# Rhythms of the Machine: Deep Learning in Music Creation

**Abhishek Paul (ap62752)**
abhishek.paul@utexas.edu

## Abstract

This paper explores the application of machine learning, particularly Recurrent Neural Networks (RNNs), to music generation. The motivation of this project is the goal of creating a tool to assist musicians in prototyping and extending compositions. The project focuses on learning the temporal dependencies and structural patterns inherent in musical sequences.

In the first phase of the project, I aimed to demonstrate whether an RNN could capture temporal relationships between notes and generate realistic extensions of monophonic melodies (maximum of one note at a given time). Using a two-layer LSTM architecture, the model successfully learned musical patterns and produced fluid, coherent extensions.

In the second phase, I expanded the scope to polyphonic music generation (multiple notes can play at a given time). I modified the data representation and expanded the architecture to include additional LSTM layers. However, the polyphonic model struggled to generate musically meaningful compositions, producing chaotic and unstructured outputs.

While the monophonic model was a success, generating fluid extensions and even entire compositions from scratch, further refinements are needed to improve the model's ability to handle polyphony and generate more expressive compositions.

## 1 Introduction

The increase in popularity of streaming services has given both well-known and aspiring musicians an avenue to distribute their music to millions of people quickly. The problem no longer lies in getting songs out there but in the generation of music. As an amateur musician, I have long been interested in utilizing machine learning in music generation. Recent advancements in Natural Language Processing (NLP) have shown how effectively models can learn structured patterns and contextual dependencies within sequential data, such as language. Music, like language, follows its own structural rules and patterns, which can be likened to its own ”grammar.”(Patel and Hagoort, 2001), (Le et al., 2024) This project explores whether NLP-inspired models such as RNNs can be adapted for music generation (Lee et al., 2017), focusing primarily on generating extensions to musical inputs.

The motivation for this project lies in creating a tool musicians can use to prototype their ideas quickly. For instance, a user could input a few seconds of a musical fragment—perhaps the beginning of a new song they are trying to write—and the model would generate a continuation, providing inspiration or a basis for further composition. This is a tool that I know that I can personally benefit from. By leveraging Long Short-Term Memory (LSTM) models, these outputs could be iteratively fed back to the model to produce complete compositions (Mangal et al., 2019).

## 2 Literature Review

Attempts at algorithmic music generation have been ongoing since at least the eighteenth century. Early examples include Mozart's Musikalisches Würfelspiel (musical dice game) (Maurer, 1999), which used predetermined rules and dice rolls to create new compositions.

The advent of computers allowed for more statistical and rule-based systems such as Markov chains (Shapiro and Huber, 2021). Seminal early computational models like David Cope's Experiments in Musical Intelligence (EMI) codified compositional styles into structured rules which could be used to generate music algorithmically (Cope, 2001), (Zhang et al., 2018). While these attempts were groundbreaking, they were inflexible and limited by predetermined rules.

The most recent advancements in using machine learning in music generation happened relatively recently in the mid-2010s with the introduction of RNNs and LSTM networks. Their ability to understand temporal dependencies enabled the generation of more coherent and expressive sequences. Many early significant advancements in this domain involve ongoing projects like Google Magenta, which has developed models such as Melody RNN (Waite, 2016) and Music Vae (Briot, 2020) these leveraged LSTMs for monophonic and polyphonic music generation.

More recent approaches have explored Variational Autoencoders (VAEs) (Roberts et al., 2018) and Generative Adversarial Networks (GANs) (Engel et al., 2019). VAEs, create novel compositions by learning compressed representations of music. GAN models like MuseGAN, can generate multi-track music and produce harmonically cohesive outputs. Transformer-based architectures, including the Music Transformer (Huang et al., 2018) and OpenAI's MuseNet (Brown et al., 2019), utilize self-attention mechanisms that capture long-term dependencies, enabling rich and coherent compositions.

## 3 Data

### 3.1 Data Representation

The choice of data representation was one of the first decisions that I needed to make for training music generation models. Two primary approaches were considered (Boulanger-Lewandowski et al., 2012), (Huang et al., 2020):

1. **Symbolic**:
   - **Musical Instrument Digital Interface (MIDI):** Encodes music as sequences of events, such as Note-On/Note-Off, with pitch, velocity, and timing information. For Note-On events, it encodes notes of the range 0-127.
   - **Piano roll:** A 2D matrix where rows represent pitches and columns represent time steps.
2. **Audio:**
   - **Wav:** Represents music as a continuous waveform.

From these choices, I decided to use the MIDI format.

## 4 MIDI

MIDI is a technical protocol for digital instruments that provides a symbolic representation of the music. It is transmitted at a high temporal sampling rate and provides information on Note On and Note Off events, the loudness of each note, and its velocity. MIDI can be visualized in the Piano roll format, as we can see in Figure 2 (Boulanger-Lewandowski et al., 2012) (Huang et al., 2020).

A midi file encodes 128 notes (0 is the lowest pitch, 127 is the highest pitch), a velocity that ranges from 0 to 31. The minimum time step is 8 ms, and for each time step, a series of instructions indicate which notes are on/off and their velocity

MIDI is ideal for sequential models such as RNNs, which was the model I wanted to investigate. There is also a significant corpus of large labeled MIDI datasets, such as the Lakh MIDI and Masetro, which could be leveraged to get training data.

### 4.1 Dataset:

For this project, I used the Maestro MIDI dataset. The Masetro dataset is a collection of 1276 performances from the International Piano-e-Competition. It contains about 200 hours of high-level MIDI piano recordings (Hawthorne et al., 2018).

While this is not the largest midi dataset, there are a few reasons I opted for this dataset:

1. **All of the data are that of piano music.** Different instruments have their unique characteristics and composition styles. This will ensure consistency in all the training and validation data.

2. **All the performances are verified to be done by humans.** There is no electronic or synthesized music. This is useful since I am training the model to mimic human play.

3. **All the music in the dataset is played by experts (participants in the competition).** A casual listener's idea of a human performance will be music played by experts.

## 5 Monophonic Extension Generation:

For the first phase of this project, I focused on generating monophonic music extensions, where

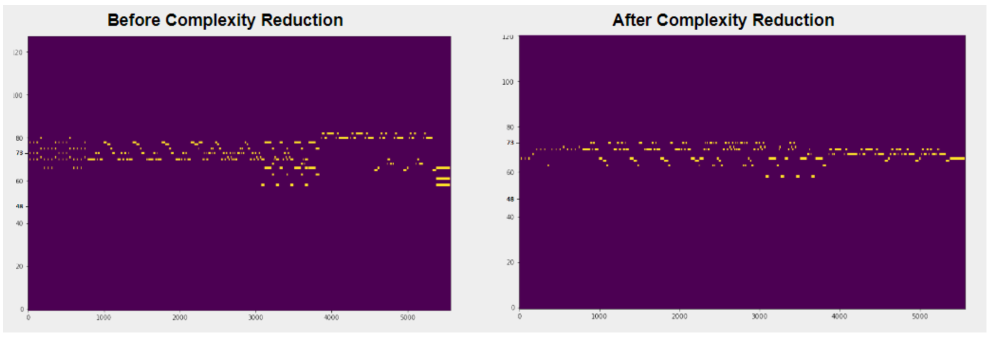

Figure 1: Shows a before and after comparison of the complexity reduction steps

| Split | Perfor mances | Duration (Hours) | Notes (Millions) |
|---|---|---|---|
| Train | 962 | 159.2 | 5.66 |
| Validation | 137 | 19.4 | 0.64 |
| Test | 177 | 20.0 | 0.74 |
| **Total** | **1276** | **198.7** | **7.04** |

Table 1: Dataset statistics for Mastro

at any given time step, either a single note will play, or there will be no note (rest).

By focusing on monophonic music, I can abstract away certain complexities, such as velocity (dynamics), and focus solely on the temporal progression of musical notes (Sajad et al., 2021).

This will lay a strong foundation for understanding core musical patterns and transitions. Further phases can incorporate more sophisticated aspects of music, such as polyphony and dynamics. This stepwise progression ensures that the model's complexity grows with its capabilities.

### 5.1 Data Transformation:

For the monophonic music generation, much of the information in the MIDI can be abstracted away. Velocity data is discarded, and the midi is converted into a 2-dimensional matrix, piano roll like the one seen in Figure 2 (Briot and Pachet, 2017).

Further simplification was needed as I needed to ensure that at each time step, there was at most one note (or no notes) playing. I needed to determine which notes to keep and which to discard. Two methods were explored:

1. Randomly selecting one note at each time step.

2. Retaining the highest note at each time step.

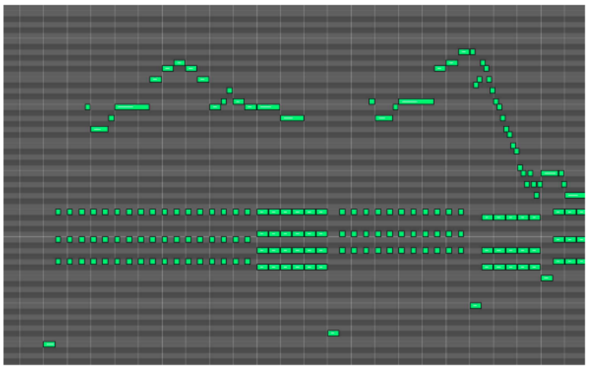

Figure 2: Piano roll visualization

I investigated both methods and compared their outputs. I saw that as long as the highest note was retained, the overall melody of the composition was preserved (Lemström and Tarhio, 2000).

We can hear the before and after transformation of a music input below:

Before Transformation
After Transformation

The example above shows that although the polyphonic to monophonic conversion drops the accompaniment, the overall melody is preserved.

The transformed data was further processed into TensorFlow's TFRecord format to optimize training efficiency.

TFRecords is a TensorFlow-specific binary file format designed to store large datasets efficiently. It organizes data as a sequence of serialized protocol buffer (protobuf) records. It speeds up the reading and processing of the data during training (Team, 2020).

### 5.2 RNN Model Overview

This project focused on the Recurrent Neural Network (RNN) model. RNNs have been used extensively for text completion problems in NLP as they capture dependencies between the past and the future. Generating music extensions shares many of these properties, so an RNN model seems the ideal choice.

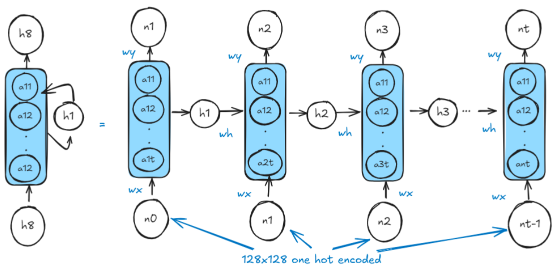

Figure 3: High-level overview of RNN structure for monophonic case

If we zoom into one of the layers in Figure 3, we can see that at each time step, the input to the model is a one-hot encoded vector representing the note being played. The label is the subsequent note in the sequence. For instance, if the sequence of notes is [n0, n1, n2, n3...], the model is trained using pairs such as (input = n0, label = n1) and (input = n1, label = n2). The hidden state in the RNN propagates information across timesteps, enabling it to learn temporal relationships between notes.

The configuration of the RNN blocks in the image is a 2-layer LSTM with 128 cells each. The input and output of each block is a 128x128 one-hot encoded matrix indicating the input and predicted note.

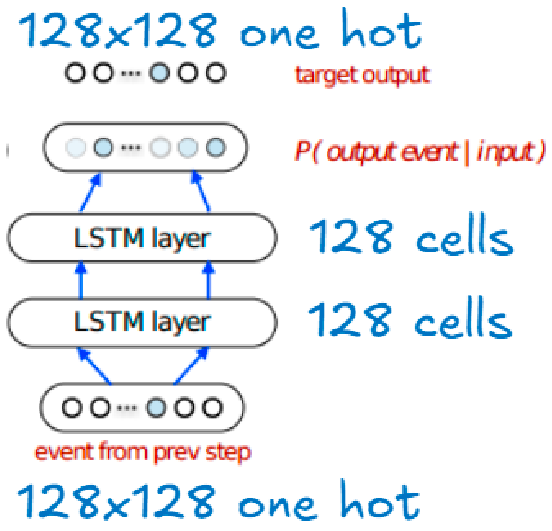

Figure 4: Detailed visualization of an RNN layer in the monophonic case

The overall goal of the model is to predict the following note given a sequence of previous notes. This is achieved by learning a probability distribution over possible following notes, conditioned on the notes that came before. This framing allows music generation to be treated as a supervised learning problem, where the model iteratively predicts the next note in the sequence.

### 5.3 Training

To evaluate the model during training, two metrics are used (Yang and Lerch, 2018):

1. **Accuracy:** The percentage of notes in a sequence correctly predicted by the model.

$$Accuracy = \frac{NumberofCorrectPredictions}{TotalNumberofPredictions}$$

2. **Perplexity:** A measure of how well the model predicts an entire sequence, with lower values indicating better predictions.

$$Perplexity = \exp\left(-\frac{1}{N}\sum_{i=1}^{N}\log P(y_i|x_i)\right)$$

Where:

- P($y_i|x_i$) is the probability assigned to the correct output $y_i$ given the input $x_i$
- N is the length of the sequence

After running the training for about 2000 steps, I got an accuracy of about 64.27% and a perplexity of 4.29.

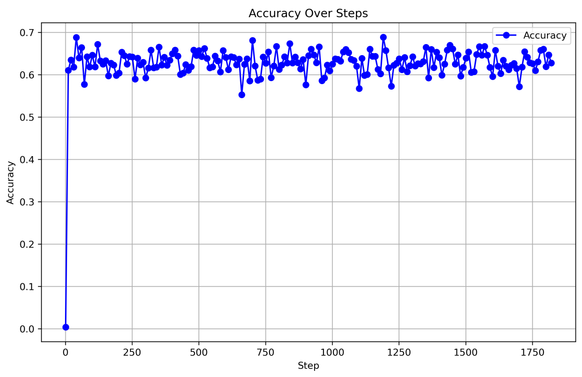

Figure 5: Change in accuracy on the training data over 2000 steps for the monophonic case

As seen in Figures 5 and 6, accuracy does not change significantly after the first 200 epochs, but perplexity continues to decline steadily.

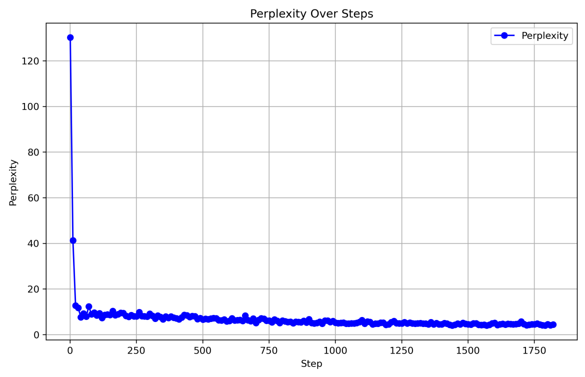

Figure 6: Change in perplexity on the training data over 2000 steps for the monophonic case

## 5.4 Evaluation of Music Generation

To create music extensions, the model used during training required some modifications. While the training process involved predicting the next note in a sequence based on past notes, the generation phase extends this idea by iteratively predicting and appending notes to an input sequence.

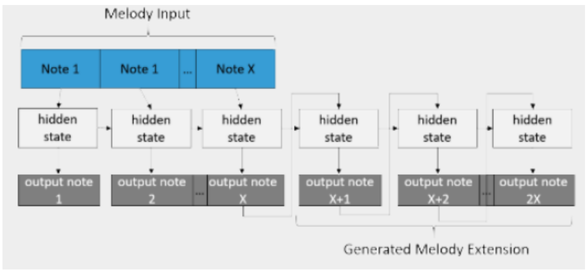

Figure 7: Extension generator process

As Figure 7 shows, the generation script performs a forward pass of the input music through the model. Using the hidden states and the last predicted note, the model continues the melody by predicting the following note in the sequence. This process is iteratively repeated, where each newly generated note and the updated hidden state become the input for the subsequent prediction, allowing the model to build an extension of the input composition.

This method enables the model to handle variable-length input sequences and generate coherent extensions while maintaining the melodic and temporal structure of the original piece.

Below are some example outputs of the model for music extension:

## 5.5 Generating short Extensions

**Example 1: Generating short extensions (15 seconds)**
This is a 5-second input that was fed into the model:
**Input Melody 1:** Input 1▶
The following are three sample extensions:
**Extension 1:** Extension 1▶
**Extension 2:** Extension 2▶
**Extension 3:** Extension 3▶

**Example 2: Generating short extensions (15 seconds)**
This is a 5-second input:
**Input Melody 1:** Input 2▶
The following are three sample extensions:
**Extension 1:** Extension 1▶
**Extension 2:** Extension 2▶
**Extension 3:** Extension 3▶

In the two examples above, I generate a 10-second extension to the input melody (15 seconds total). Upon subjective evaluation, the generated extensions are able to maintain the melodic and rhythmic characteristics of the input sequence. The transition between the original melody and the generated portion are also fluid. This suggests the model has captured some of the temporal and harmonic dependencies in the input data.

## 5.6 Generating Long Extensions

**Example 3: Generating long extensions (2 minutes)**
This is a 5-second input:
**Input Melody 1:** Input 1▶
The following are three sample extensions:
**Extension 1:** Extension 1▶
**Extension 2:** Extension 2▶
**Extension 3:** Extension3▶

In the examples above, the model generates a 2-minute extension to the input melody (totaling 2 minutes and 5 seconds), which aligns with the average length of many real-world musical compositions.

After the initial 10-15 seconds, most of the context the model utilizes comes from its generated notes. As the music progresses, less and less of the context provided by the input melody is present. This leads to a series of distinct stanzas, each rhythmically coherent and musically sound. However, this also means a lack of a consistent, overarching melody across the entire 2-minute piece.

### 5.7 Generating Fresh Composition

**Example 4: Generating long extensions (2 minutes) without input prompt (the input is a nonevent [-2])**

The following are a few sample 2-minute compositions the model created:
**Composition 1:** Composition 1▶
**Composition 2:** Composition 2▶
**Composition 3:** Composition 3▶

The final test for the model was to evaluate its ability to generate complete musical compositions from scratch. To achieve this, I initialized the model with a "non-event" ([-2]) input, representing that no notes were being played. The model had no prior musical context and had to rely solely on its learned weights and predictions to construct the composition.

The results were impressive. For each of the three generated outputs, the model produced distinct melodies. Despite no input seed, the outputs were fluid and mainly rhythmically consistent. This showed that the model demonstrated a clear understanding of 'musical grammar,' which was the primary goal of this model.

## 6 Polyphonic Extension:

My evaluation of the monophonic RNN demonstrated its ability to capture patterns within the music. This, however, was achieved by abstracting away velocity and dynamics, focusing only on monophonic music.

I will now eliminate the abstractions of monophony and extend our approach to tackling polyphonic music generation.

The architecture and data transformation strategies for this phase of the project were heavily inspired by the Performance RNN paper (Oore et al., 2020) (Simon and Oore, 2017), which provides valuable insights into modelling the nuances of polyphonic compositions.

### 6.1 Evaluating Model on polyphonic data:

Before transitioning to the polyphonic RNN, I evaluated the performance of the existing monophonic RNN on polyphonic data.

For this evaluation, I modified the training data by removing the transformation step that retained only the highest-pitched note in cases of multiple notes occurring at a single timestep. Instead, all notes were kept.

The input format was still a 128x128 one hot encoded matrix at each timestep. However, unlike the monophonic setup, this format now allowed for multiple simultaneous notes, capturing the complexity of polyphonic compositions.

After training the model for approximately 2,000 global steps, I tested it using the same input melody to evaluate its performance:

**Example 5: Evaluating polyphonic data**

The following are a few sample 2-minute compositions the model created:
**Composition 1:** Composition 1▶
**Composition 2:** Composition 2▶
**Composition 3:** Composition 3▶

From the three outputs above, it is evident that the monophonic RNN performs poorly on polyphonic music. The generated extensions have an overwhelming number of notes playing simultaneously, resulting in a chaotic sound with no discernible melody or structure.

One major limitation of the monophonic model is that it does not consider velocity and dynamics, which is essential for capturing the expressiveness of polyphonic compositions.

To design and build an effective polyphonic model, I need to fully utilize the extensive set of instructions provided by the MIDI protocol (Goel et al., 2014):

1. Note-On and Note-Off Events: When a note starts and ends. This allows for note timing and duration (0 to 127)

2. Velocity: The intensity of the note. This helps capture the dynamics and expressiveness of the music (0 to 31)

3. Pitch: Which note being played ranging from (0 to 127)

4. Tempo and Time Signature: speed and rhythmic structure of the music. Providing context for note durations and phrasing.

### 6.2 Data Transformation:

One major challenge with polyphonic data is expanding the data representation to effectively capture the simultaneous occurrence of multiple notes and their associated properties. At any given time step, each of the 128 notes could either be on or off, with each note having a velocity value between 0 and 31.

In total, for each time step, there are potentially 288 distinct events (Eibensteiner et al., 2021):

1. 128 ON-Note events: Representing the notes that are currently being played.

2. 128 Off-Note events: Representing the notes that are no longer being played.

3. 32 velocity events: Indicating the intensity (or velocity) of the notes.

The occurrence of these events at each time step are modeled by a 288x288 dimensional one-hot vector. This serves as the input to the network at a given time step. This increased information is meant help the model to learn the complex relationships between multiple simultaneous notes, their durations, and their dynamic properties.

## 7 Updated RNN Model

The increased complexity of the data requires a more sophisticated model to capture the relationships accurately. The high-level RNN architecture remains the same, as shown below; the key difference lies in the input and output.

Instead of the previous 128x128 one-hot vector, the model now uses a 288x288 one-hot vector. This vector encodes the ON-note, OFF-note, and velocity information, enabling the model to gather more context from the music.

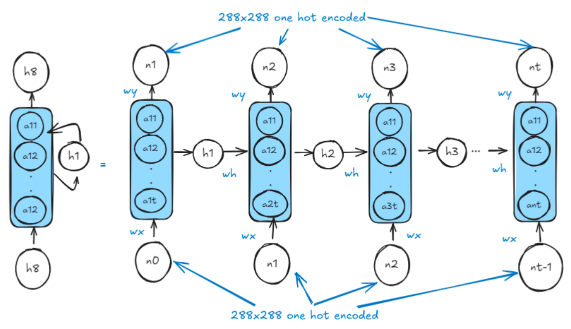

Figure 8: High-level overview of RNN structure for polyphonic case

Following the Performance RNN architecture (Oore et al., 2020) (Simon and Oore, 2017), I adapted the monophonic RNN to consist of three LSTM layers, each containing 576 cells.

This adjustment allows the model to capture more complex temporal dependencies and interactions between multiple notes in polyphonic music.

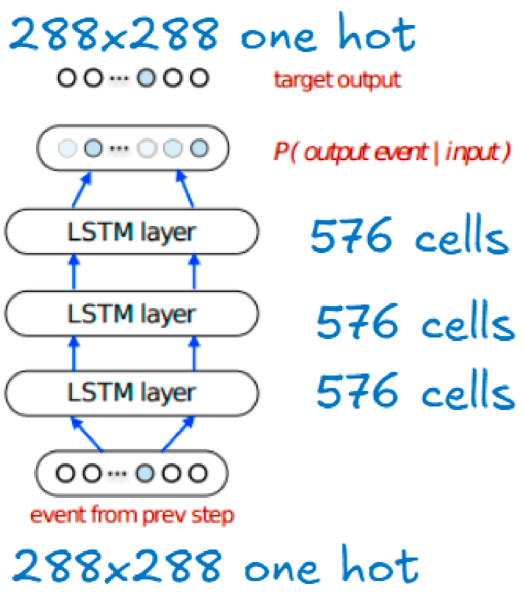

Figure 9: High-level overview of RNN structure

The overall goal of the model remains consistent with the monophonic RNN: to predict the next step in the sequence based on the previous step and the hidden state. The key difference in the polyphonic version is that, instead of predicting a single note as in the monophonic case, the model is now tasked with predicting a series of notes that could occur simultaneously, along with their corresponding velocities.

### 7.1 Training:

The training process remains the same. The model was evaluated by two metrics (Yang and Lerch, 2018):

1. **Accuracy:** The percentage of notes in a sequence correctly predicted by the model.

2. **Perplexity:** A measure of how well the model predicts an entire sequence, with lower values indicating better predictions.

Due to the complexity of the model and the limited computational resource limitation available to me, I trained the model for about 600 global steps before terminating. After 600 epochs, I got an accuracy of about 61.1% and a perplexity of 7.21.

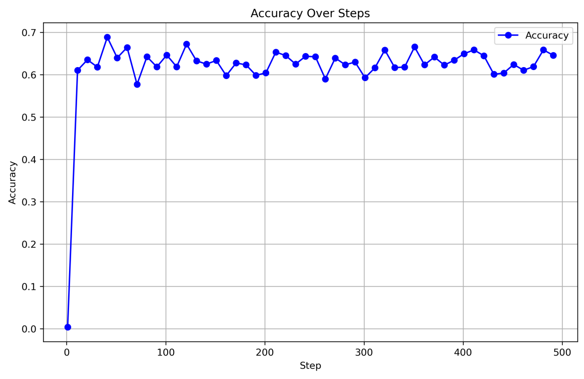

Figure 10: Change in accuracy on the training data over 2000 steps for the polyphonic case

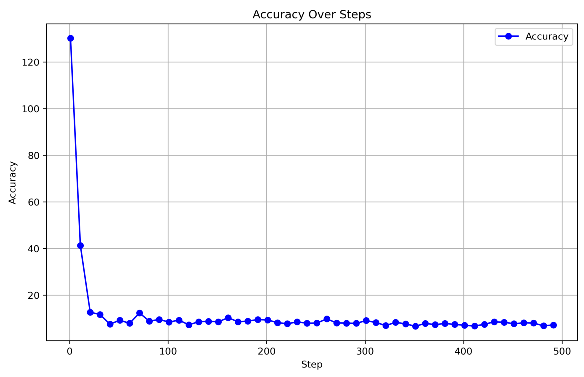

Figure 11: Change in perplexity on the training data over 2000 steps for the polyphonic case

The accuracy and perplexity scores were worse for the polyphonic model than the monophonic data. This is primarily attributed to the fact that polyphonic music is significantly more complex than monophonic data.

## 7.2 Generation and Evaluation of Music

The process of generation of polyphonic music is similar to that of monophonic music. The generation script forward passes the input music through the model. Then, the hidden state and the last predicted element of the sequence are used to predict the next step. This is iteratively repeated to create an entire composition (Godwin et al., 2022).

The three different use cases evaluated for the monophonic model were also evaluated for the polyphonic case:

### 7.2.1 Generating Short Extensions:

**Example 6: Generating short extensions (30 seconds)**

This is a 5-second input that was fed into the model:

**Input Melody 1:** Input 1▶

The following are three sample extensions:

**Extension 1:** Extension 1▶
**Extension 2:** Extension 2▶
**Extension 3:** Extension 3▶

In the examples above, I generated a 25-second extension to the input melody, resulting in a total sequence length of 30 seconds.

In contrast to the monophonic model, the generated extensions for the model lacked a discernible melody or coherent musical structure.

While the monophonic model maintained consistency between its input and extensions, the polyphonic extension could not retain any aspect of the input melody. The polyphonic model could not learn from the velocity and dynamics present in the MIDI data. As a result, the generated music remained chaotic and lacked fluidity.

### 7.2.2 Generating Long Extensions:

**Example 7: Generating long extensions (1.5 minutes)**

This is a 5-second input:

**Input Melody 1:** Input 1▶

The following are three sample extensions:

**Extension 1:** Extension 1▶
**Extension 2:** Extension 2▶
**Extension 3:** Extension3▶

In the examples above, I generated a one-minute, 30-second extension. Similar to the short extension scenario, the output was chaotic and lacked structure, indicating that the model could not capture the music's inherent qualities.

### 7.2.3 Generating Fresh Composition

**Example 4: Generating long extensions (2 minutes) without input prompt (the input is a non-event [-2])**

The following are a few sample 2-minute compositions the model created:

**Composition 1:** Composition 1▶
**Composition 2:** Composition 2▶
**Composition 3:** Composition 3▶

In the examples above, compositions were generated without any input melody. Like the previous scenarios, the outputs appeared chaotic and lacked coherence, highlighting the model's inability to produce structured or musically meaningful compositions.

Despite the changes made to the monophonic model to accommodate polyphonic data, it seems that they were insufficient for it to grasp the patterns inherent to polyphonic music. The model struggled to produce a consistent melody, rhythm, or harmony, and the overall output did not reflect the natural flow typically seen in real musical compositions. Further modifications will need to be made to the model in later works to improve its generation.

## 8 Conclusion

Overall, the project consisted of two parts. The first part was focused on demonstrating whether a RNN could understand patterns in monophonic music and temporal relationships between notes. The second part attempted to extend this to the more complex field of polyphonic music.

The primary goal of the first part was to generate realistic musical extensions based on a given input melody. A RNN model with 2 LSTM layers of 128 cells was used. The model aimed to achieve this goal by predicting the following notes in a sequence based on the previous notes and hidden states.

The evaluation demonstrated the model's capability to learn patterns in monophonic music. The generated extensions were fluid, maintaining the melody and rhythmic structure, making the model effective for generating musical continuations.

The project's second phase focused on extending the monophonic RNN model to handle polyphonic music. I modified the training data and model architecture to accommodate polyphonic compositions. More aspects of the data were included in the input. The model was modified to include three LSTM layers with 576 cells to increase its capacity to learn from this more complex data.

Despite these modifications, the model struggled with polyphonic music generation. Generated compositions lacked coherence, and the outputs were chaotic and lacked musical structure.

The results from my attempts at polyphonic music generation suggest that while the modified RNN model could handle the basic structure of polyphony, it was not enough to generate musically meaningful compositions. Further refinements, such as exploring attention mechanisms, are needed for better polyphonic music generation.

### 8.1 Strengths:

Evaluation of the outputs process highlighted several strengths of the approach:

1. **Melodic Consistency in monophonic data:** The generated output was able to maintain logical transitions, fluidity, and rhythmic coherence

2. **Versatility:** The model was able to handle different use cases, such as generating short extensions, generating long extensions and generating entire compositions from scratch

### 8.2 Limitations

There are also some major limitations in the model:

1. **Unable to effectively handle polyphonic data:** The generated extensions to polyphonic data lacked any consistent melody or fluidity.

2. **Long-Term Structure:** While monophonic-generated music is locally consistent, it lacks overarching structure. One issue might be the lack of model complexity, as it cannot store context for a large enough time interval.

### 8.3 Future work:

Several areas of improvement and exploration remain:

1. **Interactive Tools:** Develop user-friendly interfaces with electronic keyboards so musicians can interactively guide or influence the generation process (Parvian, 2018) (Simson et al., 2018).

2. **Polyphonic Music:** More work needs to be done to improve the model to handle polyphonic music. (Gautam et al., 2019) (Kumar and Ravindran, 2019)

3. **Increase model complexity:** The current RNN implementation can be improved by incorporating attention mechanisms to capture

long-term dependencies. More state models, such as Variational Auto Encoders (VAE) (Roberts et al., 2018) and Generative Adversarial Networks (GAN) (Engel et al., 2019), could also be investigated.

## References

N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent. 2012. Deep learning techniques for music generation– a survey. *Proceedings of the 13th International Society for Music Information Retrieval Conference*, pages 1–8.

Jean-Pierre Briot. 2020. From artificial neural networks to deep learning for music generation – history, concepts and trends. *Neural Computing and Applications*, 32:977–994.

Jean-Pierre Briot and François Pachet. 2017. Music generation by deep learning - challenges and directions. *CoRR*, abs/1712.04371.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Miles Brundage, and Ilya Sutskever. 2019. Musenet: A transformer-based model for music generation.

David Cope. 2001. *Experiments in Musical Intelligence*. A-R Editions, Inc., Madison, WI.

Lukas Eibensteiner, Martin Ilčík, and Michael Wimmer. 2021. Temporal-scope grammars for polyphonic music generation. In *Proceedings of the 9th ACM SIGPLAN International Workshop on Functional Art, Music, Modelling, and Design*, pages 23–34. ACM Press.

Jesse H. Engel, Kumar Krishna Agrawal, Shuo Chen, Ishaan Gulrajani, Chris Donahue, and Adam Roberts. 2019. Gansynth: Adversarial neural audio synthesis. *CoRR*, abs/1902.08710.

Aayush Gautam, Bipin Oli, Kuldip Bhutoria, and Kapildev Neupane. 2019. *Polyphonic Music Generation with RNN*. Ph.D. thesis.

Toby Godwin, Georgios Rizos, Alice Baird, Najla D. Al Futaisi, Vincent Brisse, and Björn W. Schuller. 2022. Evaluating deep music generation methods using data augmentation. *CoRR*, abs/2201.00052.

Kratarth Goel, Raunaq Vohra, and Jajati Keshari Sahoo. 2014. Polyphonic music generation by modeling temporal dependencies using a RNN-DBN. *CoRR*, abs/1412.7927.

Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse H. Engel, and Douglas Eck. 2018. Enabling factorized piano music modeling and generation with the MAESTRO dataset. *CoRR*, abs/1810.12247.

A. Huang, Z. Yang, A. Dai, R. Salakhutdinov, and Q. V. Le. 2018. Music transformer: Generating music with long-term structure. *Proceedings of the International Conference on Machine Learning (ICML)*, pages 2339–2348.

Yang Huang, R. B. M. K. M. Choi, S. H., H. R., and D. A. L. 2020. One deep music representation to rule them all? a comparative analysis of different representation learning strategies. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, New York, NY, USA. ACM.

Harish Kumar and Balaraman Ravindran. 2019. Polyphonic music composition with LSTM neural networks and reinforcement learning. *CoRR*, abs/1902.01973.

Dinh-Viet-Toan Le, Louis Bigo, Mikaela Keller, and Dorien Herremans. 2024. Natural language processing methods for symbolic music generation and information retrieval: a survey. *arXiv preprint arXiv:2402.17467*.

Eric A. N. Lee et al. 2017. Finding temporal structure in music: Blues improvisation with lstm recurrent networks. *Proceedings of the International Conference on Machine Learning (ICML)*, pages 2308–2317.

Kjell Lemström and Jorma Tarhio. 2000. Searching monophonic patterns within polyphonic sources. 2.

Sanidhya Mangal, Rahul Modak, and Poorva Joshi. 2019. LSTM based music generation system. *CoRR*, abs/1908.01080.

John A. Maurer. 1999. A brief history of algorithmic composition. Accessed: December 2, 2024.

Sageev Oore, Ian Simon, Sander Dieleman, Douglas Eck, and Karen Simonyan. 2020. This time with feeling: learning expressive musical performance. *Neural Computing and Applications*, 32(4):955–967.

T. Parvian. 2018. Connecting with music through magenta.js. Accessed: 2024-11-22.

A. D. Patel and I. A. G. Hagoort. 2001. Comparison between language and music. *Annals of the New York Academy of Sciences*, 930(1):232–258.

Adam Roberts, Jesse H. Engel, Colin Raffel, Curtis Hawthorne, and Douglas Eck. 2018. A hierarchical latent vector model for learning long-term structure in music. *CoRR*, abs/1803.05428.

Sahreen Sajad, S Dharshika, and Merin Meleet. 2021. Music generation for novices using recurrent neural network (rnn). In *2021 International Conference on Innovative Computing, Intelligent Communication and Smart Electrical Systems (ICSES)*, pages 1–6.

I. Shapiro and M. Huber. 2021. Markov chains for computer music generation. *Journal of Humanistic Mathematics*, 11(2):167–195.

I. Simon and S. Oore. 2017. Performance rnn: Generating music with expressive timing and dynamics. Accessed: 2024-11-22.

I. Simson, C. Hawthorn, and A. Roberts. 2018. Magenta.js. Accessed: 2024-11-22.

TensorFlow Team. 2020. Load data using tfrecord. Accessed: 2024-11-22.

Elliot Waite. 2016. Generating long-term structure in songs and stories. Accessed: 2024-11-22.

L.-C. Yang and A. Lerch. 2018. On the evaluation of generative models in music. *Neural Computing and Applications*, 32(9):4773–4784.

Y. Zhang, X. Liu, and Z. Chen. 2018. Comparative assessment of markov models and recurrent neural networks for jazz music generation. *Neural Computing and Applications*, 31(4):1059–1075.