode == 'major':
            tonic = element.tonic
        else:
            tonic = element.parallel.tonic
        # Transpose according to the main tonic
        gap = m21.interval.Interval(tonic, m21.pitch.Pitch('C'))
        score = score.transpose(gap)
        break
    # can not find the key
    elif isinstance(element, m21.note.Note) or
         isinstance(element, m21.note.Rest) or
         isinstance(element, m21.chord.Chord):
        break
    else:
        continue
```

Next, we iterated through each instrument, encoding the notes into elements based on sixteenth notes as the fundamental unit. Chords were encoded as tuples, notes as corresponding MIDI pitches, and rests as 'r', with '-' indicating note duration, as demonstrated here:

```
for event in part.flat.notesAndRests:
    #handle chord
    if isinstance(event, m21.chord.Chord):
        symbol = tuple(event.normalOrder)
```

```
    #handle note
    elif isinstance(event, m21.note.Note):
        symbol = str(event.pitch.midi)
    #handle rest
    elif isinstance(event, m21.note.Rest):
        symbol = "r"
    string_part.append(symbol)
    duration = event.duration.quarterLength * 4
    while(duration > 1):
        string_part.append("-")
        duration -= 1
```

6-3. Using LSTM Model for Prediction

To perform music generation, it was crucial to prepare training data for input to the neural network. In music generation tasks, we utilized the first 65 musical symbols of a sequence to predict the 65th musical symbol. Consequently, we constructed training examples, each comprising a fixed-length music sequence of 65 elements and the corresponding prediction target, as shown below:

```
x = []
y = []
num_sequences = len(int_notes) - SEQUENCE_LENGTH
for i in range(num_sequences):
    x.append(int_notes[i:i+SEQUENCE_LENGTH])
    y.append(int_notes[i+SEQUENCE_LENGTH])
#  encoding sequence to one-hot
x = keras.utils.to_categorical(x, num_classes=unique_notes_num)
y = np.array(y)
print(f"There are {len(x)} sequences.")
```

After preparing the training data, we input it into an LSTM model for training. First, we imported the necessary libraries for model construction. In this study, keras served as the primary tool for building the model, as illustrated below:

```
import tensorflow.keras as keras
from keras.models import Sequential
from keras.layers import Activation, Dense, LSTM, Dropout
from sklearn.model_selection import train_test_split
```

Before feeding the training data into the model, we partitioned 10% of the data as a test set to evaluate the performance of the trained model on unseen data. Additionally, 10% of the training data was allocated as a validation set to detect potential overfitting or underfitting issues. This process is demonstrated as follows:

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1,
  random_state=42)
```

Subsequently, we constructed an LSTM model and fine-tuned the parameter values as needed:

```
model = Sequential()
model.add(LSTM(128, input_shape=(None, unique_notes_num),
  return_sequences=True))
model.add(Dropout(0.4))
model.add(LSTM(128))
model.add(Dropout(0.4))
model.add(Dense(unique_notes_num, activation='softmax'))
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam',
  metrics=['accuracy'])
```

Finally, the training data was input into the model for training, and training results were obtained:

```
history = model.fit(x_train, y_train, epochs = 150, verbose = 1,
          batch_size = 64, validation_split = 0.1)
```

6-4. Generating Music Files

After training the model and achieving the desired performance, the next step was to use the model for music composition. Firstly, we extracted the last 64 musical symbols of the music piece we wanted to predict, transformed them into one-hot format suitable for input into the neural network, and then used the model to generate a probability vector based on the input sequence. Finally, we applied temperature sampling to generate a sequence of notes based on the model's output probabilities. This process was repeated until we generated the desired length of the music sequence. The following code demonstrates this:

```
melody = encode_part(part)[-64:]
# map seed to int
seed = [note_to_int[index] for index in melody]

for _ in range(300):
    # encoding the seed to one-hot
    one_hot_seed = keras.utils.to_categorical(seed,
        num_classes=len(note_to_int))
    one_hot_seed = one_hot_seed[np.newaxis, ...]
    # make a prediction
    probabilities = model.predict(one_hot_seed)[0]
    # [0.1, 0.2, 0.1, 0.6] -> 1
    prediction = np.log(probabilities) / 0.6
    probabilities = np.exp(prediction) / np.sum(np.exp(prediction))
    #choices = range(len(probabilities)) # [0, 1, 2, 3]
    output_int = np.random.choice(range(len(probabilities)),
        p = probabilities)
    # update seed
    seed.append(output_int)
    # map int to our encoding
    output_symbol = [k for k, v in note_to_int.items() if v == output_int][0]
    # update melody
```

```
    melody.append(output_symbol)
```

We needed to decode the generated music sequence into symbolic music data. This involved handling notes, chords, and rests, and combining them into a musical stream. Specifically, while processing the music sequence, we iterated through each element. When we encountered MIDI pitch or a rest, we created a new musical event and recorded its duration. For rests, we handled them as well. If the symbol was "-", we incremented a step counter to indicate a longer duration. Here's the code illustrating this process:

```
for i, symbol in enumerate(melody):
    # check if the symbol is "-" or if it's the last symbol in the melody.
    if symbol != "-" or i + 1 == len(melody):
        # if there's a previously encountered symbol, create a musical event.
        if start_symbol is not None:
            quarter_length_duration = 0.25 * step_counter
            # handle rest
            if start_symbol == "r":
               m21_event = m21.note.Rest(quarterLength=quarter_length_duration)
            # handle chord
            elif isinstance(start_symbol, tuple):
                m21_event = m21.chord.Chord(list(start_symbol),
                      quarterLength=quarter_length_duration)
            #handle note
            else:
                m21_event = m21.note.Note(int(start_symbol),
                      quarterLength=quarter_length_duration)
            unique_part.append(m21_event)
            # reset the step counter for the next sequence of symbols.
            step_counter = 1
        start_symbol = symbol
    # if symbol is -, increment the step counter to indicate a longer duration.
```

```
    else:
        step_counter += 1
```

Finally, we wrote the music sequence into a stream using the music21 library and saved the music file locally.

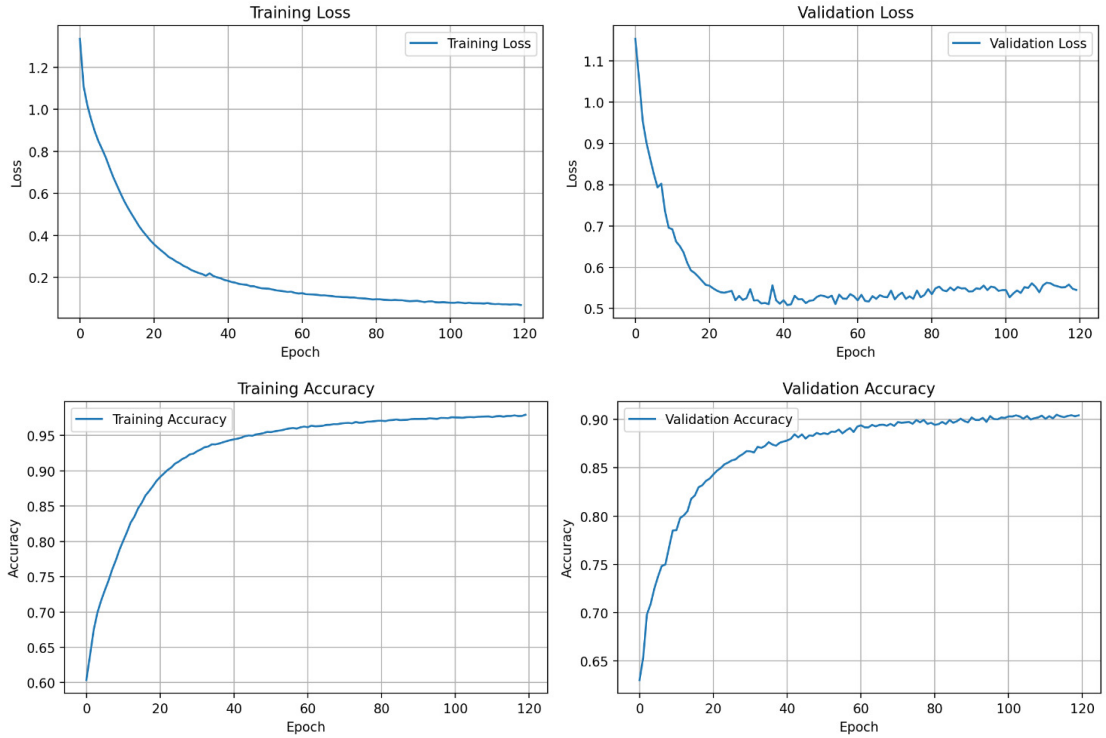
7. Results

7-1. Model Assessment

For the effective evaluation of music generation results, it not only aids in identifying the shortcomings of music generation models but also contributes to the continual improvement of the model to achieve superior music generation performance. However, evaluation in fields such as music, literature, and art often involve subjectivity, making it challenging to objectively assess the outcomes of these creative endeavors (Yamshchikov & Tikhonov, 2020). When it comes to performance evaluation of music prediction models, unlike tasks with well-defined criteria such as classification, the evaluation of music prediction models typically requires the consideration of subjective factors. While subjective assessment is more fitting in the domain of artistic creation, it presents issues with cumbersome evaluation methods and result instability. Conversely, objective evaluation metrics have distinct advantages in some cases as they provide a more reliable and actionable way to assess model performance. These objective metrics often involve quantitative analyses of model outputs, such as the structure of generated music, the coherence of notes, and the diversity of melodies. However, objective evaluation may, in some respects, become overly theoretical and fail to fully capture the complexities of the music domain.

Despite significant advancements in automated music evaluation techniques, there are still some limitations (Agarwal et al., 2018). Therefore, while considering the complexity of model evaluation and the reliability of evaluation results, this dissertation will primarily focus on objectively assessing the performance of music generation models from a mathematical perspective. This approach will combine the analysis of objective metrics with experimental results from subjective evaluations to gain a comprehensive understanding of the model's performance in various aspects, thereby providing better guidance for model refinement and optimization directions.

First, we will discuss the model's loss function values, accuracy on the training set, and loss function values and accuracy on the validation set, as shown in the following figure:



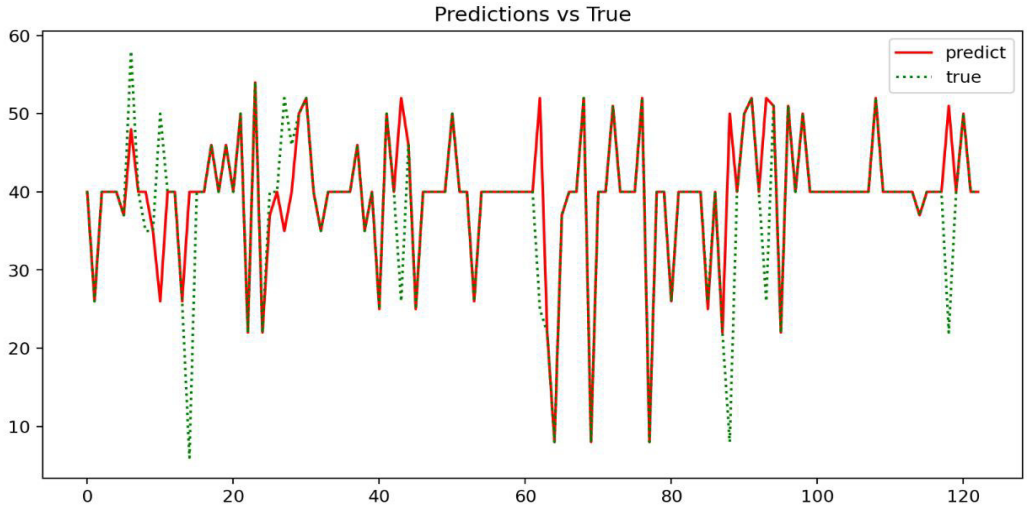
From the figure above, it can be observed that after undergoing continuous training for 3 hours, the model achieved a satisfactory accuracy of approximately 95% on the training dataset. However, its performance on the validation dataset was slightly lower, reaching only approximately 90% accuracy, and the loss function exhibited fluctuations towards the end. This observation has prompted consideration of two potential factors:

Firstly, it is possible that the model is experiencing overfitting, wherein it excessively adapts to specific features of the training data during the training process, leading to a reduced ability to generalize to new data. This becomes particularly evident when dealing with independent data, such as the validation set.

Secondly, our dataset comprises many mutually independent musical compositions, each possessing its unique musical style and characteristics. This diversity may pose challenges for the model when making predictions on the validation set, as it may contain music compositions of different styles not encountered during training. These variations in musical style could potentially have a negative impact on the model's predictive ability, as the model needs to adapt to different musical styles and melodic patterns. Increasing the size of the dataset may help alleviate this issue, as a smaller dataset

may not adequately capture the diversity and complexity of the music domain, thereby limiting the model's performance.

Next, we will assess the performance of the trained model using the test dataset, with the results presented below:



It can be observed that the performance of the model on the test dataset is generally satisfactory. It can make predictions to a certain extent on music compositions with similar musical styles that it has not previously encountered. This result indicates the reasonable effectiveness of the hyperparameters set before model training. Furthermore, through validation on the test dataset, we have further confirmed the model's generalization ability, i.e., its performance on new data. Overall, the model is well-constructed, and further evaluation of its performance can be considered.

7-2. Score Evaluation

For the evaluation of music prediction models, relying solely on results from the test set is evidently insufficient. The test set may encompass works of various musical styles, and an outstanding music prediction model should possess not only the ability to predict notes but also a certain level of musical theory knowledge, enabling it to comprehend the relationships between notes and musical rules. Ideally, as the model continues to train, the generated music should progressively approach real music, rather than merely replicating input data mechanically. Therefore, we will further validate and assess the model in this study. Generally, music possesses a certain logical structure, and predictions for a song should make sense rather than being random and devoid of logic. We expect the model's predictions to align with the inherent structure of the music itself. For example, when predicting for a song that features the repetition of a single note, the ideal prediction should also include the repetition of that single note. Similarly, when predicting ascending musical notes in a stepwise fashion, the results should continue the upward trend, which is in line with logical expectations. Consequently, in the subsequent tests, we will use music scores with specific patterns to evaluate the model's performance. This testing approach will provide a more in-depth assessment of the model's capability in music generation, verifying whether the model genuinely comprehends the underlying principles and theories of music.

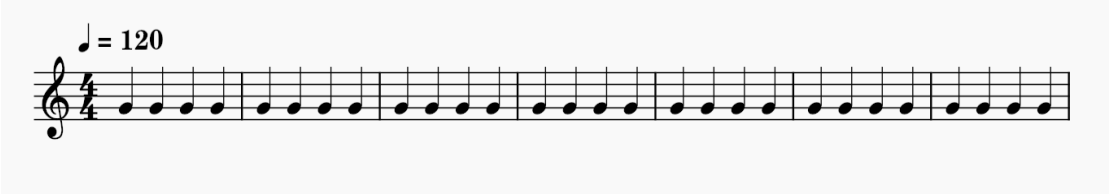
When evaluating the model's prediction ability, we have opted for a greedy sampling approach over temperature sampling for the obtained probability distribution. The primary objective behind this decision is to minimize the uncertainty introduced when the model samples probabilities as much as possible. Greedy sampling leans toward selecting predictions with the highest probabilities, thereby reducing the impact of stochastic factors in model predictions, decreasing output variability, and obtaining more stable and reliable results. This contributes to a clearer evaluation of the model's performance in music generation tasks, mitigating result fluctuations caused by randomness and enabling us to more accurately judge whether the model's predictions align with expectations.

However, during model training, as the model is not constrained to receive sequences of fixed lengths, when using shorter sequences to request predictions from the model, it takes that segment as

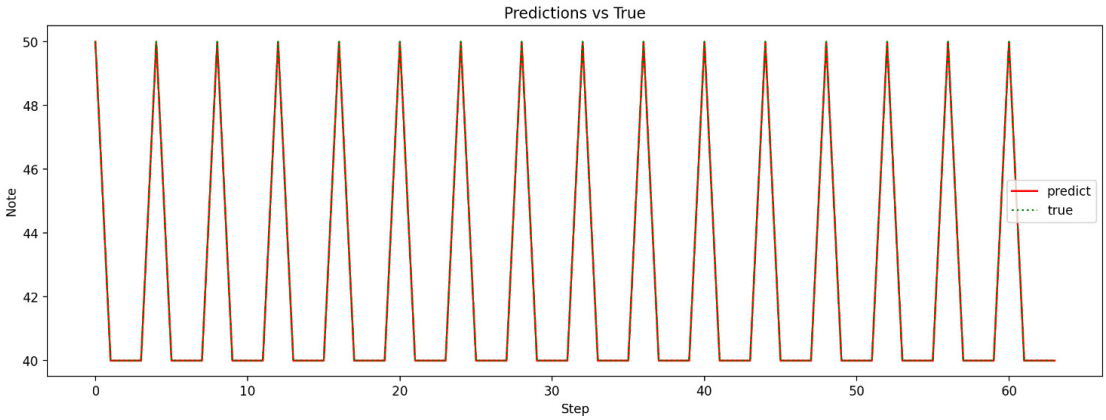
context for predicting the next outcome. Consequently, employing sequences of different lengths to predict music for the same song may yield different prediction outcomes. To ensure result consistency and comparability, in subsequent testing, we will employ fixed-length sequences to predict subsequent notes, guaranteeing that we can more accurately compare prediction results in different contexts when evaluating model performance.

In this research, four metrics will be used to evaluate the model's performance: R-squared, BLEU (Bilingual Evaluation Understudy), accuracy, and F1 score. R-squared and BLEU are employed to gauge the fitting between the model's prediction output and actual observed data, while the F1 score measures the model's performance in multi-class classification tasks.

As shown in the figure below, the presented musical score belongs to the most basic form of music notation, utilizing a 4/4-time signature, which means each measure consists of four beats, with each beat represented by a quarter note. This musical passage consists of seven consecutive measures, totaling 28 quarter notes of G4. The duration, pitch, and timing intervals between each note are consistent. However, it is essential to note that since this musical passage comprises 28 consecutive G4 quarter notes in the key of C major, it is highly likely that it is not included in the training dataset. In other words, the model's predictions for the subsequent notes in this passage rely entirely on the model's learned music patterns from the training dataset, rather than simple copying or memorization of musical patterns in the training data.



For this test, the model should be able to predict the progression of subsequent notes effortlessly. From a music theory perspective, any model should have the capability to learn the potential pattern of consecutive occurrences of the same note. If the model predicts notes different from G4 in this test, it suggests that the model has not well understood the inherent relationships between notes. This could be due to an insufficiently large dataset, a lack of diverse music patterns in the dataset, or inherent flaws in the model itself.



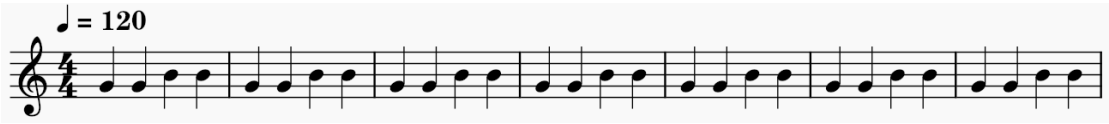
As shown in the figure above, the model demonstrates the ability to predict consecutive repeated notes, and the prediction metrics are presented in the table below:

R- squared	BLEU	Accuracy	F1 score
1.0	1.0	1.0	1.0

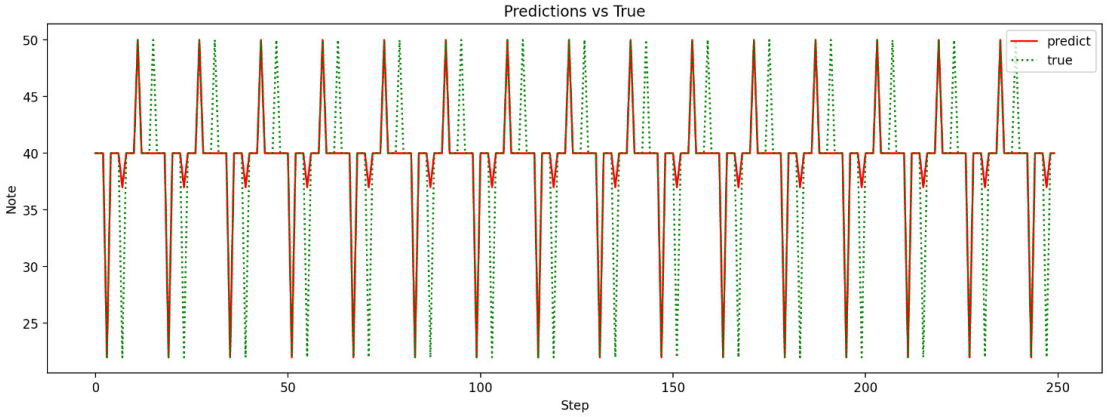
Based on the data provided in the table, we can conclude that the model exhibits a fundamental ability to handle repeated notes. This indicates that the model has at least learned to recognize and predict one of the simplest music sequence patterns, namely, the repetition of a single note. However, it is essential to note that this test scenario represents the model's performance when dealing with the simplest and highly repetitive musical segments. In practical music composition and generation, an excessive repetition of single notes is not desirable.

In the upcoming evaluation of the model's predictive capabilities, we will consider musical passages of different styles and forms to gain a comprehensive understanding of the model's potential in predicting unknown melodies. This will test the model's generalization abilities across diverse musical elements and creative styles.

As shown in the figure below, this musical score is relatively more complex compared to the previous one, still employing a 4/4-time signature. This musical passage consists of two types of notes, namely G4 and B4, which appear alternately within one measure, as illustrated below. This alternating note pattern may impose higher demands on the model's music sequence modeling because the model needs to learn how to accurately capture the alternating relationships between different notes for precise music generation.



We expect the model not only to accurately capture the pitch of the alternating notes in the music but also to precisely identify the duration of the notes. The figure below displays the model's performance in this regard:



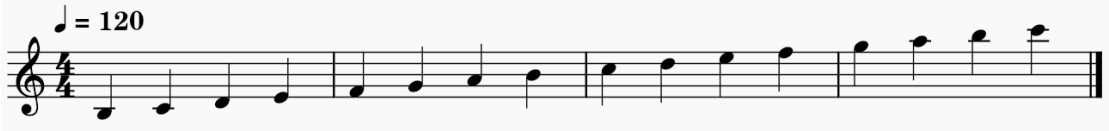
From the observation of the figure, it can be noted that when there are fluctuations in the ground truth curve, the prediction curve exhibits a similar trend of changes. Although in some fluctuation cases, they are not entirely identical, this still indicates that the model has successfully learned the rhythmic patterns of the music to some extent. However, concerning the pitch of the notes, the model's learning performance appears slightly inadequate. This suggests that the model may face some challenges in capturing pitch and requires further training to enhance the accuracy of its pitch predictions.

R- squared	BLEU	Accuracy	F1 score
0.617	0.674	0.876	0.574

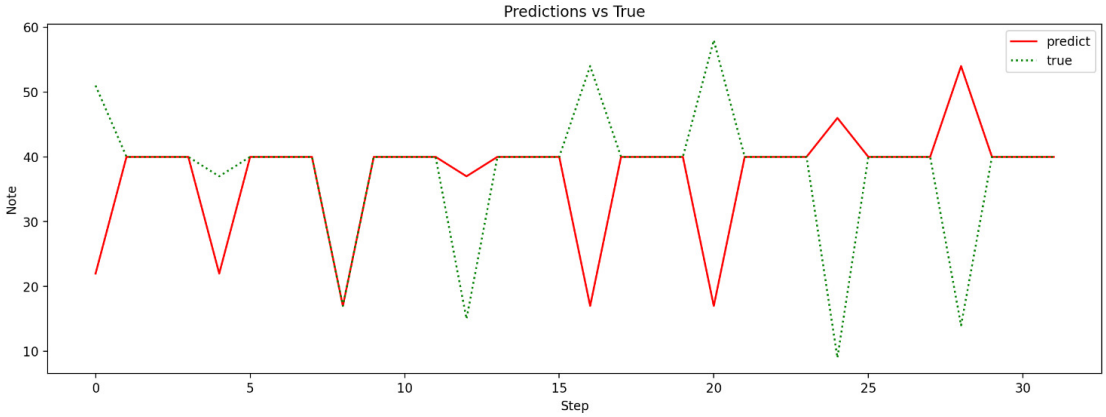
From the provided table data, it is evident that there is a clear correlation between the predicted values by the model and the actual values of the song. This indicates that the model has indeed

learned some properties related to the musical rhythm rather than making meaningless random predictions.

To test the model's level of understanding of music theory, the following musical segments were designed for testing:



As observed, this musical segment is entirely composed of gradually ascending single notes. For someone with a moderate understanding of music theory, predicting the next note should be a relatively simple task. However, for our LSTM model in this research, it may pose a challenging task. This is because the model focuses on the absolute position of notes rather than the relative position between notes. Specifically, the model's input is the MIDI pitch of the notes, not the pitch change values between notes. The comparative chart of predictions is shown below:



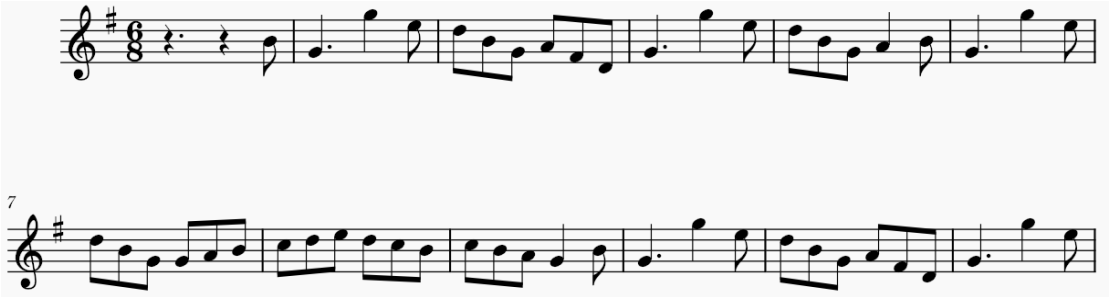
From the chart, it is evident that the model can hardly predict the direction of the next note accurately. This is primarily because the model is constructed based on the absolute values of pitch and does not consider the trend of pitch changes between notes. If we were to feed the model with a sequence of pitch change values as input, it might be easier for the model to accurately predict the pitch direction ahead.

R- squared	BLEU	Accuracy	F1 score
0.137	0.591	0.681·	0.136

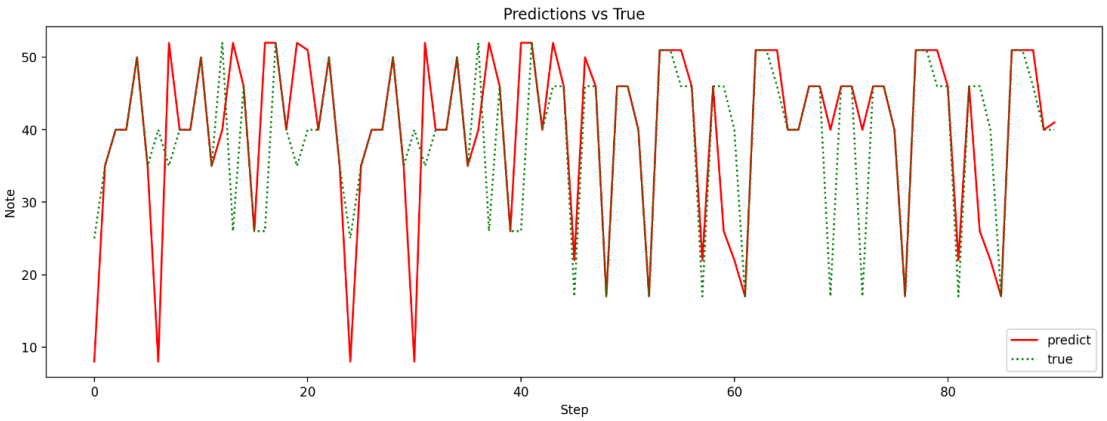
Based on the data in the table, we can conclude that the model has certain limitations in learning music theory. Its predictions for musical segments mainly rely on the time series of note occurrences in the training dataset without a deep understanding of the fundamental principles of music. Using more complex encodings for notes to encompass more essential information might help overcome the model's limitation in understanding music theory. For instance, training the LSTM model with

music transcriptions expressed in advanced notation like ABC symbols has shown promising results (Sturm et al., 2016).

In the previous model validation tests, we primarily focused on the model's performance when dealing with structured sequences of notes. However, for a more comprehensive assessment of the model's generalization capability, we need to consider a wider range of musical contexts. Therefore, the following validation tests will be conducted using segments from musical compositions to simulate more diverse musical scenarios. In this test, a completely new musical segment was used, which had not appeared in our training data. However, compared to the musical works in the training set, this musical segment shares a similar musical style and characteristics. Our expectation is that the model, when faced with such a similar style of music, can generate predictions with a certain degree of regularity and musicality. This testing scenario is crucial for evaluating the model's potential and adaptability in real music composition scenarios, as it better simulates the model's performance in real music composition situations.



When making predictions for this musical segment, if the model has successfully learned and mastered the patterns and rules of this specific music style, it is expected to apply the musical features and style it learned from the training data to produce a music segment like the training set. This also indicates the model's potential in learning and understanding music styles and the possibility of making reasonable predictions for unknown musical segments. Therefore, this test will assess the model's generalization capability when dealing with music scenarios of similar styles but different specific content. Below is a partial comparison chart of predicted results and actual results:

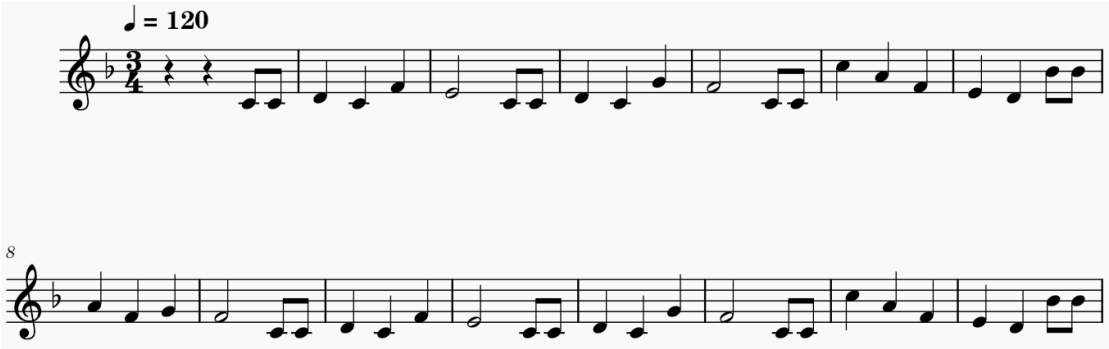


By comparing the curves of predicted values and actual values, we can observe a certain degree of similarity between them. Although the curves are not entirely identical, their overall trends are quite similar, indicating a degree of consistency between them, which means that the model can capture some important features and trends in the musical segment. The various prediction metrics are presented in the table below:

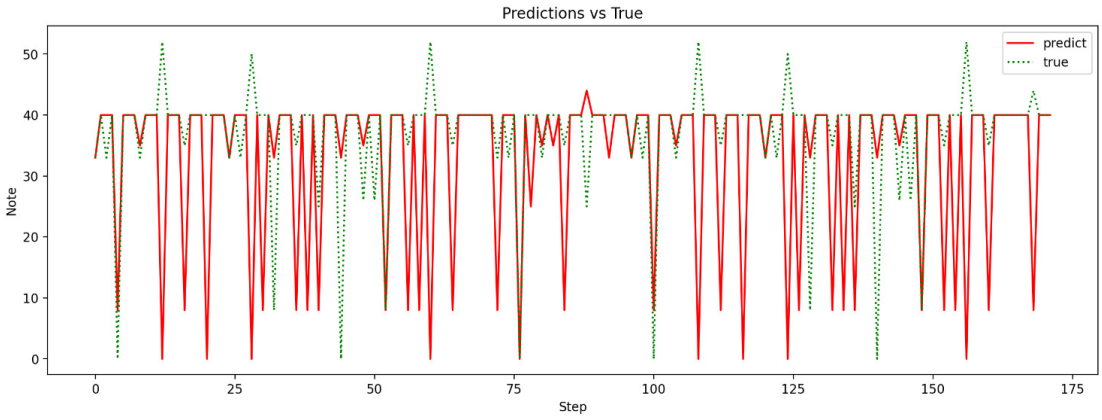
R- squared	BLEU	Accuracy	F1 score
0.214	0.727	0.797	0.477

Considering both the comparison curves and the data in the table, while the model achieves an accuracy of approximately 80% for this musical segment, the R-squared value is only 0.214. This discrepancy may be due to the model capturing relationships related to note duration but falling short in predicting pitch accurately. This observation is also supported by the BLEU score, which, while relatively high, suggests that the model's predictions are correlated with actual values but lack precision in predicting individual pitches.

The music segment tested in this study is a small part of "Happy Birthday", which is a piece of music does not present in the training set and possesses a completely independent style. This music segment incorporates various types of notes, including eighth notes, quarter notes, and half notes, forming a melody with relatively low note complexity, making it an ideal choice for validation testing. Below is the musical notation for this segment:



Different music genres typically exhibit differences in various aspects, such as rhythm complexity, note density, and note interval variations, as seen between jazz and rock music. Given this consideration, expecting the model to predict accurately is almost unrealistic. However, we expect the model to capture some patterns related to rhythm or pitch in the music. The figure below illustrates the comparison between the model's predictions and the actual values:



Based on the observed comparison, it is evident that the model's performance in accurately predicting pitch is relatively poor. However, it is worth noting that the model appears to have learned certain rhythmic pattern variations in the music. Specifically, it can distinguish differences between eighth notes and quarter notes. The inaccuracy in note prediction may be attributed, in part, to the relatively small size of the dataset. Increasing the size and complexity of the dataset could introduce more diverse music patterns, potentially improving the model's pitch prediction performance.

R- squared	BLEU	Accuracy	F1 score
0.037	0.628	0.649	0.158

Based on the data presented in the table, it can be concluded that the model's performance in this music segment is relatively poor. Its primary success lies in learning rhythmic features in the music, but it falls short in accurately predicting pitch changes. Moreover, a significant contribution to prediction accuracy comes from the prediction of note durations.

7-3. Results Analysis

In this study, the model's capabilities in music prediction were evaluated by selecting specific music segments. While subjective judgment inevitably played a role in the selection of music samples, the carefully chosen segments aimed to ensure their representativeness, covering various key aspects of music, including melody, music theory, and pitch accuracy. Through the first two experiments, we observed that the model demonstrated a certain learning ability when dealing with simple melodies. However, instances of inaccurate pitch prediction during the experiments indicated that the model has limitations when faced with complex pitch prediction tasks. Furthermore, the model performed poorly in tests related to music theory and in tests involving unknown music styles. However, it still showed some potential in predicting music for similar styles. From the validation experiments conducted, we draw the following insightful conclusions:

Firstly, when predicting ascending musical notes, the model exhibited more imitation behavior and failed to fully grasp the core principles of music theory. Its responses were more akin to simulating patterns from the training dataset rather than deeply comprehending the music theory behind the dataset.

Secondly, the model exhibited limitations in predicting pitches that were not present in the training dataset. This limitation was evident when predicting ascending notes as well. Specifically, when tasked with predicting notes that had never appeared in the training dataset, the model struggled to provide accurate predictions. This phenomenon may be related to the encoding of pitch, as the model heavily relies on existing data during training and lacks sufficient information for novel pitches.

Lastly, we observed that the model's ability to grasp music rhythm clearly surpassed its understanding of pitch. While the model exhibited relatively accurate rhythmic predictions, there was a noticeable gap in pitch prediction. This disparity may also be related to the chosen pitch encoding method, particularly when dealing with a high occurrence of quarter notes or half notes, where frequent use of '-' symbols may have led the model to learn some inaccurate patterns.

8. Conclusion and Future Work

This chapter takes the experimental results as a starting point to discuss the challenges encountered during the research and the conclusions drawn from them. Additionally, we will explore potential directions for future research and provide recommendations for further work.

8-1. Conclusion

The objective of this study was to predict the future trends in music using neural network models. After considering the requirements of the task for time series processing and the overall complexity of implementation, we opted for a music prediction model based on LSTM networks. The core task of this model was to predict the pitch and duration of the next note by extracting the temporal relationships of notes from MIDI music files. Through a series of experiments and evaluation using specific musical scores, we have arrived at some key conclusions.

Firstly, we observed that the model performed reasonably well in predicting repeated notes, achieving an accuracy of 79.7% when predicting music in the same style. This finding indicates that the model successfully captured musical patterns within these segments. It suggests that deep learning models exhibit a certain level of learning capability when dealing with melodic patterns of the same musical style. Furthermore, the model demonstrated relatively accurate predictions when dealing with known music styles, highlighting the influence of the dataset on model performance. With the continuous expansion of available datasets, it is expected that the model's performance will further improve. However, it is crucial to carefully balance dataset size and model fit, as excessively large datasets may lead to underfitting and decreased accuracy in music prediction.

On the other hand, the model's performance appeared to be less satisfactory when faced with different music styles. Although this does not invalidate the experimental results, it also underscores certain limitations inherent to the model. Experimental results indicated that when the model encountered music works of unknown styles, it could only make relatively accurate predictions of note duration while exhibiting significant errors in pitch predictions. This phenomenon may stem from the diversity of music styles and some inherent limitations of the model. Specifically, the model

has limitations in dealing with music theory and core melodies, making it challenging to accurately predict musical performance in these scenarios.

In summary, this study provides valuable insights into the field of music prediction. While the model demonstrated good learning ability in specific contexts, the results suggest that it lacks the ability to learn music theory.

8-2. Future Work

This study has conducted a series of experiments and validations using neural networks in the domain of music prediction. While the model has shown certain issues in music prediction, I firmly believe that employing neural networks for note prediction holds substantial research potential. Nonetheless, there are still various unexplored directions and areas for improvement in the current stage. Future research can expand in the following aspects:

- Dataset and Encoding: The current encoding methods employed for music data exhibit certain limitations in terms of efficiency. It may be necessary to consider the introduction of a more efficient encoding method in the future, especially when dealing with larger-scale music datasets. Additionally, the current dataset being utilized has certain limitations in terms of diversity in music styles, making it difficult to encompass a sufficiently rich range of musical styles. Therefore, in future efforts aimed at enhancing music prediction models, a dual focus on improving both dataset quality and quantity becomes imperative.
- Music Generation Patterns: Another critical consideration is that machine learning methods often imitate existing musical patterns, leading to a tendency for music generated by neural networks to exhibit monotonous rhythms. During long-term music prediction, without guidance from external variables, generated music can easily fall into meaningless repetitive loops. To enhance future music prediction models, one feasible approach is to attempt to extract the intrinsic emotions or sentiments within the music, such as identifying whether the music is cheerful, melancholic, or in another emotional state. Different emotional states in music may require distinct rhythms, tones, and harmonic patterns, thus potentially improving the accuracy of music prediction. Additionally, enhancing interaction with humans during the music generation process is an avenue worth exploring. Research indicates that incorporating AI-human interaction into music generation significantly enhances the quality of generated music, aligning it more closely with people's expectations (Huang et al., 2020). Specifically, elevating interactivity during music prediction allows for human intervention to provide guidance and direction to the model, promptly correcting any erroneous developmental directions, thereby markedly enhancing the accuracy of music prediction. The introduction of such interactivity

renders music prediction more flexible and precise, enabling better collaboration between individuals and the model to collectively create high-quality music.

- Music Generation Evaluation Methods: An appropriate evaluation method is crucial in the field of music generation. However, it is worth noting that using different evaluation criteria can yield vastly different results (Theis et al., 2015). The evaluation of the same musical composition often produces significantly different outcomes under different standards. Currently employed methods for evaluating music generation models typically include both subjective and objective aspects, with the two often complementing each other in the field of music. However, objective evaluation methods typically only involve comparing the generated music with actual music results, without giving due consideration to music theory or aesthetic factors. Meanwhile, subjective evaluation methods, although providing richer information, come with drawbacks such as requiring substantial time and resources, as well as issues related to the subjectivity and reliability of results. Therefore, future research should aim to develop more comprehensive and accurate music generation evaluation methods. This might include incorporating music theory knowledge, professional opinions from musicians, and more comprehensive aesthetic standards, thereby better assessing the quality and artistry of generated music.

9. Personal Reflection

I have always been deeply interested in the integration of music and the field of computer science. Despite my passion for music, my knowledge of music theory and related areas was relatively limited before embarking on this project. The first-time exposure to music theory took some time for me to understand the meaning of each symbol in a musical score. This lack of knowledge drove me to delve deeper into the theory and practice of music. Through this research project, I not only gained a comprehensive understanding of the intersection of music generation and deep learning but also developed a stronger interest in music theory and related domains. It can be said that this project brought me the joy and opportunity of learning new knowledge.

During the concrete implementation of the project, I encountered numerous challenges and learning opportunities. This project might well be one of the most academically and professionally demanding research endeavors I have faced thus far, especially as a student new to the field of deep learning. I have always aspired to gradually master the knowledge and skills in this domain through practical experience. From initially sifting through extensive literature to preprocessing the dataset and constructing the model, and finally evaluating the overall research results, despite the numerous difficulties, I believe all these efforts have been worthwhile.

In the data preprocessing phase, I faced various challenges. I carefully considered several methods to convert music into usable data, each with its own advantages and disadvantages. Given my relatively limited experience in data processing, I ultimately opted for a relatively straightforward approach. However, as the project progressed, I gradually realized the limitations of this method. Especially during the final evaluation stage, I discovered some flaws in this approach, resulting in lower data utilization efficiency. This experience deepened my understanding of the critical role of data preprocessing in the field of machine learning because it directly impacts the model's performance and the credibility of research results. If given the opportunity to redo the project, I would pay greater attention to the choice of data processing methods and dedicate efforts to improving data preprocessing. Nevertheless, this does not imply that I consider this project a failure; on the contrary, I believe this experiment has successfully demonstrated the potential of LSTM in melody prediction.

Undoubtedly, the most time-consuming aspect of this project was parameter tuning. Given my relatively limited experience in the field of deep learning, the selection of each parameter, such as the number of layers in the neural network, the number of neurons, and the loss function, required multiple experiments and fine adjustments. Unfortunately, hardware limitations meant that each parameter adjustment required a substantial amount of time to obtain results. Despite the time-consuming nature of this process, it is crucial for ensuring the ultimate effectiveness of the model. I have come to a profound realization that parameter tuning is an indispensable key step in optimizing deep learning model performance. Throughout this process, I accumulated a wealth of techniques and experiences related to parameter tuning. For instance, narrowing down parameter ranges to a smaller scope and conducting fine searches within that range to identify the best parameter combinations. I also gradually understood the interactions between different parameters and their impact on model performance. Finally, within the constraints of limited time, I managed to accomplish the established objectives by prioritizing tasks effectively.

I firmly believe that the fusion of machine learning and the field of music holds immense potential for widespread applications in the future. This field can not only provide new tools and methods for music composition but also drive innovation and improvement in various domains, including music education, music therapy, and the entertainment industry. Fueled by my strong interest in this field, I look forward to delving deeper into research and advancing its development in the future. Ultimately, I aspire to apply automatic music generation technology to real-life scenarios, promoting societal progress and development. Whether by providing creative inspiration to music composers or introducing innovations in the realms of music education and therapy, I hope that my research can have a practical and positive impact.

References

Agarwal, S., Saxena, V., Singal, V., & Aggarwal, S. (2018, November). Lstm based music generation with dataset preprocessing and reconstruction techniques. In *2018 IEEE symposium series on computational intelligence (SSCI)* (pp. 455-462). IEEE.

Bisong, E. (2019). Matplotlib and seaborn. *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*, 151-165. https://doi.org/10.1007/978-1-4842-4470-8_12

Breve, B., Cirillo, S., Cuofano, M., & Desiato, D. (2022). Enhancing spatial perception through sound: Mapping human movements into Midi. *Multimedia Tools and Applications*, *81*(1), 73–94. https://doi.org/10.1007/s11042-021-11077-7

Briot, J. P. (2021). From artificial neural networks to deep learning for music generation: History, concepts and trends. *Neural Computing and Applications*, *33*(1), 39–65. https://doi.org/10.1007/s00521-020-05399-0

Ciaburro, G., & Venkateswaran, B. (2017). *Neural Networks with R: Smart models using CNN, RNN, deep learning, and artificial intelligence principles*. Packt Publishing Ltd.

Colombo, F., Seeholzer, A., & Gerstner, W. (2017). Deep artificial composer: A creative neural network model for automated melody generation. In *Computational Intelligence in Music, Sound, Art and Design: 6th International Conference, EvoMUSART 2017, Amsterdam, The Netherlands, April 19–21, 2017, Proceedings 6* (pp. 81-96). Springer International Publishing.

Donahue, J., Anne Hendricks, L., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., & Darrell, T. (2015). Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2625-2634).

Eck, D., & Schmidhuber, J. (2002). Finding temporal structure in music: Blues improvisation with LSTM recurrent networks. In *Proceedings of the 12th IEEE workshop on neural networks for signal processing* (pp. 747-756). IEEE.

Haji, S. H., & Abdulazeez, A. M. (2021). Comparison of optimization techniques based on gradient descent algorithm: A review. *PalArch's Journal of Archaeology of Egypt/Egyptology*, *18*(4), 2715-2743.

He, F., Liu, T., & Tao, D. (2019). Control batch size and learning rate to generalize well: Theoretical and empirical evidence. *Advances in neural information processing systems*, *32*.

Hennig, C., & Kutlukaya, M. (2007). Some thoughts about the design of loss functions. *REVSTAT-Statistical Journal*, *5*(1), 19-39. https://doi.org/10.57805/revstat.v5i1.40

Hewitt, M. (2008). *Music theory for computer musicians*. Course Technology, CENGAGE Learning.

Huang, C. Z. A., Koops, H. V., Newton-Rex, E., Dinculescu, M., & Cai, C. J. (2020). AI song contest: Human-AI co-creation in songwriting. *arXiv preprint arXiv:2010.05388*.

Huang, C. Z. A., Vaswani, A., Uszkoreit, J., Shazeer, N., Simon, I., Hawthorne, C., ... & Eck, D. (2018). Music transformer. *arXiv preprint arXiv:1809.04281*.

Huang, G., Sun, Y., Liu, Z., Sedra, D., & Weinberger, K. Q. (2016). Deep networks with stochastic depth. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14* (pp. 646-661). Springer International Publishing.

Idris, I. (2015). *NumPy: Beginner's Guide*. Packt Publishing Ltd.

Japkowicz, N., & Shah, M. (2011). *Evaluating learning algorithms: a classification perspective*. Cambridge University Press.

Karita, S., Chen, N., Hayashi, T., Hori, T., Inaguma, H., Jiang, Z., Someki, M., Soplin, N. E., Yamamoto, R., Wang, X., Watanabe, S., Yoshimura, T., & Zhang, W. (2019). A comparative study on transformer vs rnn in speech applications. In *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)* (pp. 449-456). IEEE. https://doi.org/10.1109/asru46091.2019.9003750

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Loy, G. (1985). Musicians make a standard: The Midi phenomenon. *Computer Music Journal*, *9*(4), 8–26. https://doi.org/10.2307/3679619

Michelucci, U. (2018). Applied deep learning. *A Case-Based Approach to Understanding Deep Neural Networks*. Apress Berkeley, CA. https://doi.org/10.1007/978-1-4842-3790-8

Moore, F. R. (1988). The dysfunctions of Midi. *Computer Music Journal*, *12*(1), 19–28. https://doi.org/10.2307/3679834

Oore, S., Simon, I., Dieleman, S., Eck, D., & Simonyan, K. (2020). This time with feeling: Learning expressive musical performance. *Neural Computing and Applications*, *32*, 955–967. https://doi.org/10.1007/s00521-018-3758-9

Pang, B., Nijkamp, E., & Wu, Y. N. (2020). Deep learning with tensorflow: A review. *Journal of Educational and Behavioral Statistics*, *45*(2), 227-248. https://doi.org/10.3102/1076998619872761

Rohrmeier, M., & Rebuschat, P. (2012). Implicit learning and acquisition of Music. *Topics in*

Cognitive Science, *4*(4), 525–553. https://doi.org/10.1111/j.1756-8765.2012.01223.x

Sharma, A. (2017, March 30). *Understanding activation functions in neural networks*. Medium. https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

Sherstinsky, A. (2020). Fundamentals of Recurrent Neural Network (RNN) and long short-term memory (LSTM) network. *Physica D: Nonlinear Phenomena*, *404*, 132306. https://doi.org/10.1016/j.physd.2019.132306

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, *15*(1), 1929-1958.

Sturm, B. L., Santos, J. F., Ben-Tal, O., & Korshunova, I. (2016). Music transcription modelling and composition using deep learning. *arXiv preprint arXiv:1604.08723*.

Telea, A. C. (2014). *Data visualization: principles and practice*. CRC Press.

Theis, L., Oord, A. V. D., & Bethge, M. (2015). A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*.

Todd, P. M. (1989). A connectionist approach to algorithmic composition. *Computer Music Journal*, *13*(4), 27–43. https://doi.org/10.2307/3679551

Tymoczko, D. (2013). Review of Michael Cuthbert, *music21: A toolkit for computer-aided musicology* (http://web.mit.edu/music21/). *Music Theory Online*, *19*(3). https://doi.org/10.30535/mto.19.3.11

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.

Wu, H., & Shapiro, J. L. (2006). Does overfitting affect performance in estimation of distribution algorithms. *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, 433–434. https://doi.org/10.1145/1143997.1144078

Yamshchikov, I. P., & Tikhonov, A. (2020). Music generation with variational recurrent autoencoder supported by history. *SN Applied Sciences*, *2*(12), 1937. https://doi.org/10.1007/s42452-020-03715-w

Yang, Y., Zhou, J., Ai, J., Bin, Y., Hanjalic, A., Shen, H. T., & Ji, Y. (2018). Video captioning by adversarial LSTM. *IEEE Transactions on Image Processing*, *27*(11), 5600–5611. https://doi.org/10.1109/tip.2018.2855422

Ying, X. (2019). An overview of overfitting and its solutions. *Journal of Physics: Conference Series*, *1168*(2), 022022. https://doi.org/10.1088/1742-6596/1168/2/022022

Ketkar, N., & Ketkar, N. (2017). Introduction to keras. *Deep learning with python: a hands-on introduction*, 97-111. https://doi.org/10.1007/978-1-4842-2766-4_7

Zhou, L., Pan, S., Wang, J., & Vasilakos, A. V. (2017). Machine learning on Big Data: Opportunities and challenges. *Neurocomputing*, *237*, 350–361. https://doi.org/10.1016/j.neucom.2017.01.026