10-701 Music Generation Project Report

Ketong Chen
Chemical Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
ketongc@andrew.cmu.edu

Yucheng Dai
Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
yuchengd@andrew.cmu.edu

Zilin Ren
Statistics & Data Science
Carnegie Mellon University
Pittsburgh, PA 15213
zilinr@andrew.cmu.edu

Abstract

During music generation process, there are a few characteristics of music that need to be addressed. Music is closely related to time; music pieces are composed of different instruments; music pieces usually consist of chords. To include these characteristics, we applied Generative Adversarial Networks with Convolutional Neural Network models to take in Lakh Pianoroll Dataset as training data and generate a piece of music from scratch without any human input. Based on the baseline model, we have performed model structure modification and parameter tuning. The improved models achieved lower empty bar ratio, lower number of pitches classes per bar, and higher polyphonic rate.

1 Introduction

Music generation has always been a popular topic. It is inspiring to create our own melodies as expressions of feelings. Nowadays, with the aid of advanced technologies, people do not have to compose their own music any more - computers have the potential to do it all. There is infinite room for music generation due to music's huge domain, including different properties such as pitch, harmony, rests and chords. In our project, we will focus on a few specific genre of music and use deep learning models as powerful tool for music generation. We aim to introduce training dataset and then generate melodic music as output. To achieve our goal, we decided to build up from an existing Generative Adversarial Network (GAN) music generation model and make further improvements. In this project report, we will first cover the details of the baseline model and then propose a few new ideas and improvements. We will next introduce the implementations of our own model, followed by comparison and discussion of the results.

2 Dataset

In this project, we used the cleansed version of a piano-roll dataset which is originally derived from the Lakh MIDI Dataset (LMD)[1]. LMD is a large collection of 176,581 unique MIDI files. Based on these MIDI files, there is an existing multitrack pianoroll dataset, Lakh Pianoroll Dataset (LPD), where each MIDI file is processed with python library pretty midi and transformed to series of 128 by 96 pianoroll object. The data used in our project is further cleansed with the following procedures:

- Reduce the sparsity of pianoroll dataset. For most music files, different tracks usually have different sparsity. For example, there may only a few notes and many empty bars for the entire harp track, while the piano track is filled with notes. To balance tracks and make improve the learning process, each multi-track pianoroll is compressed into 5 tracks: drum, piano, bass, guitar, strings.
- Parse the music into meaningful phrases. To ensure the quality of training, 4 bars are considered as a phrase and longer pieces are parsed into slices of multiple phrases.

Figure 1 shows an example visualization of the cleansed dataset. The five rows represent the five instruments respectively. Each colored bar in the diagram indicates a note, where the vertical axis represents its pitch and length along the horizontal axis represents duration of the note.

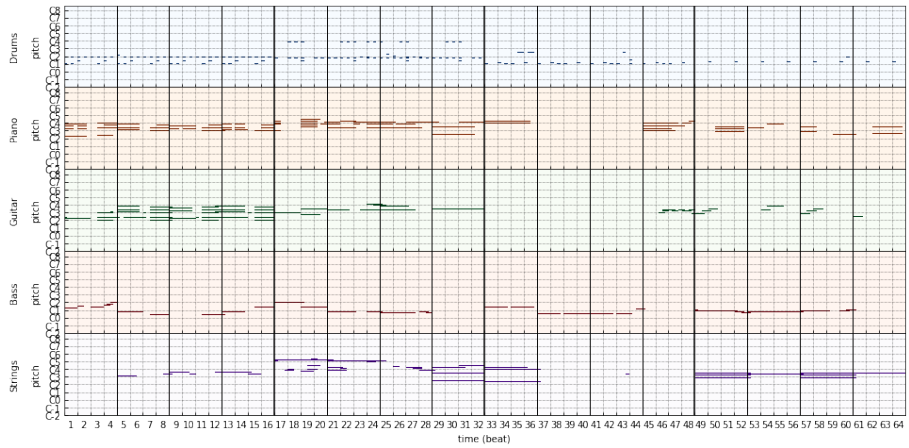

Figure 1: Visualization of cleansed LPD data

For convenience, we chose to use the MIDI format of music files over the raw mp3 format. Specifically, MIDI files allow us to separate the music into different channels and study each one of those individually. Besides, notes in MIDI files are objects that are ready to be processed and analyzed in the following steps.

3 Previous Work

3.1 General Idea

We used the MuseGAN project[2] as our baseline model, which uses GAN [3] for symbolic music generation and accompaniment. In their work, three important features of music generation were highlighted. First of all, music is closely related to time. In order to model the hierarchical character of music, a temporal structure important for music generation tasks. Secondly, music files are usually composed of different tracks, i.e. different musical instrument. Using MIDI as initial dataset, one can separate the tracks easily and conduct following analysis. Lastly, music always come in the form of chords, where multiple notes may be combined at the same time steps. Therefore, chronological orders cannot be assumed in the case of music generation.

According to the features of music mentioned above, bars instead of individual notes were regarded as basic units of the model input. Furthermore, a transposed convolutional neural network (CNN) was used because CNN is good at finding local, translational-invariant patterns. The input of CNN has a fixed dimension of 4(number of bars) * 96(number of time steps) * 84(number of note candidates) * 5(number of tracks).

A GAN-based model for multi-track was applied here. In GANs, there are two networks: a generator and a discriminator. The generator takes in randomly generated noise z and maps it to the data space. The discriminator is trained to tell the authenticity of a piece of music, whether it is real or generated by the generator, while the generator on the other hand tries to imitate real music as much

as possible to fool the discriminator. An additional gradient penalty is then used for the objective function as a regularization step. Specifically, it helps to stablize the magnitude of the gradients that the discriminator provides to the generator, and thus help stablize the training of the generator.

Three different models, namely, the jamming model, the composer model, and the hybrid model, are designed and compared as shown in Figure 2. Firstly, jamming model imitates the composition process of a group of musicians playing different instruments through improvising music without predefined arrangement. It assigns each track its own set of generator and discriminator. Secondly, the composer model assumes the behaviour of a single composer who arranges tracks by harmonic and instrumental knowledge. It has only one shared generator and discriminator for all tracks. Hybrid model combines the idea of jamming and composer model. After training and evaluation through user study, it is found that the hybrid model is preferred by "pro users" in music.

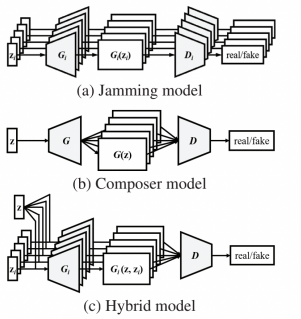

Figure 2: Jamming, Composer, and Hybrid models

3.2 Baseline Model

The baseline model is a modified composer model as shown in Figure 3. It defined a 3d-transposed-convolution layer with a batch normalization layer as one generation block. The generator contained 6 generator blocks where the first 4 blocks take in the stacked data, and the 5th and 6th block split the five tracks and process them separately. The generator takes in one shared latent variable z, and output a 5-track stacked piece of music.

The baseline discriminator model consisted of a list of discriminator blocks, each defined as a 3d-convolution layer and a layer normalization. There are a total of 7 discriminator blocks and a linear layer. The first two layers constructed as list of 5 discriminator blocks, each for one instrument track.

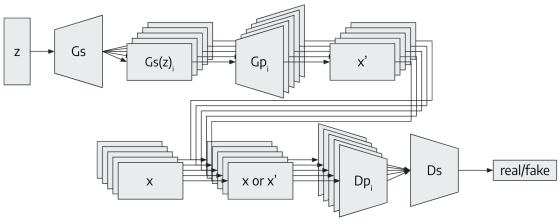

Figure 3: Baseline Model structure

The baseline model is trained on the Wasserstein GAN loss function (Equation 1) for generator and discriminator respectively.

$$
\begin{aligned}
L_D &= -\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) + \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)})) \\
L_G &= -\frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))
\end{aligned}
\tag{1}
$$

Specifically, these two loss functions follow the initial purpose of GAN. On one hand, the discriminator loss function aims to maximize real music outputs to be as close to 1 as possible and minimize all generated music outputs to be near 0. On the other hand, the generator tries to generate music that are similar to real music, so the generator loss function wants to maximize the discriminator output of generator results. The generator and discriminator loss from the baseline model is shown in Figure 4. Generator's loss has a range [-10, 15], and discriminator has narrower range of [0,15].

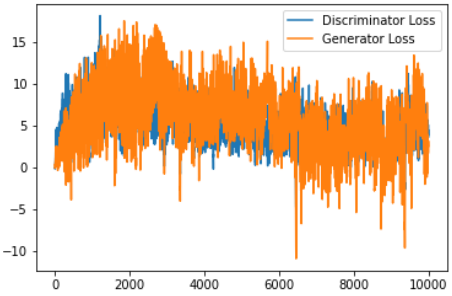

Figure 4: Baseline Model Loss

4 Methods

4.1 Data processing

In order to align with the baseline model, we use the same way to process the data. The baseline model collects a total of 26,154 samples from 5 tracks (drum, piano, bass, guitar, strings) and 13 genres (Blues, Country, Electronic, Folk, International, Jazz, Latin, New Age, Pop Rock, Rap, Reggae, R&B, and Vocal) of 7,323 songs in the data set. For each real training sample, it extracted a 4-measure piece, to form a 64x72 matrix (64 beats and 72 pitches). The final data shape is (26,154, 5, 64, 72) where the 5 tracks are stacked together.

4.2 Model Training

Our main approaches for model improvement includes:

- Add or delete layers for Generator and Discriminator.
- Change hyper-parameters.
- Change the loss function of Generator and Discriminator, and see whether the change will bring some systematic influence on the music piece.
- Change the structure of stacking and separating the tracks and see whether it will affect the consonance of the tracks. For example we can do stacked - separated - stacked or separated - stacked compared to currently what we have right now as first stacked and then separated.

4.2.1 Layer and parameter change

We have performed several changes to the baseline model, using individual or combined methods mentioned above. Firstly, we tried to increase model layers from 6 to 7 to capture more correlations through increasing complexity of the neural network. In addition, we have changed one of the parameters, number of measured per sample from 4 to 8. The reason for the parameter change is that the baseline training results show frequent repetition, and taking longer training samples would potentially reduce the repeating patterns.

4.2.2 Training with different loss function

For further improvement, we varied the loss functions[5][6]. Firstly, the Wasserstein Loss[7] into Binary Cross-Entropy(BCE) Loss. For doing so, we add a sigmoid layer at the end of the discriminator

to make the output in the range of [0,1] for indicating fake, which is indicated by 0, or real, which is indicated by 1. Then, the loss function becomes:

$$\begin{aligned} L_D &= log(D(X)) + log(1 - D(G(Z))) \\ L_G &= log(1 - D(G(Z))) \end{aligned} \tag{2}$$

In addition, we also tried Mean Square Error loss as the loss function for the discriminator and generator. This loss function (Equation 3) is used in the same setting as BCE loss, where the output of discriminator is in the range of [0,1].

$$\begin{aligned} L_D &= \frac{1}{N}\sum_{i=0}^{N}(1 - D(X))^2 + \sum_{i=0}^{N}(0 - D(G(Z))^2 \\ L_G &= \frac{1}{N}\sum_{i=0}^{N}(1 - D(G(Z)))^2 \end{aligned} \tag{3}$$

4.2.3 Changing stacked structures

Another experiment made is to change the generator structure. Originally, the generator consists of 1 shared generator section and then 5 private sections. Now they are repeated once, i.e. 1 shared generator section, 5 private sections, then another shared section, and at last 5 other private sections. Intuitively, this allows the composer to take each track's result into consideration, and each track can also modify their compositions based on the modified shared rhythm.

Specifically, the first shared generator section consists of 3 generator blocks. Then the 5 private sections only consist of 1 block each. The second shared generator section consists of 4 generator blocks. At last, the other 5 private sections consist of 1 block each.

4.3 Evaluation Metrics

To assess the music generation results, we have used several evaluation scores[2]. Evaluations used to assess the generated music include ratio of empty bars (EB), number of used pitch classes per bar (UPC), and polyphonic rate (PR). Value of EB can be used to show whether splitting into 5 tracks is appropriate. Value of UPC for individual track can be used to determine whether the track is played as the main melody or accompaniment. Value of PR measures the ratio of the number of time steps where multiple pitches are on to the total number of time steps, which indicates how many potential chords there are in the generate music.

5 Results

Music generation results improved when number of measures per sample was increased. After 30000 steps of training the modified model, the visualization of the results in shown below, compared to the baseline model results. From the diagram, the frequency of repeats in music pattern went down, as shown through comparisons between lengths of colored bars in Figure 5.

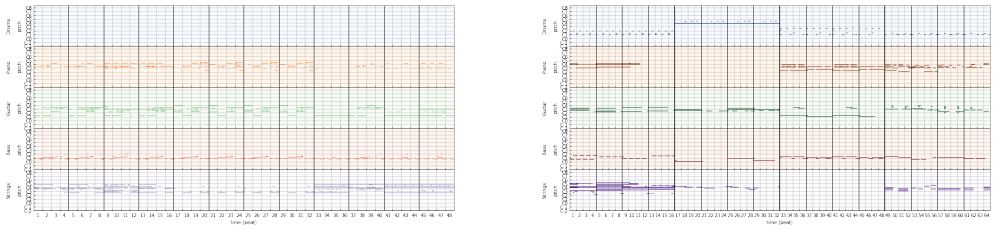

Figure 5: Baseline result(left), Increasing number of measures per sample shows less repetitions(right)

Model results improved when loss function was changed. The model with BCE loss for 30000 steps and resulted in the loss function on the left subplot in Figure 6. We can see that the generator have in

general higher losses than discriminator, and the loss value ranged in [0,5]. When examining the midi file of the music generated, we discovered that the music piece mainly contains 1/16 notes, and the patterns are repetitive for very short period. Hence, the music sounds incoherent, and not melodic.

The model with MSE loss for 30000 steps are shown in the right subplot in Figure 6. It is clear that the generator loss ranges in about [0.5, 2.00] while the discriminator loss ranges in [0, 1].

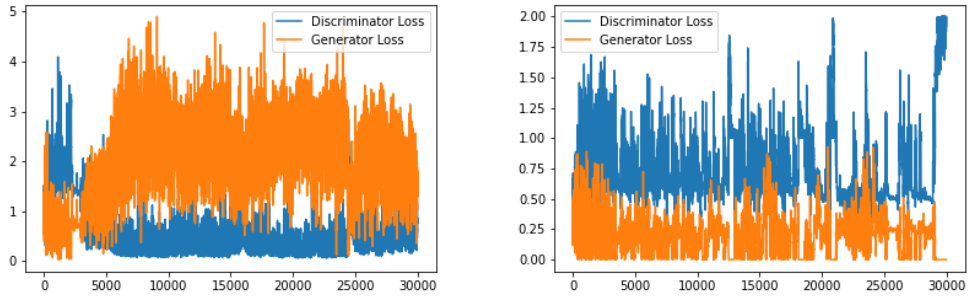

Figure 6: Model with BCE loss(left) and MSE loss(right)

The assessment results for both baseline and improved results are shown in Table 1. EB has shown general decrease in the improved models, especially the Measures-Increased model, which indicated that more bars are filled with music, instead of left blank. Lower UPC usually represent the melody-like track and higher pitch classes represent the chord-like track. UPC remains a minimum around 3 for either bass or guitar, which shows a consistent trend for the melody track. Higher PR shows a potential of chord-like composition style for each individual track. There is a general increase in PR for improved models, especially in BCE Loss model, indicating that the generated music has shown increasing complexity and more chords. Also, we can see that different models produces different performance of each instrument. It seems that the Measures Increased model will produce a good Guitar track, while the BCE loss model and Structure Changed Model will produce good Piano tracks.

6 Conclusion and Discussion

Music generation is a famous problem, as it raises our discussion of whether computers can do creative work. In this project, we discussed the possibility to generate music without a prior knowledge. Besides the baseline model, we constructed 4 models to compare the performance. The first one increased the sample measure length to include more complicated rhythm, the second and the third one changed the loss function, while the last one changed the structure of the generator. All these approaches aim to solve different aspects. They worked as intended, but they showed their drawbacks as well. Overall, most of them produce satisfying results because they mostly sound harmonic. We can combine several of the approaches in the future to further improve the quality of the music generated.

There are also possibilities of improvement in future work. We can perform more basic hyperparameter tuning, such as on numbers of measures per sample, channel size for generators and discriminators, and learning rate adjustment. From a statistical point of view, each model performs differently in each instrument. From table 1, we can see that some models almost completely ignored some instruments. In the future, it is reasonable to use different models for different tracks, and combine them later for overall harmony. In addition, due to temporal essense of music, extending the basic model from CNN to RNN or LSTM is also worth trying. There are a lot of opportunities to expand this work, and would be promising to have better outcomes in the future.

Table 1: Evaluation Metrics

Model	Instrument	Empty Bar Ratio	Pitch classes per bar	Polyphonic Rate
Baseline Model	Drums	0.35416	NA	0.0
	Piano	0.70573	4	0.07526
	Guitar	1.0	5	0.0
	Bass	0.90104	3	0.02778
	String	0.87760	4	0.0
Measures Increased Model	Drums	**0.13307**	NA	0.00846
	Piano	0.56459	6	0.18750
	Guitar	**0.16150**	4	**0.48177**
	Bass	0.75194	4	0.02778
	String	**0.46253**	5	0.66211
BCE Loss Model	Drums	0.35417	NA	0.0
	Piano	0.98177	4	**0.83398**
	Guitar	0.65885	3	0.19661
	Bass	0.82813	3	**0.27669**
	String	1.0	4	**0.80404**
MSE Loss Model	Drums	0.64583	NA	**0.10026**
	Piano	0.56250	7	0.02018
	Guitar	0.93750	3	0.53255
	Bass	**0.61458**	7	0.07812
	String	0.78385	5	0.27539
Structure Changed Model	Drums	0.21354	NA	0.0
	Piano	**0.47917**	4	0.001302
	Guitar	0.51042	5	0.05469
	Bass	0.85417	4	0.23242
	String	0.74740	6	0.45117

References

[1] The lakh MIDI dataset v0.1. (n.d.).
Retrieved November 21, 2021, from https://colinraffel.com/projects/lmd/.

[2] Dong, H., Hsiao, W., Yang, L., & Yang, Y. (2018). MuseGAN: Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment. AAAI.

[3] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Bengio, Y. (2014). Generative adversarial nets. In Advances in neural information processing systems (pp. 2672–2680).

[4] Wasserstein Gan implementation in tensorflow and pytorch. Wasserstein GAN implementation in TensorFlow and Pytorch - Agustinus Kristiadi's Blog. (n.d.).
Retrieved December 8, 2021, from https://agustinus.kristia.de/techblog/2017/02/04/wasserstein-gan/.

[5] Brownlee, J. (2020, January 9). How to code the GAN training algorithm and loss functions. Machine Learning Mastery.
Retrieved December 8, 2021, from https://machinelearningmastery.com/how-to-code-the-generative-adversarial-network-training-algorithm-and-loss-functions/.

[6] Harshit DwivediFounder and CEO of AfterShoot, Dwivedi, H., AfterShoot, F. and C. E. O. of, & on, F. me. (2021, November 12). Understanding gan loss functions. neptune.ai.
Retrieved December 8, 2021, from https://neptune.ai/blog/gan-loss-functions.

[7] Brownlee, J. (2019, July 12). How to implement Wasserstein loss for generative Adversarial Networks. Machine Learning Mastery.
Retrieved December 9, 2021, from https://machinelearningmastery.com/how-to-implement-wasserstein-loss-for-generative-adversarial-networks/.