Generate A New Classical Song Using Recurrent Neural Network

Jihun Kim
Department of Electrical Engineering and Computer Science
University of Tennessee, Knoxville
Knoxville, USA
jkim172@vols.utk.edu

Jonathan Skeen
Department of Electrical Engineering and Computer Science
University of Tennessee, Knoxville
Knoxville, USA
jskeen6@vols.utk.edu

***Abstract*—The goal of this paper is to identify the potential of using neural networks for music generation. We chose the classical music genre as our input. Inputs were sorted by the artist; thus, outputs were created based on a particular artist. We used an encoded MIDI representation of the input and produced a single MIDI representation as output. Our model was set up using the TensorFlow library and is a recurrent neural network, consisting of an autoencoder and LSTM layers. We ran various epochs to identify the ideal training range. Our results were positive overall. Due to the simplicity of the input representation, the output was also relatively simple. However, our output exhibits a strong correlation to the input as well as evidence of the new music generation.**

Keywords—Recurrent Neural Network, Long Short-Term Memory, Music Generation, Artificial Intelligence, MIDI, TensorFlow

I. Introduction and Motivation

Over the past decade, and even more so recently, there have been attempts to apply Artificial Intelligence (AI) in various fields, such as interactive artificial intelligence services or drawing pictures using input keywords. The results of these attempts have left a strong impression on people and have proved that AI-related technologies have advanced considerably in the past decade. With this in mind, we thought about the field of composition. As a new picture is created by learning a certain pattern from various pictures, we thought the same thought process could be applied to music composition; create a new song by learning a certain pattern from various songs. We believe that if this is possible, it could produce revolutionary changes in the field of music production.

With this thought in mind, we wondered if AI could analyze existing songs and compose new music which would be better than that of human composition. Just as AI could generate high-quality pictures, could it also generate high quality music. We thought it would be best to use neural networks to learn patterns and create new ones. In this study, we aimed to find out whether a song generated by a neural network could be comparable to a song written by a human.

In this project, we used a recurrent neural network (RNN) consisting of an autoencoder to create a model that can identify patterns of existing classical songs and create new classical songs using the patterns. With this model, we input various classic songs as MIDI files and output a new classical song as MIDI files. We tried to figure out how much the quality of songs produced by this model increased as we increased the number of epochs of this model.

During the experiment on this project, we found that in order to build an effective neural network, we need to build a neural network model using long short-term memory (LSTM) as well as an autoencoder to create a MIDI file consisting of more than just one note. Furthermore, we found that MIDI files were created using more diverse notes and complex scales when using a larger number of epochs.

II. Related Work

The music industry is very large and can offer life-changing success. This has led many people to consider the helpfulness of artificial intelligence pertaining to music generation. Thus, there was a substantial amount of related work available to us throughout this process.

Johnson [7] studied the representation of MIDI files as 2D binary arrays. We took the most inspiration for the encoding and decoding of data. This simple setup allowed for a much easier approach to the project and allowed for more focus on the network itself rather than encoding and decoding data. A downside to this approach is the lack of complexity we were able to represent. Music has complex events happening almost constantly. This includes changing tempos, incorporating melodies, and many other things. This approach limits many elements of the piece, simply because we are only able to record notes.

Eck and Schmidhuber [1] established the effectiveness of using Long Short-Term Memory (LSTM) units within recurrent neural networks. They used LSTM to train their model by learning the chords or melodies of a song, which allows it to learn and play long-form musical melodies and create stronger correlations across songs. We were inspired by this study on how to utilize LSTMs. Going further from their approach, we tried to build a better-trained neural network model by leveraging LSTM's layer count adjustment and repeat vectors.

The use of other types of networks has also been introduced and tested. Gunawan, Iman, and Suhartono [4] studied the use of gated recurrent units (GRUs) and incorporated them into an artificial music generation test. GRUs operate in a somewhat similar fashion to LSTMs in that they attempt to capture temporal qualities while avoiding the inclusion of a memory cell.

They built their RNN model using LSTMs and GRUs, and fed MIDI files into the model, sorted by the era in which the songs were written, such as baroque, classical, and romantic. Input MIDI files are encoded, learned, and trained. And they evaluate the output of their model using accuracy. This series of processes has been a great inspiration for establishing our model construction and evaluation methods. Based on their research, we also incorporated elements such as autoencoder and repeat vector into our model and classified the input songs according to their composers rather than according to a specific era.

AMT identifies pitch and models the acoustic signal for it. To build a more effective symbolic music prediction system utilizing AMT, Sigtia et al. [2] applied RNN. The resulting model was able to make high-level symbolic predictions and allowed the construction of more efficient algorithms.

Chuang and Su [5] used RNNs for beat tracking, as opposed to music generations. Extract information such as namely onset time, offset time (or note duration), and pitch value from MIDI data and train it through a recurrent neural network. In this process, they also utilized LSTM. Compared to the previous models, it showed excellent beat-tracking ability, but this also showed limited performance.

Yang, Chous, and Yang [9] involved using convolutional neural networks (CNNs) and generative adversarial networks (GANs). This method also produced positive results and allowed us to consider other designs for our network (i.e., the inclusion of an auto-encoder).

Colombo et al[3] involved separating notes from melodies to improve the long-range quality of the music piece. They analyzed the pitch and duration of the song and encoded them as 1's and 0's. The information collected in this way is learned in multi-layers in RNN and used for model training. They compare the score of the original song and the model's song, showing how well the model followed the original song's pattern and composed a new song.

Samuel and Pilat [6] attempted the incorporation of recurrent neural networks with multi-instrumental music. It extracts meters, keys, and chords from MIDI files, processes them, and inputs them into the RNN model. Since they want to use a variety of instruments, the input values have a much more complex form than other studies, and the fact that each instrument has a small input value seems to have made the research difficult. Still, their neural network model generated well-evaluated songs. We want to extract information from the MIDI files of several classical songs, focusing on just one instrument. By doing so, it is expected that it will be simpler and easier to input information into our model.

Chen and Miikkulainen[8] combined evolutionary algorithms and recurrent neural networks for their study. In their study, they used neural networks to generate melodies and applied evolutionary algorithms to generate better melodies. To apply the evolutionary algorithm, the fit function for pitch and rhythm was utilized. Their system worked quite well, and the evolutionary algorithm followed the constraints faithfully while generating better melodies. However, expressions such as dotted notes, rests, and chords were omitted from their results. The lack of harmony produced results that were good for each measure but not satisfactory for the overall composition.

Sheikholharm et al [10] combined genetic algorithms and recurrent neural networks in their research. A fitness function was also applied for the utilization of genetic algorithms. They utilized pitches and durations of notes via genetic algorithms and recurrent neural networks to form new melodic passages. Genetic algorithms have formed crossovers and mutations, and these mutations include various variations such as changing adjacent notes, transposing octaves, inverting groups of notes, and changing the duration of notes. The music thus created forms a new form of music while still following the pattern of the original music.

III. Method

A. *Learn from Others*

Before building a neural network model that analyzes various music files and outputs new music files, we read papers related to this topic to gain a better understanding of this topic and to see how others have approached it. As a result of reading various papers and analyzing the subject, it was found that there is a technique commonly used in various papers, and I was sure that we could create our bio-inspired model using this.

From our research, we concluded that a recurrent neural network was the most optimal solution for our goal. This is because recurrent neural networks use outputs from the previous time step as inputs to the current time step. This is important because musical notes between different time steps always have some level of correlation. Thus, if a recurrent neural network can learn certain patterns, it should be able to generate a somewhat pleasant output.

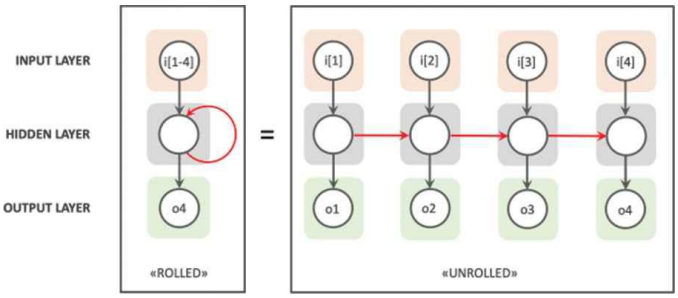

Fig. 1. Recurrent Neural Networks(RNNs) [4]

Another inspiration was the use of LSTMs. These modules have a memory cell within them and essentially work in tandem with a recurrent neural network to produce a more connected output. Since LSTMs operate with a memory cell, there is some recollection of what the previous state was. This means an LSTM can remember its previous output and modify the current output accordingly. This improves the correlation effect and when paired with the characteristics of a recurrent neural network should provide a better output.

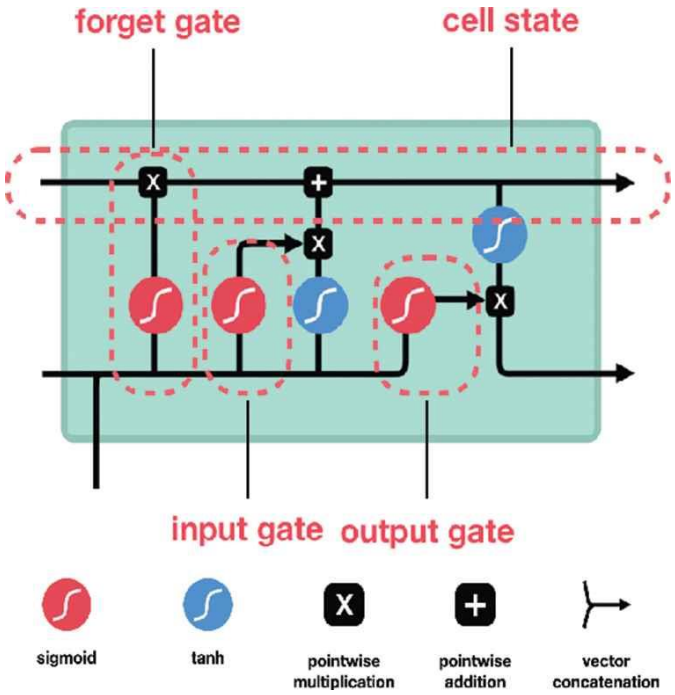

Fig. 2. Long short-term memory (LSTM) diagram [4]

The last piece to our network was the addition of an autoencoder. Autoencoders operate by compressing and decompressing data, allowing for both pattern recognition as well as generative output. These are both valuable properties of a music generator model because we want to produce a new musical piece while still having it inspired by the original input.

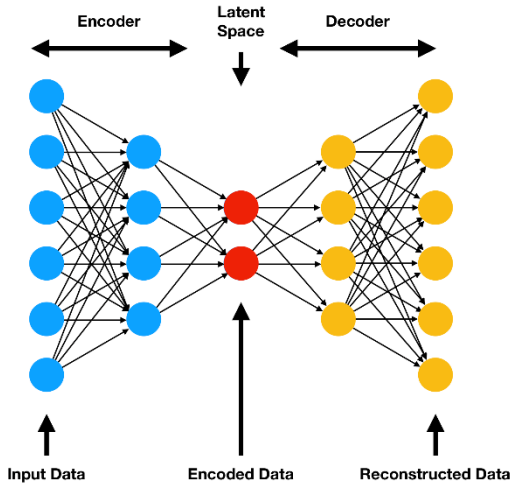

Fig. 3. Auto-encoder diagram[11]

We were inspired by the various models mentioned above. By reading various papers related to the research topic, we have improved our understanding of the structure of MIDI files and have been able to think about how our neural network model can utilize MIDI files. We also learned that LSTMs and autoencoders can be used to build more effective neural networks.

B. Overall Process of Our Model

We used Python's Mido library as our interface with MIDI files and the TensorFlow library to build our neural network. Using the Mido, Numpy, and TensorFlow libraries, we were able to create a simple recurrent neural network.

TABLE I. NUMBER OF INPUT MIDI FILES

Name of Composers	Number of MIDI Files	Total Play Time (minutes)
Tchaikovsky	12	37.11
Mendelssohn	15	37.40
Clementi	17	38.11
Schumann	24	54.80
Haydn	21	69.25
Beethoven	29	182.75

We wanted to input classical songs from various composers into the RNN model in the form of MIDI files. Also, we wanted to see if our neural network model can catch the features of each composer's songs. Therefore, we categorized MIDI files by composer and inputted each composer's MIDI file.

This model stores information about notes and chords in vectors from each incoming MIDI file. The input information is encoded in the form of a 2D binary vector so that it can be used in the neural network. This encoded information is used to train the RNN model.

C. First Attempt to Build a Model

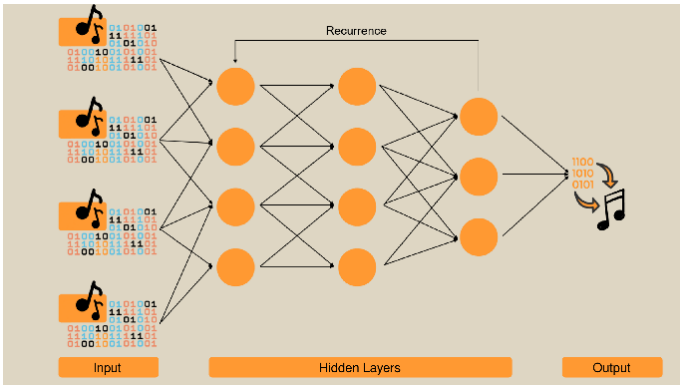

Fig. 4. Early Recurrent Neural Network Model

To build our RNN model, we used Long Short-Term Memory (LSTM) layers. This model has the advantage of allowing outputs to be reused as inputs. LSTM networks can maintain a recollection of its past outputs, which has the advantage of helping to maintain the flow and rhythm of a song and improving the overall quality of the song.

The neural network built in this way learns patterns from various MIDI files. For more effective training, the Adam optimizer was used. Moreover, the difference in the results was checked by adjusting the number of epochs which

determines the number of recurrences. The trained neural network outputs a 2D binary vector, and we adjusted it to create a song of an appropriate length. The 2D binary vector output in this way is converted into a MIDI file and output as music that everyone can hear.

However, our early model was not very good at finding patterns from MIDI files and would always output songs with only one note regardless of the number of epochs.

D. *Second Attempt to Build a Model*

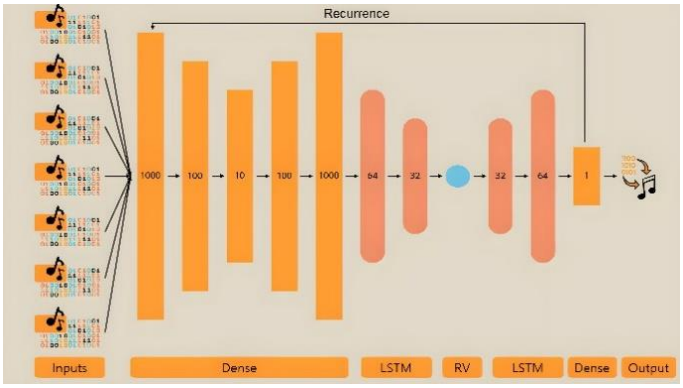

Fig. 5. Improved Recurrent Neural Network Model

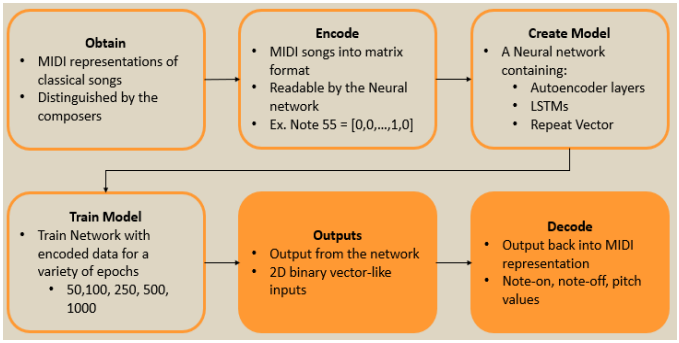

Fig. 6. Overall Process of Creating a New MIDI File

We recognized that we were missing an important part of building neural networks. Therefore, we consulted with Dr. Schuman to get advice to improve our neural network model. She recommended using an autoencoder for our neural network model.

Autoencoders go through the process of compressing and decompressing information to learn patterns. The autoencoder applied to our model starts with 1000 layers, compresses to 100 and 10 layers, and then decompresses to 100 and 1000 layers again, learning the patterns of MIDI files. Also, we used Repeat Vector which repeats the input n times to add an extra dimension to the dataset. In our new model, we used 64 layers of LSTM which compressed to 32 layers and decompressed to 64 layers. Using autoencoder, LSTM, and repeated vector, we could build a better recurrent neural network model.

Our improved model was able to create songs using multiple notes rather than just one. Also, compared to the previous model, it analyzed patterns from input MIDIs well.

IV. Results

A. *Time to Generate a New MIDI File*

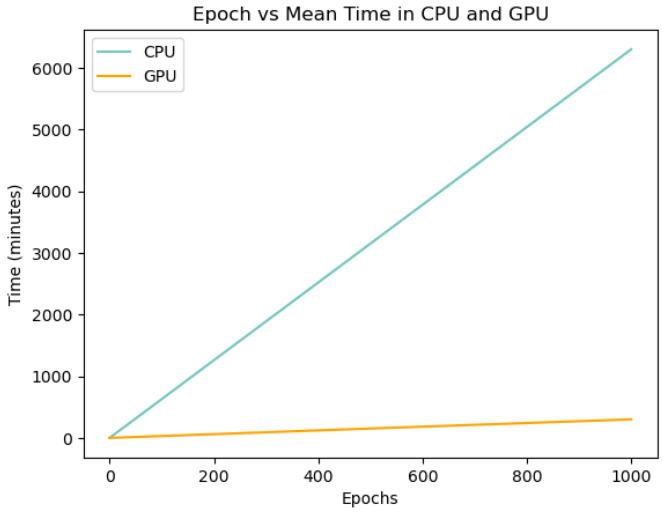

Fig. 7. Time Consumed by Number of Epoches in CPU and GPU

Our model using RNN is highly influenced by the number of epochs, which is the number of recurrences of the neural network. Using TensorFlow, users can decide whether to use CPU or GPU when running a neural network program. There is a significant difference in performance depending on using CPU or GPU, and GPU runs a program faster than the CPU does in every case.

Fig 7 shows the average time until the program is done at each epoch when using CPU and GPU. When the number of the epoch is 50, it takes an average of 94.5 minutes, or 1.5 hours, to complete the program using the GPU, while it takes an average of 315 minutes, or 5.25 hours, to complete the program using the CPU. When the number of the epoch is 1000, it takes an average of 301 minutes, or 5 hours, to complete the program using the GPU, while it takes an average of 6305 minutes, or 105 hours, to complete the program using the CPU.

Considering the difference in running time between 50 epochs and 1000 epochs, the execution time of the program is greatly affected by the number of epochs.

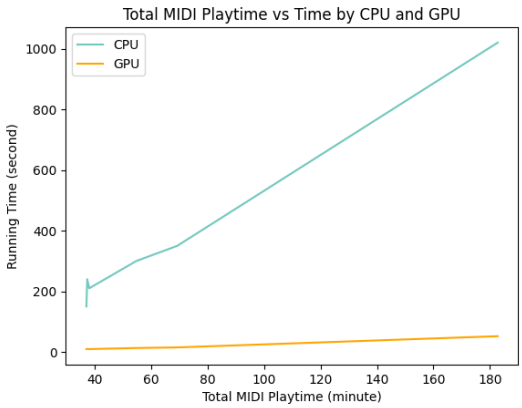

Fig. 8. Time Consumed by Total Play Time of Input MIDIs in CPU and GPU

Even if you run the program with the same number of epochs, the running time of the program can vary greatly depending on the playtime of the MIDI file, i.e., the size of the input files.

Fig 8 shows the running time of each epoch depending on the total playtime of the input files when running the program using the CPU and GPU. When a MIDI file with a total playtime of 37 minutes is input into the program, each epoch takes 150 seconds, or 2 minutes 30 minutes, using the CPU and 9.5 seconds for each epoch using the GPU. When a MIDI file with a total playtime of 182.75 minutes is input into the program, each epoch takes 1020 seconds, or 17 minutes, using the CPU, and 52 seconds for each epoch using the GPU.

Considering the difference in the time required for each epoch according to the playtime, the size of the input file greatly affects the execution time of the program, and the larger the size of the input file, the longer the execution time of the program.

B. Similarity to the Original

A neural network model analyzes patterns from input MIDI files and learns the patterns to create new MIDI files. Adjusting the way how the model learns can make a huge difference to the output. We wanted to avoid overfitting by referencing the original MIDI file too much, and we also wanted to avoid having too little similarity by referencing the original MIDI file too less. To do this, we scored the similarity score for each MIDI output and compared the scores between each epoch. The equation we used to calculate the similarity of the output MIDI file is:

$$similarity = |I \cap O| / |I \cup O| \quad (1)$$

where I is a set of input MIDI file sequences and O is a set of output MIDI file sequences. Each sequence has the information of notes in the MIDI file. This equation finds the intersection of the two sets and divides it by the union of the two sets. In other words, this equation finds how many different notes contains among whole notes. We repeated this calculation until finishing comparing all input files and output files and will find the average number of similarity scores.

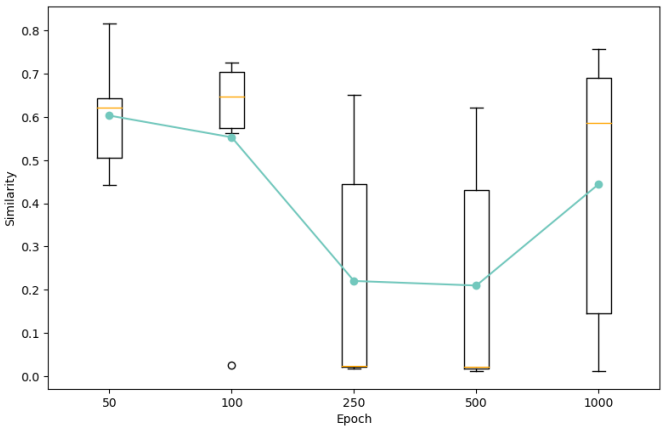

Fig. 9. Similarity Score for Each Epoch

Fig 9 shows the similarity score for each epoch. The blue line represents the average similarity score, and the box plot shows the median and variance of similarity scores. We expected that the lowest similarity score would come out when the number of epochs is 50 and 100. However, contrary to our expectations, the lowest similarity score came out when the number of epochs was 250 and 500, and the highest similarity score came out when the number of epochs was 50 and 100.

We analyzed the effect of the number of epochs on the learning of the neural network through graphs. As the number of epochs increases from 50 to 100, the neural network shows a pattern that closely resembles the original MIDI file due to a lack of training. This is a big reason for the sharp drop in similarity scores as epochs increase from 100 to 250 and 500. When the number of the epoch is 250 and 500, they learn the pattern more than when the number of the epoch is 100. However, the training is still not enough, so the similarity score is low. When the number of epochs reaches 1000, the neural network learns enough from the original MIDI file, and based on this, it can make good use of it when outputting a new MIDI file. This causes the similarity score to rise again as the number of epochs becomes 1000.

Our experiment found that when the number of epochs was 1000, the optimal MIDI file was derived by learning the pattern of the original MIDI file well. A sufficient number of epochs are required for the neural network to output a correct result through sufficient training.

C. Rhythm, Notes, and Techniques

Fig. 10. Note sequence of a MIDI file generated with Tchaikovsky's songs in 50 epochs

Fig. 11. Note sequence of a MIDI file generated with Tchaikovsky's songs in 1000 epochs

Fig. 12. Note sequence of a MIDI file generated with Schumann's songs in 50 epochs

Fig. 13. Note sequence of a MIDI file generated with Schumann's songs in 1000 epochs

Fig 10 and Fig 11 show a note sequence of MIDI files generated with Tchaikovsky's songs which have 37.11 minutes of playtime in total. Fig 12 and Fig 13 show a note sequence of MIDI files generated with Schumann's songs which have 54.8 minutes of playtime in total. Fig 10 and Fig 12 show a note sequence of a MIDI file generated with 50 epochs while Fig 11 and Fig 13 show a note sequence of a MIDI file generated with 1000 epochs.

Comparing the shape of the note sequence in Fig 10 and Fig 11 with the shape of the note sequence in Fig 12 and Fig 13, the note sequences in Fig 12 and Fig 13 use more varied notes. Also, comparing the shape of the note sequence in Figs 10 and Figs 12 with the shape of the note sequence in Figs 11 and Figs 13, the shape of the note sequence in Figs 11 and Figs 13 is more complex, and various patterns are observed.

By comparing and analyzing the differences in the note sequences shown in Fig 10 to Fig 13, we found how MIDI files could have a more complicated pattern:

- When the number of epochs is the same, the output MIDI file could have a more complicated pattern when the neural network model had a greater size of input files.
- When the input files are the same, the output MIDI file could have a more complicated pattern when the neural network model had a greater number of epochs.

V. Discussion, Conclusion, and Future Work

A. Discussion and Conclusion

To build a more effective neural network model, we utilized LSTMs and an autoencoder. We found that learning and training did not work well when we utilized only LSTMs in our neural network model, and most of the output MIDI files consisted of only one note. By using LSTMs and autoencoder together in the neural network model, better output was generated compared to using LSTM alone.

We used the neural network we built to learn and train various classic MIDI files and output a new MIDI file. In this process, we found that the learning and training time is greatly affected by the number of epochs and the size of the input MIDI files, and the time can be greatly reduced depending on the performance of the CPU or GPU.

A small number of epochs will have an insufficient number of recurrences to analyze the pattern from the original MIDI files, so the output MIDI file is too monotonous. On the other hand, a sufficiently large number of epochs can output a sufficiently learned and trained MIDI file, which sounds like the original songs but has its own pattern and has more diverse notes and more complex scales.

B. Music Theory and MIDI Improvement

In this study, we mainly focused on learning, training, and outputting note sequences from input MIDI files. In follow-up research, if we can train our neural network by dealing with variation, tempo, or chords from MIDI files in more depth, we expect to be able to obtain a MIDI file that significantly improved qualitatively. This will likely involve an alternate representation to our current one but could offer huge improvements to our outputs.

Another consideration is the addition of music theory to better define a fitness function. Currently, an outputs rating is given based on its similarity to the input. This is suboptimal since our main goal is to generate new music, rather than mimic the input. Thus, the inclusion of music theory could add more metrics to better identify the quality of the output.

C. More Epoch and More Improved Network

Moreover, the most difficult part of this study is that it takes too much time to create a single MIDI file. It was a time-consuming task that the CPU couldn't handle, so we relied mostly on the GPU to run the program. However, even with GPU, it takes quite a long time when the number of epochs is 1000. For this reason, 1000 epochs were the largest and most realistic number of epochs we could try. However, we would like to run the program with more epochs, as we have found that applying a higher number of epochs produces better-quality MIDI files.

To do this, we need to either utilize a GPU with better performance or build a more effective recurrent neural network model. Since neural networks are a vast field of research, we can try various network configurations.

For example, we can consider introducing evolutionary algorithms, genetic algorithms, or particle swarm optimization as in other previous studies. This would allow the testing of numerous different recurrent models to find which is optimal for our use case. If we do this, we expect that our model could generate a more diverse and advanced form of MIDI outputs compared to the outputs of our existing models. We expect that it is possible to build a recurrent neural network for music composition with better performance than the recurrent neural network we designed.

References

[1] D. Eck and J. Schmidhuber, “A First Look at Music Composition using LSTM Recurrent Neural Networks,” IDISA Research Report. [Online]. Available: https://people.idsia.ch/~juergen/blues/IDSIA-07-02.pdf

[2] S. Sigtia, E. Benetos, N. Boulanger-Lewandowski, T. Weyde, A. S. d'Avila Garcez and S. Dixon, "A hybrid recurrent neural network for music transcription," 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), South Brisbane, QLD, Australia, 2015, pp. 2061-2065, doi: 10.1109/ICASSP.2015.7178333.

[3] F. Colombo, S. Muscinelli, A. Seeholzer, J. Brea, and W. Gerstner, “Algorithmic Composition of Melodies with Deep Recurrent Neural Networks.” Available: https://arxiv.org/pdf/1606.07251.pdf

[4] A. A. S. Gunawan, A. P. Iman, and D. Suhartono, “Automatic Music Generator Using Recurrent Neural Network,” *International Journal of Computational Intelligence Systems*, vol. 13, no. 1, p. 645, 2020, doi: https://doi.org/10.2991/ijcis.d.200519.001.

[5] Y. -C. Chuang and L. Su, "Beat and Downbeat Tracking of Symbolic Music Data Using Deep Recurrent Neural Networks," 2020 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), Auckland, New Zealand, 2020, pp. 346-352.

[6] D. Samuel and M. Pilát, "Composing Multi-Instrumental Music with Recurrent Neural Networks," 2019 International Joint Conference on Neural Networks (IJCNN), Budapest, Hungary, 2019, pp. 1-8, doi: 10.1109/IJCNN.2019.8852430.

[7] D. Johnson, “Composing Music With Recurrent Neural Networks,” *Daniel D. Johnson*, Aug. 03, 2015. https://www.danieldjohnson.com/2015/08/03/composing-music-with-recurrent-neural-networks/

[8] C. . -C. J. Chen and R. Miikkulainen, "Creating melodies with evolving recurrent neural networks," IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No.01CH37222), Washington, DC, USA, 2001, pp. 2241-2246 vol.3, doi: 10.1109/IJCNN.2001.938515.

[9] L.-C. Yang, S.-Y. Chou, and Y.-H. Yang, “MidiNet: A Convolutional Generative Adversarial Network for Symbolic-domain Music Generation,” *arXiv:1703.10847 [cs]*, Jul. 2017, Available: https://arxiv.org/abs/1703.10847

[10] P. Sheikholharam and M. Teshnehlab, "Music Composition Using Combination of Genetic Algorithms and Recurrent Neural Networks," 2008 Eighth International Conference on Hybrid Intelligent Systems, Barcelona, Spain, 2008, pp. 350-355, doi: 10.1109/HIS.2008.46.

[11] S. Flores, “Variational autoencoders are beautiful,” Variational Autoencoders are Beautiful | Blogs, https://www.compthree.com/blog/autoencoder/ (accessed May 10, 2023).