MUSIC GENERATION USING LSTM

1st Venkat Sairam Veeranki
Data Science
Michigan Technological University
Houghton, MI 49931 USA
veerank@mtu.edu

2nd Rahul Teja Bolloju
Data Science
Michigan Technological University
Houghton, MI 49931 USA
bolloju@mtu.edu

3rd Sri Rayaleswar Mondrety
Data Science
Michigan Technological University
Houghton, MI 49931 USA
smondret@mtu.edu

***Abstract*—This project focuses on analyzing MIDI files of piano music in order to gain insights into the patterns, trends, and distinctive properties of piano music. The data was obtained from the Classical Piano Midi Page, and the dataset examined was Ludwig van Beethoven's compositions. The analysis focuses on data preparation, analysis, and building an LSTM deep neural network model using two different optimizers, Adamax and RMSprop. Finally, the models were evaluated based on categorical cross-entropy loss.**

I. Introduction

For our analysis, we have chosen to focus only on the piano files. This decision was made because the piano is a common instrument in many different genres of music, and its distinct sound allows for easier recognition and analysis of patterns and musical elements. By limiting our analysis to only piano files, we hope to gain a deeper understanding of this instrument's unique characteristics and musical features. While excluding other instruments may lead to important information being missed, this approach provides a more manageable data set that allows for more precise analysis and interpretation of the results.

In our project, we are analyzing multiple MIDI files to gain valuable insights into piano music. By analyzing patterns and trends, such as tempo, rhythm, and melody, we can uncover hidden features and nuances in piano music that may not be immediately obvious to the listener. This information can be useful for music education, composition, and even for developing AI-powered tools that can generate music based on these findings. Our process of extracting musical elements and features has proven to be an effective way to analyze and understand the intricate workings of piano music from MIDI files.

II. Related Work

In the paper[1](Yang et al., 2017) generator generates melodies in the symbolic domain one bar at a time using a CNN. It accepts a random noise vector as input as well as a conditioning signal, which can be a chord sequence or a priming melody from earlier bars. The generator then generates a series of MIDI notes, which may subsequently be converted into an audio waveform for listening.

A training dataset is used to teach the discriminator to differentiate between created and genuine melodies. It also accepts a conditioning signal as input to learn the melodies' distribution. The discriminator feeds back to the generator, urging it to develop more realistic melodies that correspond to the intended distribution.

The research also offers a unique conditional mechanism known as "skip connections," which allows the generator to include conditioning signals selectively at different stages of the CNN. This allows the model to learn from many sorts of conditioning inputs and produce more diversified and intriguing tunes.

Overall, MidiNet is a promising method for creating melodies with CNNs and GANs. It offers a versatile and adaptable framework for music generation that may combine many sorts of past information and produce music with numerous tracks.

In paper[2], DeepBach is a generative model aimed to learn polyphonic music patterns by examining Johann Sebastian Bach's compositions. The model employs a graphical model framework and a modified representation of music to simulate several musical voices at the same time.

The model may produce new music by sampling from a probability distribution over potential musical sequences using pseudo-Gibbs sampling. This methodology differs from many other automatic music creation approaches, which compose music one note at a time.

DeepBach's capacity to be guided by users is one of its distinguishing qualities. This means that by setting limits on the model, users may direct the development of new music. Users might, for example, define the notes, rhythms, or cadences they wish the model to include in the resulting music.

The authors also include a MuseScore plugin that allows users to interact with DeepBach and compose new music from within the editor. This allows users to easily experiment with different parameters and limits and hear the outcomes.

Overall, DeepBach is a unique method to autonomous music creation capable of producing believable Bach-style music. Its user-controllability and interaction with the MuseScore music editor make it a powerful tool for both music composers and music information retrieval researchers.

The paper[3], explained the recurrent neural network (RNN) architecture. LSTMs, as opposed to typical RNNs, have a more complex memory system that allows them to recall relevant events and patterns across longer periods of input data.

Traditional RNNs can learn note-by-note transition probabilities in the context of music creation, but they fail to learn the general structure of a musical composition. This is due to their inability to keep track of temporal dependencies that arise across longer time spans. LSTMs, on the other hand, are designed to better capture these temporal connections and have demonstrated effectiveness in related areas such as timing and counting, as well as context-sensitive learning.

The study referred to the use of LSTM networks to learn the structure of blues music and generate new tunes in that manner. The results demonstrated that the LSTM was capable of learning the style of blues music and producing original melodies that were pleasurable to listen to while adhering to the genre's correct structure and time. This is because the LSTM can capture the long-term temporal dependencies that are fundamental to blues music.

III. Data

Data was taken from the Classical Piano Midi Page which is a comprehensive resource for piano technique that provides valuable information on the physical aspects of playing the instrument. The website has a big library of free MIDI files for a wide range of classical music works, allowing users to listen to and play along with the music.

For our dataset we are considering Ludwig van Beethoven' compositions who was a German composer and pianist. He is widely regarded as one of the greatest composers in Western classical music history, known for his innovative use of musical form and his emotional intensity. In our dataset we have 29 MIDI files containing musical information, this dataset provides an extensive collection of Beethoven's compositions for analysis. Our method for extracting the piano's nodes and codes from each MIDI file allowed us to gain a more in-depth understanding of the intricate details of each piece. There were a total of 81,312 chords from all the MIDI files and 349 unique chords demonstrate the complexity and richness of Beethoven's musical language and the potential for further exploration and analysis of his works using this dataset.

Overall, the Classical Piano Midi Page gave us a wealth of material for our research, allowing us to delve deeper into the complexities of Beethoven's piano pieces. We were able to get vital insights into the physical and emotional qualities of his work, as well as his contributions to the larger classical music tradition, by exploiting this resource.

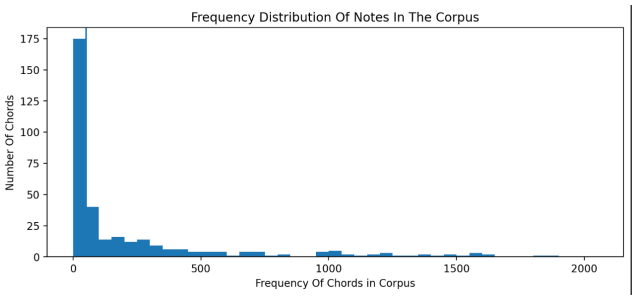
IV. Data Preprocessing

A piano's pitch and chords are essential components of its sound. Pitch refers to the frequency of a sound wave, which determines whether a note sounds high or low. Each key on a piano corresponds to a different pitch, and when a key is struck, a hammer within the piano strikes a string, generating the sound of that pitch.

Chords, on the other hand, are created by playing two or more notes simultaneously. They can be harmonic or discordant depending on the combination of notes. The keyboard layout of a piano makes it easy to play a wide range of notes together, making it ideal for playing chords. Understanding the pitch and chords of a piano is crucial for building our analysis model.

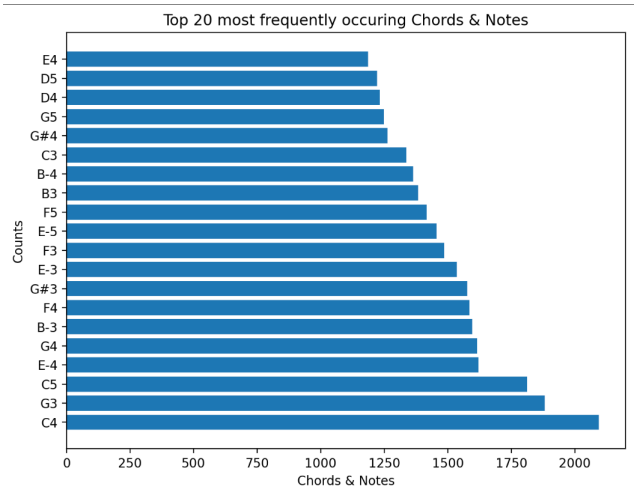
Our analysis of the Corpus revealed some interesting statistics regarding the frequency of notes. On average, a note appeared 232.99 times in the Corpus. However, there were some notable outliers in the data. The most frequent note in the Corpus was 'C4', which appeared 2094 times, making it significantly more common than other notes. Conversely, the least frequent chord in the Corpus was '11.3.5', which only appeared once. These findings may have implications for building our model's input parameters, as the frequency of certain notes could potentially influence the emotional impact of a piece of music. Further research is needed to fully understand the significance of these results.

In our analysis of the corpus, we found that the distribution of chords is skewed towards the right, indicating that some chords are more commonly used than others, as shown in Figure 1. This observation suggests that Beethoven may have had a tendency to favor certain chords over others in his compositions. Additionally, we noticed a significant increase in the frequency of chords after the vertical line at "x=50" in Figure 1, implying that there may be specific patterns or trends in chord usage in the corpus that could be further explored and studied. Furthermore, the plot highlights the most frequent chords used in the corpus, providing valuable insights for research and building models for music composition. This information could potentially inform the creation of new music and help to build better models for prediction.



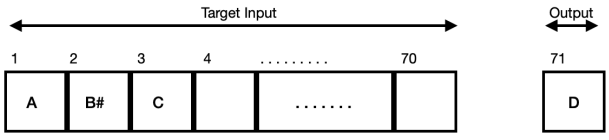
In our data analysis process, we first obtained the unique chords in our data corpus and then created a dictionary whose keys represent the chords and the corresponding values represent the frequency count of each chord in the entire corpus. We constructed a horizontal bar graph to display the top 20 chords and their corresponding frequencies. Upon

examining the graph, we discovered that the top 20 chords had frequencies ranging from 1200 to 2094. Notably, the most common chords found in Beethoven's compositions were C4, C5, and G3, in descending order.

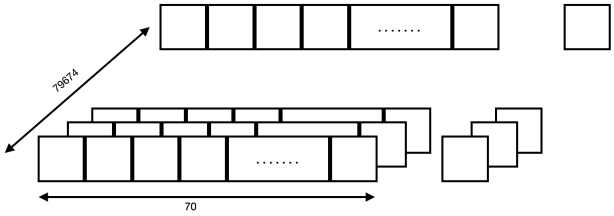


We analyzed the frequencies of each chord in order to obtain a better knowledge of the corpus. According to our findings, there were more than 175 unique chords that had a frequency count of less than 50. For our analysis, we considered words that had a frequency count greater than 50. Our final data corpus had a total of 79744 chords, which are frequent i.e., more than 50 as its frequency of occurrence. These discoveries provide insight into the underlying harmonic structure and patterns present in the corpus. We acquired a better understanding of artistic trends. This allowed us to eliminate the noise in our data and helped us in building a better model.

Our final target was to create a vector sequence of chords that can act as input to our model, with the target output being the next chord that should be played after the vector sequence of chords in Beethoven's style. To create this vector sequence of chords, we divided the data corpus into bins of size seventy. For example, we considered the first seventy chords in our data corpus, which contained various chords. The seventy-first chord in the sequence would be the target output for the vector sequence. A sample of what the vector looks like can be seen in Figure[3]. This way, we were able to create data to train our model. We obtained a data corpus of size (79674 X 70 X 1) shown in the figure[] and target outputs for those sequences of variables (79674).



The final data set has the dimension mentioned in the below diagram.



V. Model

In a deep learning architecture, the usage of LSTM (Long Short-Term Memory) as the sequential model. The LSTM is a form of Recurrent Neural Network (RNN) used to overcome the problem of disappearing gradients. Vanishing gradients is a deep learning network problem that occurs when the gradient of the error function gets too tiny during backpropagation, resulting in very sluggish or static learning. This challenge is addressed by LSTM, which employs memory cells with input, output, and forget gates that selectively keep or delete information over time.

LSTM gates are acquired during training, allowing the network to modify the flow of information as needed. The key benefit of using LSTM in a model is its ability to analyze lengthy data sequences such as natural language text, audio data, or any data with temporal dependencies. LSTM has also demonstrated cutting-edge performance in Machine learning including speech recognition, language transaction, and picture capturing.

To enable the model to store and recall information over long periods of time, LSTM employs memory cells as well as three gates (input, output, and forget). Because LSTM can capture and learn long-term relationships in the input sequence, it is particularly valuable in sequence prediction problems. It may also handle whole datasets rather than individual data points, such as photographs. Overall, LSTM has been a common choice for deep learning models including sequential data, and has demonstrated remarkable performance in a variety of machine learning applications.

During the model building process, we developed two models using LSTM. For the first model Figure[1], we initialized a Sequential model with a regularization strength of 0.1 to mitigate overfitting. Then, we created two LSTM layers, two Dropout layers, and two Dense layers. The first LSTM layer contained 256 units and was configured to accept input with a shape of (70 X 1). This layer was set to return sequences, which correspond to each input time step's hidden states. The subsequent LSTM layer contained 128 units and did not return sequences. The two Dense layers contained 128 and 1 units, respectively. The final layer utilized the softmax activation function, making it ideal for multiclass classification problems. We applied L2 regularization to each LSTM and Dense layer within the layers created. Finally, we compiled the model utilizing the Adamax optimizer, with a learning rate of 0.01.

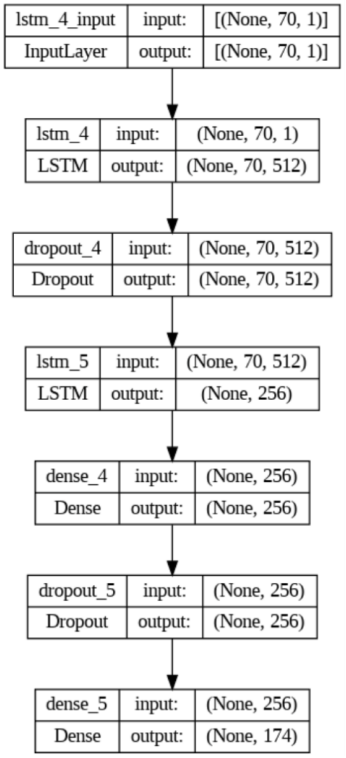
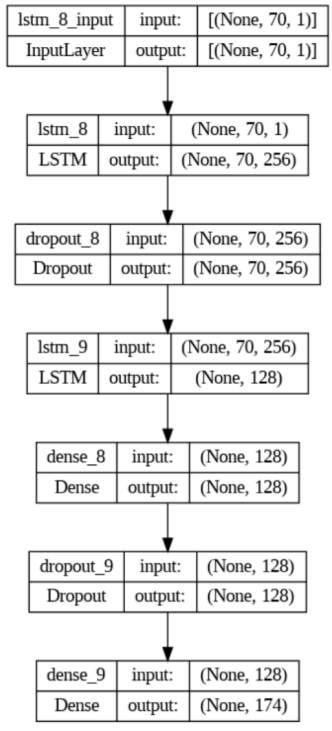


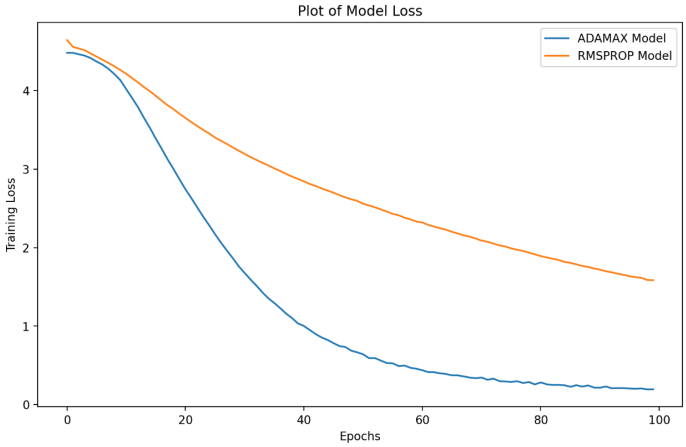
Fig. 1.

For our second model Figure[2], we utilized a comparable underlying architecture but implemented the RMSprop optimizer with a random learning rate of 0.001 and a decay rate of 0.9.



To minimize the loss and stop the training of the neural network, we used the Early Stopping function with patience of 4. This function allowed us to stop training the model when the loss didn't have any significant decrease after 4 iterations.

During the training process, we used both the Adamax and RMSprop optimizers to optimize the neural network's performance. After analyzing the results, we observed that the Adamax loss decreased with each epoch, indicating that the model was improving its ability to make accurate predictions on the training data. However, we noticed that the RMSprop loss was not decreasing as effectively, which suggests that the model was not learning as well with this optimizer. This highlights the importance of selecting an appropriate optimizer for a given task and dataset, as different optimizers may have varying effects on the model's performance.



Once the model is trained using the training data, we use a function to generate a melody in MIDI format using the trained LSTM model. First, random test sequence data is selected from the test sample. The function then generates a new note using the prediction method of the trained model, which predicts the next note in the sequence based on the current training sequence data. To add some variety to the generated melody, the function adjusts the predicted note using a logarithm and exponentiation technique before selecting the index of the maximum value to determine the next note. Each predicted note is then appended to the previously created notes and used to update the test sequence data for the next iteration. Once all the notes have been generated, the Notes_Generated list is converted into a list of chords and notes using a unique set of frequently occurring chords to create a MIDI format. Finally, the resulting melody in MIDI format is returned as the output.

VI. Results

Our final output consists of two files generated from the individual models that we trained. Each model was run for 100 epochs. The RSMprop model took less time to build, whereas the ADamax model took roughly twice as long. We have loaded both sets of generated sequences into a MIDI file format. Since the losses for the Adamax model were lower compared to RSMprop, and we manually checked the overall

output of each model, we found the music generated from RSMprop to have sharp tones and pitches that were jarring to hear. On the other hand, the output from the Adamax model was soothing to listen to.

REFERENCES

[1] Yang, L.-C., Chou, S.-Y., Yang, Y.-H. (2017). MidiNet: A Convolutional Generative Adversarial Network for Symbolic-domain Music Generation. ArXiv:1703.10847 [Cs]. https://arxiv.org/abs/1703.10847

[2] Hadjeres, G., Pachet, F., Nielsen, F. (2017). DeepBach: a Steerable Model for Bach Chorales Generation. ArXiv:1612.01010 [Cs]. https://arxiv.org/abs/1612.01010

[3] Eck, D., Schmidhuber, J. (n.d.). A First Look at Music Composition using LSTM Recurrent Neural Networks A First Look at Music Composition using LSTM Recurrent Neural Networks. Retrieved December 15, 2021, from https://people.idsia.ch/ juergen/blues/IDSIA-07-02.pdf

[4] Feng, L., Liu, S., Yao, J. (2017). Music Genre Classification with Paralleling Recurrent Convolutional Neural Network. https://arxiv.org/pdf/1712.08370.pdf