DRUM ACCOMPANIMENT GENERATION USING MIDI MUSIC DATABASE AND SEQUENCE TO SEQUENCE NEURAL NETWORK

A Thesis Submitted to
The Graduate School of Engineering and Sciences of
İzmir Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

in Electronics and Communication Engineering

by
Yavuz Batuhan AKYÜZ

July 2022
İZMİR

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my supervisor, Assoc. Prof. Dr. Şevket Gümüştekin, for giving me a chance to bring my idea to life. While his passion for music inspired, guided, and encouraged me, his knowledge and experience made significant contributions throughout the thesis. Moreover, he helped me on the way to becoming a professional.

I also thank my supervisory committee members, Assist. Prof. Dr. M. Zübeyir Ünlü, Prof. Dr. Barış Bozkurt, Prof. Dr. Mustafa A. Altınkaya, and Assist. Prof. Dr. Başak Esin Köktürk Güzel, who showed an interest and took their time to help me to improve my dissertation with their valuable comments.

Last, but not least, I would like to thank the entire IZTECH family, especially the professors of the Department of Electrical and Electronics Engineering, who have helped me since my undergraduate education and provided me with a good foundation with their meticulous attitudes, devotion, and passion.

Finally, at the end of my M.S. thesis, I would like to express my gratitude to all my family members, particularly to my father Cemil Akyüz, and my mother Tülay Akyüz, who have raised me to become who I am today, for always believing in me and supporting me.

ABSTRACT

DRUM ACCOMPANIMENT GENERATION USING MIDI MUSIC DATABASE AND SEQUENCE TO SEQUENCE NEURAL NETWORK

This thesis aims to create an artificial intelligence model to reinterpret the drum parts of musical pieces and/or to accompany music with new uniquely generated drum patterns. Besides providing rhythmic indicators, drum parts are essential to emphasize emotions. Every instrument in a musical composition is in harmony with each other to be meaningful as a whole. Based on this observation, in this thesis, a MIDI dataset and an LSTM based Seq2Seq model were used to create a link between different instruments and drums. Before the training, we created a dataset involving midi pieces with drum parts and grouped them as input and output, which are non-drum instruments, and drum parts respectively. The model was trained with six different genres and the teacher forcing method was utilized to improve the training. After the training, at the generation stage, we made it possible to adjust the complexity of the generated drum parts by changing the temperature value, which we called the complexity value, using the temperature sampling method. We also created a user interface with an instrument selection pane to give users control over the drum instruments generated. Moreover, we proposed a novel approach to generalize the idea for not only MIDI data but also WAV data. To accomplish this task, Mel-spectrogram, MFCC, and tempogram features were used. Both proposed methods are shown to produce high-quality unique drum accompaniments for different genres with adjustable complexity and freedom of choosing the desired drum instruments.

ÖZET

MIDI MÜZİK VERİTABANI VE DİZİDEN DİZİYE YAPAY SİNİR AĞI KULLANIMI İLE DAVUL EŞLİĞİ ÜRETİMİ

Bu tezde yapay zeka modelleri kullanılarak müzik parçaları içerisindeki davul kısımlarının eşsiz bir şekilde yeniden yorumlanması ve/veya yeni davul örüntüleri oluşturularak müziğe eşliği hedeflenmiştir. Davullar, müziklerde ritmi belirlemekte baş rolde bulunsalar da, bunun yanı sıra, duyguları vurgulamakta da çok başarılıdırlar. Müzik kompozisyonları bütünlük açısından bir anlam ifade etmelerini, içerisinde çalınan her enstrümanın birbiriyle bir harmoni içerisinde olmasına borçlulardır. Bu gözleme dayanarak, tezimizde, MIDI veri kümesi ve LSTM yapısına sahip olan diziden diziye modeli kullanılarak davul harici enstrümanlar ve davul enstrümanı arasında bir bağlantı kurulması hedeflenmiştir. Eğitimden önce, veri kümesi örneklenmiş ve davul harici enstrümanlara ait veriler giriş olarak, davul enstrümanına ait veriler çıkış olarak belirlenmiştir. Model, altı farklı veri kümesi kullanılarak eğitilmiş ve öğretmen zorlama tekniği kullanılarak eğitim aşaması iyileştirilmiştir. Eğitimden sonra, üretim aşamasında, üretilen davul örüntülerinin karmaşıklığını ayarlayabilmek için sıcaklık örneklemesi kullanılmış; ve sıcaklık değeri karmaşıklık parametresi olarak tanımlanmıştır. Ek olarak bir kullanıcı arayüzü geliştirilmiş, ve bu sayede, kullanıcının üretilecek olan davul enstrümanları üzerinde tam kontrol sahibi olması amaçlanmıştır. Burada sunduğumuz fikri, MIDI verileri dışında WAV verisi için de genelleştirmek amacıyla, özgün bir yaklaşım ileri sürülmüştür. Bu yaklaşımı gerçekleştirmek için Mel-spectrogram, MFCC, ve tempogram özellikleri kullanılmıştır. Sunulan ve geliştirilen iki yöntem de yüksek kalitede, farklı janra seçenekleri, değiştirilebilen karmaşıklık değeri ve enstrüman seçme özgürlüğüyle eşsiz davul eşlikleri üretimi ile sonuçlanmıştır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AI: Artificial Intelligence

ANN: Artificial Neural Network

CNN: Convolutional Neural Network

DCT: Discrete Cosine Transform

DFT: Discrete Fourier Transform

DL: Deep Learning

EOS: End of the Sequence Token

FFT: Fast Fourier Transform

GAN: Generative Adversarial Network

GPU: Graphical Processing Unit

IDCT: Inverse Discrete Cosine Transform

IDFT: Inverse Discrete Fourier Transform

LSTM: Long Short-Term Memory

MAE: Mean Absolute Error

MFCC: Mel-Frequency Cepstral Coefficients

MIDI: Musical Instrument Digital Interface

ML: Machine Learning

MP3: MPEG-1 Audio Layer 3

MSE: Mean Squared Error

NLP: Natural Language Processing

ResNet: Residual Network

RGB: Red Green Blue Image

RMSE: Root Mean Squared Error

RNN: Recurrent Neural Network

Seq2Seq: Sequence-to-Sequence

SOS: Start of the Sequence Token

STFT: Short Time Fourier Transform

VAE: Variational Autoencoder

WAV: Waveform Audio File Format

CHAPTER 1

INTRODUCTION

Life would be flat without music.
It is the background to all I do.
It speaks to the heart in its own special way
like nothing else.
-Ludwig van Beethoven-

Music is a powerful tool to express emotions. It has been existed since the beginning of mankind. At first, humans used sounds to communicate with each other, as the time passed these sounds were diversified. With this diversity, music emerged. Throughout the ages, music has been changed and improved with different methods due to the conditions of that time and newly discovered musical instruments. Besides the fact that music is a way to express emotions, many civilizations discovered its healing effect depending on their cultural and social level (Koç et al. 2016). Moreover, today, it has been proven that the music has a positive effect on people's emotional hormones (e.g., dopamine, serotonin, etc.) and physiological functions (e.g., heart rhythm, respiratory rhythm, etc.), and as a result, the quality-of-life increases. (Boşnak, Kurt and Yaman 2017). Among all the discoveries, the biggest one was the overall discovery of musical fundamentals such as measure, rhythm, harmony, notation, and scales. After this stage, for the first time, people were able to build the foundations of their musical compositions. As a consequence of the interest in music, nowadays, there are more than 300 music genres available.

Music, alone, has so many different features, such as rhythm, notes, melody, duration, amplitude, etc., and it can get more complex when these features are tried to combine to work together in a harmony. Moreover, apart from these features, the use of multiple instruments also increase complexity. In a real orchestra, every instrument has its own duty so that they work together to produce a plausible music.

Not only composers but also listeners pay attention to these features (Herremans and Chew 2017). When it comes to producing realistic music by following the music theory, the challenge is not only that there is so much information to process, but that each piece of information is somewhat related to the others. This uniqueness of features makes it harder to generalize the NLP models to generate realistic music (Dong et al. 2018). Furthermore, processing this much information requires a lot of memory. In order to relieve the models from this level of complexity, different methods are used to simplify. Some of the simplifications that are made to input data are to limit the input features to a few notes and chord progressions, a single instrument, constant tempo, etc. Moreover, different data formats and representations (e.g., MIDI, music score, symbolic representation, raw audio signal, etc.) are used to access the information more easily.

So far, we mentioned the importance of the music and the challenges of realistic music generation. In the next sections, we first review the previous studies in the field of music generation. Next, we will summarize the main objective. Lastly, we will present the organization of our thesis.

1.1. The Literature Overview

Music is not a new topic in the field of Artificial Intelligence. Over the years a significant number of scientific papers have been published about generating music by using different neural network architectures and data representations. In the late 1950s, with the invention of the first computer, the first music score was composed by computers by using stochastic models and Markov Chains (Hiller and Isaacson 1959). In the following years, specifically the late 1980s, music generation with ANNs appeared, which pioneered today's systems, and evolved significantly (Briot 2020).

One of the first attempts to generate music by using ANNs made by the Lewis' novel approach (Lewis 1988). In the creation by refinement method (CBR), which is the name of his approach, he proposed changing the standard way of using gradient descent for standard applications by reverting it.

During the training phase, Lewis used 30 examples developed by himself. Because the note intervals in these 30 examples are between unison, 3rd and 5th, and some notes that follows a stepwise motion; he described them as “well-formed” melodies. Apart from this, he also produced “poorly-formed” melodies that do not comply with this rule. In the end, these two datasets were used to train the conventional neural network. In the CBR part, the samples with random values were given to the input layer, and to maximize the positive classification result, a gradient descent optimization was applied iteratively. As a result of this process, the newly created melody became well-formed. According to Lewis, the network was able to learn the concepts of stepwise and triadic motion.

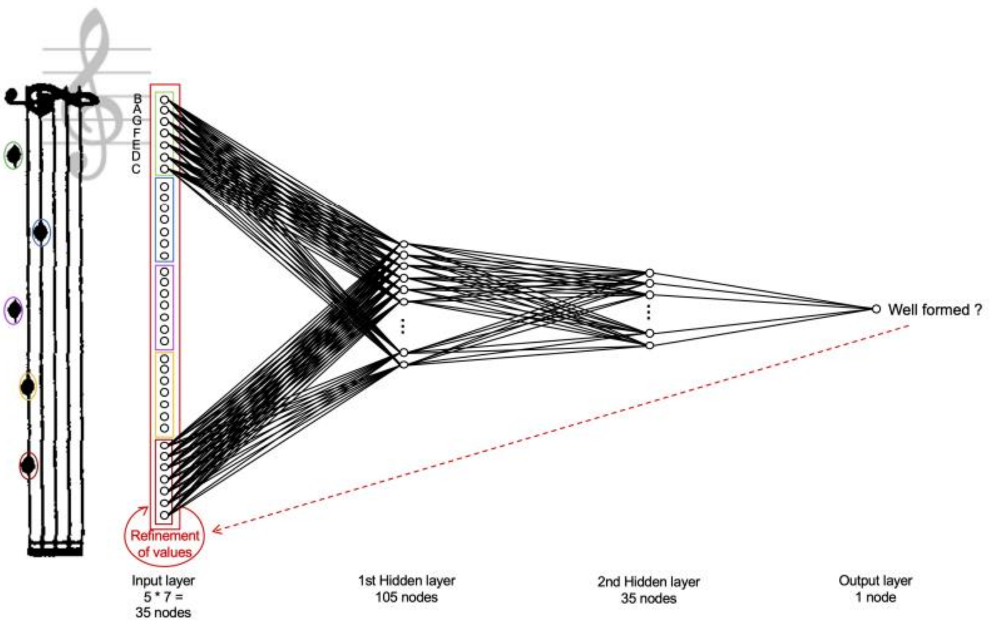

B
A
G
F
E
D
C
Well formed ?
Refinement of values
Input layer 5 * 7 = 35 nodes
1st Hidden layer 105 nodes
2nd Hidden layer 35 nodes
Output layer 1 node

Figure 1.1. CBR Architecture

(Source: Briot 2020)

Another early idea to explore generating music with an ANN was by Todd (Todd 1989). He named this design as Time-Window architecture. His main motivation was to generate a monophonic melody by using a fixed-size sliding window. Despite the architecture being capable of learning the pairwise correlations between two successive segments, the model is poor when it comes to learning long-term correlations due to the absence of explicit memory. In order to overcome this shortcoming, in his following design which he named Sequential, he divided the input layer into two parts, the context, and the plan. The context is the memory that holds the information about the melody generated so far. The plan corresponds to the melody to be learned.

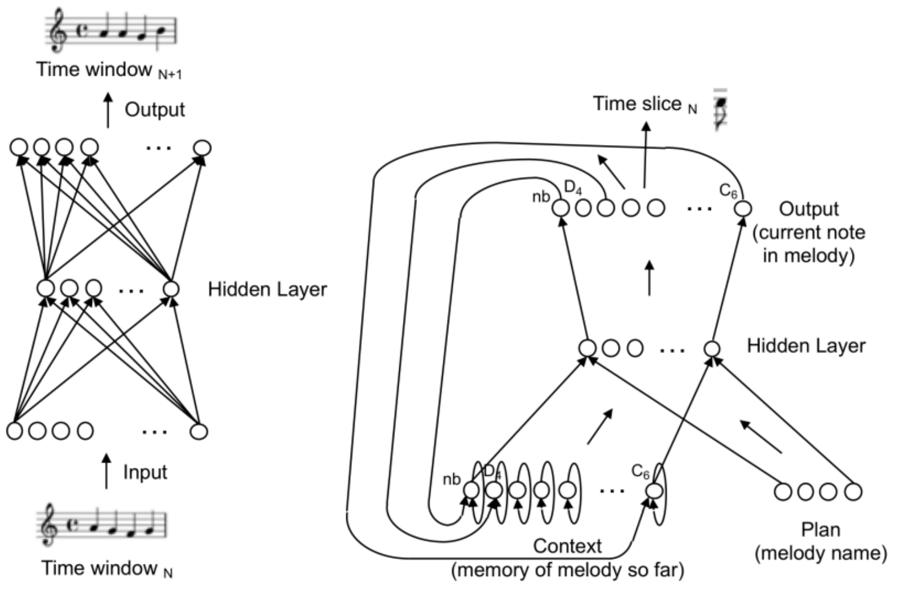


Figure 1.2. Time-Window (left) and Sequential (right) Architectures
(Source: Briot 2020)

The working principle of the idea can be summarized as follows: the training starts by choosing a plan (melody to be learned) and, then to start with a clean empty context, the activation units of each context are initialized with 0. Next, the weights are upgraded based on the comparison between the first-time step note, which is the output of the feedforwarded network, and the first-time steps note of the plan. The output values are then passed to the current context. An iteration is concluded when the last time step is reached. The process repeats for different plans. In the generation, the activation units are initialized to 0 and the network is fed with a new plan, which was not part of the training. The new melodies are generated iteratively.

So far, we discussed the initial ANN applications for music generation. Although, nowadays, due to the technological improvements we have more computational power, we can see that both Todd's and Lewis' approaches pioneered the next studies. In the following subsections, we will focus on reviewing the studies based on the widely used modern network architectures.

1.1.1. CNN-RNN based Music Generation

CNNs and RNNs are known to be good for learning stationary data but not so great for time-varying data (e.g., music generation, time-series forecasting, etc.) alone. The reason behind that is their incapability to process long-term dependencies and problems that occurred on the way. Although this is the case, there have been promising studies such as SampleRNN (Mehri et al. 2017) and MelodyRNN (Waite 2016) published to generate music by using CNN and RNN.

WaveNet (van den Oord et al. 2016), which is a study that generates raw audio waveforms published by Google DeepMind, shows that it is possible to use CNN to generate realistic music. The study uses dilated causal convolutions to predict the next sound sample conditioned to all of the previous ones. The real waveforms, which are recorded by human speakers, were used as the model input during the training. After the training phase, synthetic utterances are obtained by sampling the network. New predictions were made by taking a value from the probability distribution and fed back to the input at each step during sampling. The authors stated that although the process is computationally expensive, it was essential to generate complex and realistic audio. WaveNet has been used for different applications. The first one of these applications was in the field of text-to-speech, and listeners noted that the generated sound samples sounded more natural. Another application is in the field of music production, which produces realistic music pieces. This is an important result because CNNs are usually faster than other architecture and easier to parallelize (van den Oord et al., 2016).

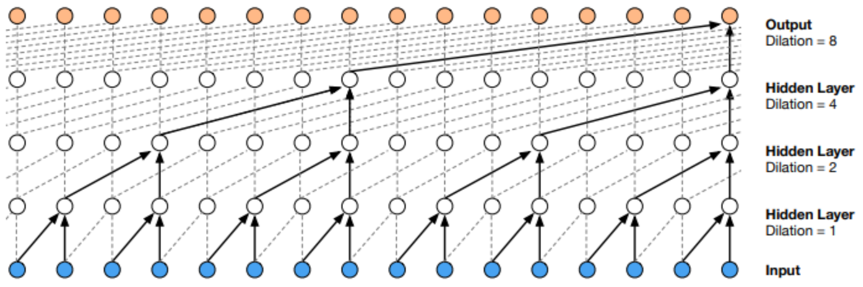


Figure 1.3. Stacked Dilated Causal CNN

(Source: van den Oord et al., 2016)

Another study done in this field is the Song from PI. Song from PI is, which is originally a YouTube video[1], a melody that is written by using digits of the number Pi (π). Motivated by the idea used in Song from PI, the authors (Chu, Urtasun and Fidler 2016) developed a model, which is conditioned on scale type, that is able to learn notes on a particular scale. In the study, they used a Pop MIDI collection and some video game music as the dataset and RNN as their model. The higher layers of the RNN focused on learning accompaniments like drums and chords. The lower layers of the RNN learned the melody. Moreover, they conducted two interesting and exciting applications. In one of them, they created a stickman that sings the generated song and performs a dance show according to it. In the other one, they extended the model capabilities so that it generates a story according to given image and converts it to a piece of pop music.

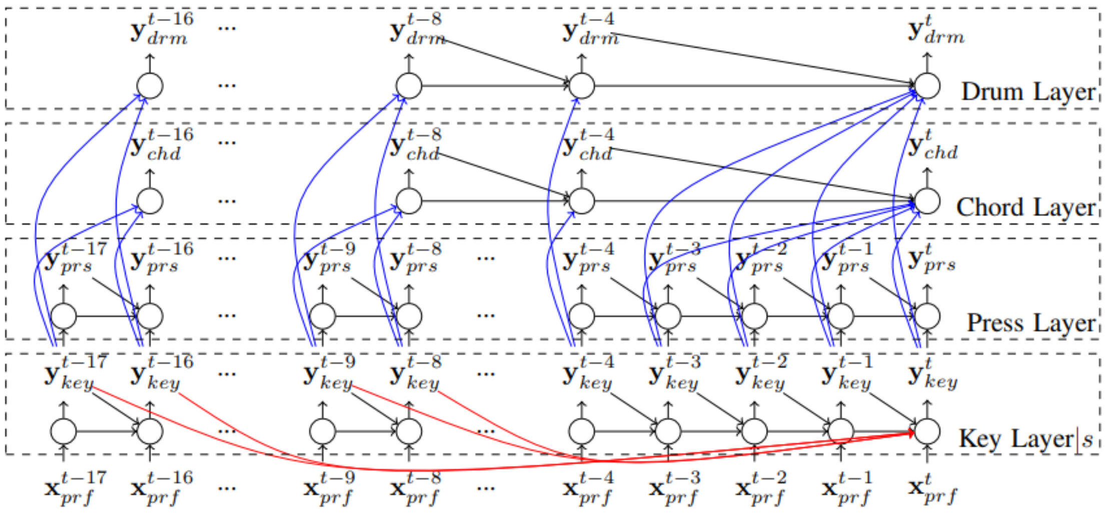


Figure 1.4. Model Represented in Music From PI

(Source: Chu, Urtasun and Fidler 2016)

1.1.2. LSTM based Music Generation

When it comes to music generation with the deep learning models, the most popular architecture that has been used is LSTM. The main reason behind it is the ability to store information of long sequences so that it can learn long-time correlations. Eck and Schmidhuber (Eck and Schmidhuber 2002) proposed two different models to learn chord

[1] "Song from π!". 2011. YouTube. https://youtu.be/OMq9he-5HUU.

structures and note structures and the learning melody that is presented throughout the music. The generated patterns were able to follow the chord progressions, but they lacked uniqueness since the model was trained with certain chord structures. Moreover, in another paper, they used LSTM to learn blues structures which resulted in blues patterns with proper timing and structure (Eck and Schmidhuber 2002). LSTM also proved its capabilities of learning different genres such as classical (Yadav et al. 2021), jazz (FRANKLIN 2006), metal (Zukowski and Carr 2018), etc.

In another study, LSTM was used by Nayebi and Vitelli (Nayebi and Vitelli 2015) to generate music with the raw audio signals as the input. The DFT is applied to inputs to extract the features. After training was completed with the generic LSTM model, the results were compared with GRU (Cho et al. 2014) which is developed to model long time dependencies of generic sequences in an effective manner. The model was previously used on the MIDI dataset and resulted that its performance being close to LSTM (Chung et al. 2014). It was concluded that LSTM was able to generate patterns that are plausible to the human ear.

Unlike using vanilla LSTM, another music generation study called DeepJ (Mao, Shin and Cottrell 2018) used Biaxial LSTM. A Biaxial LSTM (Johnson 2017) consists of two modules because of their ability to model polyphonic music by modeling each note in each time step with a conditional probability that depends on all previous time steps, and each generated note in the current time step. The time-axis module, which is used for temporal information, outputs higher note features for each note depending on the input note octaves and recurrent states from the previous step. The note-axis module, which is used for note information, takes note features from the time-axis module and based on the conditions of lower note features and higher note features, the next note is predicted. In the DeepJ, improvements on Biaxial LSTM are mainly based on the addition of volume and style parameters which lead to an overall quality increase, and additionally, the consistency problem of the model was solved.

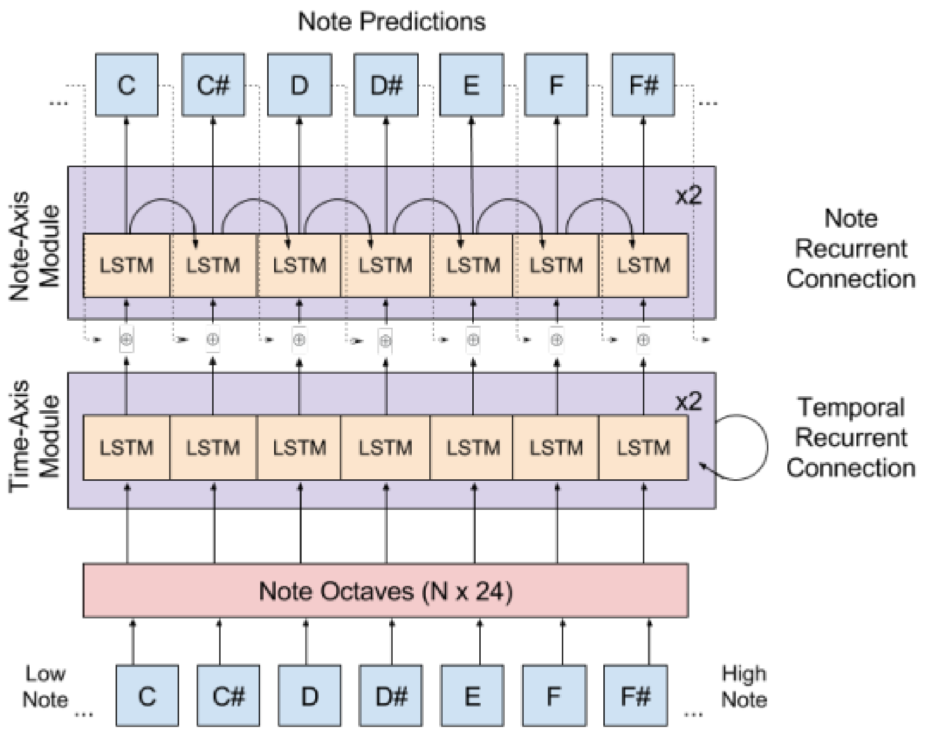


Figure 1.5. Biaxial LSTM Architecture
(Source: Mao, Shin and Cottrell 2018)

1.1.3. VAE based Music Generation

VAE (Kingma and Welling 2013) consists of two modules as encoder and decoder. By learning the probability distributions of the data, VAE generates the data similar to data in the original dataset. Since there is no target data, it is classified as an unsupervised learning algorithm.

MusicVAE (Roberts et al. 2018) is an application that used VAE to generate music. The MIDI format had been used as the dataset. In the study, they used two-layer bidirectional LSTM as the encoder and simple stacked RNN as the decoder. However, because of the vanishing gradient problem of the RNN, the model produced poorly generated sequences. To overcome this situation, they changed the decoder architecture to unidirectional LSTM.

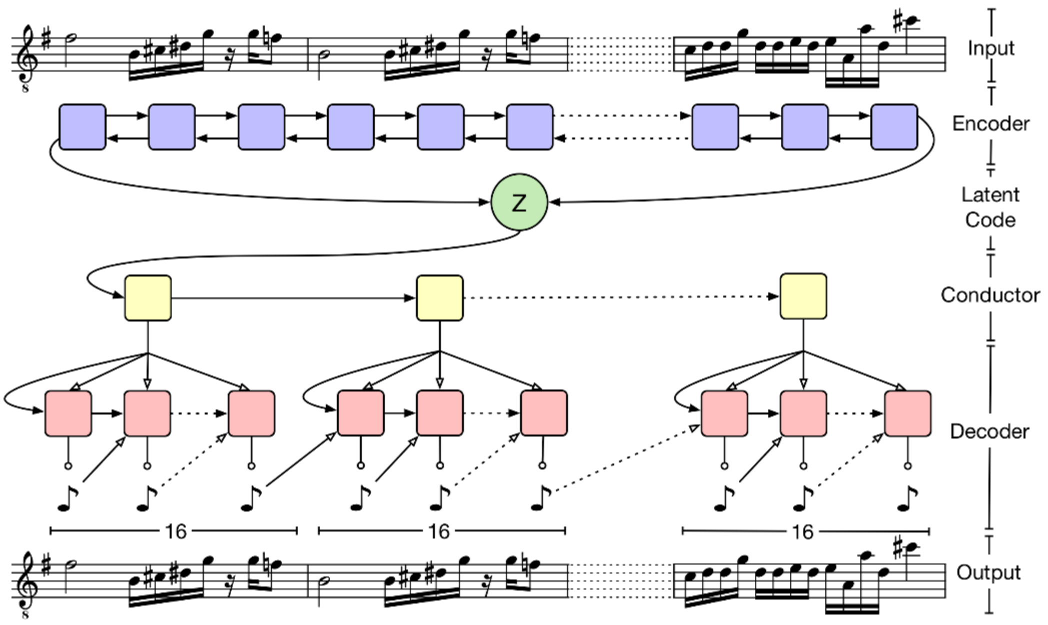


Figure 1.6. MusicVAE Architecture
(Source: Roberts et al. 2018)

1.1.4. GAN based Music Generation

Another way to generate music is to utilize GAN (Goodfellow et al. 2014). GAN consists of two models which are generator and discriminator. GAN network is basically a two-player game between generator and discriminator. Generator generates sample sized noise and passes to discriminator. Discriminator tries to discriminate real data and noise. Based on the discriminator output, generator updates its weights and generates another noise that is similar to original data. As the process continues, eventually, the generator reaches at a certain level that can generate samples similar to real data from the noise. At this point, discriminator is no longer able to discriminate between noise generated patterns from generator and original data.

Besides the vanilla GAN, there are also different GAN architectures (Creswell et al. 2018) such as Fully Connected GANs, Convolutional GANs, Conditional GANs, etc. One of the popular music generation algorithms using Convolutional GAN was the MIDINET (Yang, Chou and Yang 2017). In this study, the motivation was to generate music in symbolic domain. In the light of this goal, the data was sampled measure by measure and 2D matrices were constructed. As a result of this, they were able to represent data similar to music scores. Moreover, additional noise was added to the data with the aim of producing diverse results and improving creativity. Exploiting the powerful features of Convolutional GANs, was shown that the Convolutional GANs can be a powerful alternative to RNNs.

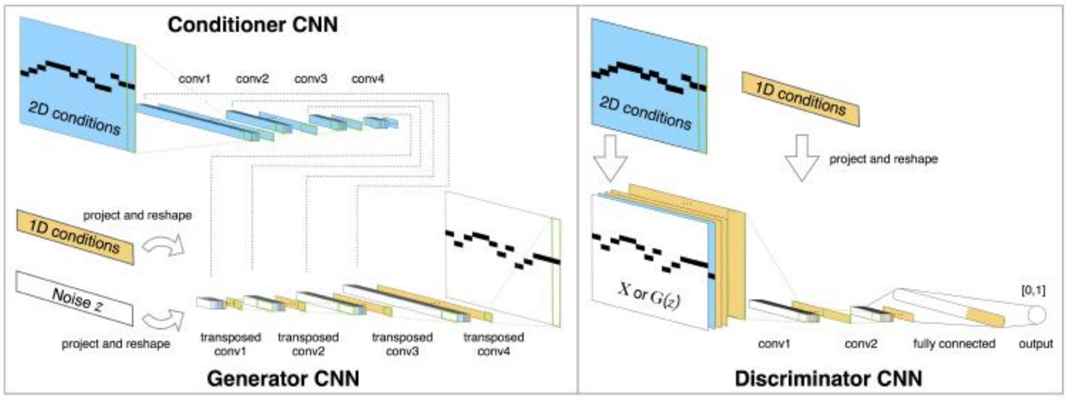


Figure 1.7. MIDINET Model

(Source: Yang, Chou and Yang 2017)

MuseGAN (Dong et al. 2018) proposed 3 different models that uses GAN as the framework, namely Jamming, Composer and Hybrid, offering both generation and accompaniment for symbolic music. The goal was to generate polyphonic music with harmony and rhythm structures, multi-track interdependency and temporal structures, while avoiding simplifications to overcome the challenges discussed at the beginning of the chapter. As a result of the study, it was observed that the model they developed could learn music, but the music produced was not close to human level production.

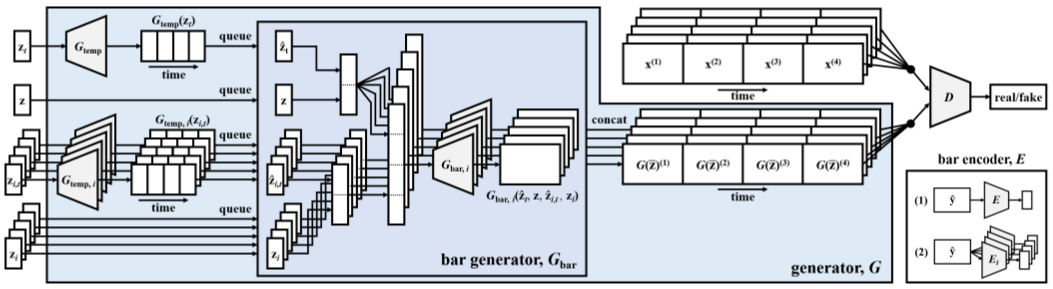


Figure 1.8. MuseGAN Model

(Source: Dong et al. 2018)

1.2. Objective of the Thesis

So far, we analyzed the previous studies and mentioned different approaches to generate music. Based on these studies, we have seen that music has many unique features that can be represented differently using various architectures. We also know that these music features (e.g., rhythm, notes, melody, duration, amplitude, etc.) vary for the different compositions. Therefore, in music generation applications, it is important to determine an analysis plan to in order to use its features properly and lower the complexity. Thus, we carried out the study by following the five-dimension analysis, used by Jean-Pierre Brioti, Gaëtan Hadjeres, and François-David Pachet (Briot, Hadjeres and Pachet 2019). The concepts of the five-dimension analysis method are defined as follows:

- **Objective:** Before designing a music generation AI tool, its objective and destination should be planned. The objective of the music can be an accompaniment, generation melody for polyphonic or monophonic music, etc. If the generated patterns are going to be performed by humans, then they should be in the form of a musical score; if they are going to be performed by machines, then the format should be in the case of an audio file.
- **Representation:** This dimension basically corresponds to feature selection and preprocessing. There are many different formats such as MIDI, raw signal, piano roll, text, etc. that can be used as the baseline. According to the format types, features can differ. For example, in the case of MIDI data, note pitch values, beat; in the case of raw audio data, spectrograms; and, in the case of piano rolls, notes and chords can be used. Moreover, in this dimension, the encoding technique is also decided.
- **Architecture:** The type of neural network architecture should be chosen so that it can learn based on the representation used. The RNN, autoencoders, LSTM, and GAN are common choices.
- **Challenges:** Knowing the limitations (e.g., quality, variability, creativity) is important for pushing the limits to the maximum level.
- **Strategy:** After setting the representation and choosing the architecture, a strategy is needed to be made to process the training and control the generation process in a way that matches the desired requirements.

Based on this framework, the objective of the study is determined as the drum accompaniment for a given audio file. In this study, two different approaches have been made for MIDI and WAV formats used as baseline. The input data of the model, represented as a sequence of notes and their velocities for the MIDI format and Mel-spectrogram, MFCC, and tempogram features were extracted for the WAV (audio signal) format. The output data of the model, for both approaches, represented as a sequence of drum notes and velocities. The one-hot encoding technique was used. The LSTM was chosen as the main architecture. In addition, in order to analyze the raw audio signal features, CNN was used as a secondary architecture. In the generation, with the aim of expanding the limitations, the temperature sampling method, which allows flexibility to adjust the complexity of the generated patterns, was used to improve the quality and the originality. Moreover, a UI was developed to provide users the freedom to choose different drum instruments based on the exact same reason given previously. The strategy was to use the Seq2Seq model as our data representation is similar to the data representation in NLP translation applications (e.g., in NLP translation applications, the input is a sequence of words and the output is a sequence of target language words; in our application, the input is a sequence of instrument variables, and the output is a sequence of originally generated drum instrument variables).

1.3. Organization of the Thesis

The following chapters are organized as follows. With the following Chapter 2, we started our discussion on a basic level introduction of the processes used throughout the thesis. In this chapter, we first started with the definition and characteristics of MIDI and WAV data. Then, we described the features that were extracted from the dataset. Lastly, we provided the mathematical expressions for model and preprocessing steps, as well as examples of these techniques, and covered their advantages.

In Chapter 3, we described our first approach to make a drum accompany network. First, we outlined the preprocessing steps that were done before the training. Then, we made a detailed analysis on the training model’s architecture and gave the reasonings behind it. Next, we explained the sampling technique used in the generator model to improve the quality of the samples. Thereafter, we provided both training results and example outputs. Lastly, we discussed the significant points of our approach and future improvements.

In Chapter 4, we extended the idea and proposed a novel approach to generalize the drum accompany network for high-quality audio signal data (i.e., WAV). As in the previous chapter, we first start by giving a detailed explanation of preprocessing steps where we extracted the features. Then, in order to analyze the characteristic of the features, we described the improvements done to the training model. Thereafter, in the generator model, we explained an algorithm that was required to determine the time localization of generated patterns. Next, we evaluated the training results and provided example outputs. Lastly, we shared our arguments about the approach and future improvements.

In Chapter 5, we mainly discussed about the challenges faced throughout the thesis. Moreover, based on these challenges and the techniques developed against them, we presented ideas that can be potential scientific research directions.

CHAPTER 2

BACKGROUND

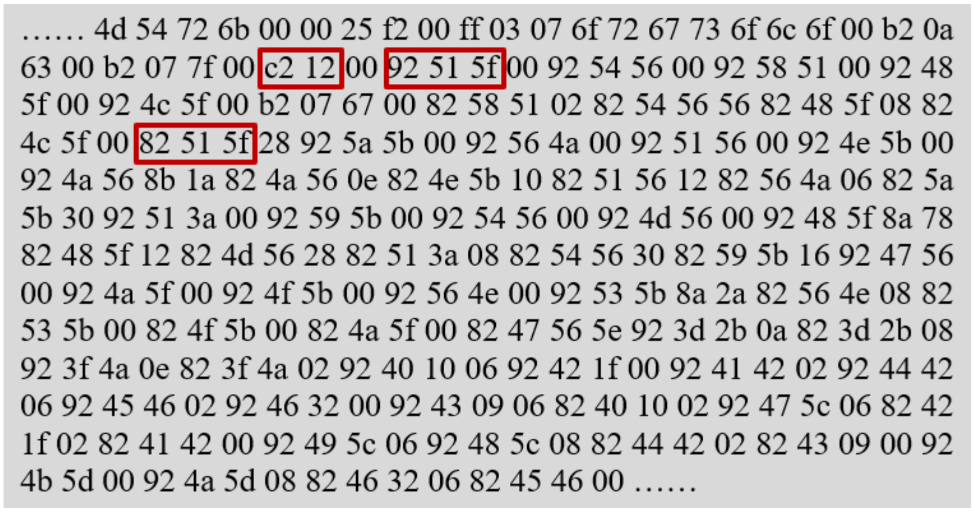
If you can't explain something in simple terms,
you don't understand it.
-Richard P. Feynman-

This chapter provides background information for the topics used throughout the thesis. The main goal is to create a machine learning algorithm for music accompaniment. In order to train the network, six different dataset genres which are jazz, hip-hop-rap, pop, blues, rock, and shuffle containing all previous genres are used. In the following subsections, we first give a brief explanation of the dataset format, preprocessing steps and the network architecture. In addition, supplementary methods are explained which are used to improve the architecture's loss and convergence rate for training as well as a sampling method for prediction.

2.1. The MIDI Format

The choice of input data format is important for the efficacy neural network training and efficient memory usage. Necessary variables, inside the data, should be easily accessible and not complicated. When it comes to audio data, there are several different data formats such as WAV, MIDI, MP3, etc. The widely used data format for audio signal processing is the WAV format. Although this format is better than artificially generated MIDI format sound-wise, it takes up too much memory and many preprocessing steps are required to acquire the desired information such as instruments, notes pitches, and notes velocities. For some recordings involving multiple instruments, noise, etc. it may not be possible to extract any meaningful information. On the other hand, the MIDI format not only provides all the required information such as pitches, durations, bpm, etc.

The MIDI format is a compact representation of music that is used for creating realistic simulations by a computer. In addition, MIDI data is usable in serial transmission between different instruments and computers. The format stores information in bytes under the name called MIDI messages. MIDI messages can be divided into two main categories, which are channel messages and system messages, at the highest level. There is total of 16 channels, and channel 9 is reserved for percussion instruments. Channel messages are valid for specific channels which are determined by the leading status bytes. On the contrary, system messages are not channel-specific and channel number is not indicated by the status bytes. Furthermore, channel voice messages and channel mode messages are two subcategories of channel messages. Channel voice message is the most important part of this format. Musical information which are note on, note off, note velocity, program change (instrument change), control change, channel pressure, and pitch bend change is contained in this message type (Back 1999). In order to ease the understanding of midi format data, the following example is demonstrated. Let's assume that we have a song in a midi format, a part of the representation of raw midi data of the song is given in figure below.



```
…… 4d 54 72 6b 00 00 25 f2 00 ff 03 07 6f 72 67 73 6f 6c 6f 00 b2 0a
63 00 b2 07 7f 00 c2 12 00 92 51 5f 00 92 54 56 00 92 58 51 00 92 48
5f 00 92 4c 5f 00 b2 07 67 00 82 58 51 02 82 54 56 56 82 48 5f 08 82
4c 5f 00 82 51 5f 28 92 5a 5b 00 92 56 4a 00 92 51 56 00 92 4e 5b 00
92 4a 56 8b 1a 82 4a 56 0e 82 4e 5b 10 82 51 56 12 82 56 4a 06 82 5a
5b 30 92 51 3a 00 92 59 5b 00 92 54 56 00 92 4d 56 00 92 48 5f 8a 78
82 48 5f 12 82 4d 56 28 82 51 3a 08 82 54 56 30 82 59 5b 16 92 47 56
00 92 4a 5f 00 92 4f 5b 00 92 56 4e 00 92 53 5b 8a 2a 82 56 4e 08 82
53 5b 00 82 4f 5b 00 82 4a 5f 00 82 47 56 5e 92 3d 2b 0a 82 3d 2b 08
92 3f 4a 0e 82 3f 4a 02 92 40 10 06 92 42 1f 00 92 41 42 02 92 44 42
06 92 45 46 02 92 46 32 00 92 43 09 06 82 40 10 02 92 47 5c 06 82 42
1f 02 82 41 42 00 92 49 5c 06 92 48 5c 08 82 44 42 02 82 43 09 00 92
4b 5d 00 92 4a 5d 08 82 46 32 06 82 45 46 00 ……
```

Figure 2.1. Raw MIDI Data

The file is the combination of data bytes as shown above. The MIDI file should be parsed to be meaningful. In this example, the channel voice message is focused and parsed since it contains the most important information. The messages, we are most interested in are the program change messages, note on and note off messages.

2.1.1. Program Change Message – c2 12

The status byte of the program change message is represented by 0xC2. The high nibble of the byte refers that it is a program change, and the low nibble represents the channel number. The second byte, 0x12, refers to the instrument number. This byte corresponds to 18 in decimal and maps to percussive instrument (See Appendix A for MIDI Instrument Map).

2.1.2. Note On Message – 92 51 5f

Note on event is represented at the high nibble of the status byte which is 9. The low nibble corresponds to channel number, which is the same as program change, where note is going to be played. The second byte represented by 0x51 refers to note pitch number. This byte corresponds to 81 in decimal and maps to note value A (See Appendix A for MIDI Note Number Map). The velocity/volume of this note is represented in the last byte, 0x5F. There are a total of 128 pitch and velocity values, the lowest being 0 and the highest being 127.

2.1.3. Note Off Message – 82 51 5f

There are two different types of data streams to understand that note is off. The first type, seen in this example, the high nibble of status byte is 8. This means that note is off. The note off message can also be 92 51 00. This is the second type, and note is understood to be off by the last byte which refers to 0 velocity. The second byte corresponds to note number as in note on message.

2.2. The WAV Format

The WAV format is an audio waveform format that is developed by IBM and Microsoft. Unlike MIDI file type, note’s pitch, notes velocity and instrumental information in WAV files are not easy to extract. They should be extracted by using signal preprocessing techniques. An example of a decoded WAV file can be seen below, which is basically a raw signal.

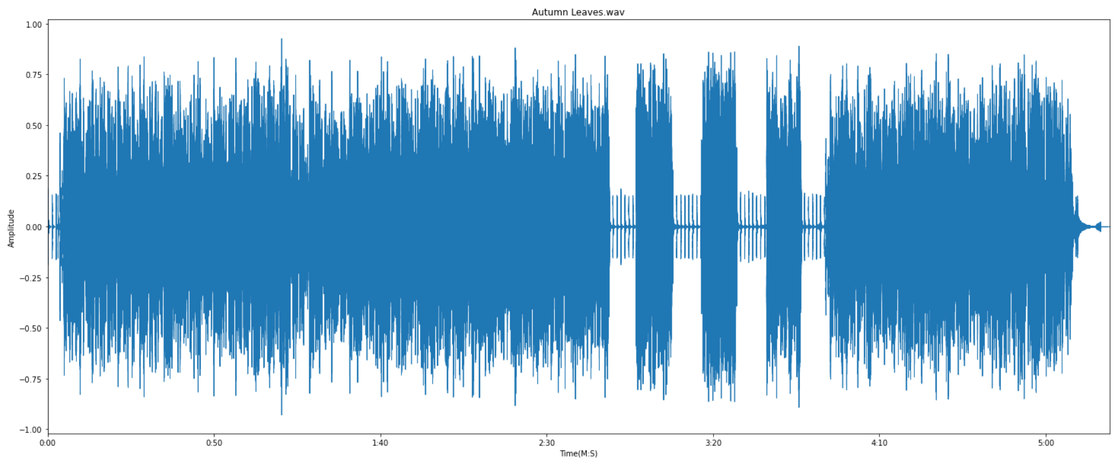


Figure 2.2. Raw WAV Signal

2.3. Data Preprocessing

Data preprocessing is an important step to extract the features and make data ready for the neural networks. This section provides background information on the preprocessing steps used throughout our application.

2.3.1. Mel Spectrogram

Mel spectrogram is a type of a spectrogram which has been used in ML application such as music genre classification, audio classification, music instrument classification, etc. The difference between classic spectrogram and Mel spectrogram is the representation of frequency. The frequency representation in classical spectrograms is linear where the melodic features in a signal become hard to differentiate from each other. Humans hear frequencies logarithmically. This is another problem that we encounter in classical spectrograms, because of their linearity. Therefore, using Mel spectrograms to extract features is better than using classical spectrograms.

In order to understand the problem with classical spectrograms better, we will consider the following example. Let's assume that we are playing two different sequences of notes on a piano. The first sequence is from C2 to C4, whose frequencies are from 65 Hz to 262 Hz. The second sequence is from G6 to A6, whose frequencies are from 1568 Hz to 1760 Hz. Although the pitch tones of these two sequences sound different from one another, both samples have a difference of nearly 200 Hz between the initial and last notes played. This problem occurs because the nature of pitches is nonlinear.

Mel spectrograms use Mel scale which is a perceptual scale of pitches that separated in equal distances from one another. The word "Mel" in Mel scale, which maps frequencies to Mel frequencies, is an abbreviation for melody.

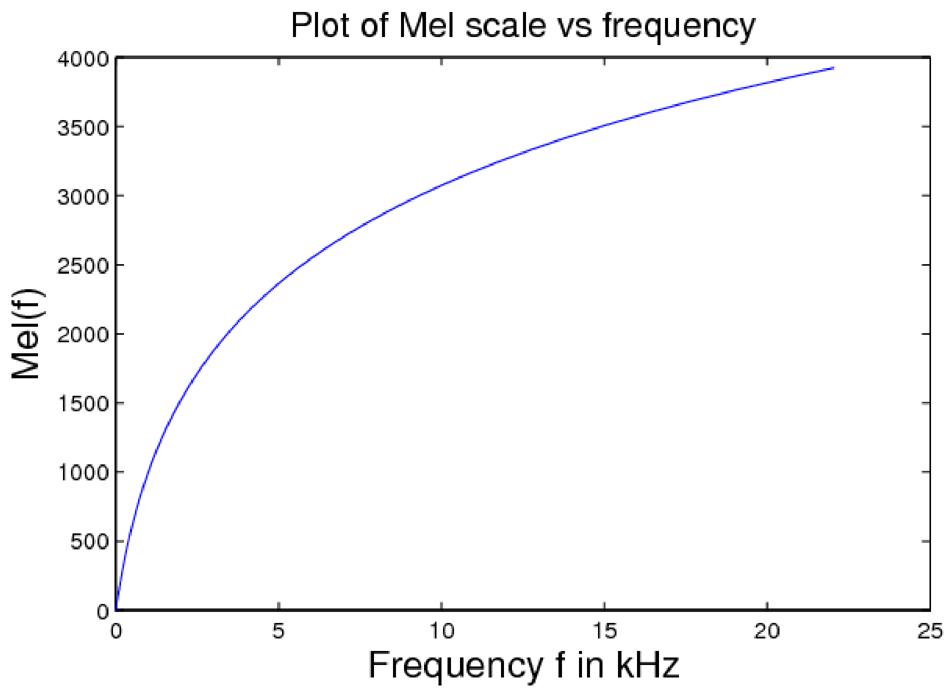


Figure 2.3. Mel Scale

(Source: Ramaseshan 2013)

The forward and inverse formulas for the conversion of frequencies to Mel frequencies are given below respectively (O'Shaughnessy 2000).

$$m = 2595 \log_{10}\left(1 + \frac{f}{700}\right) \tag{2.1}$$

$$f = 700\left(10^{\frac{m}{2595}} - 1\right) \tag{2.2}$$

The instructions for extracting the Mel spectrogram features are as follows,

- Obtain classical spectrogram by using STFT (See Appendix B.1 for STFT).
- Convert amplitudes to decibels
- Convert frequencies to Mel frequencies by using formula (2.1)

The Mel spectrogram of a given raw signal in Figure 2.2 can be seen below.

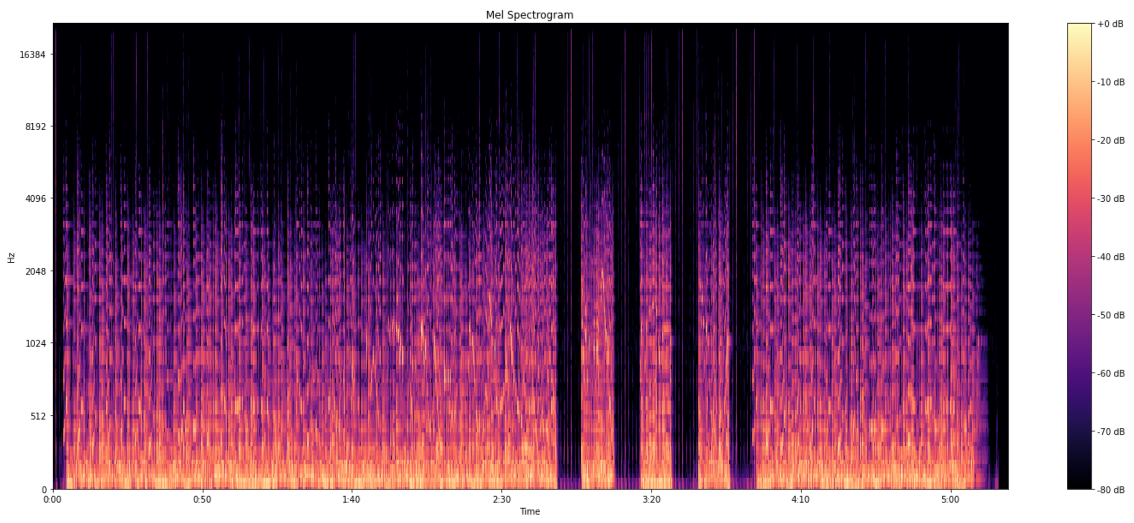


Figure 2.4. An Example of a Mel Spectrogram

2.3.2. Mel-Frequency Cepstral Coefficients (MFCC)

In order to understand these features clearly, we want to start with the concept of the cepstrum which is used the compute cepstral coefficients. The reason the process is called a cepstrum is that it computes the spectrum of a spectrum. The cepstrum was developed in 1963 by B.P. Bogert, M.J. Healy, and J.W. Tukey for seismic signal studies for the first time (Bogert, Healy and Tukey 1963). Thereafter, in the 1967, it was used in speech analysis to determine the voice pitch of a voiced speech signal (Noll 1967). Later on, cepstrum became commonly used tool for music processing applications.

Cepstrum can be computed as follows,

$$C\big(x(t)\big) = F^{-1}[\log\left(F[x(t)]\right)] \qquad (2.3)$$

where $C(.)$ represents the cepstrum of a signal, $x(t)$ is a time-domain signal, $F[.]$ and $F^{-1}[.]$ are DFT and IDFT respectively (See Appendix B.2 for DFT and IDFT). Let us visualize the process step by step to the ease the understanding. The following signal, $x(t)$, will be used as an input for the next steps.

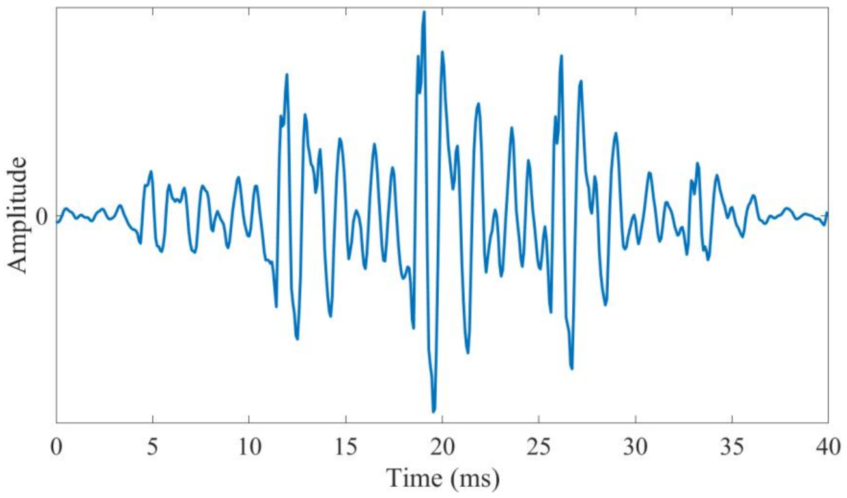


Figure 2.5. A Windowed Speech Signal

(Source: Bäckström 2019)

Step 1. After by taking the DFT of the input signal, $F[x(t)]$, we obtain the following signal

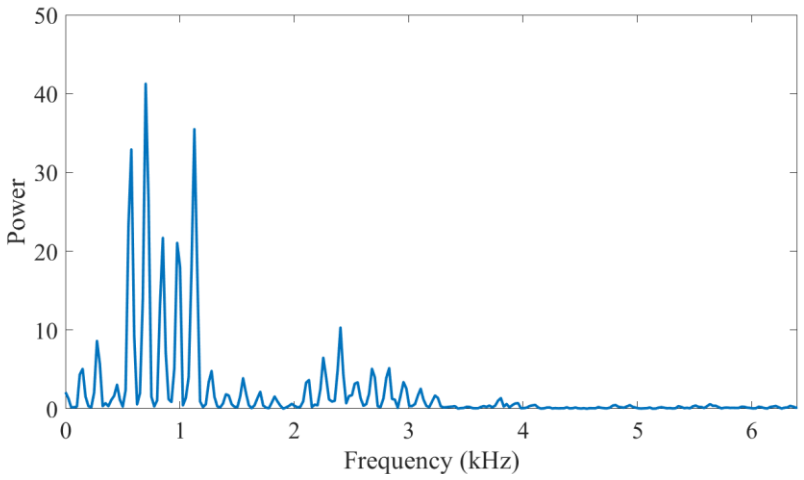


Figure 2.6. DFT of the Input Signal

(Source: Bäckström 2019)

Step 2. The log-power spectrum is obtained by taking the logarithm of the power spectrum, $log(F[x(t)])$

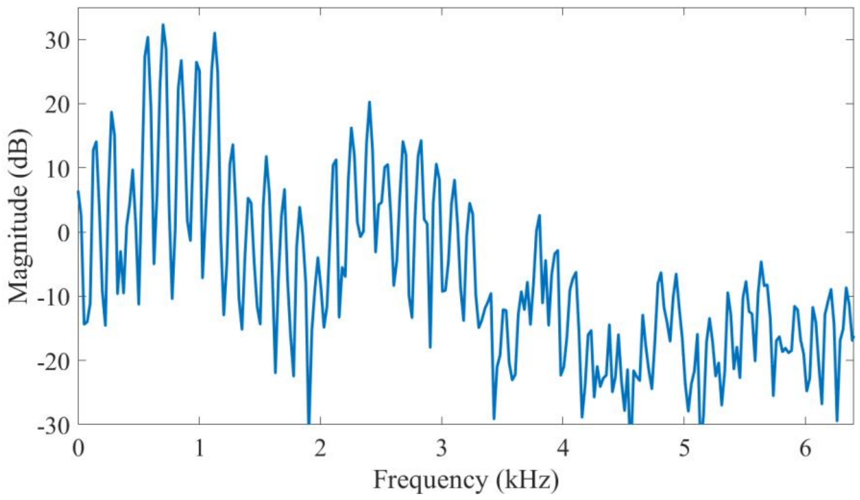


Figure 2.7. Log-Power Spectrum

(Source: Bäckström 2019)

Notice that the log-power spectrum has a continuous and periodic structure. This is the result of the harmonics in the input signal, $x(t)$. Thus, the log-power spectrum satisfies the conditions of the DFT.

Step 3. By taking the IDFT of the log-power spectrum, $F^{-1}[\log(F[x(t)])]$, we obtain the Cepstrum

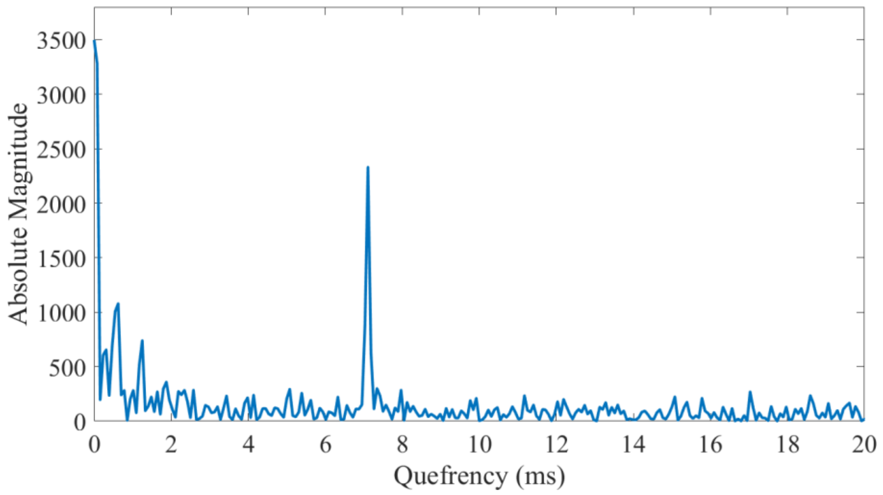


Figure 2.8. Cepstrum of the given Input Signal

(Source: Bäckström 2019)

The x-axis of the cepstrum is called Quefrency. In cepstrum, we compute the two time-frequency transforms of a time-domain signal. As a result of that, the cepstrum is somewhat similar to the time-domain signals so that the unit of the quefrency axis is given in milliseconds.

The peaks in cepstrum represents the presence of quefrencies in the log-power spectrum. In Figure 2.8, we have the highest peak around quefrency 7ms. These kinds of peaks are called rhamonic which is basically the cepstrum definition of harmonic. Rhamonics and quefrencies are related to fundamental frequency of the original signal so that it is used for peak detection. For example, the fundamental frequency, f_0, of the Figure 2.8 can be calculated as follows

$$\begin{aligned} f_0 &= \frac{1}{7ms} \\ &= 142.85\ Hz \end{aligned} \tag{2.4}$$

Since the concept of the cepstrum is clear now, we can calculate the MFCC by following the given instructions below.

- Take the DFT of the time-domain input signal
- Obtain the log-power spectrum by taking the logarithm
- Map the frequencies to Mel-Frequencies by using the Mel scale
- Obtain the MFCC by taking the DCT (See Appendix B.3 for DCT)

The reason of using DCT to obtain MFCC instead of IDFT is that to decorrelate the signal. This benefits the machine learning algorithms since we want inputs to be least correlated with each other. The MFCC of a given raw signal in Figure 2.2 can be seen below.

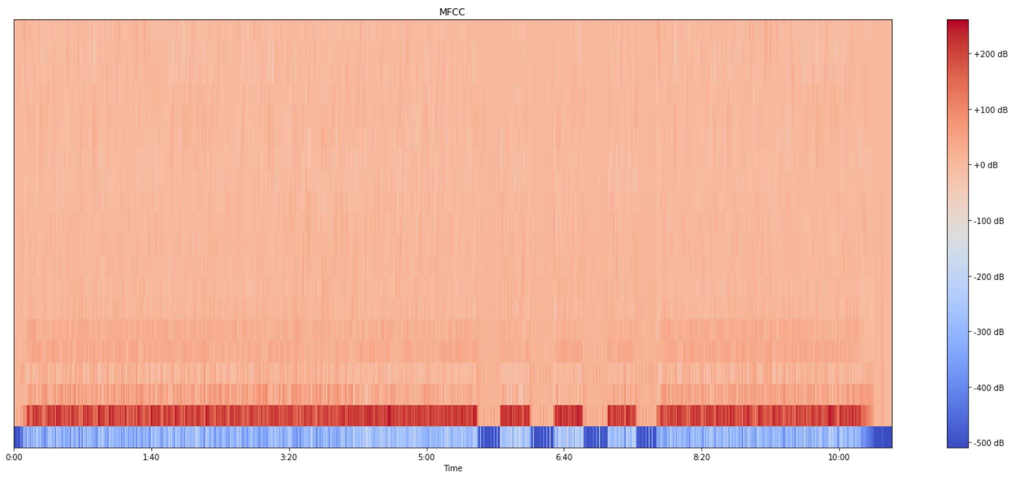


Figure 2.9. An Example of MFCC

2.3.3. Tempogram

Tempo is the speed that determines the how fast a part of the music plays. The unit of tempo is given by BPM. The tempo may change throughout the musical pieces, even in some we can observe more than one tempo, which is called tempi. The tempo information along with time signature forms the rhythm of a musical piece. In addition, it allows us to determine a note's duration.

Tempogram, which is developed for the characterization of tempi and local pulse of an audio file, represents the tempo information in an audio file (Tian et al. 2015). In the context of estimating tempo, many different methods have been proposed. The general approach to accomplish this task is usually based on using the novelty curve. Therefore, the local tempo and beat information can be estimated from novelty curve by using different methods such as autocorrelation (Ellis 2007), Fourier (Peeters 2005; Grosche and Muller 2009), or comb filter methods (Scheirer 1998) .

In order to compute the tempogram, first novelty curve should be calculated. The novelty curve, $\Delta: Z \rightarrow R_{\geq 0}$, of a given magnitude spectrum of an audio signal, $|X|$, can be calculated by using the following (Grosche, Muller and Kurth 2010),

$$\Delta(t) = \sum_{k=1}^{K} |Y(t+1,k) - Y(t,k)|_{\geq 0} \tag{2.5}$$

for $t \in Z$, where K is the number of Fourier coefficients and $Y = log(1 + C \cdot |X|)$ represents the log magnitude of the spectrum for a constant $C > 1$. The study (Grosche, Muller and Kurth 2010) discusses two different types of tempograms which are based on autocorrelation and Fourier Transform. The autocorrelation tempogram will further be explained since it was used in the thesis.

The calculation of autocorrelation tempogram starts by creating a box window, $W: Z \to R$, that is centered at $t = 0$ with window length N. Thus, the unbiased local autocorrelation, $\mathcal{A}(t, \ell)$, can be calculated as follows (Grosche, Muller and Kurth 2010)

$$\mathcal{A}(t, \ell) = \frac{\sum_{n \in Z} \Delta(n)\Delta(n + \ell) \cdot W(n - t)}{2N + 1 - \ell} \tag{2.6}$$

for time $t \in Z$ and time lag ℓ. Therefore, the autocorrelation tempogram, $\mathcal{T}^A(t, \tau)$, is defined as (Grosche, Muller and Kurth 2010)

$$\mathcal{T}^A(t, \tau) = \mathcal{A}(t, \ell) \tag{2.7}$$

where $\tau = 60/(r \cdot \ell)$ represents the tempo, and the r (in the study $r = 0.023\ seconds$) corresponds to the step size of each time parameter, t.

The tempogram of a given raw signal in Figure 2.2 can be seen below.

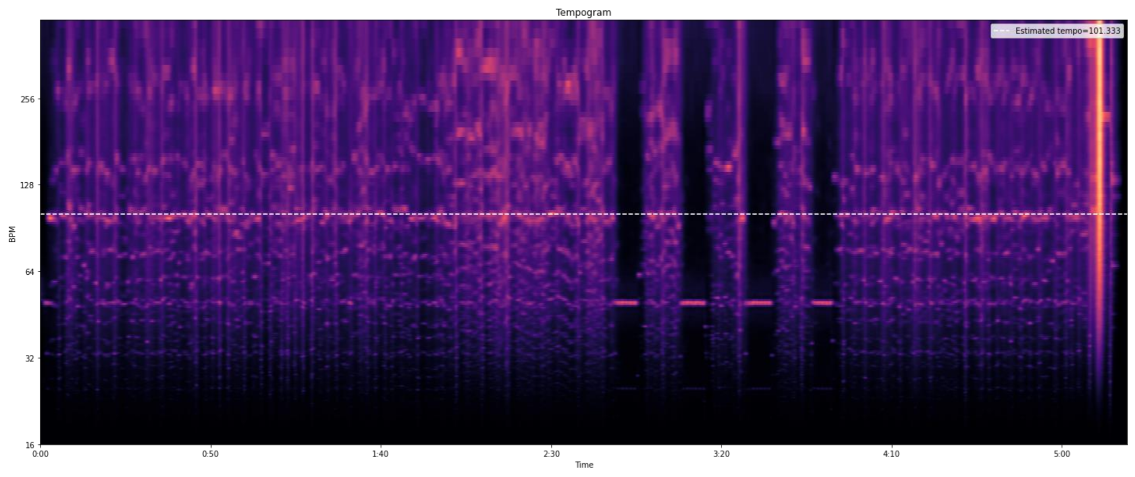


Figure 2.10. An Example of Tempogram

2.3.4. Zero-Padding

Zero-padding is a process of adding zeros to missing input time instants. The method is used for different purposes depending on the application. It is used to preserve the features at the edges of pictures in convolutional neural networks. Another application area of this method is in NLP (Natural Language Process) neural networks where it is used to equalize the size of sentences. Zero padding can be applied either in front of the sentences or at the end of them. Location of zeros don’t affect the network training since input gets masked. Specifically for our thesis application and NLP applications, the padding value doesn't have to be zero. Any number that is not in the vocabulary can be used.

2.3.5. Masking

Zero padding is applied because neural networks require the same sized input vectors. As a consequence of zero padding, the original information may be lost or get less attention by the network. Thus, the network may work poorly. In order to overcome this situation masking is used. Masking is a process to tell the network to skip missing timesteps/zero padded values.

2.3.6. Categorical Variable Encoding

The performance of the neural network depends on the model and the hyperparameters used, as well as the type of the data given to the input ("What Is Categorical Data | Categorical Data Encoding Methods" 2020). Due to its nature, artificial neural networks cannot receive and process raw data (e.g., sentences, symbolic music note notations, etc.) as it is. The input is expected to be in form of numerical values. Therefore, categorical variable encoding becomes a necessary step.

Categorical variable encoding is used to encode the categories in a dataset into numerical categorical variables. Datasets are divided into two main groups according to their variable properties. These are numerical (quantitative) and categorical (qualitive).

Datasets usually belong to one of the further division of variables as described in the figure below.

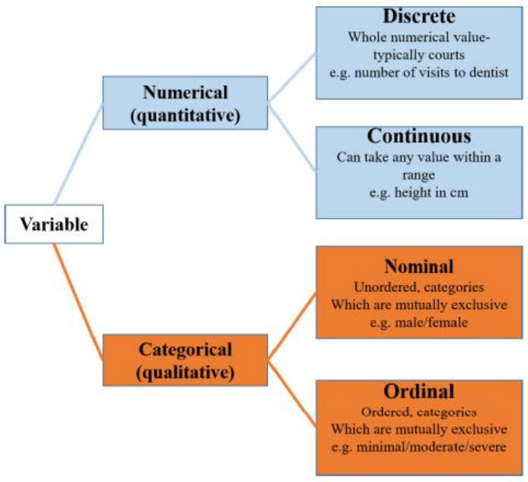

Variable
Numerical (quantitative)
Discrete
Whole numerical value-typically courts
e.g. number of visits to dentist
Continuous
Can take any value within a range
e.g. height in cm
Categorical (qualitative)
Nominal
Unordered, categories
Which are mutually exclusive
e.g. male/female
Ordinal
Ordered, categories
Which are mutually exclusive
e.g. minimal/moderate/severe

Figure 2.11. Variables: Quantitative (Numerical) vs Qualitative (Categorical) (Source: Dahouda and Joe 2021)

In an encoding scheme, number of unique categories are stored in a vector called vocabulary. There are several different types of variable encoding techniques. In this context, we describe the three most used ones, but emphasizing on the one-hot encoding since it is the preferred encoding scheme in the thesis

2.3.6.1. Label Encoding

The scheme for assigning integer values to each categorical variables are called label encoding. Label encoding is used in ordered categorical features. The following table can be given as an example for the label encoding.

Table 2.1. Label Encoder Example

Category	**Label Encoded Value**
Excellent	0
Good	1
Bad	2

This encoder example will inefficient because the algorithm won’t be able to understand the reasoning behind the given values. For this reason, one-hot encoding is usually preferred.

2.3.6.2. Binary Encoding

Binary encoding is an ordered encoding scheme. Categories are label encoded first, then label encodes are transformed to their corresponding binary value. Each bit in the binary number corresponds to a feature.

Table 2.2. Binary Encoding Example

Category	**Label Encoded Value**	**Binary Value**	**Encoded Value**
Red	0	000	[0, 0, 0]
Green	1	001	[0, 0, 1]
Blue	2	010	[0, 1, 0]
Pink	3	011	[0, 1, 1]

If we look at the given example, there are several categories sharing the same features. For example, second feature is shared between blue and pink, third feature is shared between green and pink. These shared features lead to undesired predictions in the model especially as the dataset gets larger. This misrepresentation can be avoided by using one-hot encoding.

2.3.6.3. One-Hot Encoding

One-hot encoding technique encodes each categorical variable into a Boolean variable. One-hot encoding technique encodes each categorical variable into a Boolean variable. The value of 1 represents the presence of that category whereas 0 represents the absence of that category. The categories inside the vocabulary can be mapped to any numbers. In general, each categorical variable usually maps to their corresponding positional number. The following example can be given for one-hot encoding.

Assume that we have the given array of a dataset

$$fruits = [banana, watermelon, banana, apple, kiwi]$$

In this dataset, our vocabulary is

$$vocabulary_{fruits} = [banana, watermelon, apple, kiwi]$$

We have 4 different categories which means that the length of one hot representation of each category will be 4. Then, each categorical variable represented in one-hot encoded form is as follows

$$\begin{array}{cccc} 1. & banana & \rightarrow & [1,0,0,0] \\ 2. & watermelon & \rightarrow & [0,1,0,0] \\ 3. & apple & \rightarrow & [0,0,1,0] \\ 4. & kiwi & \rightarrow & [0,0,0,1] \end{array}$$

Thus, the one-hot encoded dataset becomes as follows

$$encoded_{fruits} = \begin{bmatrix} [1,0,0,0] \\ [0,1,0,0] \\ [1,0,0,0] \\ [0,0,1,0] \\ [0,0,0,1] \end{bmatrix}$$

In the end, we can see that the representation of each category is mutually exclusive from each other which results model to be more accurate. The disadvantage of this method is that it is a memory inefficient as the length of each category representation is equal to the vocabulary (Dahouda and Joe 2021). In order to overcome the inefficiency in large datasets, data pipeline can be used.

2.4. Artificial Neural Networks

In this section, we also want to explain the differences between artificial intelligence (AI), machine learning (ML), artificial neural network (ANN), and deep learning (DL). Although they are very close and often confused with each other, they are quite different. They can be considered subsets of one another (Kavlakoglu 2020).

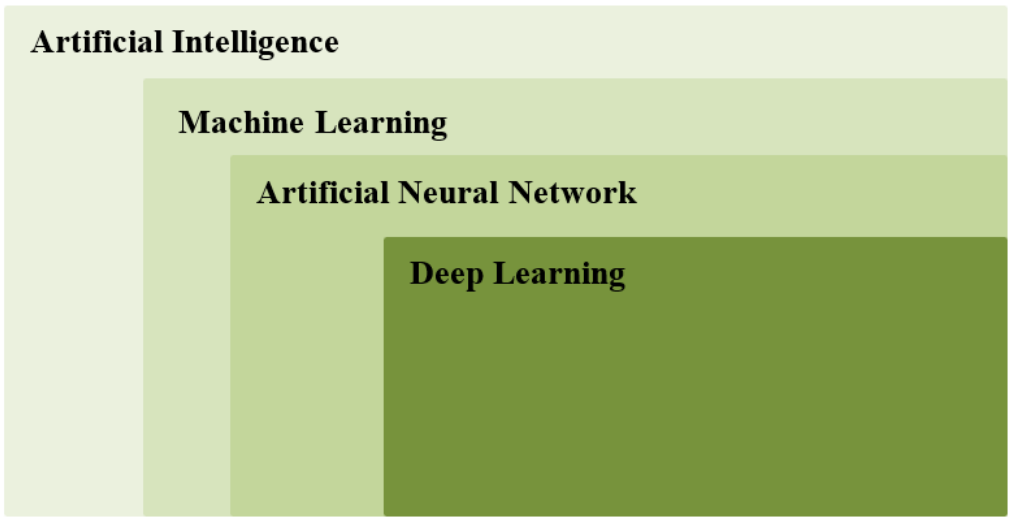


Figure 2.12. Set of Artificial Intelligence, Machine Learning, Artificial Neural Network and Deep Learning

Starting from top to the bottom, AI is known as mimicking the human behavior. AI requires defining every possible situation in programming by using if-else statements and loops. However, this doesn’t work very well. Imagine defining every situation and every combination of those situations for self-driving cars. There would be infinite numbers of situations and combinations, therefore defining every possible situation and combination is theoretically not possible. This problem is overcome by ML.

Machine learning uses human-provided features instead of explicit programming. ML is divided into two problems which are classification and regression. Classification problems, as the name suggests, are used to classify different categories such as dogs-cats, colors, etc. On the other hand, regression problems are used for continuous values. Predicting the cost of a car based on given features like its brand, construction year, transmission, etc., is a regression problem. A regression problem can be transformed into a classification problem. For example, instead of predicting the cost of a car in the given example, we can just map cost to categories like high, medium, or low. The problem with this technique requires a human to choose and provide the features. This is what makes it inefficient. This method is improved by using ANNs which take inputs and determine features without the help of humans.

Artificial neural networks are the subset of ML. The inspiration of ANNs is human neurons inside the brain (Education 2020). In our brain, there are billions of neurons that carry information by electrical signals. Inside a neuron, the dendrites receive the external signal which can be something we touch, see, taste, etc. This external signal is processed in the nucleus and turns to an output signal which is carried with axons. These axons lead to the next neuron dendrites. The connection between the previous and next neuron is provided with synapses between axon tips and dendrites (Mahanta 2017).

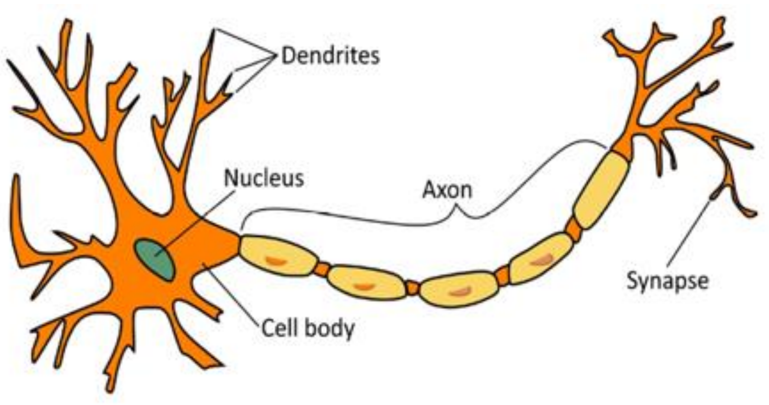


Figure 2.13. Neuron Structure

The simplest mathematical model inspired by a neuron structure is called perceptron. Perceptron is a single layered ANN structure.

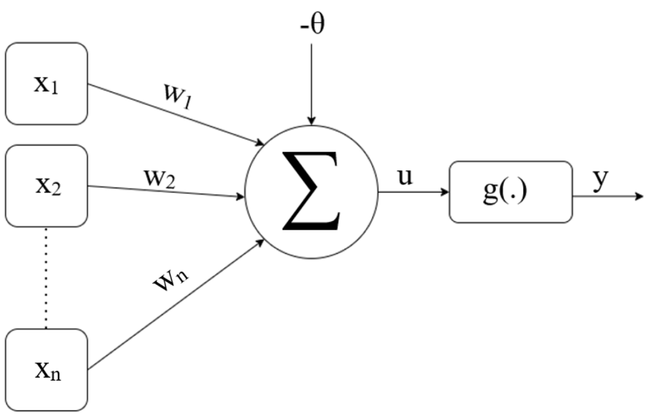


Figure 2.14. Perceptron Architecture

As can be seen in the figure above, there are similarities between neuron structure and perceptron architecture. Dendrites, biological input signal receivers, are similar to inputs in the architecture. The connections between inputs and perceptron are quantified by weights which simulate activation levels of synapses. The neuron structure's nucleus corresponds to the perceptron which combines weighted input signals to compute output. Lastly, axons, which carry information to the next neurons, are represented by output.

In this architecture, a set of input signals $\{x_1, x_2, \ldots, x_n\}$ are presented to the neuron. The input signals are used to calculate the weighted sum with a set of synaptic weights $\{w_1, w_2, \ldots, w_n\}$ and bias term θ is added. At the end of this calculation, activation potential u is obtained. Finally, the output is computed by employing the activation function, $g(.)$, to the activation potential. Two mathematical expressions to calculate the output of the perceptron, proposed by McCulloch and Pitts, is given as follows (da Silva et al. 2017)

$$u = \sum_{i=1}^{n} w_i x_i - \theta \tag{2.8}$$

$$y = g(u) \tag{2.9}$$

For every training sample, ANN updates its weights to become more accurate. During training, a loss is calculated. In the simplest way, the loss can be computed by taking the absolute value of the difference between the true value and the output produced by the network. Then this loss along with the learning rate of activation function, η, is used to compute the updated weights. The vector expression for weight updates (da Silva et al. 2017) can be given by

$$loss = \left|d^{(k)} - y\right| \tag{2.10}$$

$$\boldsymbol{w}^{current} = \boldsymbol{w}^{previous} + \eta \cdot \left(d^{(k)} - y\right) \cdot \boldsymbol{x}^{(k)} \tag{2.11}$$

where $\boldsymbol{x}^{(k)}=[-1 \;\; x_2^{(k)} \;\; \ldots \;\; x_n^{(k)}\;]^T$ represents the kth training sample, vector $\boldsymbol{w}=[\theta \;\; w_1 \;\; \ldots \;\; w_n]^T$ contains bias term and weights, $d^{(k)}$ denotes the desired value for kth training sample.

The training is completed when the desired loss value reached. There are different types of losses and activation functions depending on the problem type. Regression problems generally use mean absolute error (MAE), mean squared error (MSE), root mean squared error (RMSE), and such. On the other hand, binary cross entropy, categorical cross entropy, hinge losses are the some of the loss functions that are used for classification problems. Furthermore, Sigmoid, SoftMax, tanh, ReLU, Leaky ReLU can be given as examples for different activation functions.

Deep learning, subset of ANN, is basically an improved version of ANNs. The main difference between ANN and DL is the number of hidden layers inside the model.

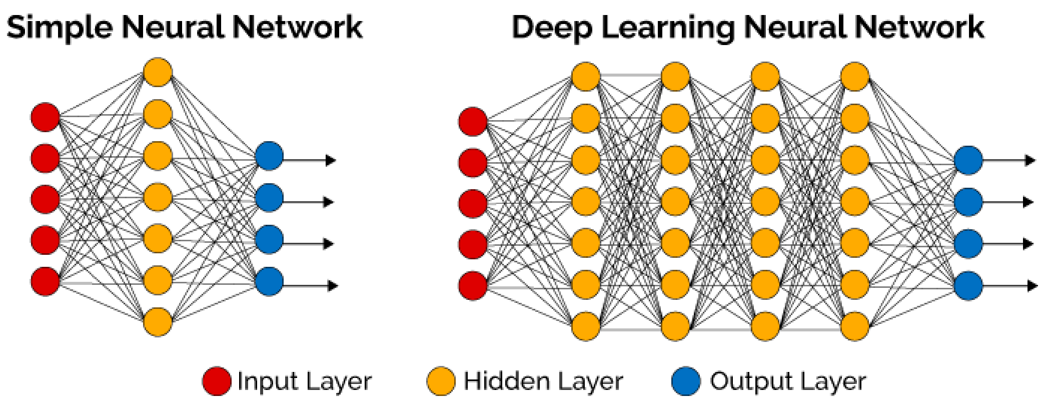


Figure 2.15. Neural Network and Deep Learning Architectures

(Source: Mostafa et al. 2020)

As the number of hidden layers goes up, the accuracy of the output is improved. Despite the positive effect of the additional in hidden layers, a proper number of hidden layers should be chosen. Any more or less than a sufficient number of hidden layers could cause overfitting or underfitting. If the number of hidden layers is larger than the complexity of the problem, then overfitting occurs. Underfitting, occurs when the number of hidden layers is smaller than the complexity of the problem (Uzair and Jamil 2020). Besides the number of hidden layers, another factor that determines the performance of deep learning is the optimization technique that is employed. In general, an artificial neural network and a deep learning update its weights to minimize the loss by finding the global or local minima. This is expected to be done in the shortest time and with high accuracy (Okewu, Adewole and Sennaike 2019). In order to do this efficiently, there are various optimization techniques that are specific to each model and each problem.

Gradient Descent, Stochastic Gradient Descent, RMSprop, Adagrad, AdaDelta, Adam are some of the widely used optimization algorithms.

2.4.1. Activation Functions

Activation functions are nonlinear functions with the purpose of helping the ANNs to classify the categories correctly by creating to the decision hyperplane or decision surface that separates the categories from each other during the training phase. In other words, it helps to separate different classes even if they are not linearly separable.

2.4.1.1. Sigmoid

Sigmoid is a “S” shaped non-linear function. The expression of sigmoid (Nilsson 1998) is given by

$$f(s) = \frac{1}{1 + e^{-s}} \tag{2.12}$$

where, *s* corresponds the input and *f(s)* is the output. The sigmoid function is shown in the figure below.

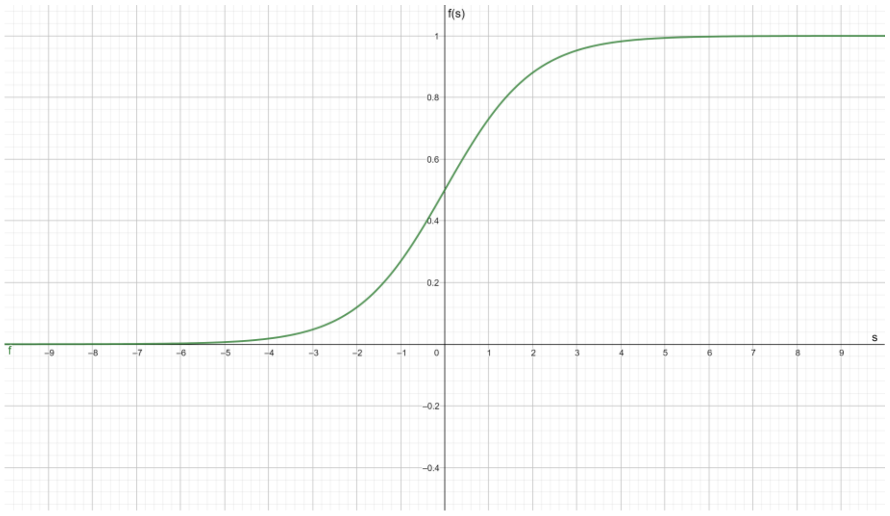


Figure 2.16. Sigmoid Function

In neural network models, as data gets more complicated, we can add more hidden layers. If Sigmoid is used for every hidden layer, model will eventually stop learning. This is caused because Sigmoid squeezes information. This causes the gradient to get smaller and smaller during backpropagation. This is known as vanishing gradient problem. The negative effect of the Sigmoid function can be alleviated by using different activation functions.

2.4.1.2. Hyperbolic Tangent (tanh)

Hyperbolic tangent (tanh) is one of the activation functions that helps ANN to prevent vanishing gradient problem. However, it has a negative effect on training. Due to its saturation rate, it is slower in terms of training time (Krizhevsky, Sutskever and Hinton 2017). This function can be expressed as:

$$\begin{aligned} f(x) &= \tanh(x) \\ &= \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \end{aligned} \tag{2.13}$$

where, *x* corresponds the input and, *f(x)* is the output. The tanh function is shown in the figure below.

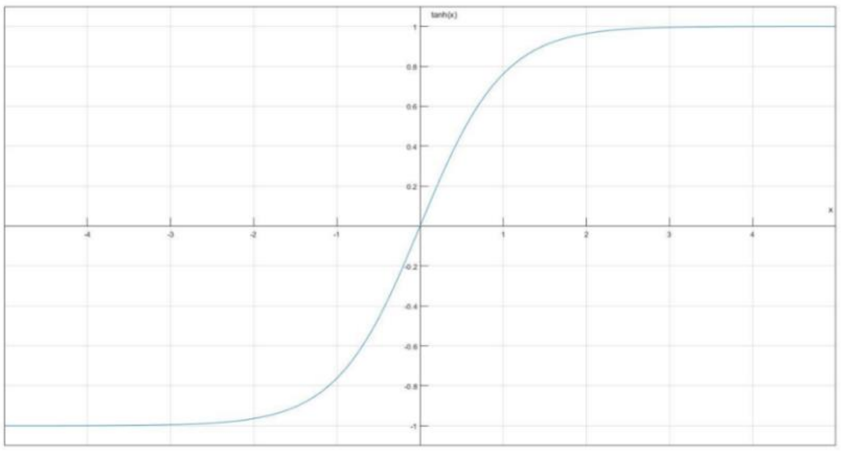


Figure 2.17. Hyperbolic Tangent Function

2.4.1.3. SoftMax

SoftMax is an activation function that takes inputs and scales them to relative probabilities. It is used for multiclass classification problems where it represents probabilities of each class (Nwankpa et al. 2018). SoftMax function (Goodfellow, Bengio and Courville 2016) is defined as

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{2.14}$$

where x_i corresponds to elements of the vector $\boldsymbol{x}$, and n is the number of classes. SoftMax function can be visualized from the given figure below.

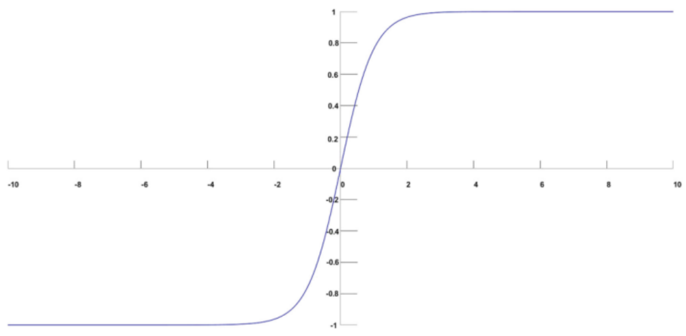


Figure 2.18. SoftMax Activation Function

(Source: Shen et al. 2018)

2.4.2. Categorical Cross Entropy Loss

Categorical cross entropy is a loss function used to classify multiple classes. This loss is very good at distinguishing between two discrete probability distributions. The formula to calculate categorical cross entropy is given by

$$Loss = -\sum_{i=1}^{\substack{output\\size}} y_i \log \hat{y}_i \tag{2.15}$$

where *$\hat{y}_i$* is the *i*th model output value, *y_i* denotes the *i*th target value. The purpose of the minus sign in front of the equation is to reduce the loss due to the convergence of the probability distributions. Categorical cross entropy is usually preferred for one-hot encoded data.

2.4.3. Adam Optimizer

Adam gets its name from the adaptive moment estimation. It is a first order stochastic gradient-based optimization technique with little memory requirement. The strength of the method comes from the use of the adaptive learning rate method to find the individual learning rates for each parameter (Kingma and Ba 2014).

Adam adapts the learning rate by utilizing the first and second moments of gradient estimations. Moments can be calculated by taking the expected value of a random variable. Therefore, Nth moment of a random variable is given by

$$m_n = E[X^n] \tag{2.16}$$

where *m* denotes moment, *E[.]* is the expectation operator (See Appendix B.4 for Expected Values), *X* is the random variable.

The gradient of the cost function of a neural network is usually evaluated on some random batches, because of that it can be considered as a random variable. Exponential moving average is used by this method to estimate the moments. Then, moving averages on a mini batch is given as follows

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{2.17}$$

$$v_t = \beta_2 \, v_{t-1} + (1 - \beta_2) g_t^2 \tag{2.18}$$

where *t* denotes the timestep, *m* denotes the moving averages of the gradient *v* is the moving averages of the squared gradient, g corresponds to gradient on a mini batch, *β_1* and *β_2* are the exponential decay rates which are hyperparameters with suggested default values 0.9 and 0.999 respectively (Kingma and Ba 2014).

In order to write the equation for weight updates with Adam optimizer, some preprocessing is needed. First of all, we know that moving average vectors are initialized to 0's at the first iteration. We also know that g is considered to be a random variable because it is evaluated on random mini data batches. In addition, first and second moments are given by estimates of m and v. Then by using Eq. (2.16), we expect the following properties

$$E[m_t] = E[g_t] \tag{2.19}$$

$$E[v_t] = E[g_t^2] \tag{2.20}$$

If the given properties hold, this leads us to have an unbiased estimator. However, moving averages don't hold these properties. This can be easily shown by remembering that moving average vectors are initialized with 0's. Then Eq. (2.17) can be unpacked as follows

$$m_0 = 0 \tag{2.21}$$

$$\begin{aligned} m_1 &= \beta_1 m_0 + (1 - \beta_1) g_1 \\ &= (1 - \beta_1) g_1 \end{aligned} \tag{2.22}$$

$$\begin{aligned} m_2 &= \beta_1 m_1 + (1 - \beta_1) g_2 \\ &= \beta_1 (1 - \beta_1) g_1 + (1 - \beta_1) g_2 \end{aligned} \tag{2.23}$$

$$\begin{aligned} m_3 &= \beta_1 m_2 + (1 - \beta_1) g_3 \\ &= \beta_1^2 (1 - \beta_1) g_1 + \beta_1 (1 - \beta_1) g_2 + (1 - \beta_1) g_3 \end{aligned} \tag{2.24}$$

$$\vdots$$

Recall that m is initialized with 0's. Then using all the previous timesteps of moving average, m_t, can be written as

$$m_t = (1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i}\, g_i \tag{2.25}$$

By following the same steps, similar equation for moving averages of squared gradient can be written as

$$v_t = (1-\beta_2)\sum_{i=1}^{t}\beta_2^{t-i}\, g_i^2 \tag{2.26}$$

Adam uses bias correction terms. The bias correction term for moving average gradient, m_t, can be calculated by first taking the expected value of Eq. (2.25). Then, g_i is approximated with g_t and a new term ζ is introduced to represent the error resulting from this approximation.

$$\begin{aligned} E[m_t] &= E\left[(1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i}\, g_i\right] \\ &= E[g_t](1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i} + \zeta \\ &= E[g_t](1-\beta_1^t) + \zeta \end{aligned} \tag{2.27}$$

Applying similar steps for v_t yields as,

$$\begin{aligned} E[v_t] &= E\left[(1-\beta_2)\sum_{i=1}^{t}\beta_2^{t-i}\, g_i^2\right] \\ &= E[g_t^2](1-\beta_2)\sum_{i=1}^{t}\beta_2^{t-i} + \zeta \\ &= E[g_t^2](1-\beta_2^t) + \zeta \end{aligned} \tag{2.28}$$

If ζ = 0, this means that $E[g_t]$ and $E[g_t^2]$ are stationary. If not, ζ should be kept small by choosing proper values for exponential decay rates β1 and β2 to assign small weights to gradients. Then, final bias corrected estimators, m̂t and ĝt, for mt and vt can be given as follows

$$\widehat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.29}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{2.30}$$

Finally, weight updates using Adam optimizer can be written as (Bushaev 2018)

$$w_t = w_{t-1} - \eta \frac{\widehat{m}_t}{\sqrt{\hat{v}}_t + \in} \tag{2.31}$$

where w_t denotes weight of the current timestep, w_{t-1} denotes weight of the previous timestep, η represents the learning rate, $\hat{m}_t$ is bias corrected estimator of moving average of gradient, $\hat{g}_t$ is bias corrected estimator of moving averages of squared gradient, and ∈ denotes the step size.

2.4.4. Long Short-Term Memory Architecture

Long Short-Term Memory (LSTM) is first presented by Sepp Hochreiter and Jürgen Schmidhuber in 1997. LSTM is an efficient recurrent network architecture that uses gradient-based learning. It is proposed to overcome exploding and vanishing gradient problem that occurs in recurrent neural networks (RNN) (Hochreiter and Schmidhuber 1997) (See Appendix C for Vanishing Gradient Problem). LSTMs are good for training time series because of the cell state and the hidden state they have. The cell state is the memory of the model that stores useful information for use in subsequent layers. The hidden state is a working memory that decides which information is relevant to write or to forget.

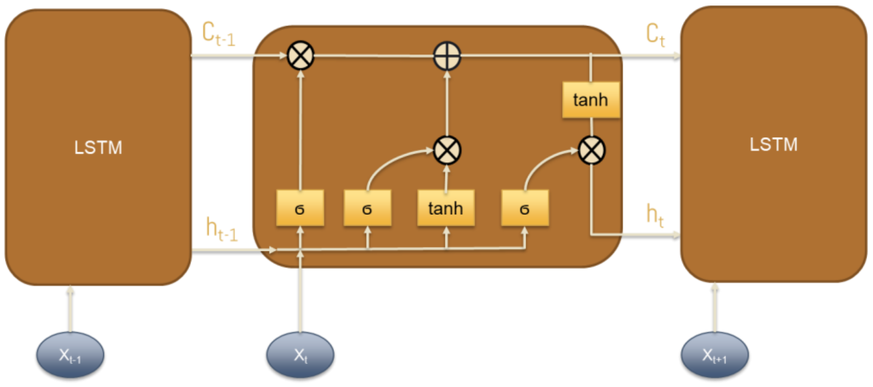


Figure 2.19. An Illustration of LSTM Cell

An LSTM cell consists of three gate structures which are forget gate, input gate and output gate. Additionally, it has a memory cell that progresses by making small changes without any hindrance.

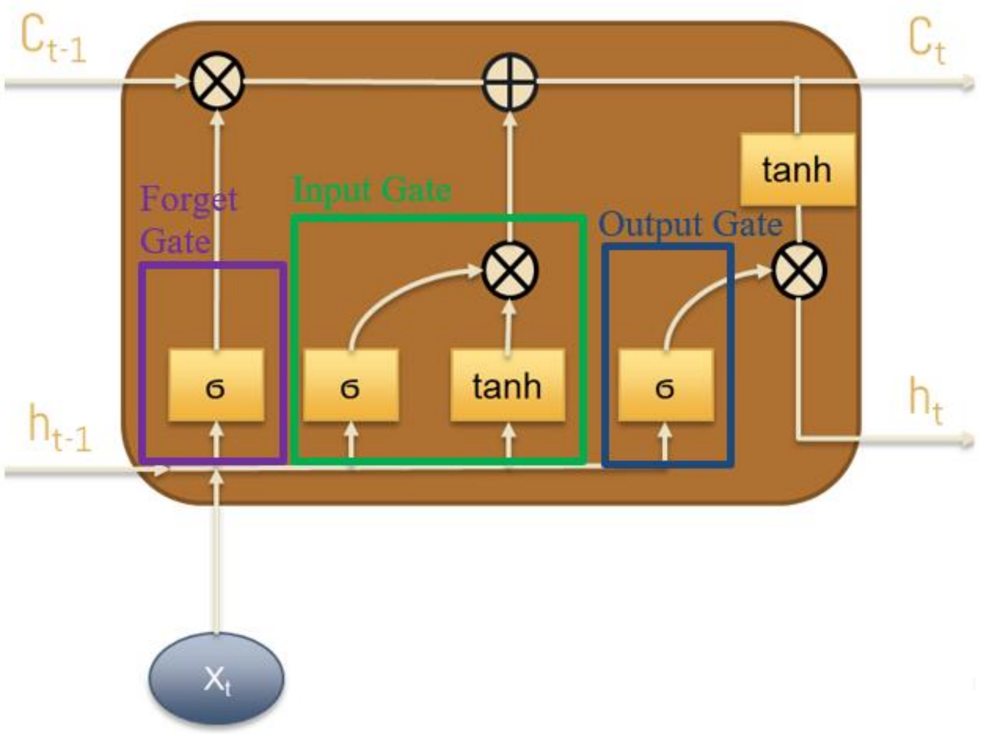


Figure 2.20. Inside of an LSTM Cell

In the given structure, $\vec{x}_t$ denotes the input vector, $\vec{h}_{t-1}$ denotes previous hidden state, $\vec{c}_{t-1}$ is the previous cell state, $\vec{c}_t$ is the current cell state, and $\vec{h}_t$ corresponds to current hidden state. Activation function sigmoid is represented by σ symbol.

The forget gate's job is to forget unnecessary information from the previous cell state. This gate uses sigmoid activation to perform this action. It is known that the output of sigmoid activation is between 1 or 0. Using this information and applying element-wise multiplication between the output of the forget gate and the previous cell state, the gate determines which value will be forgotten. The equations for the forget gate are given by (Olah 2015)

$$\vec{f}_t = \sigma\left[\vec{W}_{f_h}\vec{h}_{t-1} + \vec{W}_{f_x}\vec{x}_t + \vec{b}_f\right] \tag{2.32}$$

$$\vec{C}_{t_f} = \vec{C}_{t-1} \odot \vec{f}_t \tag{2.33}$$

where $\vec{f}_t$ is the output of the first sigmoid activation function, $\vec{W}_{f_h}$ is the weight vector of forget gate's hidden state, $\vec{W}_{f_x}$ denotes the weight vector forget gate's input, $\vec{b}_f$ is the bias vector of the forget gate and $\vec{C}_{t_f}$ denotes the updated cell state. The $\odot$ denotes the Hadamard Product (See Appendix B.5 for Hadamard Product).

The goal of the input gate is to determine which values of the input is worth to remember. The input gate uses two activation functions which are sigmoid and tanh. The duty of sigmoid function is the same as forget gates. It determines which input information is important. The tanh's job is to choose candidates for the input gate. At the end, the network is able to forget unnecessary candidates and keeps relevant information from the input. Finally, outputs of these activation functions are element-wise multiplied and added to updated cell state, $\vec{C}_{t_f}$, to use as the next LSTM cell's previous cell state. The updated equations for the input gate are given by (Olah 2015)

$$\vec{\imath}_t = \sigma\left[\vec{W}_{i_h}\vec{h}_{t-1} + \vec{W}_{i_x}\vec{x}_t + \vec{b}_i\right] \tag{2.34}$$

$$\vec{g}_t = tanh\left[\vec{W}_{g_h}\vec{h}_{t-1} + \vec{W}_{g_x}\vec{x}_t + \vec{b}_g\right] \tag{2.35}$$

$$\vec{\tilde{C}}_t = \vec{\imath}_t \odot \vec{g}_t \tag{2.36}$$

$$\vec{C}_t = \vec{C}_{t_f} \odot \vec{\tilde{C}}_t \tag{2.37}$$

where $\vec{\iota}_t$ denotes the output of the second sigmoid activation function, $\overrightarrow{W}_{i_h}$ is the weight vector of input gate's hidden state, $\overrightarrow{W}_{i_x}$ denotes the weight vector input gate's input, $\vec{b}_i$ is the bias term of the input gate, $\vec{g}_t$ denotes the candidate variables which is the output of the first tanh activation, $\overrightarrow{W}_{g_h}$ is the weight vector of candidate variables' hidden state, $\overrightarrow{W}_{g_x}$ denotes the candidate variables' input, $\vec{b}_g$ corresponds to the bias vector of the candidate variables.

The last variable, the hidden state for the next LSTM cell's previous hidden state, is calculated by the output gate. The output gate takes previous hidden state $\vec{h}_{t-1}$, the input $\vec{x}_t$ and put them through the third sigmoid activation function. Next, the cell state $\vec{C}_t$, is passed through tanh function. Lastly, the outputs of the activation functions are element-wise multiplied to decide which information should be carried over to next cell. The equations for the output gate are given by (Olah 2015)

$$\vec{o}_t = \sigma\left[\overrightarrow{W}_{o_h}\vec{h}_{t-1} + \overrightarrow{W}_{o_x}\vec{x}_t + \vec{b}_o\right] \tag{2.38}$$

$$\vec{h}_t = \tanh\left(\vec{C}_t\right) \odot \vec{o}_t \tag{2.39}$$

where $\vec{o}_t$ corresponds to output of the third sigmoid activation, $\overrightarrow{W}_{o_h}$ is the weight vector of output gate's hidden state, $\overrightarrow{W}_{o_x}$ denotes the weight vector of output gate's input and lastly $\vec{b}_o$ is the bias vector of the output gate.

2.4.5. Sequence-to-Sequence Model

Sequence-to-sequence (Seq2Seq) is a two multilayered LSTM (Sutskever, Vinyals and Le 2014) that has been popularly used in real life problems such as Google's Machine Learning Translation (Wu et al. 2016) and Apple's Siri speech to text (Siri Team 2017). The first multilayered LSTM, known as encoder, takes input in form of a sequence and turns into a context vector. The context vector is a fixed-length vector that is used to capture the similarity of use among the input (Gallant 2000). The second multilayered LSTM, known as decoder, decodes the target sequence from the context vector to output.

In order to train a Seq2Seq model, encoder's and decoder's input sequence requires a special token to represent the end of the sequence (EOS), but decoder's input sequence requires an additional special token to represent the start of the sequence (SOS). The aim of encoder is to understand the meaning of input sequence and generate the context vector for the decoder. The encoder stops when EOS is received, the context vector and the SOS is passed to the decoder. The decoder starts generating outputs sequentially when the SOS is received. The training is completed when the EOS is generated by the decoder.

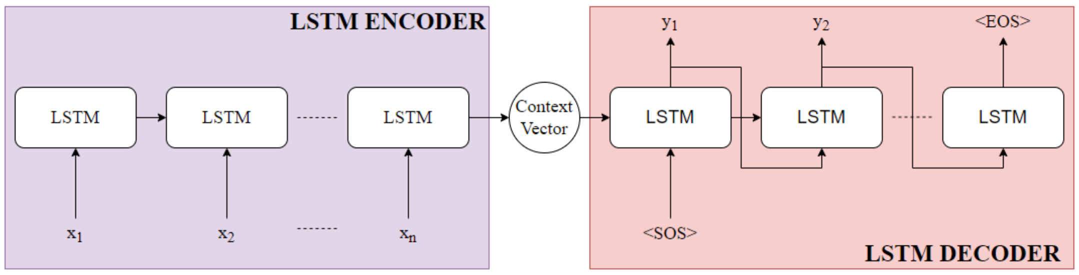


Figure 2.21. Seq2Seq Encoder-Decoder Model

To facilitate the understanding let's assume that we have an input sequence $\vec{X}$ and an output sequence $\vec{Y}$ represented as the following equations ("Write A Sequence To Sequence (Seq2seq) Model — Chainer 7.8.0 Documentation", n.d.)

$$\begin{aligned}\vec{X} &= \{x_1, x_2, \dots, x_n\} \\ &= (x_i)_{i=1}^{n}\end{aligned} \tag{2.40}$$

$$\begin{aligned}\vec{Y} &= \{y_1, y_2, \dots, y_n\} \\ &= (y_j)_{j=1}^{n}\end{aligned} \tag{2.41}$$

The model generates output by using the conditional probability. In high level, we can say that we are generating an output sequence $\vec{Y}$ given by an input sequence $\vec{X}$ whose conditional probability can be written as ("Write A Sequence To Sequence (Seq2seq) Model — Chainer 7.8.0 Documentation", n.d.)

$$P(\vec{Y}|\vec{X}) = \frac{P(\vec{Y} \cap \vec{X})}{P(\vec{X})} \tag{2.42}$$

where, *P(.)* is the probability operator. However, the model does not use the given Eq. (2.42). Recall that Seq2Seq gets a sequence of inputs and starts generating outputs sequentially. Then, the actual probability that model uses is given by

$$P(y_j|\vec{Y}_{<j}, \vec{X}) = \frac{P(y_j \cap (\vec{Y}_{<j}, \vec{X}))}{P(\vec{Y}_{<j}, \vec{X})} \tag{2.43}$$

where y_j is the jth element of output sequence, $\vec{Y}_{<j}$ denotes the output sequence from 1 to j-1. By using this notation, Eq. (2.42) can be rewritten as

$$P_\theta(\vec{Y}|\vec{X}) = \prod_{j=1}^{n+1} P_\theta(y_j|\vec{Y}_j, \vec{X}) \tag{2.44}$$

So far, this was the basic structure of a Seq2Seq model was given. This model's training can be improved by using Teacher Forcing Method. The teacher represents the ground truth (Keneshloo et al. 2019). The ground truth is given to decoder during training for improvement. As a result of that the model converges faster and the predictions are improved.

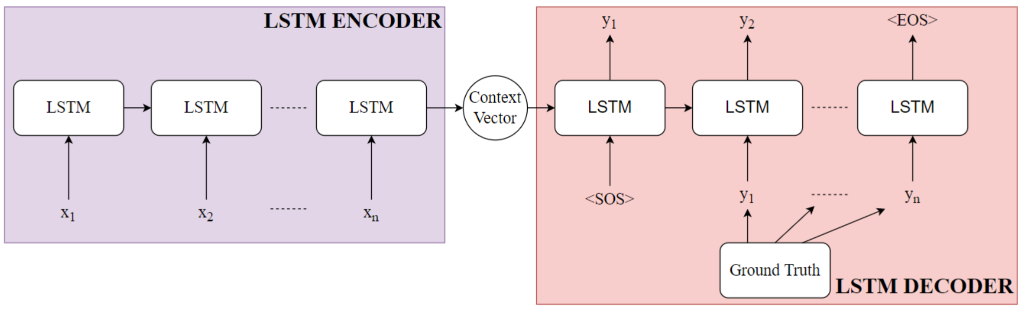


Figure 2.22. Seq2Seq Model with Teacher Forcing

2.4.6. Temperature Sampling

Temperature sampling, a standard technique for language models, is used to improve the quality of samples (Stewart 2016). The sampling method is inspired by statistical thermodynamics. In statistical thermodynamics, the probability of encountering low energy states is more likely when the temperature is high. The same idea can be applied to probability models. In probability models, logits, which are the predictions, correspond to energy states. The logits are divided by the temperature values before the SoftMax (Mann 2019). The temperature sampling expression is given by

$$q_i = \frac{e^{(\frac{z_i}{T})}}{\sum_j e^{(\frac{z_j}{T})}} \tag{2.45}$$

where, q_i is the element of $\vec{q} = \{q_1, q_2, \dots, q_n\}$ which is the new probabilities, and both z_i and z_j are the elements of $\vec{z} = \{z_1, z_2, \dots, z_n\}$ which is the previous probabilities.

As $T \to 1$, the sampling method generates more repetitive and low-quality samples. On the contrary, samples produced as $T \to 0$ will be more diverse and unique. When $T = 0$, model is corresponding to max likelihood and as $T \to \infty$ model turns into a uniform sampling.

CHAPTER 3

MIDI DRUM ACCOMPANIMENT NETWORK

Whatever the mind can conceive and believe,
it can achieve.
-Napeloen Hill-

3.1. Introduction

In this chapter, we describe the steps of the development of a midi drum accompaniment network. The main idea of the midi drum accompaniment network is to create a relationship between drum patterns and other instruments. In order to accomplish the idea, the dataset contents are sampled by time as the first step. Secondly, these samples are separated into two different vectors which are going to form the input and output of the network for training. The input vector contains every information of all the instruments' note pitches and velocities except for the drums. The output vector only contains the drums' note pitches and velocities. However, for the output vector, we have considered the most used 22 drum notes instead of every MIDI note. The reason is that to simplify the model and eliminate the unnecessary information. In generation part, the same idea is applied. However, in this part only the input vector is taken into the consideration and previously trained weights are used. A user interfaces are developed for the network and players.

The following sections are organized as follows. In Section 3.2, we describe the proposed work to train and develop the midi drum accompaniment network. In Section 3.2.1, we provide a detailed explanation of preprocessing steps of the midi dataset. In Section 3.2.2, we explain the neural network training. In Section 3.2.3, we provide information on generating new unique drum patterns. In Section 3.3, we summarize and present the training and prediction results.

3.2. Proposed Work

We propose using Seq2Seq model on MIDI data not to create new songs but make an accompaniment network. Seq2Seq model along with LSTM architecture works well on time series data. Since each instrument's note in a MIDI data has their own time to play, it is considered to be a time series data. The main idea of Seq2Seq is to make a relationship between two sequences. In the thesis, we trained the network by using both note pitches and velocities. By using the combination of two information, generated patterns become more humanized. In the generation part, we used a sampling method that has been used widely in NLP applications. This sampling method is called temperature sampling. Temperature sampling increases the quality of generated patterns so that they become more unique and more exciting. In addition, we add a user parameter called complexity which is basically the temperature value in the sampling. The complexity value allows user to choose if generated patterns have a more complex combination of notes or not.

The network is trained by using six different datasets are used. These datasets are from the different genres which are rock, pop, hiphop-rap, jazz, blues, and the combination of these genres which is named as the shuffle. The flowchart of the training and generating part of the network can be seen in the following figure

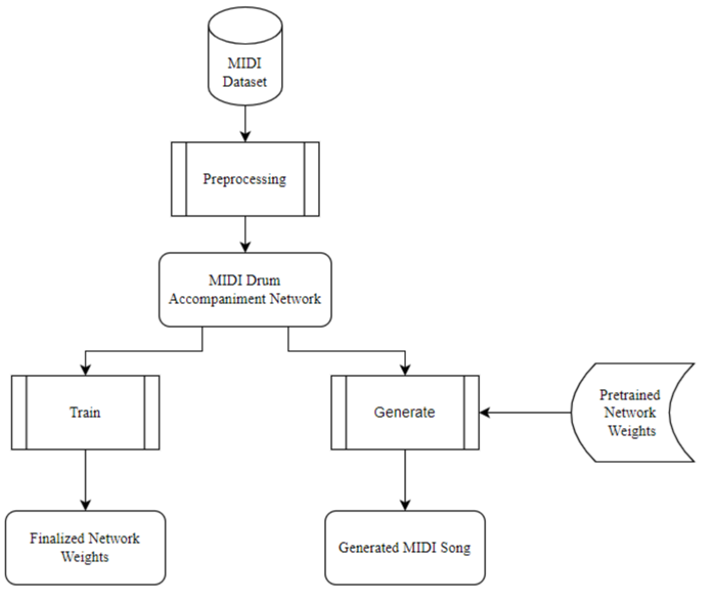
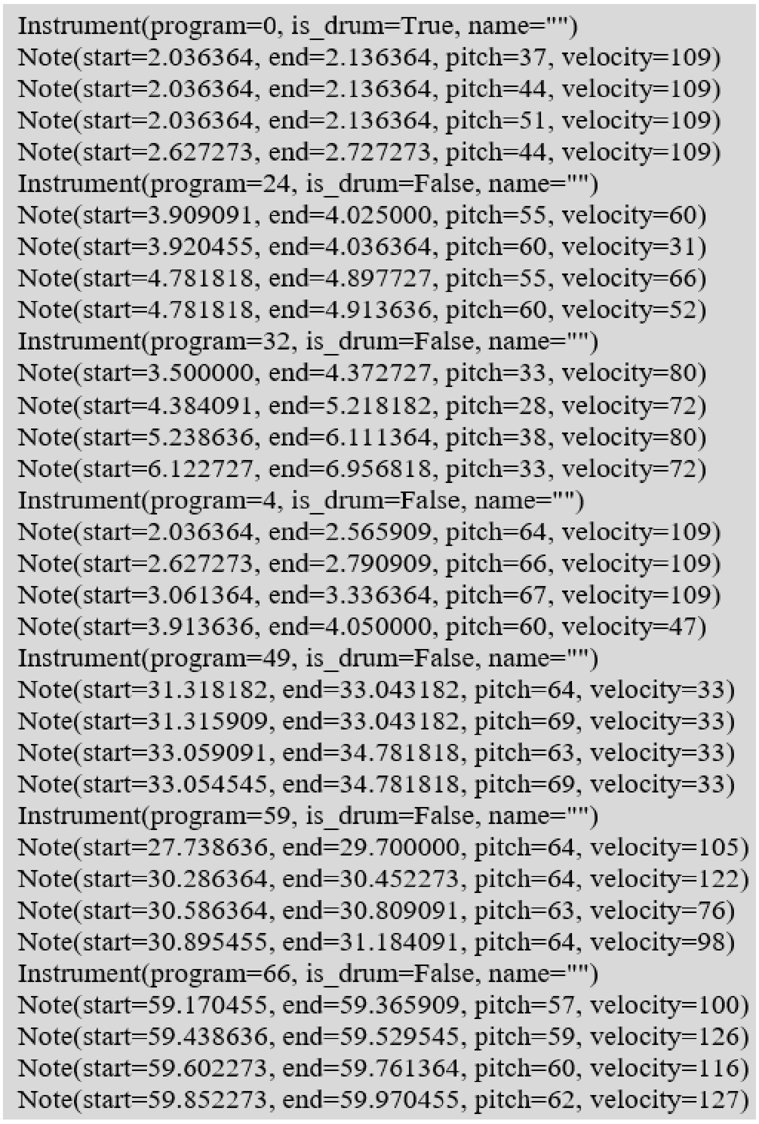

MIDI Dataset
Preprocessing
MIDI Drum Accompaniment Network
Train
Generate
Pretrained Network Weights
Finalized Network Weights
Generated MIDI Song

Figure 3.1. MIDI Drum Accompaniment Network Flowchart

3.2.1. Preprocessing

In order to parse midi files, an open-source library called “pretty_midi” was used (Raffel and Ellis, 2014). The library takes a midi file as an input and outputs the information such as instrument, pitch value, notes starting and ending times, and velocity, etc. By default, the library reads the midi file in instrument order, and outputs the note information for each instrument. An example of a parsed midi file can be seen in the following figure.



```
Instrument(program=0, is_drum=True, name="")
Note(start=2.036364, end=2.136364, pitch=37, velocity=109)
Note(start=2.036364, end=2.136364, pitch=44, velocity=109)
Note(start=2.036364, end=2.136364, pitch=51, velocity=109)
Note(start=2.627273, end=2.727273, pitch=44, velocity=109)
Instrument(program=24, is_drum=False, name="")
Note(start=3.909091, end=4.025000, pitch=55, velocity=60)
Note(start=3.920455, end=4.036364, pitch=60, velocity=31)
Note(start=4.781818, end=4.897727, pitch=55, velocity=66)
Note(start=4.781818, end=4.913636, pitch=60, velocity=52)
Instrument(program=32, is_drum=False, name="")
Note(start=3.500000, end=4.372727, pitch=33, velocity=80)
Note(start=4.384091, end=5.218182, pitch=28, velocity=72)
Note(start=5.238636, end=6.111364, pitch=38, velocity=80)
Note(start=6.122727, end=6.956818, pitch=33, velocity=72)
Instrument(program=4, is_drum=False, name="")
Note(start=2.036364, end=2.565909, pitch=64, velocity=109)
Note(start=2.627273, end=2.790909, pitch=66, velocity=109)
Note(start=3.061364, end=3.336364, pitch=67, velocity=109)
Note(start=3.913636, end=4.050000, pitch=60, velocity=47)
Instrument(program=49, is_drum=False, name="")
Note(start=31.318182, end=33.043182, pitch=64, velocity=33)
Note(start=31.315909, end=33.043182, pitch=69, velocity=33)
Note(start=33.059091, end=34.781818, pitch=63, velocity=33)
Note(start=33.054545, end=34.781818, pitch=69, velocity=33)
Instrument(program=59, is_drum=False, name="")
Note(start=27.738636, end=29.700000, pitch=64, velocity=105)
Note(start=30.286364, end=30.452273, pitch=64, velocity=122)
Note(start=30.586364, end=30.809091, pitch=63, velocity=76)
Note(start=30.895455, end=31.184091, pitch=64, velocity=98)
Instrument(program=66, is_drum=False, name="")
Note(start=59.170455, end=59.365909, pitch=57, velocity=100)
Note(start=59.438636, end=59.529545, pitch=59, velocity=126)
Note(start=59.602273, end=59.761364, pitch=60, velocity=116)
Note(start=59.852273, end=59.970455, pitch=62, velocity=127)
```

Figure 3.2. Parsed MIDI Data of a MIDI File

The library uses a special condition called “is_drum” because percussive instruments have a special channel, which is 9. The parsed midi files are still needed to be reordered in time since the order of the notes in a song is important. The problem was solved by reordering every midi file data according to time and storing into a data frame by using the library called “pandas” (McKinney 2010).

Table 3.1. MIDI Data Sorted by Start Timings

Index	Start	End	Pitch	Instrument	Velocity
0	2.03636	2.13636	37	0-Drum	109
1	2.03636	2.13636	44	0-Drum	109
2	2.03636	2.13636	51	0-Drum	109
3	2.03636	2.56591	64	4-Electric Piano 1	109
4	2.62727	2.72727	44	0-Drum	109
5	2.62727	2.79091	66	4-Electric Piano 1	109
6	3.06136	3.16136	37	0-Drum	109
7	3.06136	3.16136	44	0-Drum	109
8	3.06136	3.16136	51	0-Drum	109
9	3.06136	3.33636	67	4-Electric Piano 1	109
10	3.40909	3.50909	51	0-Drum	118
11	3.40909	5.39545	72	4-Electric Piano 1	118
12	3.5	4.37273	33	32-Acoustic Bass	80
13	3.8	3.9	35	0-Drum	30
14	3.8	3.9	51	0-Drum	30
15	3.90909	4.00909	51	0-Drum	60
16	3.90909	4.025	55	24-Acoustic Guitar (nylon)	60
17	3.91364	4.05	60	4-Electric Piano 1	47
18	3.91364	4.05	64	4-Electric Piano 1	47
19	3.91364	4.05	67	4-Electric Piano 1	47
20	3.91364	4.05	69	4-Electric Piano 1	47
21	3.92045	4.03636	60	24-Acoustic Guitar (nylon)	31
22	4.23636	4.33636	42	0-Drum	90
23	4.23636	4.33636	51	0-Drum	90
24	4.34773	4.44773	35	0-Drum	80
25	4.34773	4.44773	51	0-Drum	80
26	4.38409	5.21818	28	32-Acoustic Bass	72
27	4.65227	4.75227	42	0-Drum	45
28	4.65227	4.75227	53	0-Drum	45
29	4.65227	4.81136	60	4-Electric Piano 1	45
30	4.65227	4.81136	64	4-Electric Piano 1	45
31	4.65227	4.81136	67	4-Electric Piano 1	45
32	4.65227	4.81136	69	4-Electric Piano 1	45
33	4.78182	4.88182	35	0-Drum	100
34	4.78182	4.88182	51	0-Drum	100
35	4.78182	4.89773	55	24-Acoustic Guitar (nylon)	66
36	4.78182	4.91364	60	24-Acoustic Guitar (nylon)	52

(cont. on next page)

Table 3.1. (cont.)

37	5.10455	5.20455	51	0-Drum	70
38	5.21818	5.31818	35	0-Drum	80
39	5.21818	5.31818	42	0-Drum	80
40	5.23864	6.11136	38	32-Acoustic Bass	80

As pointed out in the prologue of the chapter, additional preprocessing steps are needed to form the input and output for the training. First, dataset contents were sampled at each starting time point. Then, the samples were given as

$$t \rightarrow t_1:\ sample_1 = \left[\begin{bmatrix} n_{1_1} \\ v_{1_1} \\ b_{1_1} \end{bmatrix}, \begin{bmatrix} n_{1_2} \\ v_{1_2} \\ b_{1_2} \end{bmatrix}, \dots, \begin{bmatrix} n_{1_k} \\ v_{1_k} \\ b_{1_k} \end{bmatrix}\right] \tag{3.1}$$

$$t \rightarrow t_2:\ sample_2 = \left[\begin{bmatrix} n_{2_1} \\ v_{2_1} \\ b_{2_1} \end{bmatrix}, \begin{bmatrix} n_{2_2} \\ v_{2_2} \\ b_{2_2} \end{bmatrix}, \dots, \begin{bmatrix} n_{2_k} \\ v_{2_k} \\ b_{2_k} \end{bmatrix}\right] \tag{3.2}$$

$$\vdots$$

$$t \rightarrow t_m:\ sample_m = \left[\begin{bmatrix} n_{m_1} \\ v_{m_1} \\ b_{m_1} \end{bmatrix}, \begin{bmatrix} n_{m_2} \\ v_{m_2} \\ b_{m_2} \end{bmatrix}, \dots, \begin{bmatrix} n_{m_k} \\ v_{m_k} \\ b_{m_k} \end{bmatrix}\right] \tag{3.3}$$

where $t_1, t_2, \dots, t_m$ denotes the starting time of each sample, *m* denotes the number of samples, *k* denotes the data point inside a sample, and, *n*, *v*, *b* are the note's pitch, velocity, and instrument values respectively. The instrument type is a binary value where $b = 1$ corresponds to drums, and $b = 0$ means that the note and velocity value of that sample corresponds to any other instrument. In the end, each sample which didn't have the combination of drums and other instruments was removed from the sample set.

In the next step, the sample contents were separated to form the input and output. The input vector contained every instrument's note pitch and velocity information except drums. On the other hand, the output vector contained only the drums' note pitch and velocity information. Notice that each sample can have different number of drum and other instrument combinations. As a result, the number of input and output nodes may need to be flexible. In order to achieve this, we used a hyperparameter called maximum pattern length. The hyperparameter is usually set to the number of most used instrument in a sample. However, the pattern length also effects the network's learning process. When the pattern length is too short or too long, network starts to learn very slowly or not at all. In such a case, we may want to increase or decrease the pattern length. To keep the preprocessing simple, we initialized both input and output vectors with zeros. With this initialization, any pattern less than the maximum pattern length became zero padded. On the contrary, for patterns longer than the maximum pattern length, only elements up to the first maximum pattern length were used. In the thesis, the pattern length was chosen to be 20 by trial and error. Therefore, the input vector and output vector were given as

$$input_{MxNx2}, output_{MxNx2} = 0_{MxNx2} \tag{3.4}$$

$$input_{MxNx2}, output_{MxNx2} = \begin{cases} \begin{bmatrix} n_{m_k} \\ v_{m_k} \end{bmatrix} \in input_{m,k} & if \quad b_{m_k} = 0 \\ \begin{bmatrix} n_{m_k} \\ v_{m_k} \end{bmatrix} \in output_{m,k} & if \quad b_{m_k} = 1 \end{cases} \tag{3.5}$$

where $input_{MxNx2}$, $output_{MxNx2}$, and 0_{MxNx2} are the input vector, output vector and the zero matrix respectively with the shape of $MxNx2$ where M denotes the number of samples, and K denotes the maximum pattern length, and the first column represents the note pitch values whereas second column represents the velocity values.

As the last step input and output vectors were one-hot encoded for the training. Recall that in Section 2.1.2, we mentioned that there are total of 128 note pitch and velocity values in a MIDI file. On the other hand, we have only 22 drum notes and an extra zero value that comes from the zero padding. As a result of that, the model's input was one-hot encoded with 128 elements, but the model's note output and the model's velocity were one-hot encoded with 23 and 128 elements respectively.

To facilitate the understanding, we want to give an example of preprocessing a dataset. Let's assume that we are given the Table 3.1 as our dataset. Then, samples are as follows

$$t \to 2.03636s: \quad sample_1 = \left[\begin{bmatrix} 37 \\ 109 \\ 1 \end{bmatrix}, \begin{bmatrix} 44 \\ 109 \\ 1 \end{bmatrix}, \begin{bmatrix} 51 \\ 109 \\ 1 \end{bmatrix}, \begin{bmatrix} 64 \\ 109 \\ 0 \end{bmatrix} \right] \tag{3.6}$$

$$t \rightarrow 2.62727s: \quad sample_2 = \left[\begin{bmatrix} 44 \\ 109 \\ 1 \end{bmatrix}, \begin{bmatrix} 66 \\ 109 \\ 0 \end{bmatrix}\right] \tag{3.7}$$

$$t \rightarrow 3.06136s: \quad sample_3 = \left[\begin{bmatrix} 37 \\ 109 \\ 1 \end{bmatrix}, \begin{bmatrix} 44 \\ 109 \\ 1 \end{bmatrix}, \begin{bmatrix} 51 \\ 109 \\ 1 \end{bmatrix}, \begin{bmatrix} 67 \\ 109 \\ 0 \end{bmatrix}\right] \tag{3.8}$$

$$t \rightarrow 3.40909s: \quad sample_4 = \left[\begin{bmatrix} 51 \\ 118 \\ 1 \end{bmatrix}, \begin{bmatrix} 72 \\ 118 \\ 0 \end{bmatrix}\right] \tag{3.9}$$

$$t \rightarrow 3.90909s: \quad sample_5 = \left[\begin{bmatrix} 51 \\ 60 \\ 1 \end{bmatrix}, \begin{bmatrix} 55 \\ 60 \\ 0 \end{bmatrix}\right] \tag{3.10}$$

$$t \rightarrow 4.65227s: \quad sample_6 = \left[\begin{bmatrix} 42 \\ 45 \\ 1 \end{bmatrix}, \begin{bmatrix} 53 \\ 45 \\ 1 \end{bmatrix}, \begin{bmatrix} 60 \\ 45 \\ 0 \end{bmatrix}, \begin{bmatrix} 64 \\ 45 \\ 0 \end{bmatrix}, \begin{bmatrix} 67 \\ 45 \\ 0 \end{bmatrix}, \begin{bmatrix} 69 \\ 45 \\ 0 \end{bmatrix}\right] \tag{3.11}$$

$$t \rightarrow 4.78182s: \quad sample_7 = \left[\begin{bmatrix} 35 \\ 100 \\ 1 \end{bmatrix}, \begin{bmatrix} 51 \\ 100 \\ 1 \end{bmatrix}, \begin{bmatrix} 55 \\ 66 \\ 0 \end{bmatrix}, \begin{bmatrix} 60 \\ 52 \\ 0 \end{bmatrix}\right] \tag{3.12}$$

Notice that we have only 7 samples out of 40 data points. As mentioned earlier, we exclude the samples that don't include drums and other instruments. Each sample must have at least one drum and one another instrument information. Also remember that input and output vectors must be initialized according to maximum pattern length. In this example, longest sample is the 6th sample. 6th sample has 2 drum information and 4 other instrument information. Therefore, the maximum pattern length is selected to be 4. Then, by using Eq. (3.4) input and output vectors are initialized as follows

$$\begin{aligned} input &= 0_{7x4x2} \\ &= \begin{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \cdots & \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \vdots & \ddots & \vdots \\ \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \cdots & \begin{bmatrix} 0 \\ 0 \end{bmatrix} \end{bmatrix}_{7x4x2} \end{aligned} \tag{3.13}$$

$$output = 0_{7x4x2}$$
$$= \begin{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \cdots & \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \vdots & \ddots & \vdots \\ \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \cdots & \begin{bmatrix} 0 \\ 0 \end{bmatrix} \end{bmatrix}_{7x4x2} \tag{3.14}$$

and by using Eq. (3.5) input and output vectors can be written as

$$input = \begin{bmatrix} \begin{bmatrix} 64 \\ 109 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 66 \\ 109 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 67 \\ 109 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 72 \\ 118 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 55 \\ 60 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 60 \\ 45 \end{bmatrix} & \begin{bmatrix} 64 \\ 45 \end{bmatrix} & \begin{bmatrix} 67 \\ 45 \end{bmatrix} & \begin{bmatrix} 69 \\ 45 \end{bmatrix} \\ \begin{bmatrix} 55 \\ 66 \end{bmatrix} & \begin{bmatrix} 60 \\ 52 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} \end{bmatrix}_{7x4x2} \tag{3.15}$$

$$output = \begin{bmatrix} \begin{bmatrix} 37 \\ 109 \end{bmatrix} & \begin{bmatrix} 44 \\ 109 \end{bmatrix} & \begin{bmatrix} 51 \\ 109 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 44 \\ 109 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 37 \\ 109 \end{bmatrix} & \begin{bmatrix} 44 \\ 109 \end{bmatrix} & \begin{bmatrix} 51 \\ 109 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 51 \\ 118 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 51 \\ 60 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 42 \\ 45 \end{bmatrix} & \begin{bmatrix} 53 \\ 45 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 35 \\ 100 \end{bmatrix} & \begin{bmatrix} 51 \\ 100 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} \end{bmatrix}_{7x4x2} \tag{3.16}$$

Before moving on the last step, let’s define an expression to represent the one-hot encoded vector for simplification. The expression is given as

$$OHE[i,j] = [0 \quad \ldots \quad i \quad \ldots \quad 0]_{1xj} \tag{3.17}$$

where $i, j \in Z^{+},\ j \geq i$ and i denotes the location of 1, and j denotes total number of elements inside the vector. For example,

$$OHE[5,8] = [0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0]_{1x8} \tag{3.18}$$

$$OHE[2,3] = [0 \quad 1 \quad 0]_{1x3} \tag{3.19}$$

$$OHE[1,5] = [1 \quad 0 \quad 0 \quad 0 \quad 0]_{1x5} \tag{3.20}$$

Recall that we decided to use the most frequently used 22 drum notes. Furthermore, we zero padded the output and ended up with total of 23 different note values instead of 128 different MIDI notes. In addition, each element's position doesn't correspond the actual integer value as it did for the input notes and velocity values. Therefore, the values should be sorted from smallest to largest first, and then each value should be represented by their position. The sorted drum note values are given as follows

$$\begin{aligned} &drum_{note_{values}} \\ &= [0,35,36,37,38,40,41,42,43,44,45,46,47,48,49,50,51,52,53,55,56,57,59] \end{aligned} \tag{3.21}$$

Thus, the same OHE expression can be used with a slight change in meaning. This change of meaning can be seen in the example below, where the corresponding value in the OHE expression is the position of the drum note value instead of the actual value. With an extra step the actual drum value can be obtained easily.

$$\begin{aligned} OHE[5,23] &= [0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad \dots \quad 0]_{1x128} \\ &= drum_{values}[5] \\ &= 38 \end{aligned} \tag{3.22}$$

After applying the one-hot encoding, the model's input, $encoded_{input}$, and model's outputs, $encoded_{notes_{output}}$ and $encoded_{velocity_{output}}$, can be written as

$$encoded_{input} = \begin{bmatrix} \begin{bmatrix} OHE[65,128] \\ OHE[110,128] \end{bmatrix} & \begin{bmatrix} OHE[1,128] \\ OHE[1,128] \end{bmatrix} & \begin{bmatrix} OHE[1,128] \\ OHE[1,128] \end{bmatrix} & \begin{bmatrix} OHE[1,128] \\ OHE[1,128] \end{bmatrix} \\ \begin{bmatrix} OHE[67,128] \\ OHE[110,128] \end{bmatrix} & \begin{bmatrix} OHE[1,128] \\ OHE[1,128] \end{bmatrix} & \begin{bmatrix} OHE[1,128] \\ OHE[1,128] \end{bmatrix} & \begin{bmatrix} OHE[1,128] \\ OHE[1,128] \end{bmatrix} \\ \begin{bmatrix} OHE[68,128] \\ OHE[110,128] \end{bmatrix} & \begin{bmatrix} OHE[1,128] \\ OHE[1,128] \end{bmatrix} & \begin{bmatrix} OHE[1,128] \\ OHE[1,128] \end{bmatrix} & \begin{bmatrix} OHE[1,128] \\ OHE[1,128] \end{bmatrix} \\ \begin{bmatrix} OHE[73,128] \\ OHE[119,128] \end{bmatrix} & \begin{bmatrix} OHE[1,128] \\ OHE[1,128] \end{bmatrix} & \begin{bmatrix} OHE[1,128] \\ OHE[1,128] \end{bmatrix} & \begin{bmatrix} OHE[1,128] \\ OHE[1,128] \end{bmatrix} \\ \begin{bmatrix} OHE[56,128] \\ OHE[61,128] \end{bmatrix} & \begin{bmatrix} OHE[1,128] \\ OHE[1,128] \end{bmatrix} & \begin{bmatrix} OHE[1,128] \\ OHE[1,128] \end{bmatrix} & \begin{bmatrix} OHE[1,128] \\ OHE[1,128] \end{bmatrix} \\ \begin{bmatrix} OHE[61,128] \\ OHE[46,128] \end{bmatrix} & \begin{bmatrix} OHE[65,128] \\ OHE[46,128] \end{bmatrix} & \begin{bmatrix} OHE[68,128] \\ OHE[46,128] \end{bmatrix} & \begin{bmatrix} OHE[70,128] \\ OHE[61,128] \end{bmatrix} \\ \begin{bmatrix} OHE[56,128] \\ OHE[67,128] \end{bmatrix} & \begin{bmatrix} OHE[61,128] \\ OHE[53,128] \end{bmatrix} & \begin{bmatrix} OHE[1,128] \\ OHE[1,128] \end{bmatrix} & \begin{bmatrix} OHE[1,128] \\ OHE[1,128] \end{bmatrix} \end{bmatrix}_{7x4x2x128} \quad (3.23)$$

$$encoded_{notes_{output}} = \begin{bmatrix} [OHE[4,23]] & [OHE[10,23]] & [OHE[17,23]] & [OHE[1,23]] \\ [OHE[10,23]] & [OHE[1,23]] & [OHE[1,23]] & [OHE[1,23]] \\ [OHE[38,23]] & [OHE[10,23]] & [OHE[17,23]] & [OHE[1,23]] \\ [OHE[17,23]] & [OHE[1,23]] & [OHE[1,23]] & [OHE[1,23]] \\ [OHE[17,23]] & [OHE[1,23]] & [OHE[1,23]] & [OHE[1,23]] \\ [OHE[8,23]] & [OHE[19,23]] & [OHE[1,23]] & [OHE[1,23]] \\ [OHE[3,23]] & [OHE[17,23]] & [OHE[1,23]] & [OHE[1,23]] \end{bmatrix}_{7x4x1x23} \quad (3.24)$$

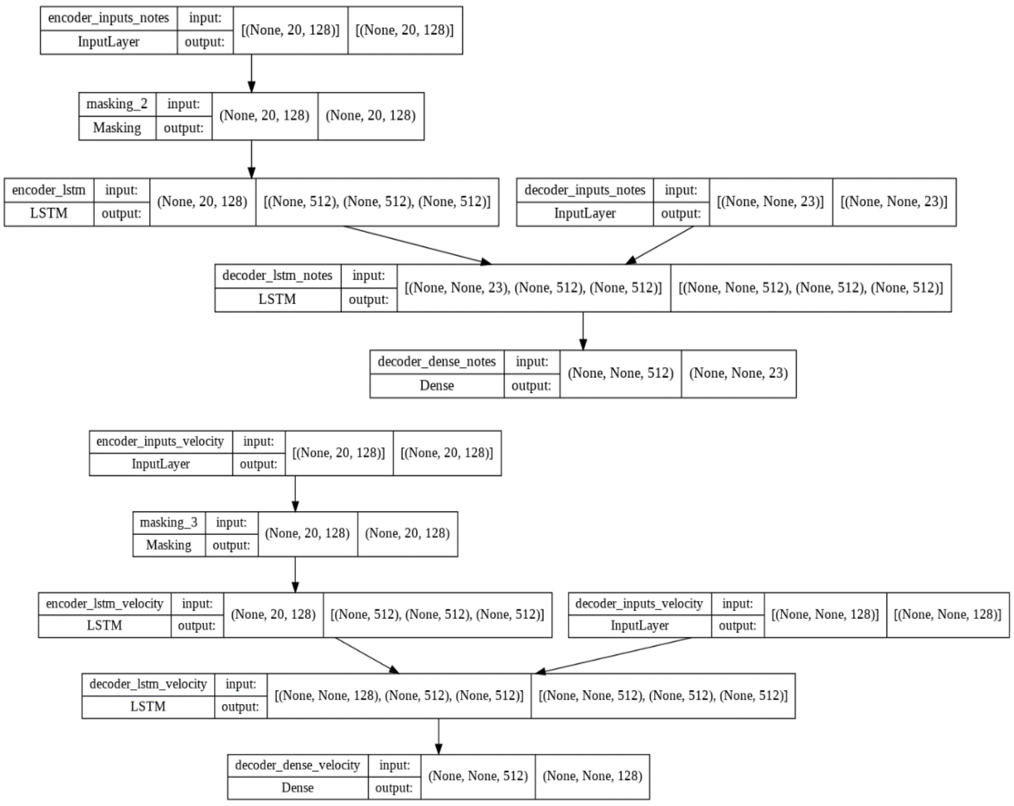
$$encoded_{output} = \begin{bmatrix} [OHE[110,128]] & [OHE[110,128]] & [OHE[110,128]] & [OHE[1,128]] \\ [OHE[110,128]] & [OHE[1,128]] & [OHE[1,128]] & [OHE[1,128]] \\ [OHE[110,128]] & [OHE[110,128]] & [OHE[110,128]] & [OHE[1,128]] \\ [OHE[119,128]] & [OHE[1,128]] & [OHE[1,128]] & [OHE[1,128]] \\ [OHE[61,128]] & [OHE[1,128]] & [OHE[1,128]] & [OHE[1,128]] \\ [OHE[46,128]] & [OHE[46,128]] & [OHE[1,128]] & [OHE[1,128]] \\ [OHE[101,128]] & [OHE[101,128]] & [OHE[1,128]] & [OHE[1,128]] \end{bmatrix}_{7x4x1x128} \quad (3.25)$$

In this section, we described the preprocessing steps that had applied during the thesis. The main reason we used these specific preprocessing steps was to improve the network. In the following section, we are going to describe the type of neural network that is preferred, and training processes used in the thesis.

3.2.2. Training Model

The network training was completed by using open-source libraries called "Tensorflow" (Abadi et al. 2015) and "Keras" (Gulli and Pal 2017). A total of three models were created, one for training and the other two for prediction. Each sample was separated into notes and velocities before feeding the training input. Then, inputs were masked with the one-hot encoded version of "0" because we used "0" for padding in preprocessing. By masking the input values, we were essentially telling that the masked value had no effect on the dataset.

In order to construct the Seq2Seq model, we started off by using LSTM architectures with 512 layers. Then, instead of taking the output of the LSTM layers, we used their hidden states and passed them to another LSTM. By concatenating two LSTMs and passing the context vector, we were able to develop an Encoder-Decoder Seq2Seq model where encoder is the first LSTM, and the decoder is the second LSTM. In addition, SoftMax activation was used in the last layer of the models. The graphical representation of training model can be seen in the figure given below.



encoder_inputs_notes | InputLayer | input: [(None, 20, 128)] | output: [(None, 20, 128)]
masking_2 | Masking | input: (None, 20, 128) | output: (None, 20, 128)
encoder_lstm | LSTM | input: (None, 20, 128) | output: [(None, 512), (None, 512), (None, 512)]
decoder_inputs_notes | InputLayer | input: [(None, None, 23)] | output: [(None, None, 23)]
decoder_lstm_notes | LSTM | input: [(None, None, 23), (None, 512), (None, 512)] | output: [(None, None, 512), (None, 512), (None, 512)]
decoder_dense_notes | Dense | input: (None, None, 512) | output: (None, None, 23)
encoder_inputs_velocity | InputLayer | input: [(None, 20, 128)] | output: [(None, 20, 128)]
masking_3 | Masking | input: (None, 20, 128) | output: (None, 20, 128)
encoder_lstm_velocity | LSTM | input: (None, 20, 128) | output: [(None, 512), (None, 512), (None, 512)]
decoder_inputs_velocity | InputLayer | input: [(None, None, 128)] | output: [(None, None, 128)]
decoder_lstm_velocity | LSTM | input: [(None, None, 128), (None, 512), (None, 512)] | output: [(None, None, 512), (None, 512), (None, 512)]
decoder_dense_velocity | Dense | input: (None, None, 512) | output: (None, None, 128)

Figure 3.3. Encoder Training Model for Notes and Velocities

As it can be seen in Figure 3.3, there are two different inputs which are for encoder and decoder. The reason for that is the Teacher Forcing Method. In this method, we provide ground truth values to the decoder to increase the efficiency of the network. To explain the shapes that are seen in the figure, we start off with encoder inputs. Encoder inputs have the shape of (None, 20, 128). The first shape variable represents the total number of samples in a dataset, but since it is not given during the model creation it is labeled as none. Recall that in the previous section we decided to preprocess data with a maximum pattern length of 20, which is also represented by the second shape variable. The last variable shape, 128, represents the total number of features. The reason for the last variable shape to be 128 is because there are total of 128 different notes and velocities in a MIDI file. Next, encoder inputs were masked in order to skip the timesteps that were used for padding and fed into LSTM architecture with 512 layers. The encoder has three different outputs with each of them having the shape of (None, 512). The reason for that is the LSTM has three different outputs which are network output, hidden state, and cell state. Hidden state and Cell state forms the context vector. Then, decoder input is fed into decoder LSTM. Notice that note’s decoder input has the shape of (None, None, 23). The reason for the difference between note’s decoder input and velocity’s decoder input is that there are possibilities for 23 drum notes.

The model was trained on a Tesla K80 GPU, which is provided by Google Colaboratory (Google Research 2017), in order to accelerate the learning process. In addition, the model trained by using six different datasets. Each dataset was trained with 200 epochs and 64 batch size. Categorical Cross Entropy and Adam was used as the loss function and as the optimizer respectively. To improve the learning capability of the network, the learning rate was decreased gradually each time when the difference of three consecutive losses was no more than 0.01. At the end, model weights were saved for prediction. The number of samples each dataset contained is given in the following table.

Table 3.2. Number of Samples in Datasets

Dataset Name	**Number of MIDI Files**	**Number of Samples**
Blues	54	32039
HipHop-Rap	42	24595
Jazz	43	24941

(cont. on next page)

Table 3.2. (cont.)

Pop	53	35245
Rock	42	39999
Shuffle	58	41414

3.2.3. Generator Model

The goal of designing this model was to generate drum patterns that feels human-made and unique. However, the training network cannot be directly used for the prediction. The reason for that is due to using the teacher forcing method for training. The teacher forcing method expects ground truth/correct output, but during prediction we won’t be able to know the ground truth for a given input. Thus, we were required a new Seq2Seq model for prediction. The new model didn’t use the teacher forcing method. However, we used the context vector of the training model through the saved training weights. The graphical representation of new Seq2Seq model’s encoder and decoder for the prediction can be seen in the following figures.

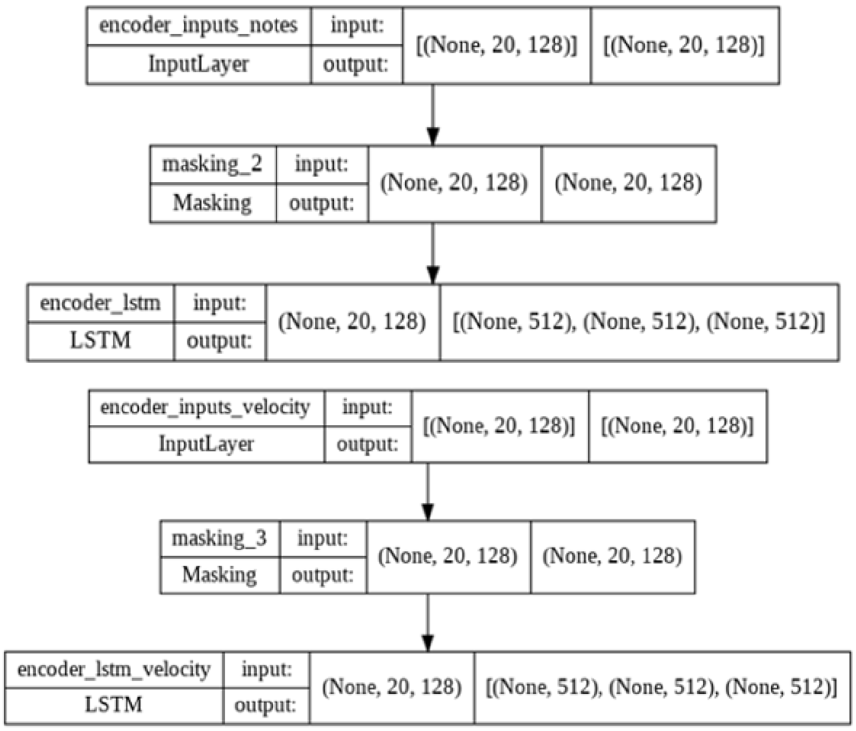


Figure 3.4. Encoder Prediction Model for Notes and Velocities

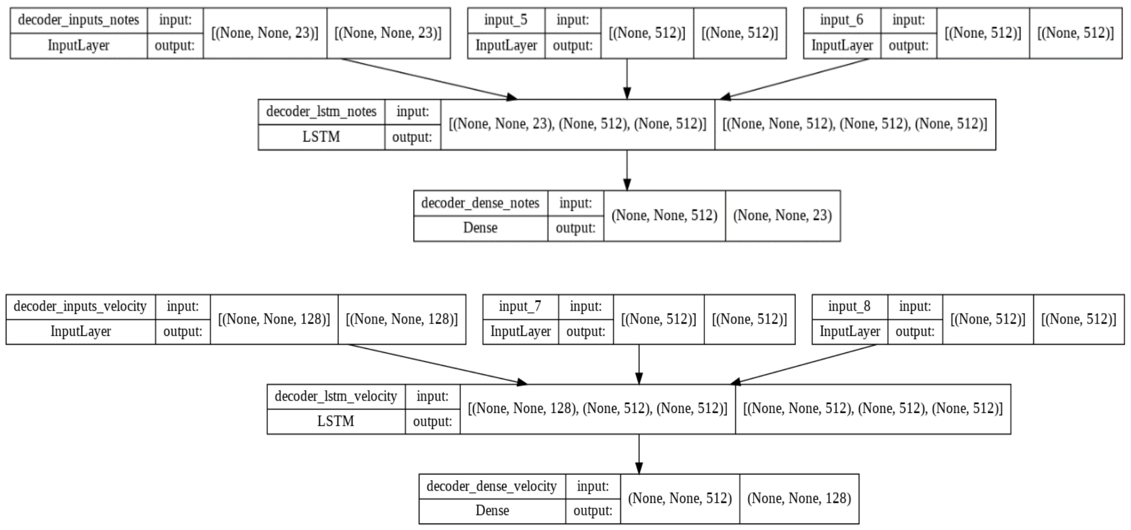
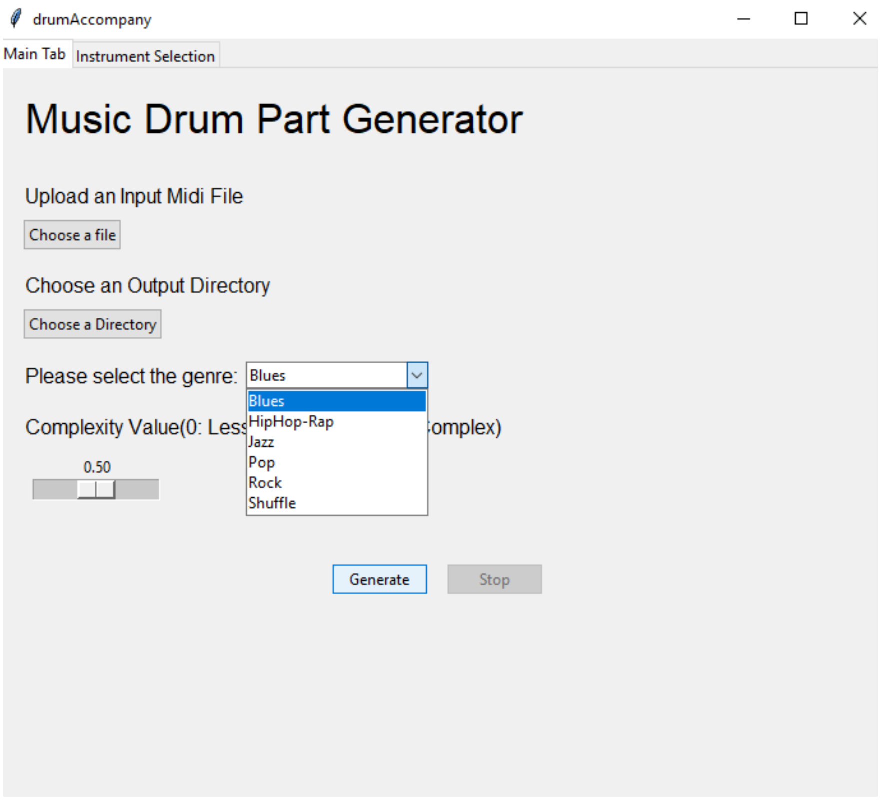

decoder_inputs_notes InputLayer input: output: [(None, None, 23)] [(None, None, 23)]
input_5 InputLayer input: output: [(None, 512)] [(None, 512)]
input_6 InputLayer input: output: [(None, 512)] [(None, 512)]
decoder_lstm_notes LSTM input: output: [(None, None, 23), (None, 512), (None, 512)] [(None, None, 512), (None, 512), (None, 512)]
decoder_dense_notes Dense input: output: (None, None, 512) (None, None, 23)
decoder_inputs_velocity InputLayer input: output: [(None, None, 128)] [(None, None, 128)]
input_7 InputLayer input: output: [(None, 512)] [(None, 512)]
input_8 InputLayer input: output: [(None, 512)] [(None, 512)]
decoder_lstm_velocity LSTM input: output: [(None, None, 128), (None, 512), (None, 512)] [(None, None, 512), (None, 512), (None, 512)]
decoder_dense_velocity Dense input: output: (None, None, 512) (None, None, 128)

Figure 3.5. Decoder Prediction Model for Notes and Velocities

Encoder prediction model is using the same input layer, masking layer and LSTM architecture as the encoder training model. Weights were loaded from the context vector of training model. The decoder prediction model's input layer was provided with an initial input which starts with the SOS special token since it requires a dynamical information. Then, decoder continued to predict new output by using previous time instant output as its input. In addition, during prediction we used a complexity value to improve the quality of the generated patterns. The complexity value is the temperature value mentioned in the temperature sampling method described in the previous chapter. In general, these types of models use highest probabilistic values as the next output. By using the complexity value, we improved the quality of the patterns.

A user interface (UI) was developed to ease the user access by using an open-source python library called "tkinter" (Lundh 1999). The UI was designed to increase user's flexibility and experience. The UI consists of two tabs which are called "Main Tab" and "Instrument Selection". In main tab, user is able to choose the input and output file directories, music genre, and the complexity value. Choosing complexity value closer to 0 produces less complex drum patterns whereas choosing complexity value closer to 1 result in the most complex patterns. The second tab, "Instrument Selection", is provided for users who want more control over the generated drum parts. In this tab, the user can adjust the presence of the drum instruments and decide which instruments should be played.



Music Drum Part Generator

Upload an Input Midi File

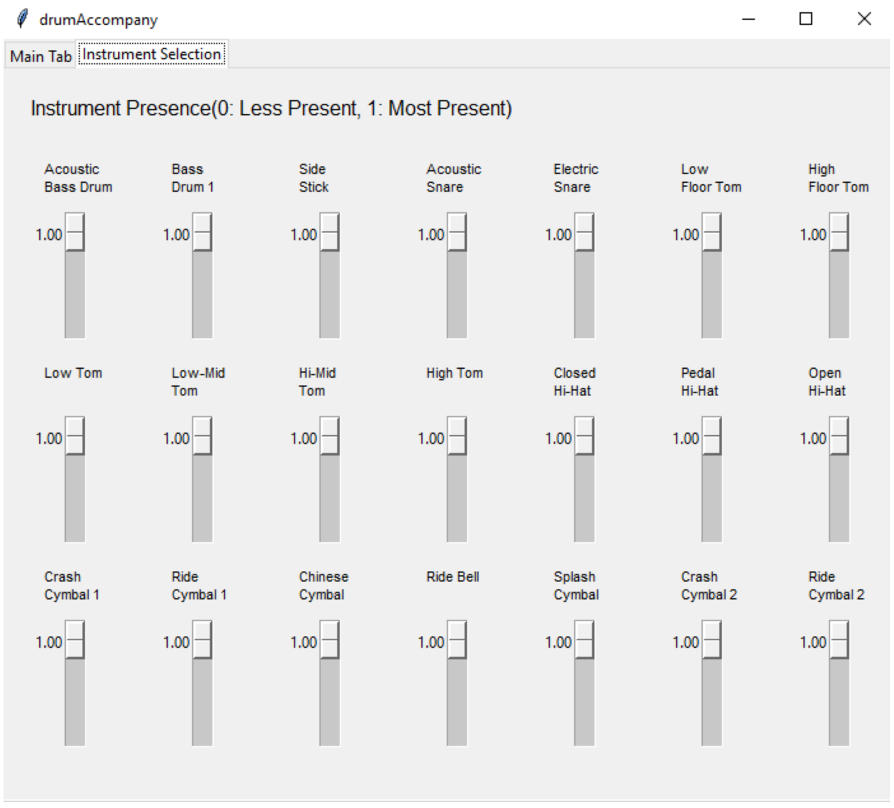
Choose an Output Directory

Please select the genre: Blues

Complexity Value(0: Less … omplex)

0.50

Figure 3.6. UI Main Tab



Instrument Presence(0: Less Present, 1: Most Present)

Acoustic Bass Drum 1.00
Bass Drum 1 1.00
Side Stick 1.00
Acoustic Snare 1.00
Electric Snare 1.00
Low Floor Tom 1.00
High Floor Tom 1.00
Low Tom 1.00
Low-Mid Tom 1.00
Hi-Mid Tom 1.00
High Tom 1.00
Closed Hi-Hat 1.00
Pedal Hi-Hat 1.00
Open Hi-Hat 1.00
Crash Cymbal 1 1.00
Ride Cymbal 1 1.00
Chinese Cymbal 1.00
Ride Bell 1.00
Splash Cymbal 1.00
Crash Cymbal 2 1.00
Ride Cymbal 2 1.00

Figure 3.7. UI Instrument Selection Tab

3.3. Results

In the scope of this study, the goal was to create an AI software that is able to assist with drums for the music groups, music producers, and any individual who is interested in music. The idea was to understand the harmony between the drum notes and other notes in each interval taken from the song. In this study, MIDI was chosen as the dataset file format because it has easy access to a lot of music information. Every MIDI file was separated into samples and the input, and the output were constructed for the neural network model as they are given in Eq. (3.15) and (3.16). Afterward, the input and output were one-hot encoded and made ready for training. Seq2Seq neural network model trained with a total of six different music genres which are blues, jazz, hiphop-rap, pop, rock, and shuffle. The training continued for 200 epochs for each dataset and learning rate was gradually decreased the improve the loss and accuracy. In the Figure 3.9, 3.10 and 3.11, we present the improvement of loss and accuracy and change of learning rate over epoch graphs for each dataset during the training respectively.

During the training, the dataset was not split as the training and validation so that we looked for qualitative results rather than quantitative results. The main reason is that the application is being about art. In order to provide additional insight to these results, the trained model was used to generate drum parts for a MIDI file that the network never seen before. The newly created MIDI files can be listened by scanning the QR code provided at the end of the chapter. By training the model with a diversity of genres, we aimed to give the user freedom in the production of drum notes so that more original and interesting drum parts could be generated. Due to easy access to musical information in MIDI files, MIDI Drum Accompaniment Network also covers already existing drum parts of a song by replacing them.

Figure 3.8. QR Code for Example MIDI Outputs[2]

[2] "MIDI Drum Accompaniment Network". 2022. YouTube. https://www.youtube.com/playlist?list=PLFhr-SY0apszYp-tto-QvjUIgwpfNnB17. (Short URL: https://tinyurl.com/bdh44mxp)

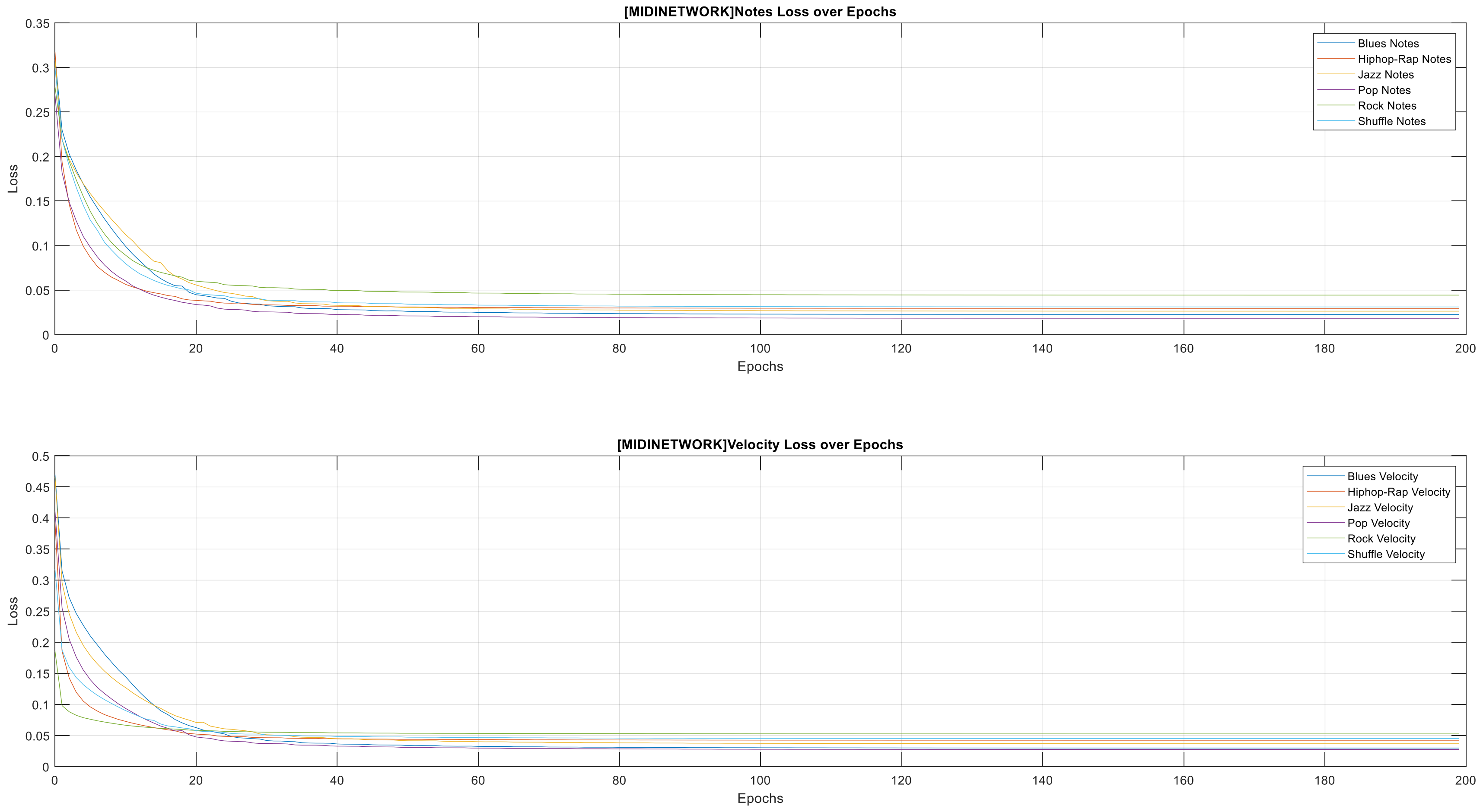


Figure 3.9. MIDINETWORK - Losses of Note and Velocity over Epochs

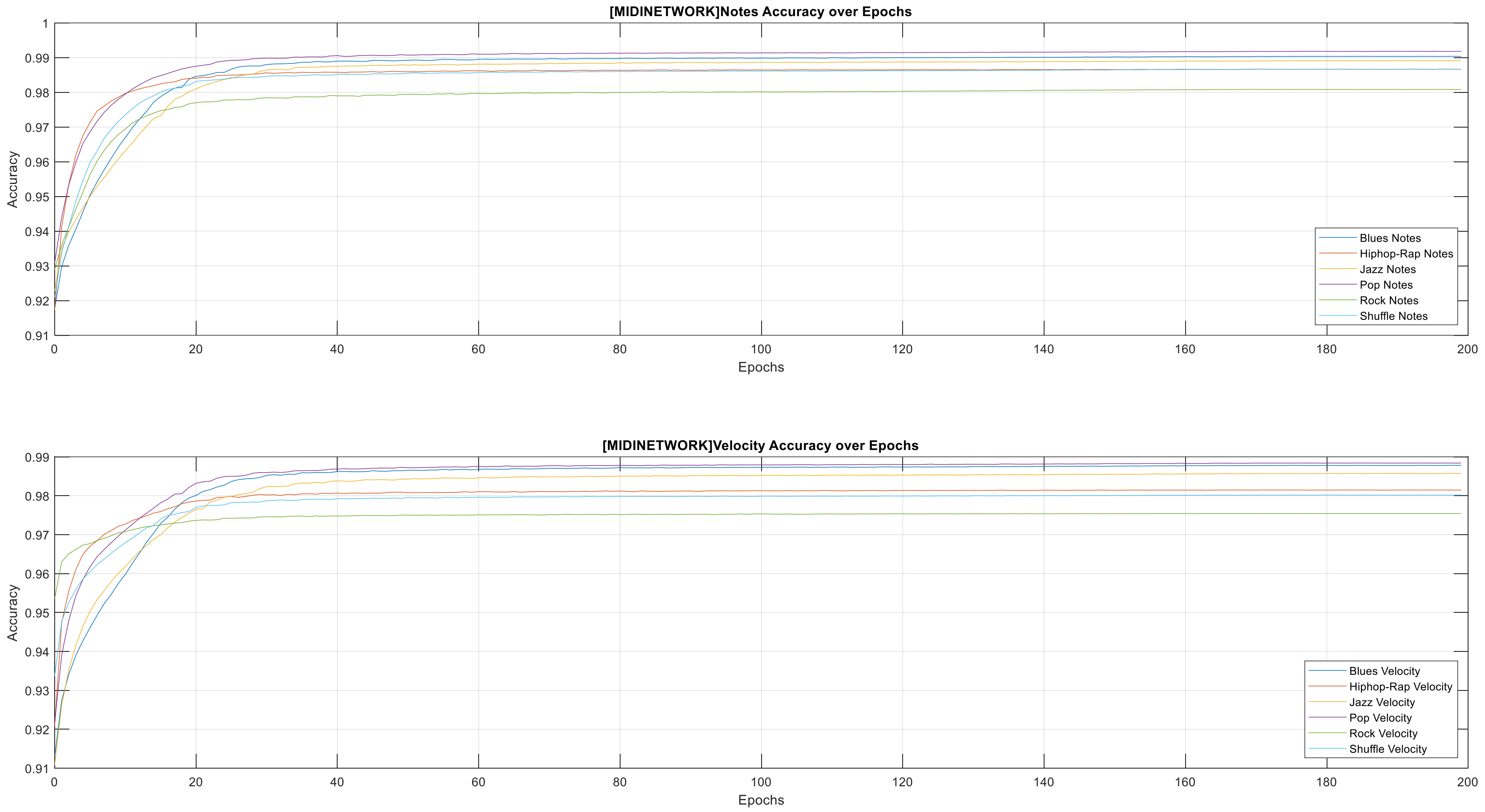


Figure 3.10. MIDINETWORK - Accuracies of Note and Velocity over Epochs

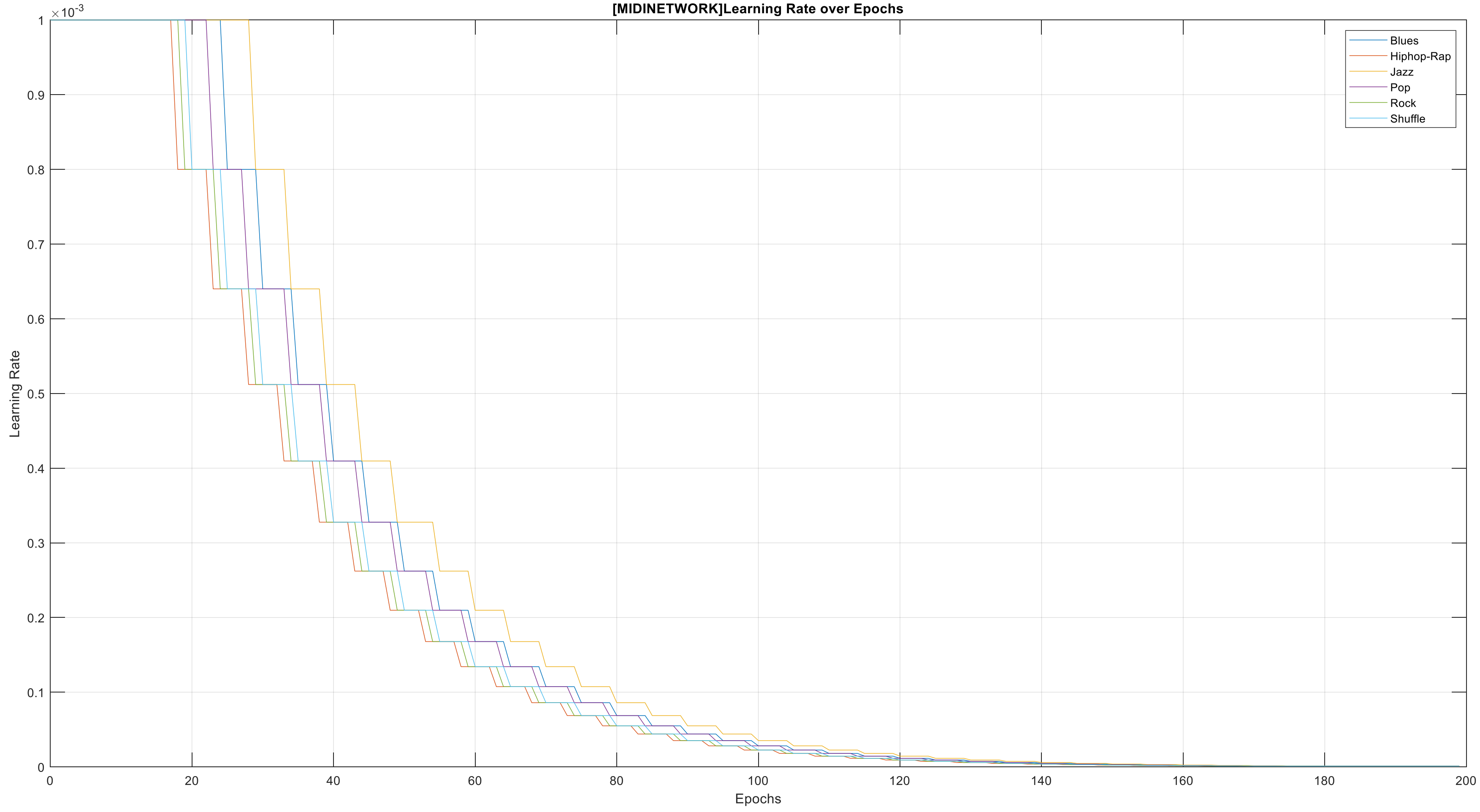


Figure 3.11. MIDINETWORK - Change of Learning Rate over Epochs

3.4. Discussion

In this chapter, we gave a detailed explanation of the MIDI accompany network’s working principles. The goal was to make a model that was able to accompany and/or rewrite the drum parts of a song. In order to accomplish this goal, the necessary features were extracted from the MIDI format. Moreover, Seq2Seq model was used to understand the relationship between input data and drum output data. In addition, a UI was created to ease the access for the user. The network was made for people who enjoys the music, for music groups to give an inspiration, and for music producers.

In the study, even though each piece of data in MIDI format belongs to an instrument, we treated the input information as it belongs to a single instrument. Moreover, we only used two features which were notes and their corresponding velocities. These decisions reduced the model’s complexity and accelerated the training, and the model became simpler.

The algorithm described in this chapter can be improved in different ways. First and foremost, the simplest way to improve the model's capabilities is to train the model not only for drums but for other instruments individually. Thus, the model will be able to accompany the track with any instrument that the user prefers. The second idea of the improvement is to use the starting time difference between two successive notes as well as the duration of the notes as the additional features. Hence, we can also produce these features in output during generation, allowing our model to have a distinct influence on rhythm. Furthermore, the model can be implemented to work in real-time. Lastly, the model’s quantitative results can be calculated by using basic rules of making drum patterns as the reference.

CHAPTER 4

WAV DRUM ACCOMPANIMENT NETWORK

Creativity is seeing what others see and
thinking what no one else has ever thought.
-Albert Einstein-

4.1. Introduction

In this chapter, we propose a novel approach to generate drum parts for the complex raw waveform signals in WAV files. Since waveform signals have a mixture of multiple frequencies and noise in them, it is hard to extract every note's information. We utilize the MIDI dataset combined with waveform signals generated from this dataset for the training. By using this idea, we are able to reach necessary information to create the model's input and output. The main idea is to extract signal’s characteristic features such as Mel spectrogram, MFCC, and tempogram from its MIDI form. Then, we use a neural network model in a way that it can extract the relationship between these features and originally accompanied drum notes. Thus, in the generation part we are able to drop out MIDI data and use any WAV file as input.

The remainder of this chapter is organized as follows: In Section 4.2, we describe the proposed work to train and develop the WAV drum accompaniment network. In Section 4.2.1, we provide a detailed explanation of preprocessing steps followed by model training in Section 4.2.2. In Section 4.2.3 we explain the generating part of the model. In Section 4.3, we present the training and prediction results.

4.2. Proposed Work

We propose using the idea of image captioning models to make a relationship between the features and drum notes. Image captioning models usually use a pretrained CNN networks before their LSTM models. However, since the characteristics of signals are different than pretrained images of a pretrained CNN, we decided to retrain it. In addition, we decided to use Seq2Seq model due to its success in MIDI accompaniment network.

We start by sampling the MIDI dataset. Then, the model's input was converted to WAV files and Mel spectrogram, MFCC, and tempogram features were extracted. The model's output was kept as the sequence of drum notes and velocities. Although there are 128 different notes in a MIDI file, we decided to choose 22 notes that are mostly used in drum instruments. The idea was to make neural network model to understand the relationship between actual signal's features and drum information so that we will be able to accompany any given WAV file when the training is complete. Similarly, the temperature sampling method was used in the generator model to obtain unique and exciting drum patterns. The network was trained by using a dataset containing six different genres. Additionally, a UI was created to ease the user access.

4.2.1. Preprocessing

After converting MIDI input sequence to waveform with a sample rate of 44100 Mel spectrogram, MFCC, and tempogram were computed using 1024 point FFT where hop length is 512, and number of MFCC is 128. Then, these features were stacked on top of each other, and a 3-channel RGB image-like feature array was created. Thus, we were able to generate an input that can also be extracted from the raw WAV files, which can then be used in the generator model.

We aimed to obtain time-frequency information of the input notes by using Mel spectrogram. The main reason for using Mel spectrogram was due to its logarithmic scale which is a good approximation for human hearing. In addition, we used MFCC as another feature to cover any missing information from the Mel spectrogram. Lastly, the tempogram was used to get the tempo characteristic of the song. Although the input's pattern length didn't need to be limited, the maximum pattern length was used as same as the before, which was 20, for the model's output.

Before going further and explain each step deeper, we want to give the preprocessing flowchart below to facilitate the understanding of the idea.

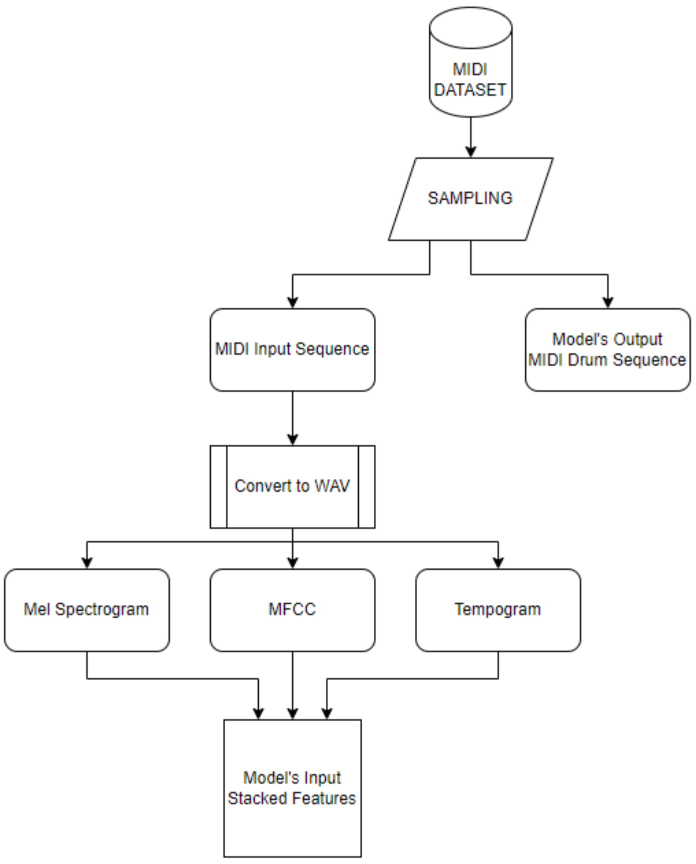


Figure 4.1. Preprocessing Steps of the WAV Accompaniment Network

Given that each input data is sampled with a rectangular window, the modified equations, Eq. (3.1), (3.2), and (3.3), for the sampling can be written as

$$t_0 \leq t < t_1 : sample_1 = \left[\begin{bmatrix} n_{1_1} \\ v_{1_1} \\ b_{1_1} \end{bmatrix}, \begin{bmatrix} n_{1_2} \\ v_{1_2} \\ b_{1_2} \end{bmatrix}, \dots, \begin{bmatrix} n_{1_k} \\ v_{1_k} \\ b_{1_k} \end{bmatrix} \right] \tag{4.1}$$

$$t_1 \leq t < t_2 : sample_2 = \left[\begin{bmatrix} n_{2_1} \\ v_{2_1} \\ b_{2_1} \end{bmatrix}, \begin{bmatrix} n_{2_2} \\ v_{2_2} \\ b_{2_2} \end{bmatrix}, \dots, \begin{bmatrix} n_{2_k} \\ v_{2_k} \\ b_{2_k} \end{bmatrix} \right] \tag{4.2}$$

$$\vdots$$

$$t_{m-1} \leq t < t_m : sample_m = \left[\begin{bmatrix} n_{m_1} \\ v_{m_1} \\ b_{m_1} \end{bmatrix}, \begin{bmatrix} n_{m_2} \\ v_{m_2} \\ b_{m_2} \end{bmatrix}, \ldots, \begin{bmatrix} n_{m_k} \\ v_{m_k} \\ b_{m_k} \end{bmatrix} \right] \tag{4.3}$$

where t_0 corresponds to initial starting time of the MIDI file and the sample's next time instant is calculated as $t_{i+1} = \sum_{i=0}^{m} t_i + t_w$. t_w denotes the window length in seconds, n, v are note's pitch, and velocity respectively, the number of samples denoted by $m = \{\eta : \eta \geq 1, \eta \in Z^+\}$, and b is the instrument number where $b = 1$ corresponds to drums, and $b = 0$ means that the note and velocity value of that sample corresponds to any other instrument but drums.

In the thesis, the rectangular window's length, t_w, was chosen as 1 second. If the window size is chosen too short, the model's training time goes up due to the increased number of input-output pairs, which also improves the performance of the model. On the other hand, if the sampling rate is chosen too high, the model's training time goes down and the performance of the model is compromised.

We changed the input type to be an m-tuple instead of a matrix, where m is the number of samples. The reason was that the input samples didn't need to be zero padded since they were going to be converted into WAV signals. However, before the calculations of the features any WAV file whose length is less than 1 second were zero padded. Furthermore, Eq (3.5) was used to calculate the input and Eq. (3.4) and (3.5) were used for the output. Then, every row of input tuple converted to WAV file by using FluidSynth library (Newmarch 2017). The Mel spectrogram, MFCC, and tempogram features were extracted from the sample by using librosa library (McFee et al. 2015). Thereafter, these features were stacked on top of each other, and the model's input vector was made ready. No additional steps were required for the model's output vector, and they were calculated by using Eq. (3.4), (3.5), (3.14), (3.17), (3.21), (3.24) and (3.25).

To ease the understanding of the steps better, we would like to explain the concept with an example. Let's assume that we are given the Table 3.1 as our dataset and the rectangular window's length, t_w, is chosen as 1 second as described above. Lastly, we choose the maximum pattern length to be 20. Furthermore, we choose number of FFT to be 1024, hop length to be 512, and number of MFCC to be 20 for this example. Then, samples can be written as follows

$2.03636s \leq t < 3.03636s$:

$$sample_1 = \left[\begin{bmatrix}37\\109\\1\end{bmatrix}, \begin{bmatrix}44\\109\\1\end{bmatrix}, \begin{bmatrix}51\\109\\1\end{bmatrix}, \begin{bmatrix}64\\109\\0\end{bmatrix}, \begin{bmatrix}44\\109\\1\end{bmatrix}, \begin{bmatrix}66\\109\\0\end{bmatrix}\right] \tag{4.4}$$

$3.03636s \leq t < 4.03636s$:

$$sample_2 = \left[\begin{bmatrix}37\\109\\1\end{bmatrix}, \begin{bmatrix}44\\109\\1\end{bmatrix}, \begin{bmatrix}51\\109\\1\end{bmatrix}, \begin{bmatrix}67\\109\\0\end{bmatrix}, \begin{bmatrix}51\\118\\1\end{bmatrix}, \begin{bmatrix}72\\118\\0\end{bmatrix}, \begin{bmatrix}33\\80\\0\end{bmatrix}, \dots, \begin{bmatrix}60\\31\\0\end{bmatrix}\right] \tag{4.5}$$

$4.03636s \leq t < 5.03636s$:

$$sample_3 = \left[\begin{bmatrix}42\\90\\1\end{bmatrix}, \begin{bmatrix}51\\90\\1\end{bmatrix}, \begin{bmatrix}35\\80\\1\end{bmatrix}, \begin{bmatrix}51\\80\\1\end{bmatrix}, \begin{bmatrix}28\\72\\0\end{bmatrix}, \begin{bmatrix}42\\45\\1\end{bmatrix}, \begin{bmatrix}53\\45\\1\end{bmatrix}, \begin{bmatrix}60\\45\\0\end{bmatrix}, \dots, \begin{bmatrix}60\\52\\0\end{bmatrix}\right] \tag{4.6}$$

$5.03636s \leq t < 6.03636s$:

$$sample_4 = \left[\begin{bmatrix}51\\70\\1\end{bmatrix}, \begin{bmatrix}35\\80\\1\end{bmatrix}, \begin{bmatrix}42\\80\\1\end{bmatrix}, \begin{bmatrix}38\\80\\0\end{bmatrix}\right] \tag{4.7}$$

Since we want to obtain WAV files of the input sequences, we create an m-tuple instead of a matrix as described earlier. Then, by using Eq. (3.5) for the input and Eq. (3.4) and (3.5) for the output, we obtain the following

$$input = \begin{pmatrix} \begin{bmatrix}64\\109\end{bmatrix} \quad \begin{bmatrix}66\\109\end{bmatrix} \\ \begin{bmatrix}67\\109\end{bmatrix} \quad \begin{bmatrix}72\\118\end{bmatrix} \quad \begin{bmatrix}33\\80\end{bmatrix} \quad \dots \quad \begin{bmatrix}60\\31\end{bmatrix} \\ \begin{bmatrix}28\\72\end{bmatrix} \quad \begin{bmatrix}60\\45\end{bmatrix} \quad \dots \quad \begin{bmatrix}60\\52\end{bmatrix} \\ \begin{bmatrix}38\\80\end{bmatrix} \end{pmatrix}_4 \tag{4.8}$$

$$output = \begin{bmatrix} \begin{bmatrix} 37 \\ 109 \end{bmatrix} & \begin{bmatrix} 44 \\ 109 \end{bmatrix} & \begin{bmatrix} 51 \\ 109 \end{bmatrix} & \begin{bmatrix} 44 \\ 109 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \cdots & \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 37 \\ 109 \end{bmatrix} & \begin{bmatrix} 44 \\ 109 \end{bmatrix} & \begin{bmatrix} 51 \\ 109 \end{bmatrix} & \begin{bmatrix} 51 \\ 118 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \cdots & \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 42 \\ 90 \end{bmatrix} & \begin{bmatrix} 51 \\ 90 \end{bmatrix} & \begin{bmatrix} 35 \\ 80 \end{bmatrix} & \begin{bmatrix} 51 \\ 80 \end{bmatrix} & \begin{bmatrix} 42 \\ 45 \end{bmatrix} & \begin{bmatrix} 53 \\ 45 \end{bmatrix} & \cdots & \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 51 \\ 70 \end{bmatrix} & \begin{bmatrix} 35 \\ 80 \end{bmatrix} & \begin{bmatrix} 42 \\ 80 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \cdots & \begin{bmatrix} 0 \\ 0 \end{bmatrix} \end{bmatrix}_{4x20x2} \tag{4.9}$$

As the next step, new MIDI files are created for each row of the input and their true note starting and note ending times as well as their instrument type are used from the Table 3.1. By converting these MIDI files to WAV and zero padding in order to complete to a second, we get the following signal waveforms,

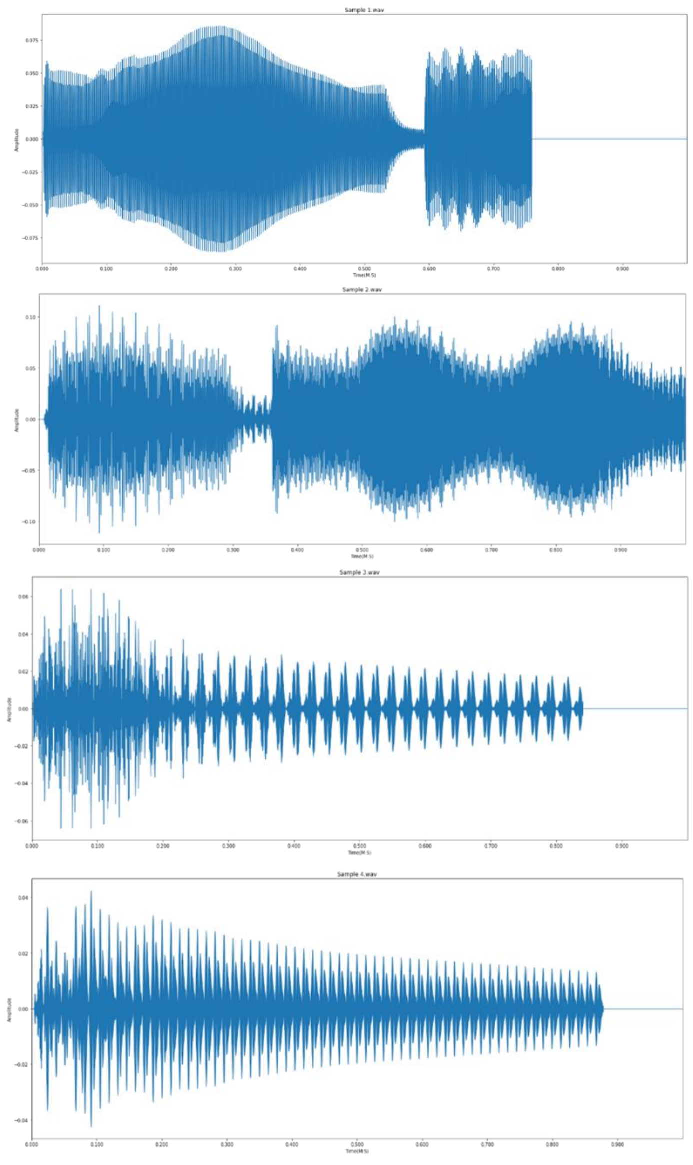


Figure 4.2. Wav Signal Forms of the Samples

Now that we have the signal waveforms of the MIDI data, we can calculate the input features. First, by following the instructions described in Section 2.3.1 we obtain the following Mel spectrograms for each sample

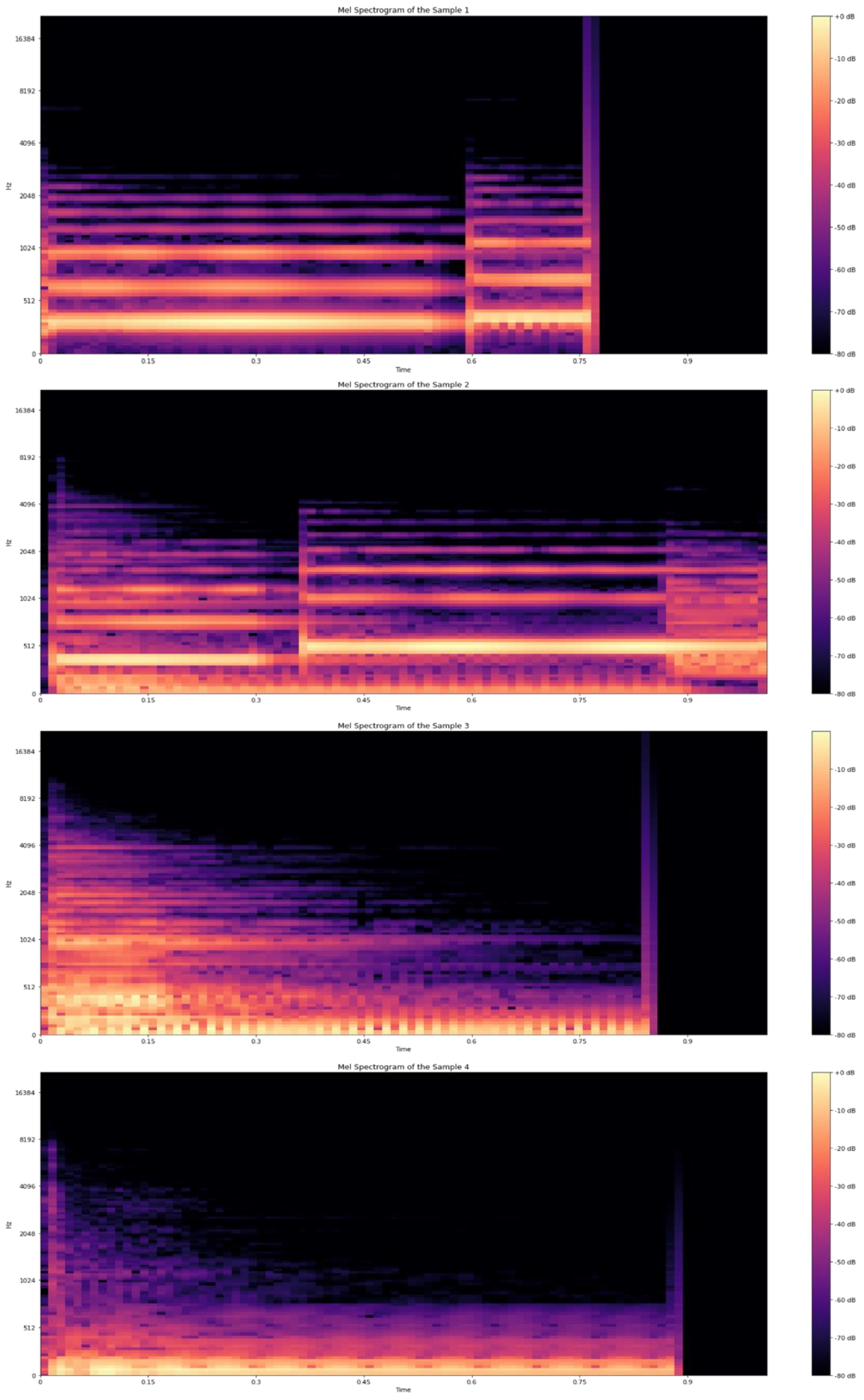


Figure 4.3. Mel Spectrogram of the Samples

As the next step, the MFCC features of the samples are calculated by following the instructions described in Section 2.3.2. Thus, we obtain the following MFCCs

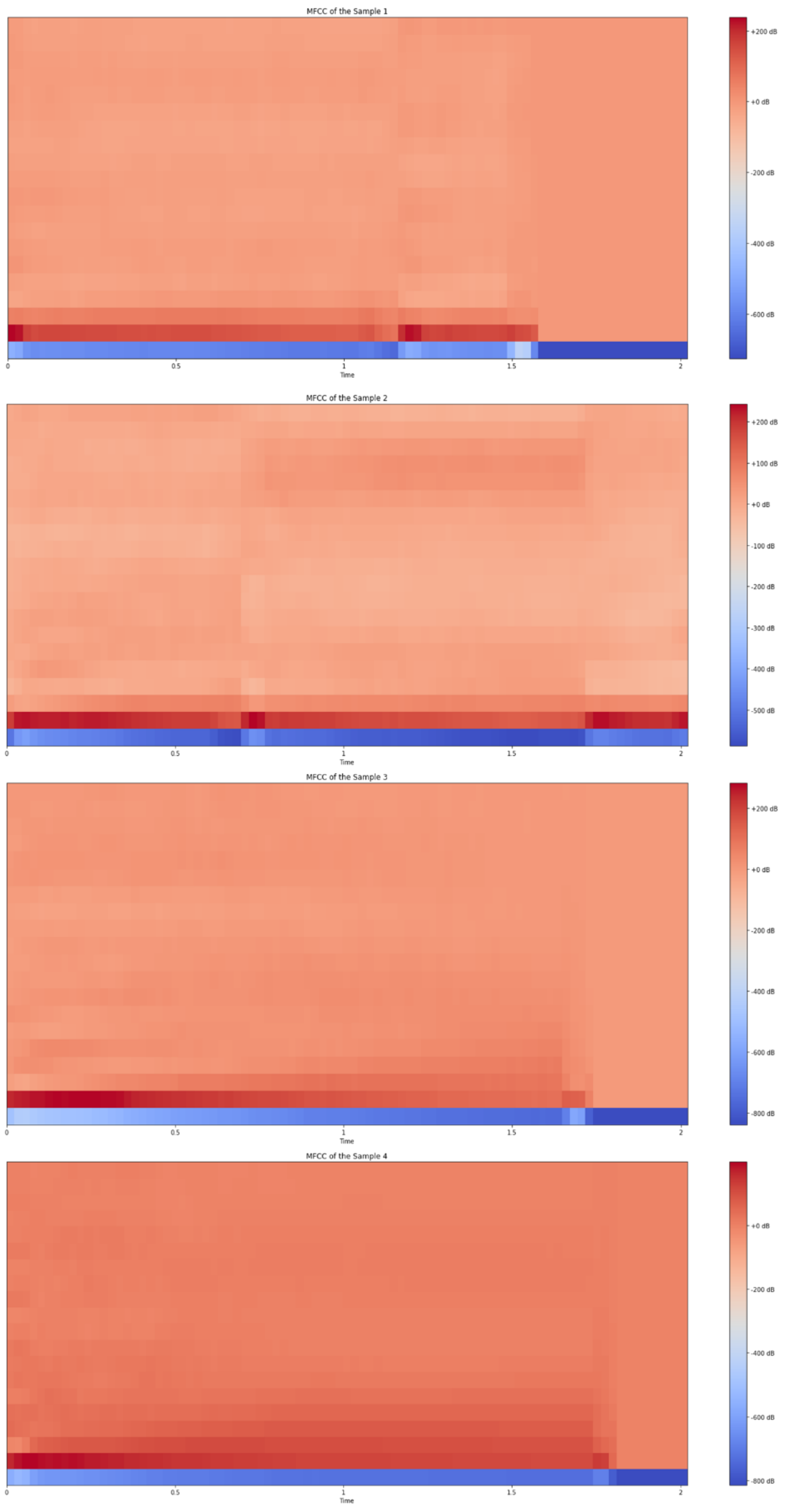


Figure 4.4. MFCCs of the Samples

Lastly, by using the Eq. (2.5), (2.6), and (2.7) given in Section 2.3.3 the tempograms are obtained as follows

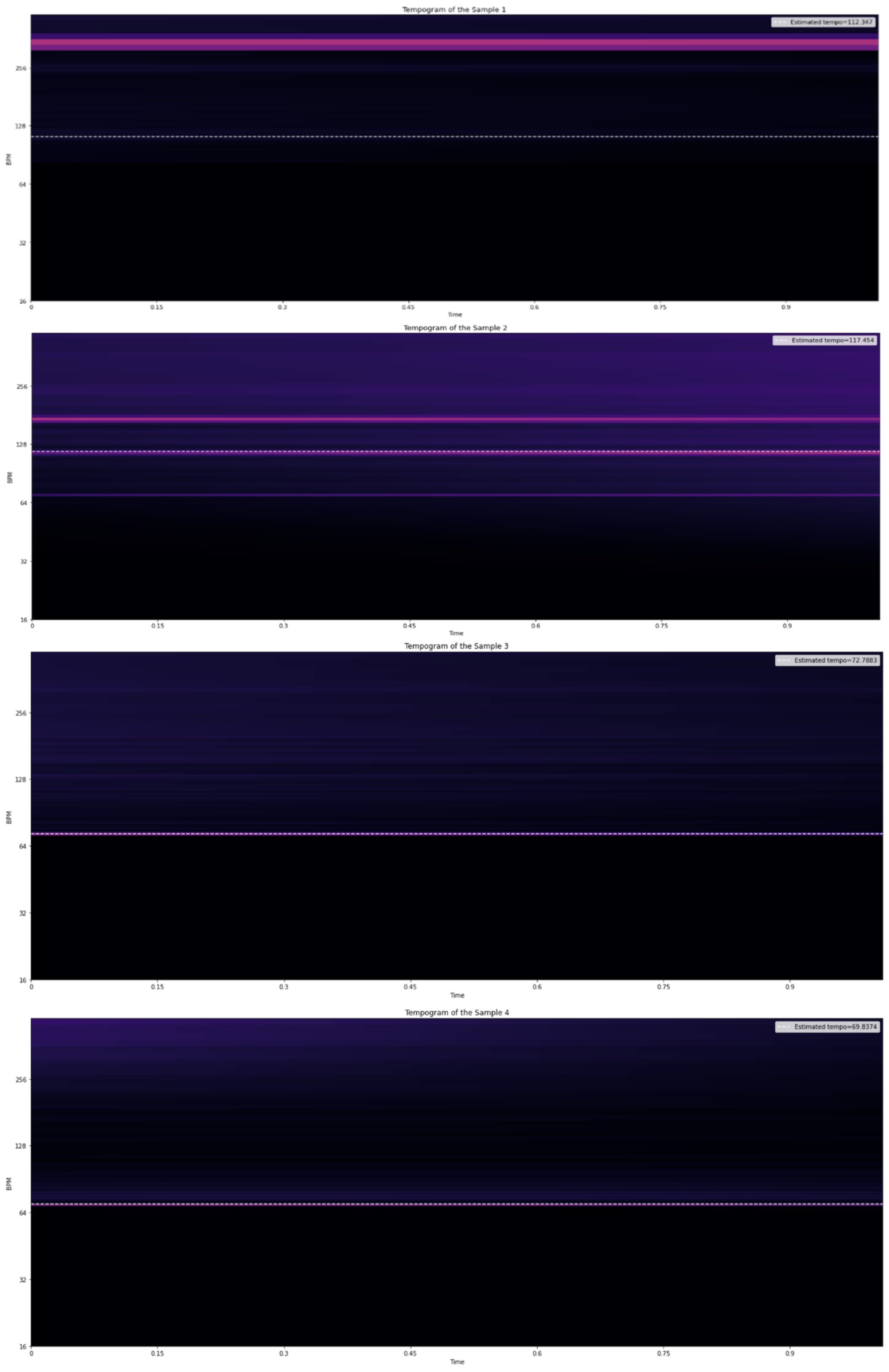


Figure 4.5. Tempogram of the Samples

At the end, these features are concatenated along the z-axis to create 3 featured input vectors for the model's input. Moreover, by using Eq. (3.17) and Eq. (3.21) the encoded outputs, $encoded_{notes_{output}}$ and $encoded_{velocity_{output}}$, for the model's output are obtained as follows

$$encoded_{notes_{output}} = \begin{bmatrix} [OHE[4,23]] & [OHE[10,23]] & [OHE[17,23]] & \dots & [OHE[1,23]] \\ [OHE[4,23]] & [OHE[10,23]] & [OHE[17,23]] & \dots & [OHE[1,23]] \\ [OHE[8,23]] & [OHE[17,23]] & [OHE[2,23]] & \dots & [OHE[1,23]] \\ [OHE[17,23]] & [OHE[2,23]] & [OHE[8,23]] & \dots & [OHE[1,23]] \end{bmatrix}_{4x20x1x23} \quad (4.10)$$

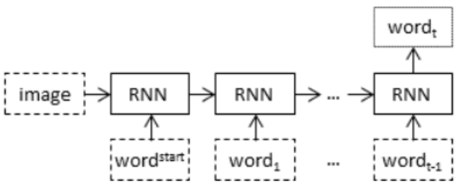
$$encoded_{velocity_{output}} = \begin{bmatrix} [OHE[38,128]] & [OHE[45,128]] & [OHE[52,128]] & \dots & [OHE[1,128]] \\ [OHE[38,128]] & [OHE[45,128]] & [OHE[52,128]] & \dots & [OHE[1,128]] \\ [OHE[43,128]] & [OHE[52,128]] & [OHE[36,128]] & \dots & [OHE[1,128]] \\ [OHE[52,128]] & [OHE[36,128]] & [OHE[43,128]] & \dots & [OHE[1,128]] \end{bmatrix}_{4x20x1x128} \quad (4.11)$$

In this section, the preprocessing steps are described. These steps prepare the WAV files for further processing. In the next section training procedure is discussed using the inputs and outputs described above.

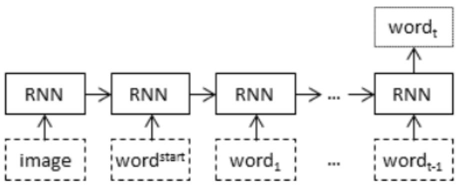
4.2.2. Training Model

The traditional neural network models used in time series forecasting aren't suitable for raw signal forms alone. In addition, we don't have a time series input pattern like we had in Chapter 3. Instead, the input data was preprocessed, and three features were extracted. Therefore, we require additional steps to analyze the characteristics of the input features before feeding into a Seq2Seq model. Recall that these features were stacked on top of each other and represented as a 3-channel RGB image. As a consequence of this, one of the best models, CNNs, were preferred to analyze this type of data. Thus, we were able to transform a Seq2Seq application into an image captioning application.

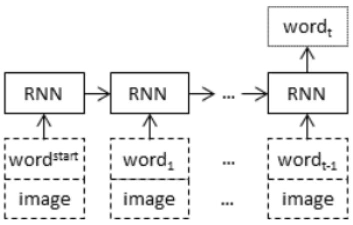
Image captioning is a neural network application which describes contents of an image with sentences. The model consists of encoder-decoder architecture where the encoder is a CNN, and the decoder is an RNN or an LSTM. The model requires two inputs one is an image to be described for the encoder, and the other one is a text sequence for the decoder. The main question that should be answered with this architecture is where to place the image in a model. To answer this question, the following figure should be studied with the description of each type.



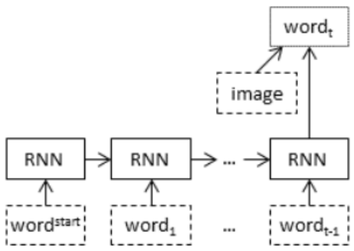
(a) Init-inject: The image vector is used as an initial hidden state vector for the RNN.



(b) Pre-inject: The image vector is used as a first word in the prefix.



(c) Par-inject: The RNN accepts two inputs at once in every time step: a word and an image.



(d) Merge: The image vector is merged with the prefix outside of the RNN.

Figure 4.6. Different Types of the Image Captioning Models

(Source: TANTI, GATT and CAMILLERI 2018)

Moreover, there are three additional architectures that are proposed by Andrej Karpathy (Karpathy and Fei-Fei 2017), Google (Vinyals et al. 2015), and Microsoft (Tran et al. 2016). Since Google's architecture was used in the thesis, we are going to explain it further.

In contrast to other architectures, Google’s architecture uses LSTM instead of RNN. The architecture is given in the following figure.

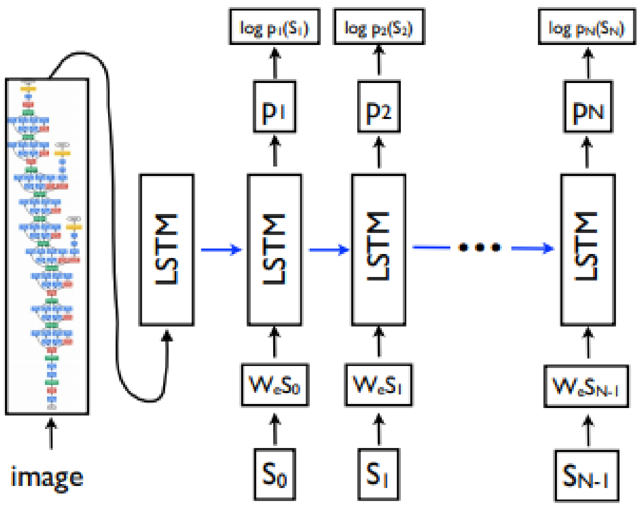


Figure 4.7. Google's Image Captioning Model

(Source: Vinyals et al. 2015)

The image is fed to the input of the encoder and its features are extracted via a CNN. Subsequently, the extracted features are given to LSTM as an input, and the context vector is calculated. The context vector becomes the decoder's initial hidden state and the decoder's input is fed with the corresponding text sequence. Hence, the model trains so that it is able to make a relationship between the image's context vector and the target text sequence. The equations of the architecture (Vinyals et al. 2015) are given as follows

$$x_{-1} = CNN(I) \tag{4.12}$$

$$x_t = W_e S_t \tag{4.13}$$

$$p_{t+1} = LSTM(x_t) \tag{4.14}$$

where I denotes the input image, S_t is the one-hot encoded representation of the text description of the image, $t \in \{0, \dots, N-1\}$ and N is the sentence length, W_e represents the embedding's weights, p_{t+1} is the probability distribution which is the SoftMax applied output of the LSTM given in Eq. (2.39).

Now that the working principle of image captioning model is described, we can update the Seq2Seq model used in MIDI accompany network to work with WAV files.

Recall that the inputs preprocessed and 3-channel RGB image like features were created. Therefore, using the idea of the image captioning model is suitable for the application. The model given in Figure 2.22 was updated by adding a CNN architecture. Thus, the new encoder is a combination of CNN and LSTM, but the decoder is still the same as in our MIDI based model. The new updated Seq2Seq model with teacher forcing can be seen in the following figure.

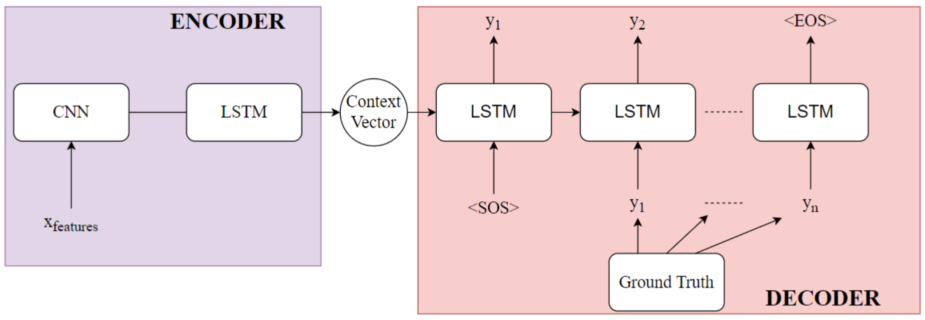


Figure 4.8. Updated Seq2Seq Model

In this thesis, ResNet's model (He et al. 2016) was used as the CNN. ResNet is a widely used pretrained CNN model for image recognition. Even though the pretrained network weights are provided, we retrained the network due to the specific nature of our input images.

The model was trained on a Tesla K80 GPU, which is provided by Google Colaboratory (Google Research 2017), in order to accelerate the learning process. The model completed its training in 200 epochs for the six different dataset and same training parameters were used in Section 3.2.2. The number of samples each dataset contained is given in the following table.

Table 4.1. Number of Samples in Datasets

Dataset Name	**Number of MIDI Files**	**Number of Samples**
Blues	50	9194
HipHop-Rap	42	8314
Jazz	43	7622

(cont. on next page)

Table 4.1. (cont.)

Pop	50	9607
Rock	41	9496
Shuffle	50	10600

4.2.3. Generator Model

The goal of this application is to give input in a signal form and provide a drum pattern that is in harmony with the original track. In order to accomplish the task, the model was trained, using MIDI files, with the physical features that are extracted from the signal waveforms. Therefore, without the need of the MIDI format, the raw WAV files can be used as the input for the generator model. In this model, we were not able to feed the model with the ground truths. Therefore, we used the saved training weights and dropped out the teacher forcing method. In addition, the temperature sampling method was used to improve the quality of the generated patterns. Moreover, by their nature, we know that human drum performers can’t play more than four notes simultaneously. Therefore, we limited the generator model to generate four consecutive drum samples for each time instant. In order to generate more natural patterns.

The steps that the generator model follows can be explained further as follows: the model first takes a given WAV file and by using preprocessing methods mentioned in Section 4.2.1, the input is prepared for the generation. The features are fed into an LSTM, which is the last part of the encoder, to extract the context vector. The context vector is given to the decoder as the initial hidden state, and the SOS token is given as the input of the decoder in order to start the generation process. Lastly, the model produces new drum patterns, that includes no more than four drum instruments, for each sample.

Recall that we sampled each WAV file with a rectangular window with the window length of a second. While there's nothing wrong with that, the newly created WAV file does feel very artificial when the generated drum patterns are combined with the original track. This is because the drum patterns formed by sampling WAV files at one second intervals. The previous work, which is the MIDI accompany network, didn’t encounter such a problem because each sample of the MIDI file was in sync with the rhythm so that the generated drum patterns were able to blend in. Thereby, we needed either a new sampling technique that is not in a certain time interval or we had to obtain each note's starting time as in MIDI format. To alleviate these problems, we preferred to use the start times of the notes in each sample so that the generated drum patterns will be able to keep up with the rhythm. One of many ways to solve this problem was to use the following onset detection algorithm.

The beginning of a music notes or a sound is called an onset. Onset detection algorithms can work in the time domain, frequency domain, phase domain, and complex domain. Although the concept of onset, transient, and attack are related to each other, there are clear differences between them. The onset of a note corresponds to the beginning of the transient. The attack represents the envelope increase of the amplitude (Bello et al. 2005).

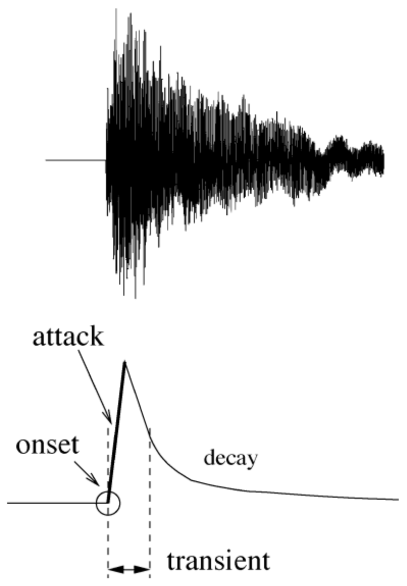


Figure 4.9. Attack, Transient, Decay, Onset of a Single Note
(Source: Bello et al. 2005)

The onset detection algorithm is an active area in music signal processing that has been used for many applications. One of the applications was used in the research about generating thumbnails of Irish traditional music (Kelly et al. 2010). The following figure, which is taken from the research, can be given an example to show capability of the onset detection to detect note timings. Despite some inaccuracies, we can see that the onset detection was able to identify the note's starting positions.

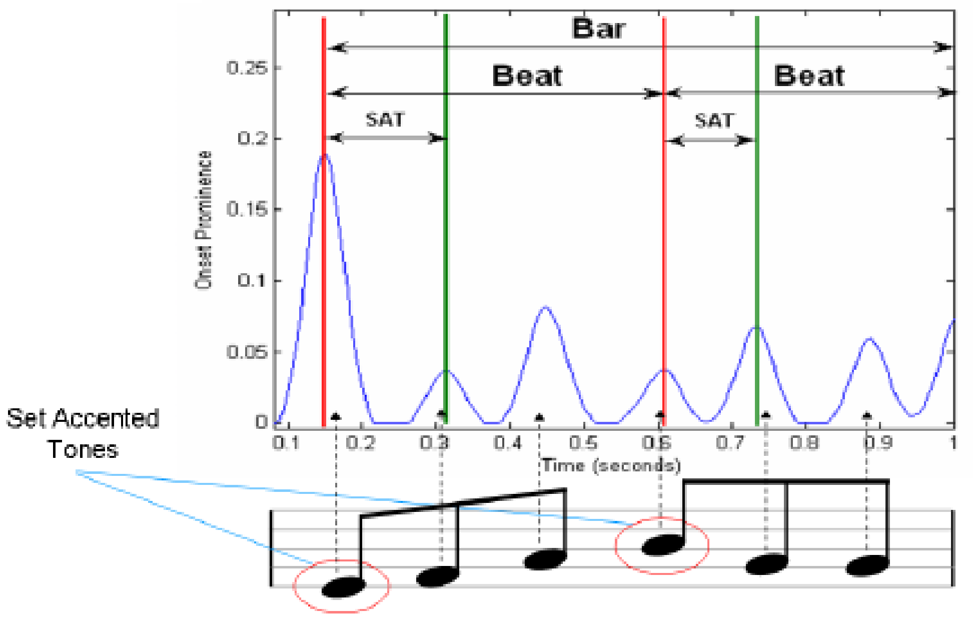


Figure 4.10. Onset Detection of an Irish Traditional Music
(Source: Kelly et al. 2010)

By using the onset detection algorithm on the sampled WAV files, we were able to obtain each note's starting time. With only a few extra steps, model has started to produce more humanized and unique results. In addition, the UI, given in Section 3.2.3, was updated to work with WAV files.

4.3. Results

We used the MIDI format to help analyze the WAV files. As it in the previous application, MIDI format first separated to input and output parts. The input is converted to WAV and its Mel spectrogram, MFCC, and tempogram are used as the input features. The output remained same and the information that belongs to drums were used. Moreover, the model was trained with six different datasets with 200 epochs each.

In neural network applications, datasets are usually split as training and validation samples to prevent overfitting and evaluate the model's performance during training. However, in this application, the dataset was not split into two and remained as one training set since the size of the dataset is limited and the main idea of our study is in the field of art. Therefore, the qualitative results were preferred instead of quantitative. In the figure 4.12, 4.13, and 4.14, we present the graphs on the improvement of loss and accuracy and change of learning rate over epoch for each dataset. From the graphs, the one can see that the model is able to converge in 200 epochs.

In the following figure, we provide a QR code to show the example outputs that has produced by the algorithm. The user is able to generate unique and interesting drum patterns due to the diversity of genres used in training. As can be seen from these examples, we can see that using MIDI format to analyze WAV files is a good approach to overcome the complexity of the raw signals.

Figure 4.11. QR Code for Example WAV Outputs[3]

[3] "WAV Drum Accompaniment Network". 2022. YouTube. https://www.youtube.com/playlist?list=PLFhr-SY0apsyTBya35OZM_XbYCZJhPIek. (Short URL: https://tinyurl.com/3twf9ht4)

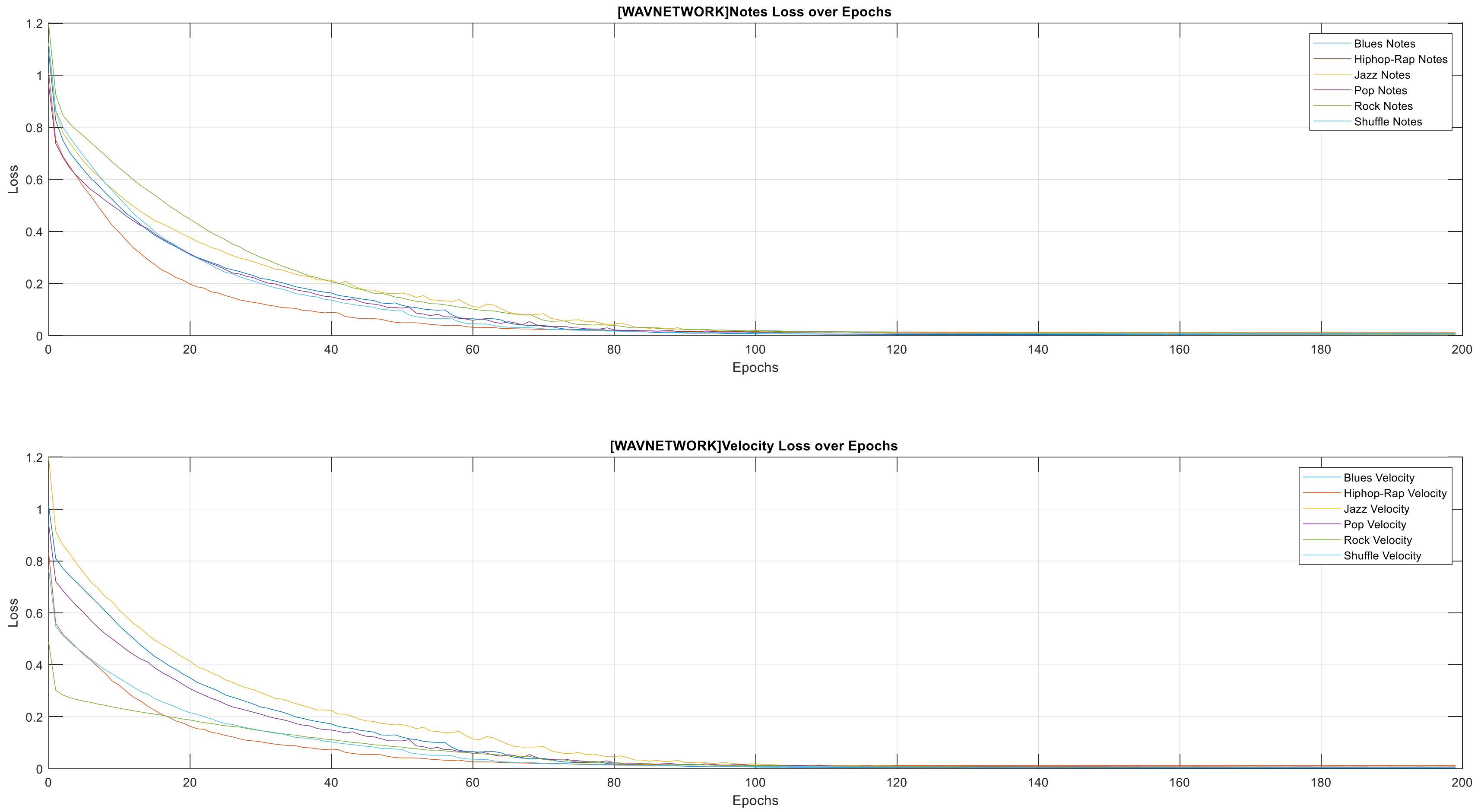


Figure 4.12. WAVNETWORK - Losses of Note and Velocity over Epochs

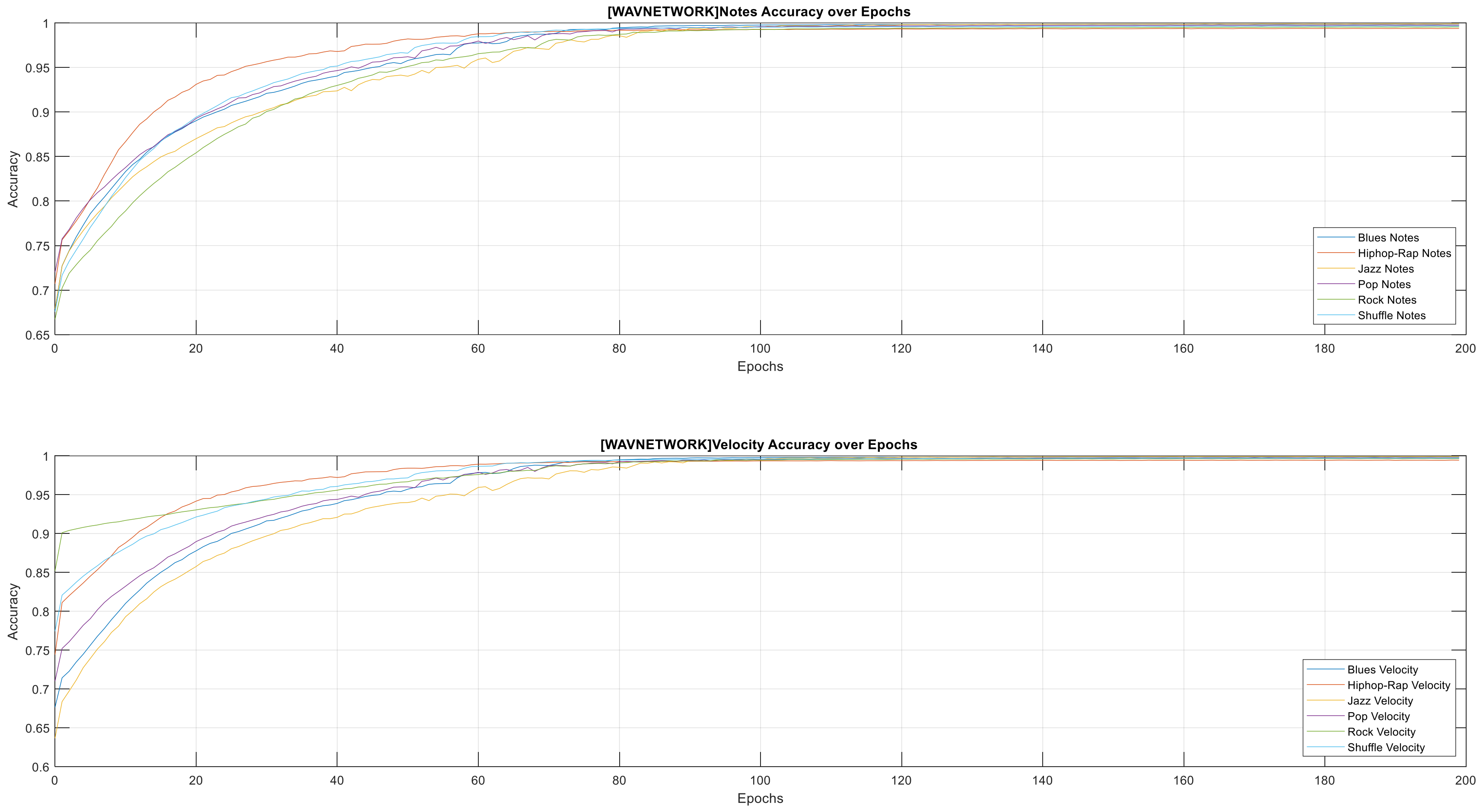


Figure 4.13. WAVNETWORK - Accuracies of Note and Velocity over Epochs

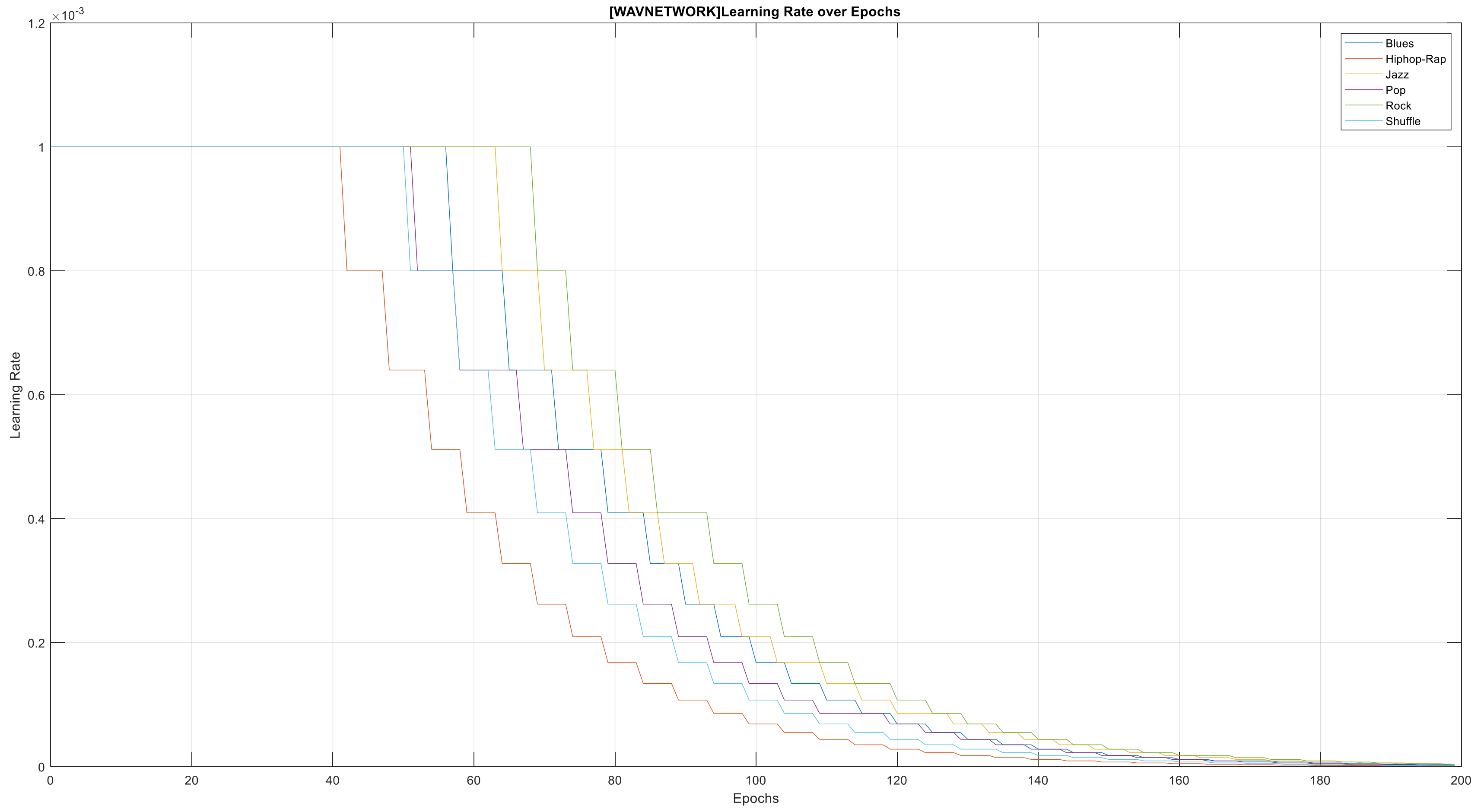


Figure 4.14. WAVNETWORK - Change of Learning Rate over Epochs

4.4. Discussion

The main motivation to propose this novel approach was to improve the MIDI accompaniment network so that it can be used for raw recordings of music. The MIDI format contains useful information such as instrument types, exact note timings, note values, velocities, etc. On the other hand, the WAV files don't have this simplicity, but Mel spectrogram, MFCC, tempogram and such characteristic features can be obtained from them. Hence, in order to update the MIDI accompaniment network, the idea we came up with was to use the best features of the two worlds. Therefore, we first take MIDI dataset and split into input and output as defined in Chapter 3. Then, we convert the input data to WAV file and extracted three features which are Mel spectrogram, MFCC, and tempogram. The model's input became the three channel data, and the output just one-hot encoded MIDI drum sequence. The model was able to understand the relationship between input and output, but we want to mention a few possible improvements.

As it was discussed in Section 4.2.3, our model's generated drum patterns are located at the time instants that are calculated by the onset detection algorithm. Although this works fine, the timing of the rhythm can be improved by including the true note timings as another feature. Furthermore, different types of signal features or more features can be used. On the contrary, the model might have become overwhelmed due to too many features so using fewer features may increase the quality of generated patterns. In addition, the model can be implemented to work in real-time. Since the features are not really images but signal characteristics, a different type of architecture can be used instead of CNN. Moreover, by using the basic rules of music theory, the quantitative results can be generated.

CHAPTER 5

CONCLUSION AND FUTURE DIRECTIONS

If I have the belief that I can do it,
I shall surely acquire the capacity to do it
even if I may not have it at the beginning.
-Mahatma Gandhi-

The drums not only accompany other instruments in a band but, also, play a deciding factor for the rhythm, dynamics, and melody (Jack, n.d.). In addition, they are referred to be a backbone of a song. Good drum patterns maintain a steady groove while changing the dynamics by increasing/decreasing the volume and intensity (Stams 2013). Moreover, drum patterns keep other instruments as a whole during the move from one part of the song to another (e.g., verse to pre-chorus, chorus to bridge, etc.). When these unique features are used in the correct way, drums become a powerful tool to bring songs to life.

In the light of these special characteristics of drums, our perspective was to enhance the feelings of songs by adding drum patterns. In this sense, this thesis proposes two algorithms for two different types of music data formats. The first algorithm, presented in Chapter 3, uses MIDI format due to the easy access to useful features. In this application, firstly, features were grouped according to their instrument type where features that belong to drums were specified as the output and the others were specified as the input. Thereafter, both input and output were one-hot encoded and became ready for the model. The Seq2Seq neural network model was chosen to train the algorithm. In order to improve the results, the temperature sampling method was used. In the second algorithm, presented in Chapter 4, our purpose was to design a drum accompaniment network using audio signal waveforms as input. In this algorithm, MIDI format files were used to support the training. The rationale behind this idea was that the same features used in the MIDI accompaniment network were difficult to extract from WAV files. Hence, after the feature extraction from MIDI, we converted input data to WAV and extracted signal features using Mel spectrogram, MFCC, and tempogram. This led our model to learn the relationship between actual signal features and the suitable drum patterns.

The performance results were provided for both models in Section 3.3 and in Section 4.3. It was shown that both models were able to converge for given datasets with the help of the teacher forcing method. The results generated by these models can be improved with minor user interaction to achieve qualitatively pleasing results. However, quantitative evaluation is not attempted. Therefore, it is not certain that the model has the capability to generate perfect suitable drum patterns for every song. This is a critical point to be noted in order to generalize and improve the algorithm by developing new techniques to measure quantitative results in any area of creative art.

To summarize, two algorithms were that address the unique characteristics of the drums. The main goal of both algorithms was to enhance the feelings of musical pieces, and we were able to achieve acceptable results. Lastly, we showed that using waveforms with MIDI formats to analyze music is a reasonable approach which creates possibilities for the on-the-fly creation of original drum parts in real-time performances by human performers.

5.1. Future Directions

In the light of our motivation, we were able to accomplish production of drum patterns to accompany a given musical piece. During the thesis work, we saw that some commonly used methods were insufficient for solving some of the difficulties we encountered. On the other hand, some of the techniques we used worked well. Based on these experiences, we want to provide the following sections to discuss potential scientific research directions.

5.1.1. Multi-Instrumental Music Generation

In our application, we focused on only generating drum patterns for a given song. However, looking at the information MIDI format provides, we can easily observe that multi-instrumental music generation at a MIDI level is possible. One possible approach can be making a deep model to analyze each instrument's order, and then their corresponding notes, timings, and velocities. Moreover, by using the MIDI format as the supporting factor as done in the thesis, the capabilities can be expanded for audio waveform formats as well.

Furthermore, the correlation between non-drum and drum instruments can be measured, and highly correlated instruments can contribute more to the decision of generated drum pitches and velocities. This can be done by optimizing and training a word2vec embedding but for instruments instead of words. The word2vec is a vocabulary encoding which learns word embeddings and their similarity. Similarly, co-occurrence matrix can be used to see relationship of the instruments with each other.

5.1.2. Methods of Measuring Quantitative Results in the Arts

In the field of natural language processing, similar applications are measured with an automatic evaluation metric called BLEU score. BLEU score is a number between 0 and 1 that measures the similarities between machine-translated text to a reference text (Papineni et al. 2001). Although this method works well for machine translation models, we believe that a different approach should be developed for applications on artistic performance.

Even though in real life, every person has a different taste in art, we believe that it is possible to measure quantitative results at a significant level by using basic rules of the art of interest. One example that can be given from our application is as follows: we could use the music theory at the basic level and the common drum patterns as the reference. In this way, we can try to calculate the distance between the generated and the common patterns. Additionally, we can check if the generated patterns obey the rules of the music theory.

5.1.3. Roles of the Drums in Music

Drums can play many roles in a band. They can be used to ornament the notes, create steady grooves, fill in gaps or create solos where they become the headliner of the song. Out of all possibilities, in this work, we treated drum patterns as an ornamentation rather than grooves. In upcoming improvements, we believe that exploiting different characteristics of the drums will lead to more musically pleasing. For example, the model can be trained first to create its own groove, and then to ornament the groove with different drum instruments (e.g., cymbals, bass drums, etc.) in order to pass the feeling of the song better.

5.1.4. Inclusion of the Timing Information

The correct timing of the notes is crucial for making music since it defines the certain musical features such as rhythm, etc. In the thesis work, in order to lower the complexity of the network, we decided to use already existed note timings in the MIDI generated patterns, and onset detection algorithm in WAV generated patterns. Although the idea was worked well enough, we believe that the capability of the network and uniqueness of generated patterns can be improved by using timing as another feature. There are different ways to represent the timing information. In the following bullet points, we are going to describe the two possible ideas.

- **Discretization of the time axis:** The time axis of the musical piece can be discretized by defining the sampling period (T_s) to a specific note length. The time axis can be grouped and divided by sixteenth notes. Therefore, the generated notes can be placed into one of these sample points. To facilitate the understanding, the following example is provided. Let's assume that we have a musical piece with ***120 bpm*** where its sixteenth note length is ***125 milliseconds***.

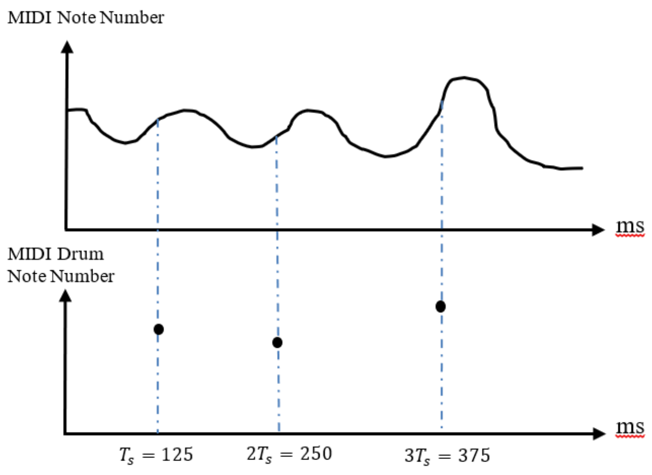


Figure 5.1. Discretized Time Axis

In order to create realistic human-like performance, random offset values can be used to place drumbeats in slightly varying positions.

- **Note length representation:** A more elegant way to include timing information is to consider every possible note length value. In music theory, there are two concepts that are related to each other and defines the timing. These are bpm, which describes how fast the song is going to be played in terms of quarter notes, and note lengths, which are timing representation of notes such as quarter note, whole note, half note, etc. Moreover, the note lengths can be converted to actual seconds according to bpm of the song. Some of bpm to the most used note length formulas can be seen in the following table.

Table 5.1. Note Length Formulas

Note Type	Normal (in seconds)	Dotted (in seconds)	Triplet (in seconds)
Half	$120/BPM$	$180/BPM$	$80/BPM$
Quarter	$60/BPM$	$90/BPM$	$40/BPM$
Eighth	$30/BPM$	$45/BPM$	$20/BPM$
Sixteenth	$15/BPM$	$22.5/BPM$	$10/BPM$

Furthermore, note lengths, pitches, and velocity information can be represented in a 3D matrix where x-axis is the timestep of the note pitch in each sample, y-axis corresponds to note length values, z-axis is the note pitch values which are between 0 - 128. Therefore, the following example is given to ease the understanding of the idea.

Let us have the following input of note pitches and velocities along with their note lengths with sample rate of 5.

$$notes = [35\ \ 32\ \ 44\ \ 32\ \ 35] \tag{5.1}$$

$$noteLengths = [half\ \ dotted - half\ \ quarter\ \ sixteenth\ \ eighth] \tag{5.2}$$

Therefore, the data representation for the given sample that contains timing information can become as follows,

$$sample = \begin{array}{r|ccccc}
half & 35 & 0 & 0 & 0 & 0 \\
quarter & 0 & 0 & 44 & 0 & 0 \\
eighth & 0 & 0 & 0 & 0 & 35 \\
sixteenth & 0 & 0 & 0 & 32 & 0 \\
dotted - half & 0 & 32 & 0 & 0 & 0 \\
dotted - quarter & 0 & 0 & 0 & 0 & 0 \\
dotted - eighth & 0 & 0 & 0 & 0 & 0 \\
dotted - sixteenth & 0 & 0 & 0 & 0 & 0 \\
triplet - half & 0 & 0 & 0 & 0 & 0 \\
triplet - quarter & 0 & 0 & 0 & 0 & 0 \\
triplet - eighth & 0 & 0 & 0 & 0 & 0 \\
triplet - sixteenth & 0 & 0 & 0 & 0 & 0 \\
\hline
 & 1 & 2 & 3 & 4 & 5
\end{array}$$

Even though this can be one of the best approaches to include actual true note timings, there is a shortcoming. Generally, there are 130 different note length values although some of them are not used most of the time. The dimension of this representation depends on number of note length values and sampling rate. Therefore, this idea may end up being very memory inefficient if its complexity is not simplified according to the specific application.

REFERENCES

"Deep Learning For Siri’S Voice: On-Device Deep Mixture Density Networks For Hybrid Unit Selection Synthesis". 2017. *Apple Machine Learning Research*. https://machinelearning.apple.com/research/siri-voices.

"What Is Categorical Data | Categorical Data Encoding Methods". 2020. *Analytics Vidhya*. https://www.analyticsvidhya.com/blog/2020/08/types-of-categorical-data-encoding/.

"Write A Sequence To Sequence (Seq2seq) Model — Chainer 7.8.0 Documentation". n.d. *Docs.Chainer.Org*. https://docs.chainer.org/en/v7.8.0/examples/seq2seq.html.

Abadi, Martín, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, and Greg S. Corrado et al. 2015. "Tensorflow: Large-Scale Machine Learning On Heterogeneous Distributed Systems". https://www.tensorflow.org/.

Back, David. 1999. "Standard MIDI File Format, Updated". *Music.Mcgill.Ca*. http://www.music.mcgill.ca/~ich/classes/mumt306/StandardMIDIfileformat.html#BMA1_5.

Bäckström, Tom. 2019. "Cepstrum And MFCC - Introduction To Speech Processing – Aalto University Wiki". *Wiki.Aalto.Fi*. https://wiki.aalto.fi/display/ITSP/Cepstrum+and+MFCC.

Bello, J.P., L. Daudet, S. Abdallah, C. Duxbury, M. Davies, and M.B. Sandler. 2005. "A Tutorial On Onset Detection In Music Signals". *IEEE Transactions On Speech And Audio Processing* 13 (5): 1035-1047. doi:10.1109/tsa.2005.851998.

Bogert, B. P., M. J. R. Healy, and J. W. Tukey. 1963. *Proceedings Of The Symposium On Time Series Analysis*. New York: Wiley.

Boşnak, Mehmet, Akif Hakan Kurt, and Selma Yaman. 2017. "Beynimizin Müzik Fizyolojisi". *Kahramanmaraş Sütçü İmam Üniversitesi Tıp Fakültesi Dergisi* 12 (1): 35-44. doi:10.17517/ksutfd.296621.

Briot, Jean-Pierre, Gaëtan Hadjeres, and François Pachet. 2019.” Deep Learning Techniques For Music Generation. Computational Synthesis And Creative Systems”. *Springer International Publishing*.

Briot, Jean-Pierre, Gaëtan Hadjeres, and François-David Pachet. 2019. "Deep Learning Techniques For Music Generation -- A Survey". *Deep Learning Techniques For Music Generation, Computational Synthesis And Creative Systems*. doi:10.48550/arXiv.1709.01620.

Briot, Jean-Pierre. 2020. "From Artificial Neural Networks To Deep Learning For Music Generation: History, Concepts And Trends". *Neural Computing And Applications* 33 (1): 39-65. doi:10.1007/s00521-020-05399-0.

Bushaev, Vitaly. 2018. "Adam—Latest Trends In Deep Learning Optimization.". *Medium*. https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c.

Cho, Kyunghyun, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. "On The Properties Of Neural Machine Translation: Encoder–Decoder Approaches". *Proceedings Of SSST-8, Eighth Workshop On Syntax, Semantics And Structure In Statistical Translation*. doi:10.3115/v1/w14-4012.

Chu, Hang, Raquel Urtasun, and Sanja Fidler. 2016. "Song From PI: A Musically Plausible Network For Pop Music Generation". doi: 10.48550/arXiv.1611.03477

Chung, Junyoung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. "Empirical Evaluation Of Gated Recurrent Neural Networks On Sequence Modeling". *NIPS 2014 Deep Learning And Representation Learning Workshop*. doi:10.48550/arXiv.1412.3555.

Creswell, Antonia, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A. Bharath. 2018. "Generative Adversarial Networks: An Overview". *IEEE Signal Processing Magazine* 35 (1): 53-65. doi:10.1109/msp.2017.2765202.

da Silva, Ivan Nunes, Danilo Hernane Spatti, Rogerio Andrade Flauzino, Luisa Helena Bartocci Liboni, and Silas Franco dos Reis Alves. 2017. *ARTIFICIAL NEURAL NETWORKS*. SPRINGER.

Dahouda, Mwamba Kasongo, and Inwhee Joe. 2021. "A Deep-Learned Embedding Technique For Categorical Features Encoding". *IEEE Access* 9: 114381-114391. doi:10.1109/access.2021.3104357.

Dong, Hao-Wen, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. 2018. "Musegan: Multi-Track Sequential Generative Adversarial Networks For Symbolic Music Generation And Accompaniment". *Proceedings Of The 32nd AAAI Conference On Artificial Intelligence (AAAI)*.

Eck, Douglas, and Jurgen Schmidhuber. 2002. "A First Look At Music Composition Using LSTM Recurrent Neural Networks". *Technical Report No. IDSIA-07-02*.

Eck, Douglas, and Jürgen Schmidhuber. 2002. "Learning The Long-Term Structure Of The Blues". *Artificial Neural Networks — ICANN* 2002, 284-289. doi:10.1007/3-540-46084-5_47.

Education, IBM. 2020. "What Are Neural Networks?". *Ibm.Com*. https://www.ibm.com/cloud/learn/neural-networks.

Ellis, Daniel P. W. 2007. "Beat Tracking By Dynamic Programming". *Journal Of New Music Research* 36 (1): 51-60. doi:10.1080/09298210701653344.

Franklin, Judy A. 2006. "Jazz Melody Generation Using Recurrent Networks and Reinforcement Learning". *International Journal On Artificial Intelligence Tools* 15 (04): 623-650. doi:10.1142/s0218213006002849.

Gallant, Stephen I. 2000. "Context Vectors: A Step Toward A “Grand Unified Representation”". *Lecture Notes In Computer Science*, 204-210. doi:10.1007/10719871_14.

Goodfellow, Ian J., Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. "Generative Adversarial Networks". doi: 10.48550/arXiv.1406.2661

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.

Google Research. 2017. "Google Colaboratory". *Colab.Research.Google.Com*. https://colab.research.google.com/.

Grosche, Peter, and Meinard Muller. 2009. "Computing Predominant Local Periodicity Information In Music Recordings". *2009 IEEE Workshop On Applications Of Signal Processing To Audio And Acoustics*. doi:10.1109/aspaa.2009.5346544.

Grosche, Peter, Meinard Muller, and Frank Kurth. 2010. "Cyclic Tempogram—A Mid-Level Tempo Representation For Musicsignals". *2010 IEEE International Conference On Acoustics, Speech And Signal Processing*. doi:10.1109/icassp.2010.5495219.

Gulli, Antonio, and Sujit Pal. 2017. *Deep Learning With Keras*. Packt Publishing Ltd.

Hadjeres, Gaëtan, François Pachet, and Frank Nielsen. 2017. "Deepbach: A Steerable Model For Bach Chorales Generation". *34th International Conference on Machine Learning.* doi: 10.48550/arXiv.1612.01010 .

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. "Deep Residual Learning For Image Recognition". *2016 IEEE Conference On Computer Vision And Pattern Recognition (CVPR)*. doi:10.1109/cvpr.2016.90.

Herremans, Dorien, and Elaine Chew. 2017. "Morpheus: Generating Structured Music With Constrained Patterns And Tension". *IEEE Transactions On Affective Computing* 10 (4): 510-523. doi:10.1109/taffc.2017.2737984.

Hiller, Lejaren Arthur, and Leonard Maxwell Isaacson. 1959. “Experimental Music: Composititon Witth An Electronic Computer”. McGraw-Hill.

Hochreiter, Sepp, and Jürgen Schmidhuber. 1997. "Long Short-Term Memory". *Neural Computation* 9 (8): 1735-1780. doi:10.1162/neco.1997.9.8.1735.

Jack, Dylan. n.d. "Why Are Drums Important In A Band? (This Is Why) – Drum Sector". *Drumsector.Com*. https://drumsector.com/why-are-drums-important-in-a-band/.

Johnson, Daniel D. 2017. "Generating Polyphonic Music Using Tied Parallel Networks". *Computational Intelligence In Music, Sound, Art And Design*, 128-143. doi:10.1007/978-3-319-55750-2_9.

Karpathy, Andrej, and Li Fei-Fei. 2017. "Deep Visual-Semantic Alignments For Generating Image Descriptions". *IEEE Transactions On Pattern Analysis And Machine Intelligence* 39 (4): 664-676. doi:10.1109/tpami.2016.2598339.

Kavlakoglu, Eda. 2020. "AI Vs. Machine Learning Vs. Deep Learning Vs. Neural Networks: What’S The Difference?". *Ibm.Com*. https://www.ibm.com/cloud/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks.

Kelly, C., M. Gainza, D. Dorran, and E. Coyle. 2010. "Audio Thumbnail Generation Of Irish Traditional Music". *IET Irish Signals And Systems Conference (ISSC 2010)*. doi:10.1049/cp.2010.0504.

Keneshloo, Yaser, Tian Shi, Naren Ramakrishnan, and Chandan K. Reddy. 2019. "Deep Reinforcement Learning For Sequence-To-Sequence Models". *IEEE*

Transactions On Neural Networks And Learning Systems, 1-21. doi:10.1109/tnnls.2019.2929141.

Kingma, Diederik P., and Jimmy Lei Ba. 2015. "Adam: A Method For Stochastic Optimization". *3rd International Conference For Learning Representations*. doi:10.48550/arXiv.1412.6980.

Kingma, Diederik P., and Max Welling. 2013. "Auto-Encoding Variational Bayes". doi:10.48550/arXiv.1312.6114.

Koç, Esra Meltem, Duygu Ayhan Başer, Rabia Kahveci, and Adem Özkara. 2016. "Ruhun Ve Bedenin Gıdası: Geçmişten Günümüze Müzik Ve Tıp". *Konuralp Tıp Dergisi* 8 (1). doi:10.18521/ktd.83286.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. 2017. "Imagenet Classification With Deep Convolutional Neural Networks". *Communications Of The ACM* 60 (6): 84-90. doi:10.1145/3065386.

Lewis. 1988. "Creation By Refinement: A Creativity Paradigm For Gradient Descent Learning Networks". *IEEE International Conference On Neural Networks* 2: 229-233. doi:10.1109/icnn.1988.23933.

Lundh, Fredrik. 1999. "An Introduction To Tkinter."

Mahanta, Jahnavi. 2017. "Introduction To Neural Networks, Advantages And Applications". *Medium*. https://towardsdatascience.com/introduction-to-neural-networks-advantages-and-applications-96851bd1a207.

Mann, Ben. 2019. "How To Sample From Language Models". *Medium*. https://towardsdatascience.com/how-to-sample-from-language-models-682bceb97277.

Mao, Huanru Henry, Taylor Shin, and Garrison Cottrell. 2018. "Deepj: Style-Specific Music Generation". *2018 IEEE 12th International Conference On Semantic Computing (ICSC),* 377-382. doi:10.1109/icsc.2018.00077.

McFee, Brian, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. 2015. "Librosa: Audio And Music Signal Analysis In Python". *Proceedings Of The 14Th Python In Science Conference* 8.

McKinney, Wes. 2010. "Data Structures For Statistical Computing In Python". *Proceedings Of The Python In Science Conference* 445: 56-61. doi:10.25080/majora-92bf1922-00a.

Mehri, Soroush, Kundan Kumar, Ishaan Gulrajani, Rithesh Kumar, Shubham Jain, Jose Sotelo, Aaron Courville, and Yoshua Bengio. 2017. "SampleRNN: An Unconditional End-To-End Neural Audio Generation Model". *ICLR*. doi: 10.48550/arXiv.1612.07837

Mostafa, Bossy, Noha El-Attar, Samy Abd-Elhafeez, and Wael Awad. 2020. "Machine And Deep Learning Approaches In Genome: Review Article". *Alfarama Journal Of Basic &Amp; Applied Sciences*. doi:10.21608/ajbas.2020.34160.1023.

Nayebi, Aran, and Matt Vitelli. 2015. "GRUV: Algorithmic Music Generation Using Recurrent Neural Networks."

Newmarch, Jan. 2017. "Fluidsynth". *Linux Sound Programming,* 351-353. doi:10.1007/978-1-4842-2496-0_20.

Nilsson, Nils J. 1998. *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann Publishers, Inc.

Noll, A. Michael. 1967. "Cepstrum Pitch Determination". *The Journal Of The Acoustical Society Of America* 41 (2): 293-309. doi:10.1121/1.1910339.

Nwankpa, Chigozie Enyinna, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. 2018. "Activation Functions: Comparison Of Trends In Practice And Research For Deep Learning". doi:10.48550/arXiv.1811.03378.

Okewu, Emmanuel, Philip Adewole, and Oladipupo Sennaike. 2019. "Experimental Comparison Of Stochastic Optimizers In Deep Learning". *Computational Science And Its Applications – ICCSA 2019*, 704-715. doi:10.1007/978-3-030-24308-1_55.

Olah, Christopher. 2015. "Understanding LSTM Networks -- Colah's Blog". *Colah.Github.Io*. http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

O'Shaughnessy, Douglas. 2000. *Speech Communications*. New York: Institute of Electrical and Electronics Engineers.

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2001. "BLEU: A Method For Automatic Evaluation Of Machine Translation". *Proceedings Of The 40Th Annual Meeting On Association For Computational Linguistics - ACL '02*. doi:10.3115/1073083.1073135.

Peeters, Geoffroy. 2005. "Time Variable Tempo Detection And Beat Marking". *The International Computer Music Conference*.

Raffel, Colin, and Daniel P.W. Ellis. 2014. "Intuitive Analysis, Creation and Manipulation of MIDI Data with pretty_midi". *15th International Conference On Music Information Retrieval Late Breaking And Demo Papers*.

Ramaseshan, Ajay. 2013. "Application Of Multiway Methods For Dimensionality Reduction To Music", 29. https://www.researchgate.net/publication/259479391_Application_of_Multiway_Methods_for_Dimensionality_Reduction_to_Music.

Roberts, Adam, Jesse Engel, Colin Raffel, Curtis Hawthorne, and Douglas Eck. 2018. "A Hierarchical Latent Vector Model For Learning Long-Term Structure In Music". *ICML*. doi: 10.48550/arXiv.1803.05428

Scheirer, Eric D. 1998. "Tempo And Beat Analysis Of Acoustic Musical Signals". *The Journal Of The Acoustical Society Of America* 103 (1): 588-601. doi:10.1121/1.421129.

Shen, Leixian, Qingyun Zhang, Guoxu Cao, and He Xu. 2018. "Fall Detection System Based On Deep Learning And Image Processing In Cloud Environment". *Advances In Intelligent Systems And Computing*, 594. doi:10.1007/978-3-319-93659-8_53.

Stams, Erik. 2013. "Why Drummers Really Are The Backbone Of The Band". *Drum Expo*. https://www.musicradar.com/news/drums/why-drummers-really-are-the-backbone-of-the-band-580362.

Stewart, Russell. 2016. "Maximum Likelihood Decoding With Rnns - The Good, The Bad, And The Ugly - The Stanford Natural Language Processing Group". *Nlp.Stanford.Edu*. https://nlp.stanford.edu/blog/maximum-likelihood-decoding-with-rnns-the-good-the-bad-and-the-ugly/.

Sutskever, Ilya, Oriol Vinyals, and Quoc V Le. 2014. "Sequence To Sequence Learning With Neural Networks". doi:10.48550/arXiv.1409.3215.

Tanti, Marc, Albert Gatt, and Kenneth P. Camilleri. 2018. "Where To Put The Image In An Image Caption Generator". *Natural Language Engineering* 24 (3): 467-489. doi:10.1017/s1351324918000098.

Tian, Mi, Gyorgy Fazekas, Dawn A. A. Black, and Mark Sandler. 2015. "On The Use Of The Tempogram To Describe Audio Content And Its Application To Music Structural Segmentation". *2015 IEEE International Conference On Acoustics, Speech And Signal Processing (ICASSP)*. doi:10.1109/icassp.2015.7178003.

Todd, Peter M. 1989. "A Connectionist Approach To Algorithmic Composition". *Computer Music Journal* 13 (4): 27-43. doi:10.2307/3679551.

Tran, Kenneth, Xiaodong He, Lei Zhang, and Jian Sun. 2016. "Rich Image Captioning In The Wild". *2016 IEEE Conference On Computer Vision And Pattern Recognition Workshops (CVPRW)*. doi:10.1109/cvprw.2016.61.

Uzair, Muhammad, and Noreen Jamil. 2020. "Effects Of Hidden Layers On The Efficiency Of Neural Networks". *2020 IEEE 23rd International Multitopic Conference (INMIC)*. doi:10.1109/inmic50486.2020.9318195.

van den Oord, A., Kalchbrenner, N., Vinyals, O., Espeholt, L., Graves, A. and Kavukcuoglu, K., 2016. “Conditional image generation with PixelCNN decoders”. *NIPS'16: Proceedings of the 30th International Conference on Neural Information Processing Systems*, pp.4797–4805.

van den Oord, Aaron van den, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. 2016. "Wavenet: A Generative Model For Raw Audio". doi: 10.48550/arXiv.1609.03499

Vinyals, Oriol, Alexander Toshev, Samy Bengio, and Dumitru Erhan. 2015. "Show And Tell: A Neural Image Caption Generator". *2015 IEEE Conference On Computer Vision And Pattern Recognition (CVPR)*. doi:10.1109/cvpr.2015.7298935.

Waite, Elliot. 2016. "Generating Long-Term Structure In Songs And Stories". *Magenta*. https://magenta.tensorflow.org/2016/07/15/lookback-rnn-attention-rnn.

Wu, Yonghui, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, and Maxim Krikun et al. 2016. "Google's Neural Machine Translation System: Bridging The Gap Between Human And Machine Translation". doi: 10.48550/arXiv.1609.08144

Yadav, Omprakash, Darryl Fernandes, Vishal Dube, and Myron D'Souza. 2021. "Apollo: A Classical Piano Composer Using Long Short-Term Memory". *IETE Journal Of Education* 62 (2): 60-70. doi:10.1080/09747338.2021.1966843.

Yang, Li-Chia, Szu-Yu Chou, and Yi-Hsuan Yang. 2017. "Midinet: A Convolutional Generative Adversarial Network For Symbolic-Domain Music Generation". *ISMIR (International Society Of Music Information Retrieval) Conference*. doi:10.48550/arXiv.1703.10847.

Zukowski, Zack, and CJ Carr. 2018. "Generating Black Metal And Math Rock: Beyond Bach, Beethoven, And Beatles". doi:10.48550/arXiv.1811.06639

APPENDIX A

MIDI SOUND SET

Table A.1. General MIDI Level 1 Instrument Patch Map

PC#	Instrument Name	PC#	Instrument Name	PC#	Instrument Name
1.	Acoustic Grand Piano	33.	Acoustic Bass	65.	Soprano Sax
2.	Bright Acoustic Piano	34.	Electric Bass (finger)	66.	Alto Sax
3.	Electric Grand Piano	35.	Electric Bass (pick)	67.	Tenor Sax
4.	Honky-tonk Piano	36.	Fretless Bass	68	Baritone Sax
5.	Electric Piano 1	37.	Slap Bass 1	69.	Oboe
6.	Electric Piano 2	38.	Slap Bass 2	70.	English Horn
7.	Harpsichord	39.	Synth Bass 1	71.	Bassoon
8.	Clavi	40.	Synth Bass 2	72.	Clarinet
9.	Celesta	41.	Violin	73.	Piccolo
10.	Glockenspiel	42.	Viola	74.	Flute
11.	Music Box	43.	Cello	75.	Recorder
12.	Vibraphone	44.	Contrabass	76.	Pan Flute
13.	Marimba	45.	Tremolo Strings	77.	Blown Bottle
14.	Xylophone	46.	Pizzicato Strings	78.	Shakuhachi
15.	Tubular Bells	47.	Orchestral Harp	79.	Whistle
16.	Dulcimer	48.	Timpani	80.	Ocarina
17.	Drawbar Organ	49.	String Ensemble 1	81.	Lead 1 (square)
18.	Percussive Organ	50.	String Ensemble 2	82.	Lead 2 (sawtooth)
19.	Rock Organ	51.	SynthStrings 1	83.	Lead 3 (calliope)
20.	Church Organ	52.	SynthStrings 2	84.	Lead 4 (chiff)
21.	Reed Organ	53.	Choir Aahs	85.	Lead 5 (charang)
22.	Accordion	54.	Voice Oohs	86.	Lead 6 (voice)
23.	Harmonica	55.	Synth Voice	87.	Lead 7 (fifths)
24.	Tango Accordion	56.	Orchestra Hit	88.	Lead 8 (bass + lead)
25.	Acoustic Guitar (nylon)	57.	Trumpet	89.	Pad 1 (new age)
26.	Acoustic Guitar (steel)	58.	Trombone	90.	Pad 2 (warm)
27.	Electric Guitar (jazz)	59.	Tuba	91.	Pad 3 (polysynth)
28.	Electric Guitar (clean)	60.	Muted Trumpet	92.	Pad 4 (choir)
29.	Electric Guitar (muted)	61.	French Horn	93.	Pad 5 (bowed)
30.	Overdriven Guitar	62.	Brass Section	94.	Pad 6 (metallic)
31.	Distortion Guitar	63.	SynthBrass 1	95.	Pad 7 (halo)
32.	Guitar harmonics	64.	SynthBrass 2	96.	Pad 8 (sweep)

(cont. on next page)

Table A.1. (cont.)

PC#	**Instrument Name**	**PC#**	**Instrument Name**	**PC#**	**Instrument Name**
97	FX 1 (rain)	108.	Koto	119.	Synth Drum
98	FX 2 (soundtrack)	109.	Kalimba	120.	Reverse Cymbal
99	FX 3 (crystal)	110.	Bag pipe	121.	Guitar Fret Noise
100	FX 4 (atmosphere)	111.	Fiddle	122.	Breath Noise
101	FX 5 (brightness)	112.	Shanai	123.	Seashore
102	FX 6 (goblins)	113.	Tinkle Bell	124.	Bird Tweet
103.	FX 7 (echoes)	114.	Agogo	125.	Telephone Ring
104.	FX 8 (sci-fi)	115.	Steel Drums	126.	Helicopter
105.	Sitar	116.	Woodblock	127.	Applause
106.	Banjo	117.	Taiko Drum	128.	Gunshot
107.	Shamisen	118.	Melodic Tom		

Table A.2. General MIDI Level 1 Instrument Families

PC#	**Family Name**
1-8	Piano
9-16	Chromatic Percussion
17-24	Organ
25-32	Guitar
33-40	Bass
41-48	Strings
49-56	Ensemble
57-64	Brass
65-72	Reed
73-80	Pipe
81-88	Synth Lead
89-96	Synth Pad
97-104	Synth Effects
105-112	Ethnic
113-120	Percussive
121-128	Sound Effects

The PC# stands for the MIDI Program Change which corresponds to the instrument number. MIDI Channel 10 is reserved for percussion instruments. The corresponding pitches of MIDI note numbers (Key#) are the same for every instrument.

Table A.3. General MIDI Note Chart

Note	**Octave**										
	-1	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
C	0	12	24	36	48	60	72	84	96	108	120
C#	1	13	25	37	49	61	73	85	97	109	121
D	2	14	26	38	50	62	74	86	98	110	122
D#	3	15	27	39	51	63	75	87	99	111	123
E	4	16	28	40	52	64	76	88	100	112	124
F	5	17	29	41	53	65	77	89	101	113	125
F#	6	18	30	42	54	66	78	90	102	114	126
G	7	19	31	43	55	67	79	91	103	115	127
G#	8	20	32	44	56	68	80	92	104	116	
A	9	21	33	45	57	69	81	93	105	117	
A#	10	22	34	46	58	70	82	94	106	118	
B	11	23	35	47	59	71	83	95	107	119	

Table A.4. General MIDI Level 1 Percussion Key Map

Key #	**Drum Sound**	**Key #**	**Drum Sound**	**Key#**	**Drum Sound**
35	Acoustic Bass Drum	52	Chinese Cymbal	69	Cabasa
36	Bass Drum 1	53	Ride Bell	70	Maracas
37	Side Stick	54	Tambourine	71	Short Whistle
38	Acoustic Snare	55	Splash Cymbal	72	Long Whistle
39	Hand Clap	56	Cowbell	73	Short Guiro
40	Electric Snare	57	Crash Cymbal 2	74	Long Guiro
41	Low Floor Tom	58	Vibraslap	75	Claves
42	Closed Hi-Hat	59	Ride Cymbal 2	76	Hi Wood Block
43	High Floor Tom	60	Hi Bongo	77	Low Wood Block
44	Pedal Hi-Hat	61	Low Bongo	78	Mute Cuica
45	Low Tom	62	Mute Hi Conga	79	Open Cuica
46	Open Hi-Hat	63	Open Hi Conga	80	Mute Triangle
47	Low-Mid Tom	64	Low Conga	81	Open Triangle
48	Hi-Mid Tom	65	High Timbale		
49	Crash Cymbal 1	66	Low Timbale		
50	High Tom	67	High Agogo		
51	Ride Cymbal 1	68	Low Agogo		

APPENDIX B

MATHEMATICAL EXPRESSIONS and TRANSFORMS

B.1. Short Time Fourier Transform (STFT)

STFT is a Fourier Transform method used to calculate the frequency and phases of a signal based on its change over time. The output of the STFT is called spectrogram. Spectrogram contains both time and frequency information of a signal, a trade off exists between them. As the width of the window increases, the time resolution increases but frequency resolution decreases and vice versa. The continuous-time and discrete-time STFT equations, respectively, can be given as follows

$$STFT\{x(t)\} = X(\tau,\omega) = \int_{-\infty}^{\infty} x(t)w(t-\tau)e^{-j\omega t}dt \qquad \text{(B.1)}$$

$$STFT\{x[n]\} = X(m,\omega) = \sum_{n=-\infty}^{\infty} x[n]w[n-m]e^{-j\omega n} \qquad \text{(B.2)}$$

where τ and m corresponds to continuous and discrete time axis respectively, ω corresponds to frequency axis, $w(.)$ and $w[.]$ are the continuous-time and discrete-time window function respectively.

B.2. Discrete Fourier Transform (DFT)

DFTs are used to calculate the frequency spectrum of discrete-time signals. For the signal $\{x[n]\} := x[0], x[1], \dots, x[N-1]$ the forward and reverse DFT equations, respectively, can be given as

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}kn} \quad \text{(B.3)}$$

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{j\frac{2\pi}{N}kn} \quad \text{(B.4)}$$

where $\{X[k]\} \coloneqq X[0], X[1], \ldots, X[N-1]$ is a sequence of complex numbers.

B.3. Discrete Cosine Transform (DCT)

DCT represents a finite sequence of points, which is the cosine sum of different oscillating frequencies. DCT is widely used in signal processing applications, especially in image compression, due to the advantage it provides over DFT. The forward and inverse equations DCT, respectively, are given by

$$X_C[k] = \sum_{n=0}^{N-1} x[n] \, cos \, cos \left(\frac{2\pi(2n+1)k}{4N} \right) \; for \; k = 0 \colon N-1 \quad \text{(B.5)}$$

$$x[n] = \frac{1}{N} X[0] + \frac{2}{N} \sum_{k=1}^{N-1} X[k] \, cos \, cos \left(\frac{2\pi(2n+1)k}{4N} \right) \quad \text{(B.6)}$$

B.4. Expected Value

The expected value of a random variable, X, that is governed by a probability density function, $f_x(x)$, is defined by

$$E[X] = \int_{-\infty}^{\infty} x f_x(x) dx \quad \text{(B.7)}$$

B.5. Hadamard Product

Hadamard product, which is known as element-wise product, multiply two matrices, with same dimensions, and results another matrix with same dimension. The Hadamard product is undefined for the multiplication of matrices that has different dimensions. The Hadamard product for two matrices, A and B, is defined as follows

$$A_{MxN} \odot B_{MxN} = (A)_{mn}(B)_{mn} \tag{B.8}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \odot \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} & a_{13}b_{13} \\ a_{21}b_{21} & a_{22}b_{22} & a_{23}b_{23} \\ a_{31}b_{31} & a_{32}b_{32} & a_{33}b_{33} \end{bmatrix} \tag{B.9}$$

where M and N are rows and columns of the matrices, m and n, row and column, denotes the indices respectively.

APPENDIX C

THE VANISHING GRADIENT PROBLEM

In general, deep learning models update their weight vectors to learn the relationship between the input and the target data. This is accomplished by first calculating the error and then updating its weights with gradient descent. The vanishing gradient problem appears during the backpropagation when the gradient of the information starts fading. To understand the problem clearly, let us look into the following example.

Assume that the following deep learning model is created with 1 input layer, k hidden layers and 1 output layer. Moreover, the sigmoid function, σ, is chosen as the activation function.

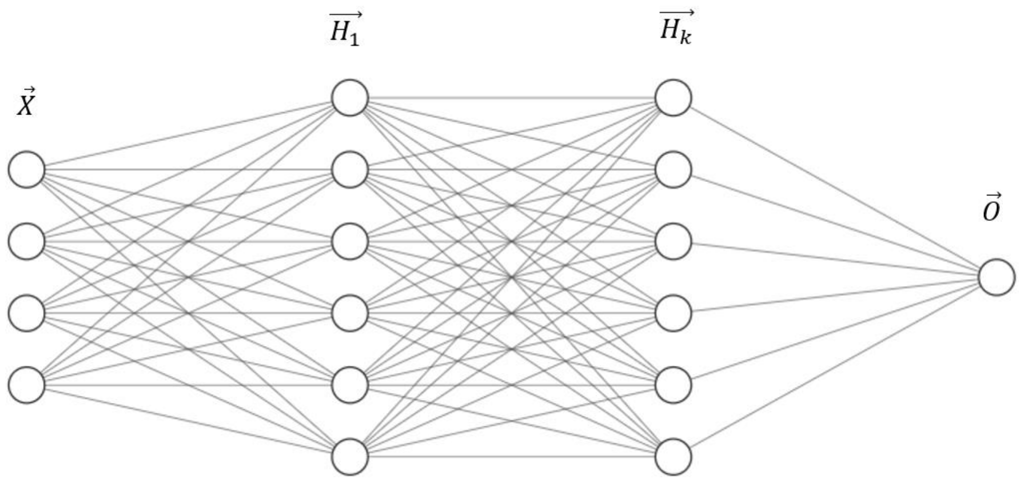


Figure C.1. A Fully Connected Deep Learning Model

During forward propagation the output is calculated as follows,

$$\vec{Z}_{n+1} = \overrightarrow{W_n}\vec{H}_n \tag{C.1}$$

$$\vec{H}_n = \sigma(\vec{Z}_{n+1}) \tag{C.2}$$

$$\vec{Z}_{k+1} = \vec{W}_k \vec{H}_k \tag{C.3}$$

$$\vec{O} = \sigma(\vec{Z}_{k+1}) \tag{C.4}$$

where $n = \{0,1,2,\ldots,k\}$ and $\vec{H}_0 = \vec{X}$, $\sigma(.)$ denotes the element wise sigmoid activation, and $\vec{W}_n$ are the weight matrices for the corresponding layers.

After the forward propagation, the model's weights are updated by calculating the error, E, and gradient. To understand the problem clearly, instead of looking overall weight updates, let us focus on the weight update between the input and first hidden layer. Therefore, first the error then by using the chain rule the calculation for weight update $\vec{W}_0$ can be written as

$$\vec{E} = \left(\vec{O} - \vec{Y}\right)^2 \tag{C.5}$$

$$\frac{\partial \vec{E}}{\partial \vec{W}_0} = \frac{\partial \vec{Z}_1}{\partial \vec{W}_0} \frac{\partial \vec{H}_1}{\partial \vec{Z}_1} \frac{\partial \vec{Z}_2}{\partial \vec{H}_1} \frac{\partial \vec{H}_2}{\partial \vec{Z}_2} \cdots \frac{\partial \vec{H}_k}{\partial \vec{Z}_k} \frac{\partial \vec{Z}_{k+1}}{\partial \vec{H}_k} \frac{\partial \vec{O}}{\partial \vec{Z}_{k+1}} \frac{\partial \vec{E}}{\partial \vec{O}} \tag{C.6}$$

where $\frac{\partial \vec{Z}_{k+1}}{\partial \vec{H}_k}$ terms correspond to the change of $\vec{Z}_{k+1}$ due to $\vec{H}_k$, and $\frac{\partial \vec{H}_k}{\partial \vec{Z}_k}$ terms represent the change of $\vec{H}_k$ due to $\vec{Z}_k$. In order words, by recalling the Eq. (C.1) and (C.2.), these terms represent the changes due to the weights and sigmoid function respectively.

The vanishing gradient problem occurs on the terms $\frac{\partial \vec{H}_k}{\partial \vec{Z}_k}$. In order to see this clearly, first the sigmoid function and its derivative are provided as follows

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{C.7}$$

$$\begin{aligned} \frac{d\sigma(x)}{dx} &= \frac{e^{-x}}{(e^{-x} + 1)^2} \\ &= \sigma(x)(1 - \sigma(x)) \end{aligned} \tag{C.8}$$

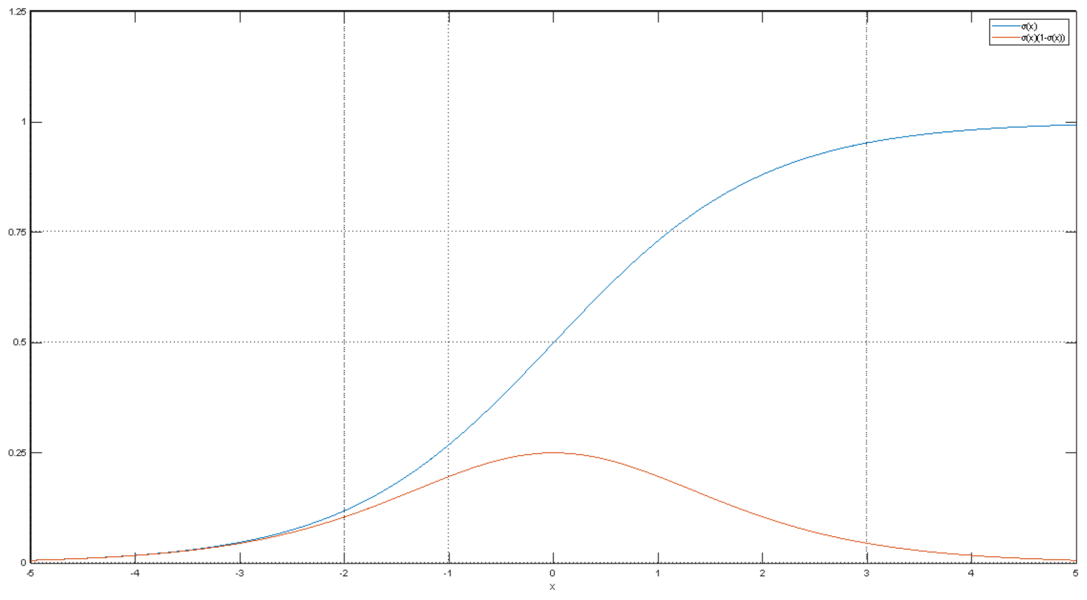


Figure C.2. Graph of Sigmoid and its Derivative

Furthermore, from the figure given above, notice that the derivative of the sigmoid can have maximum value of 0.25. Therefore, by using this knowledge and the identity provided in Eq. (C.8), the inequality for $\frac{\partial \vec{H}_k}{\partial \vec{Z}_k}$ terms inside Eq. (C.6) can be written as

$$\frac{\partial \vec{H}_k}{\partial \vec{Z}_k} = \vec{H}_k \left(1 - \vec{H}_k\right) \leq 0.25 \tag{C.9}$$

Realizing that the number of these terms are proportional to number of hidden layers. Although every term ends up with the maximum value of 0.25, the multiplication of these terms become closer to 0. This issue is known as the vanishing gradient problem. Hence, the model won't be able to update its weights and unable to learn/improve.

There are different types of solutions for this problem. The problem can be resolved by using different activation functions (e.g., ReLU) than sigmoid, and different neural network architectures such as LSTM.