Chapter 1

Introduction

The realm of music composition has long been shaped by human creativity, intuition, and innovation. Yet, the emergence of artificial intelligence and machine learning techniques has ushered in a new era in music creation, pushing the boundaries of what is conceivable in the realm of musical expression. Among these techniques, Generative Adversarial Networks (GANs) stand out as a powerful paradigm for generating music autonomously.

In this report, we delve into the world of Automatic Music Generation using GANs. GANs, initially popularized in visual art generation, have now found their place in the domain of music composition, enabling the creation of harmonies, melodies, and rhythms that resonate with human-like creativity. The inherent capacity of GANs to capture complex patterns, learn from data distributions, and generate novel outputs has ignited a wave of exploration in AI-driven music composition.

In this report, we will go over the representation used to handle music notes, the different kinds of models used to generate music autonomously, the model I created, the dataset used for training, and my attempts to make a better model.

Chapter 2

Literature Review

In the literature on machine learning, the field of music creation has gotten a lot of attention. In earlier studies, the preferred method was not machine learning; instead, researchers tried to combine ideas from emotional theory and music theory. For instance, Wallis et al. [1] used an algorithm to create music using the Arousal-Valence model of emotions. These early algorithms, however, did not produce music in a way that was comparable to human composition. These studies used algorithms to combine pre-made melodies, musical elements, and templates to create a musical composition rather than operating at the level of individual musical notes. This strategy limited the originality of the music that was created.

Subsequently, Machine Learning algorithms emerged as a more effective approach for generating music at the granular level of musical notes. Within this literature review, we provide an overview of the progression of model architectures employed in music generation, the symbolic representations of musical notes employed by Machine Learning models, and take a look into evaluation metrics for such models.

I Music representation

The first step in using musical data to train ML models is representing that data in a way that can be used by the models. ML models can work with vectors and matrices. Music can be represented using sheet music data, symbolic data, and audio data. Music sheets and audio files are extremely hard to use with ML models therefore, this project will only focus on symbolic music representation as it is the easiest way to represent music on a computer and feed it to a ML model. As we focus on symbolic data, MIDI is the most widely used symbolic music representation [2].

MIDI can be envisioned as a group of simple time series of events. The events are as follows ‘note X on‘, ‘note X off‘, ‘increase loudness‘, ‘decrease loudness‘, etc. Note that the previous events are not the exact events used by the MIDI representation but rather a simplification of them. Each of those time series is called a track. Each track is dedicated to an instrument. Each of those events is timestamped in milliseconds. Therefore, for a certain instrument, the MIDI representation shows when to play a certain note of it, for how long, and how loud. The MIDI representation keeps a recording of every possible note by every possible instrument. It then plays the appropriate recording when the appropriate event happens.

This representation as a group of time series makes it perfect to use MIDI as a data representation for seq-to-seq models as it can be transformed into a vector or matrix representation which in turn can be converted into tokens to be fed to seq-to-seq models. That will allow the use of MIDI data as input to multiple models such as RNNs and Transformers.

II Music Generation Architectures

There are multiple methods to generate music using machine learning. In this section we will go over some methods such as LSTMs, RNNs, CNNs, GANs, etc. In order to understand how each model has been used in previous research.

A. CNNs (Convolutional Neural Networks)

Mohammad Akbari and Jie Liang [3] presented a hybrid VAE-GAN model where they utilized CNNs in the generator and discriminator to generate sequential data. While that model was not used specifically in generating music, it was tested on piano music generation from a few starting notes and it showed great promise in being able to generate whole pieces of music.

Dong et al. [4] also used GANs with CNNs to generate symbolic music. Music has a hierarchical structure, with higher-level building blocks made up of smaller recurrent patterns. Their model used that fact to create a hierarchical representation of music over the time dimension. The result of the previous models is not an audio file that can be played but rather a MIDI file that can be used with a MIDI player to play audio.

B. RNNs (recurrent neural networks)

Even though RNNs are not looked at as state of the art models there have been multiple usages of them in generating music. Nadeem et al. [5] created an RNN model that can produce pleasant music where 34% of the test subject were not able to distinguish it from human music. In their model, they trained two separate layers where one of them is trained on chord data and the other on note data while insuring both inputs are considered at all steps of generation. After that the output of both layers is combined to create the music.

C. LSTMs (Long-Short Term Memory)

LSTMs are a variant of RNNs and while they are not state of the art models there have been multiple studies showing the use of LSTMs in generating pleasant music. Gino Brunner et al. [6] showcase this with their LSTM models. They created two LSTMs models, one which predicts chord progression based on chord embedding and one predicts the notes in that chord. Their simple model was able to create pleasant music and shows that even simple models can produce desired results.

D. Transformers

Transformers has been the most interesting development in seq-to-seq models in ML in recent history. [7] While Transformers are focused on language prediction as language models, they can be used to predict tokens to generate symbolic music. This method, however, is sub-optimal because "the full attention cannot efficiently model the typically long music sequences (e.g., over 10,000 tokens), and the existing models have shortcomings in generating musical repetition structures." [8]. Yu et al. [8] proposed Museformer, a Transformer model that is supposed to combat those inefficiencies. Museformer has two types of attention (fine- and coarse-grained attention). This allows the model to "capture both music structure-related correlations via the fine-grained attention, and other contextual information via the coarse-grained attention" [8] and " it is efficient and can model over 3X longer music sequences compared to its full-attention counterpart" [8]. This shows that Transformers are a great tool to use in music generation in ML.

E. Hybrid models

There have been multiple attempts to utilize different models and combine them in ways to increase their performance. Jiang et al. [9] have created a hybrid VAE-Transformer model which allows hierarchical music generation. Qin et al. [10] suggested a hybrid LSTM-Transformer model where they embedded notes using 2-LSTM layers and then embedded chords using a Transformer and then using the Transformer to decode the music.

III Evaluation Metrics

Evaluation of music is extremely difficult because it is hard to create objective metrics for it. Due to the nature of the problem subjective evaluation is prevalent and while subjective evaluation is a good way to direct our research, it suffers from drawbacks such as biases. There have been attempts to create as fair as possible subjective scores such as Mean Opinion Score which we can use to evaluate our work [11]. We can try to use some general objective measures, such as, training time, inference time, etc. There have been also attempts to create objective metrics for music generation specially. Wang et al. [12] have created Armor: A Benchmark for Meta-evaluation of Artificial Music which we can use to guide our work in creating music.

IV Conclusion

In this chapter, we went over the representation of music that can be used for ML purposes. We went over how MIDI representation works. We looked at multiple different architectures for music generation such as RNNs and Transfromers.

We finally took a look at some evaluation metrics that we can use to guide our work in music generation.

Chapter 3

Dataset

For this project, I am going to use Classical Music MIDI dataset [1]. This dataset contains music samples represented in MIDI which is what we need to use with seq-to-seq models.

[1]https://www.kaggle.com/datasets/soumikrakshit/classical-music-midi/data

Chapter 4

Initial Model

I Model Design

For the first model developed, I decided to go with something simple. I can use this simple model as a way to understand how difficult the problem is and as a way to create my preprocessing and inference codes.

For the preprocessing, I read multiple MIDI files of piano records of a certain composer, then I converted them to a list of strings where each string represented a certain note. After that I counted the number of times a certain note occurred in the music piece (a.k.a frequency) and filtered the notes based on frequency. I wanted to filter out the less repeating notes as to decrease the training time. I also looked at the unique notes as rather a sort of noise. After a little experimentation I decided to put the threshold for the filter at 50. After that, I created 2 dictionaries. One will map a certain note to a number and the other will map numbers to their corresponding notes. This will make the inference phase easier. After that we create the input and output sequences. The first input sequence is all the notes in the range [0, timestamp] and the output for the first sequence is the note at the index timestamp + 1. The second input sequence is all the notes from [1, timestamp + 1] and the output is the note at index timestamp + 2 and so on. After that we have our input sequences and output notes and we split them using the train test split function from sklearn.

After that, we start building the model. I used 2 stacked LSTM layers with a dropout rate of 0.2. Dropout basically prevents over-fitting while training the model. Finally I used a fully connected Dense layer for the output. I used an LSTM model because it is simple enough to experiment with and create the pre-processing and inference stages with while also having an idea about the quality of the output of such a simple model.

Output dimension of the Dense Layer will be equal to the length of our unique notes along with the ‘softmax’ activation function which is used for multi-class classification problems.

In the inference stage, I take random note from the test set. Then I predict the second note from it, then I append them and predict the third note from that sequence and so on until I predict a certain number of notes and create a long sequence that represents a MIDI file. After that, I convert that sequence of the MIDI representation into an actual MIDI file and save it.

II Results

I only relied on subjective evaluation of the result so far since this is a preliminary result. I have noticed that while the music does not sound bad, it does not sound too good either. The music sounds slightly creepy as if it is the background music for a horror movie or a video game. That does not mean that it is bad, only that it feels slightly "off". The music also seems to change a lot at the end of the sequence, going to different keys and chords from the start of the sequence. I assume this is due to the memory loss effect of LSTMs. The music file's length is 100 seconds, I have attached the music file titled "output" with this report. If the model is diverging with a 100 second piece music file then it cannot be used to create long music meaning we need to look for better models.

III Challenges

The first challenge was the training time. The model took about 30s per epoch which was much longer than what I expected. This signals that I need to find better training hardware when I move to more complex models. Second challenge was solving the problem of diverging output sequences, this will probably be solved by moving to a more complex model.

Chapter 5

Expansion Attempt

After implementing the first successful model, I tried to improve on it by using a GAN training framework. A similar feat was done by O. Mogren [13]. I attempted to implement a C-RNN-GAN network. Despite the potential of this model in generating continuous and coherent sequences, complexities in architecture design, training instability, and sequence coherence hindered successful implementation.

I Methodology

The implementation process commenced with an extensive study of Continuous-RNN-GAN [13] architectures and related literature. Understanding the interactions between RNNs and GANs, particularly in the context of generating continuous sequences, was crucial for the implementation efforts.

II Challenges Faced

- Code Implementation Complexity: Translating the theoretical concepts of Continuous-RNN-GAN into functional code proved highly challenging with a steep learning curve that I was not able to go through. The intricate interactions between RNNs, continuous sequence representation, and adversarial training added complexity to code development.

- Musical Data Representation: Representing musical data in a format suitable for training the Continuous-RNN-GAN model was intricate. Capturing musical nuances, temporal relationships, and ensuring continuity between notes posed a significant challenge.

- Temporal Coherence: Ensuring seamless transitions between musical notes and maintaining the overall coherence of the generated sequences across extended periods presented a formidable challenge.

III Failed Implementation Attempts

- Code Development: Multiple attempts were made to write code for Continuous-RNN-GAN architecture, incorporating continuous sequence representations and the adversarial training setup. However, complexities in integrating these components and achieving functionality hindered successful code development.

- Musical Representation: Despite efforts to preprocess musical data and represent it suitably for training, creating a representation that effectively captured musical nuances and coherence remained elusive.

IV Key Learnings

- Coding Complexity: Implementing complex neural network architectures like Continuous-RNN-GAN demands a deep understanding of network interactions and their intricate connections within the code.
- Musical Representation Challenges: Representing music in a manner that aligns with continuous sequences while preserving temporal coherence requires innovative encoding methods tailored specifically for music generation.
- Temporal Continuity: Ensuring seamless transitions and temporal coherence within generated musical sequences pose significant challenges that necessitate specialized handling within the model architecture and training process.

V Conclusion

The endeavor to implement Continuous-RNN-GAN for music generation faced formidable challenges in translating theoretical concepts into functional code and effectively representing musical data. Despite the unsuccessful implementation, the insights gained into the complexities of generating continuous music sequences will guide future research efforts towards developing more effective and robust models for music generation.

Chapter 6

Conclusion

In this report, we have went over the previous works related to this topic in such as CNNs, RNNs, LSTMs, Transfomers, and hybrid models. We have also went over the music representation used and the evaluation metrics to be used. We have also talked about the dataset used to train the model and the model created that can generate music autonomously. We also talked about our attempted to create a better model, the challenges faced, and the lessons learnt.

Bibliography cited

[1] T. I. I. Wallis and E. Campana, “Computer-generating emotional music: The design of an affective music algorithm,” p. 7, 2008, Accessed: October 8, 2022. doi: 10.1.1.212.5626. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?rep=rep1&type=pdf&doi=10.1.1.212.5626.

[2] G. Loy, “Musicians make a standard: The midi phenomenon,” *Computer Music Journal*, vol. 9, no. 4, pp. 8–26, 1985, Accessed: October 9, 2023. doi:10.2307/3679619. [Online]. Available: https://www.jstor.org/stable/3679619.

[3] M. Akbari and J. Liang, “Semi-recurrent cnn-based vae-gan for sequential data generation,” pp. 2321–2325, 2018, Accessed: October 15, 2023. doi:10.1109/ICASSP.2018.8461724. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8461724.

[4] R. Becker, F. Corò, G. D’Angelo, and H. Gilbert, “Balancing spreads of influence in a social network,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 01, pp. 3–10, Apr. 2020, Accessed: October 15, 2023. DOI: 10.1609/aaai.v34i01.5327. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/5327.

[5] M. Nadeem, A. Tagle, and S. Sitsabesan, "Let's make some music," pp. 1–4, 2019, Accessed: October 15, 2023. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8706447. DOI: 10.23919/ELINFOCOM.2019.8706447.

[6] G. Brunner, Y. Wang, R. Wattenhofer, and J. Wiesendanger, "Jambot: Music theory aware chord based generation of polyphonic music with lstms," pp. 519–526, 2017, Accessed: October 15, 2023. doi:10.1109/ICTAI.2017.00085. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8371988. DOI: 10.1109/ICTAI.2017.00085.

[7] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," vol. 30, I. Guyon, U. V. Luxburg, S. Bengio, *et al.*, Eds., 2017, Accessed: October 15, 2023. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[8] B. Yu, P. Lu, R. Wang, *et al.*, "Museformer: Transformer with fine- and coarse-grained attention for music generation," vol. 35, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., pp. 1376–1388, 2022, Accessed: October 15, 2023. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/092c2d45005ea2db40fc24c470663416-Paper-Conference.pdf.

[9] J. Jiang, G. G. Xia, D. B. Carlton, C. N. Anderson, and R. H. Miyakawa, "Transformer vae: A hierarchical model for structure-aware and interpretable music representation learning," pp. 516–520, 2020, Accessed: Oc-

tober 15, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/9054554. DOI: 10.1109/ICASSP40776.2020.9054554.

[10] Q. Y., X. H., D. S., and et al., “Bar transformer: A hierarchical model for learning long-term structure and generating impressive pop music,” *Appl Intell*, 2023, Accessed: October 15, 2023. [Online]. Available: https://link.springer.com/article/10.1007/s10489-022-04049-3. DOI: 10.1007/s10489-022-04049-3.

[11] R. Madhok, S. Goel, and S. Garg, “Sentimozart: Music generation based on emotions,” in *Proceedings of the 10th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART,*, Accessed: October 15, 2023. [Online]. Available: https://www.scitepress.org/Documents/2018/65977/, INSTICC, SciTePress, 2018, pp. 501–506, ISBN: 978-989-758-275-2. DOI: 10.5220/0006597705010506.

[12] S. Wang, Z. Bao, and J. E, “Armor: A benchmark for meta-evaluation of artificial music,” MM ’21, pp. 5583–5590, 2021, Accessed: October 15, 2023. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3474085.3475700. DOI: 10.1145/3474085.3475700. [Online]. Available: https://doi.org/10.1145/3474085.3475700.

[13] O. Mogren, “C-rnn-gan: Continuous recurrent neural networks with adversarial training,” 2016, Accessed: Nov 15, 2023. [Online]. Available: https://arxiv.org/abs/1611.09904. DOI: 10.48550/arXiv.1611.09904. [Online]. Available: https://arxiv.org/abs/1611.09904.