CIS 522 – Final Project – Technical Report

PopMusicNet

April 2022

Team Members:

- Guanwen Qiu; pennkey: guanwenq; Email: `guanwenq@seas.upenn.edu`
- Zhihao Yan; pennkey: zhihyan; Email: `zhihyan@seas.upenn.edu`
- Mingxuan Zhang; pennkey: mzhangz; Email: `mzhangz@seas.upenn.edu`

Abstract

Music generation and classification using deep learning techniques have been a topic of interest for the past two decades. This project aims to create novel neural network architectures to classify and even generate new music using 20,000 pianoroll samples of different genres from the Lakh Piano Dataset [Don+17] [Lib70], a popular benchmark dataset for recent music generation tasks. We trained a simple feedforward 2D Convolutional Neural Network for classifying music genres as our first step. The model achieves 60% overall accuracy with ten labels when the test data is distributed evenly across genres. Our main body of the project is about generating a piece of music given a prelude. We use raw pianorolls directly to learn the syntax and semantics of music. We trained two different models to learn music generation and analyze their performance. GRU was our first attempt, but it doe not perform well. The failure of this model provides us with much insight and inspiration to develop a successful music generation ensemble model, which we name VAE-RNN. We hope that people with different music levels can use our projects to make music. For people who have absolutely no composition experience, this model could generate melody and accompaniments just for fun and satisfy their curiosity about making their own music. For an amateur musician, the model could be used to generate accompaniment for a provided piano melody with bass, drums, strings, and guitar. Musicians can also use the model to do style transfer or highly customized music generation.

1 Introduction

It can be foreseen that an intelligent music generation will bring a huge impact on our society and the music industry. For example, NetMusic has developed a fully functional system for people with different levels of music skills to compose, arrange and mix music. Amateur musicians could save the money they would have to pay to the arranger and mixer in order to complete their music.

Music learning proves to be a challenging task because of the two characteristic natures of music: horizontally, music data is continuous and resemble several streams of a signal whose strength(volume) and frequency(pitch) change drastically across time. Vertically, music consists of multiple instruments that can play at different frequencies and are perceived by humans as chords. A common approach to do music generation is through tokenization and formulating it to become a text generation problem where we can apply natural language processing techniques and text generation models that we are more familiar with. However, for our project, we choose to explore a completely different methodology. Instead of using tokenization, we keep music in its original format throughout the training and generation process. In this way, we are able to keep all music information such as tempo, pitches, chords, and duration intact and have them expressed in the most natural form.

For the music classification part, we design a highly regularized Convolution Neural Network to classify the genre of a song into ten labels. We then unplug all the non-linear layers and regularization such as dropout, batch normalization, and Relu to make it a linear model. We then compare the results of these two models.

For the music generation part, we develop two models. One uses Gated Recurrent Unit(GRU) as its main component, and the second is an ensemble deep learning architecture where we use 5 Variational Auto Encoders (VAE) to do feature extraction and 5 Recurrent Neural Networks (RNN) to do music generation. Our result is promising. The VAE-RNN model allows us to generate music with great flexibility and variety. The music generated shows both interesting long-term and short-term developments.

2 Related Work

Most music data are provided either as midi files where music information are made discrete in the first place or recorded audio files which contain complex audio signal. To generate music using midi data, most researchers reduce the task to text generation by tokenizing music information at each time step into a text string that contains pitch and duration information. Similar to how attention networks and transformers have been used in recent improvements in NLP, attempts have been made to utilize transformers in music production. Shaw [Sha19] designed MusicAutobot, a multi-task engine that can both generate new music and construct harmony conditional on other instruments using a combination of BERT, Transformer-XL, and Seq2Seq. DeepMind demonstrated the usefulness of WaveNet, which uses dilated convolutions to generate raw audio in 2016. Dong expanded on the GAN concept in 2017 with MuseGAN, which employs several generators to create synthetic multi-instrument music that takes into account instrument dependencies. For higher training stability, [Don+17] used the Wasserstein-GAN with Gradient Penalty (WGAN-GP). Following the same ideology, Yang [YCY17] developed MidiNet, which uses Deep Convolutional Generative Adversarial Networks (DCGANs) to generate multi-instrument music sequences based on midi files. However, it is worth noticing that both of these methods use only one bar of music to generate a single bar melody. The VAE-RNN model we developed is much more capable of generating music with long-term development when compared to the previous methods. Also, it is able to generate five tracks of different instruments played at the same time instead of a single melody line.

3 Dataset and Features

We use the lpd-cleansed 5-track dataset from the Lakh Pianoroll Dataset which is derived from the Lakh MIDI Dataset by combining different tracks into five general instrumental tracks:drums,piano,strings,guitar,bass and each track has 128 pitches. It contains 21,425 multitrack pianorolls collected from lpd-matched dataset. Each track of the multitrack pianoroll file contains the information about what pitches are hit at each time step. The resolution of this dataset is 24, which means one beat in the original music is divided into 24 time steps. See Figure 1 for an example of a typical pianoroll.

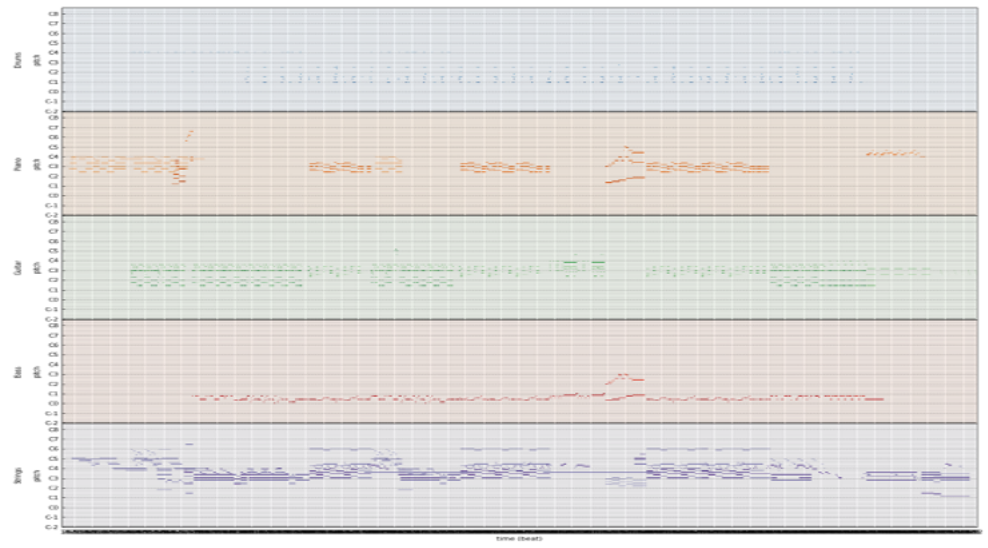


Figure 1: Example of a Pianoroll

We convert the music information to a matrix. We use velocity or volume as our unit of the matrix. For instance: [1,0,0,3,0,0,0] represents Pitch C and F hit at the same time with volumes 1 and 3, respectively. We also normalize the volume of each track individually during training. The training data thus has the following shape (track numbers, time sequence, pitches). Noted that we will use the term midi and pianoroll, note and pitch interchangeably for the rest of the paper. It is also worth mentioning that the dataset is quite dirty in the sense it contains pianorolls of low quality and wrong genre labels that are hard to filter.

4 Methodology

We first develop a linear CNN to classify music. We then turn the model into a non linear version by adding regularization and activation function to it. The results are then analyzed. There are mainly two approaches to generating new music using midis. First, we can use raw midi directly to learn the syntax and semantics of music. The second approach is to tokenize all notes hit (chord) and duration at a time step into text, and apply the text generation model. We choose to explore the first methodology because of the following reasons:

- We think it is a more representative form of music
- Enable more flexible manipulation of the structure and generation process

4.1 Linear CNN

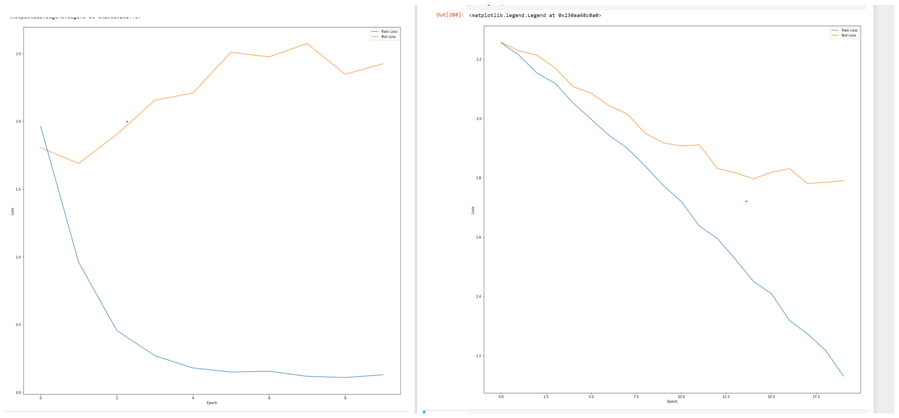
The original dataset did not come with genre labels. We have to retrieve the genre info by using other music genre datasets. The genre dataset we choose to use is MSD AllMusic Top Genre Dataset (TopMAGD) provided in the Million Song Dataset Benchmarks [Doe16] [SMR12] [Ber+11]. We are able to retrieve the genre label for around 7000 pianorolls. We then calculate the average size of time steps of our dataset, which is 5000 time steps for each song. Thus each song is padded to 5000 or clipped to 5000 time steps during the data pre-processing step.

Our labels are heavily imbalanced, with 4345 samples of Pop-rock music and below 100 samples for jazz, blues, and new-age. Also, after some manual checking, we found that many labels are not accurate, and there is no standard, consistent definition of each music genre. The decision boundaries between different genres are often very ambiguous and highly subjective to a listener. For example, for most of us, it is hard to tell blues from jazz, electronic from pop, and country from folk. To alleviate such problems, we merge similar labels into one general label such as blues, jazz, and reggae. We reduce the number of labels from 15 to 10, and our final label set contains Country, Electronic, International, Jazz, Latin, New-Age, Rap, RnB, Rock, and Vocal.

The model comprises of five convolution filters which are long in width(time step) and x-stride and short in height(pitches) and y-stride in order to reduce the size of the time step but keep more pitches information. This design is justified by the fact that most pitches do not change between a few time steps of a song. We have also decided to train the model with only the bass, piano, and drums tracks because a large number of our samples do not contain guitar and string tracks. To address the class imbalance problem, we assign each class a different weight calculate as

$$W = N_{total}/N_{genre}$$

where N total is the total number of samples and N genre is the number of samples of a specific genre. Figure 2 shows the train and test loss curve of the linear and non-linear models. It can be seen that the linear model starts to overfit at the early stage of the training, while the non-linear variant shows a much smoother and steadily decreased loss curve for both training and testing loss. This indicates that the non-linear model is able to learn more general features from the training data. Figure 3 show the prediction heat map for linear and non-linear variant, respectively. The non-linear variant is able to make more accurate predictions than the linear variant in nearly all genres except the rock genre.



```
Out[288]: <matplotlib.legend.Legend at 0x230aa68c0a0>
```

Figure 2: Left: Linear CNN Loss Curve Right: Non Linear CNN Loss Curve

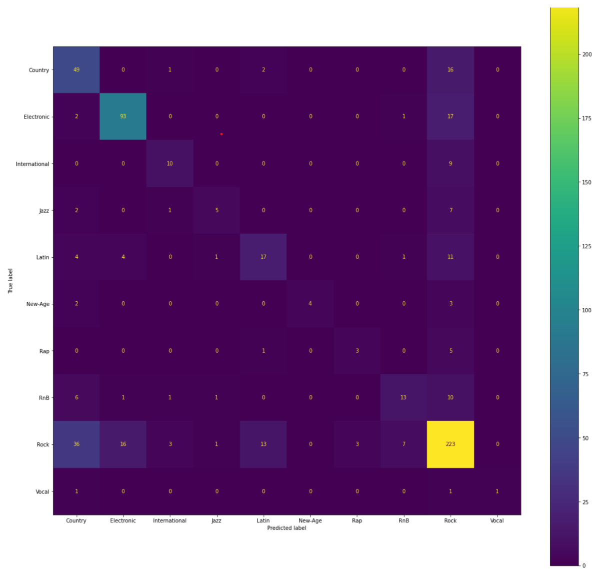


Figure 3: Linear CNN Loss Curve

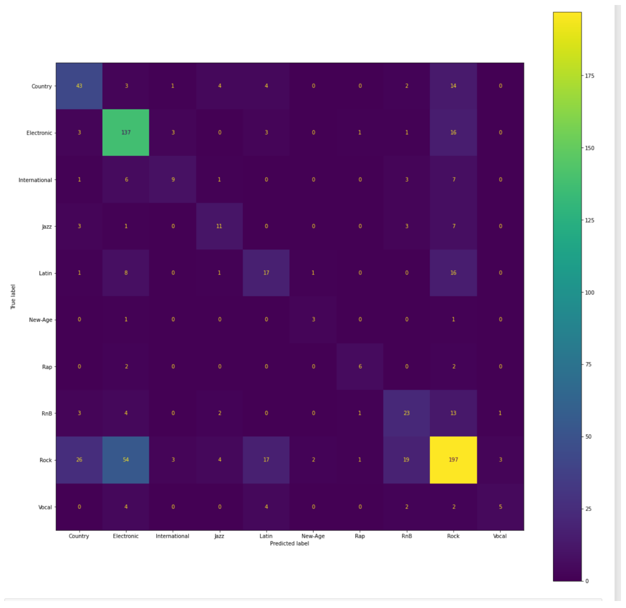


Figure 4: Non Linear CNN Loss Curve

4.2 Generation - GRU

The first attempt we made to generate music was through GRU. The problem is to predict the pitches that should sound at the next time step, which is quite similar to the RNN-based language models used in natural language processing. We chose the Gated Recurrent Unit (GRU) over the vanilla RNN for implementation since it has a greater ability to keep long-term dependencies. At each time step, the model will receive input and hidden state from the last time step and output pitches that should sound at the current time step.

All valid pianorolls are concatenated in the time steps axis into an enormously long matrix. We then partition it along the time axis in sequence, preserving its original order. For our partition, the window is set to 48, which is exactly 4 beats or a bar in music. Using sequential partitioning, we are able to reuse hidden states at previous timestep throughout the training process. We also binarize the matrix, so all hit pitches are marked as 1, which allows us to formulate the question as predicting the probability of each pitch sounding at the next time step using binary cross-entropy loss. The loss function at a given time step t for our model is:

$\mathrm{L}_t = -1/N \sum_{i=1}^{N} y_i * log(p(y_i)) + (1 - y_i) * log(1 - p(y_i))$

The training phrase has the below architecture as shown in Figure 4:

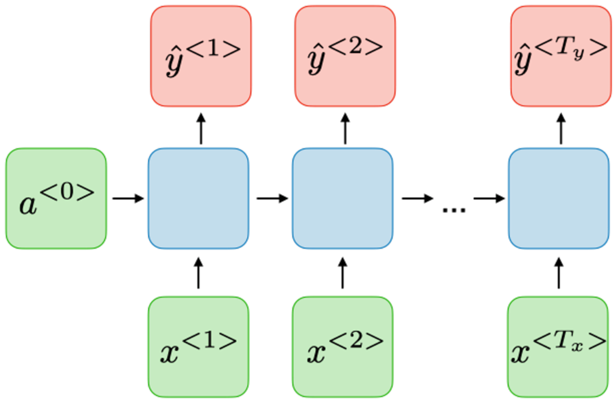


Figure 5: Y represent pitches hit at next time step. Essentially, $\mathrm{Y}_t = X_t + 1$

During the generation step, we feed the model a prelude(sequence of time steps) to warm up its hidden state and then iteratively generate music at the next time step. The results are then collected and binarized by using a threshold value in the range (0,1). An illustration of this process is shown in Figure 5.

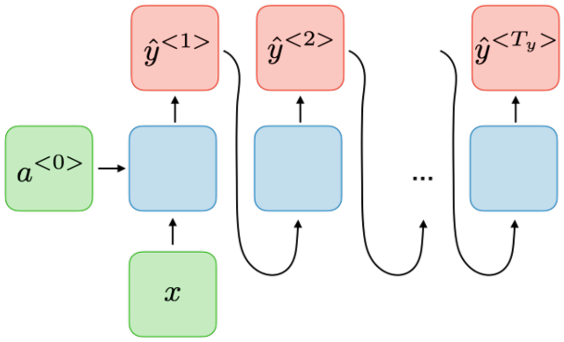


Figure 6: GRU generation phrase

Figure 7 shows the loss curves of this model, and Figure 8 shows the piano roll plot of a piece of music combined given prelude and generated music. It can be seen that the model fails to make meaningful music, and the notes being played are mostly stationary across time. The loss curve give us a hint that the model is not able to perform learning even for the training data otherwise we should see overfit effect. We initially thought this is because the GRU model we develop do not have enough complexity and thus add more layers and hidden size to it. However, no matter how many layers or hidden size we add, the loss curves stay the same pattern. This illustrate that GRU is inherently not able to perform learning on the raw pianoroll.

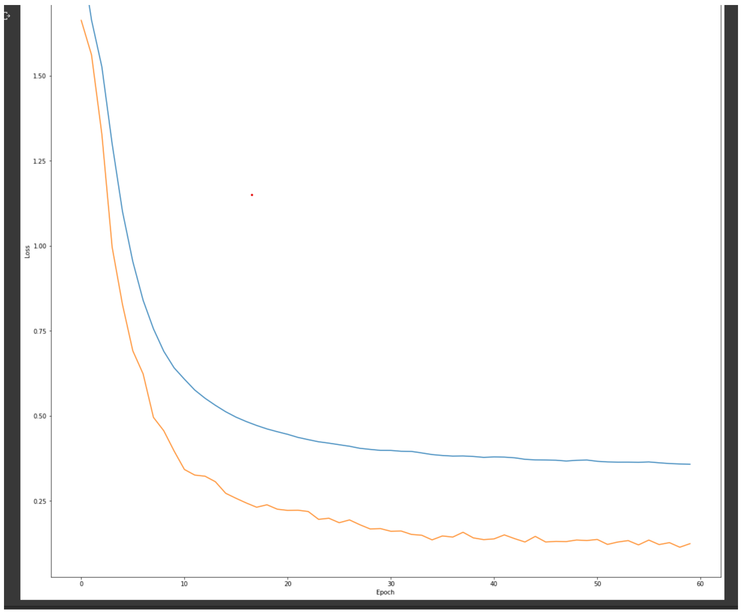


Figure 7: GRU Loss

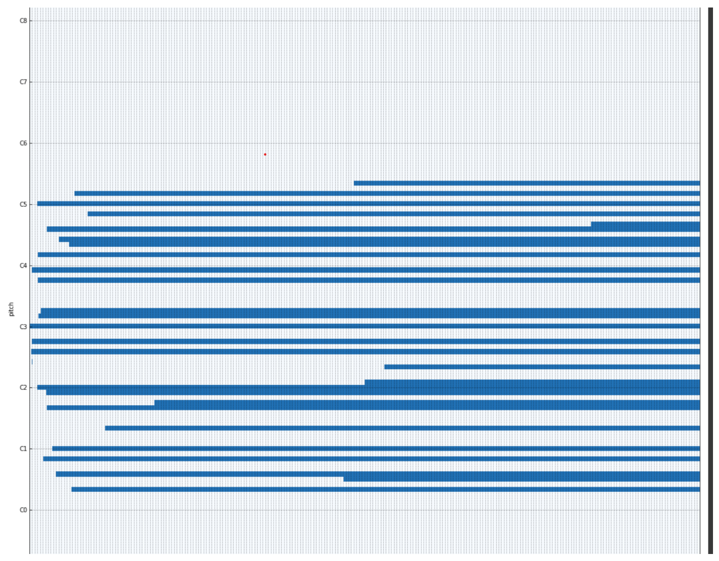


Figure 8: Pianoroll Generated Using GRU

We attribute the failure to four reasons. First, window size or training sequence length at 48 is not enough for the model to perform actual learning of the music. Second, music is inherently different from text generation because, at each time step, pitches hit are either 0 or 1. Third, for most time steps, sounding notes stay the same, which makes the model biased toward keeping note sounding at this time step into the next time step. Fourth, the prediction error during generation increases exponentially since if the model makes a bad prediction, it will continue to generated predictions based on that bad one.

To solve such problems, we design a new, novel deep learning architecture to generate music.

4.3 VAE-RNN

In summary, VAE-RNN is a deep learning architecture that is able to perform long-term music generation. It combines the advantages of VAE (good at learning and reconstructing regional information) and RNN (good at making predictions based on previous observation).

A variational autoencoder (VAE) is an autoencoder with regularized training to ensure that the latent space has good properties for generative processes. The two most important features are continuity and completeness[[**5**]]. Continuity means close points in latent space should bed decoded to similar things that are comparable to some extent. Completeness means a point sampled from latent space should decode to meaningful material.

The training process for our model has the following steps. Assume the current time step as t:

- Train VAE for each instrument
- Run the encoder part of the VAE through all the pianorolls we have for that instruments and get a long (timestep, latent size) dataset contains learned latent variables of a track at each time step.
- Train a RNN to predict latent features of the piano track at t given latent features up to t-1
- Train a RNN for each track of other instruments to predict its latent variable at t given the history of latent features of Piano up to t.

The model architecture is illustrated below:

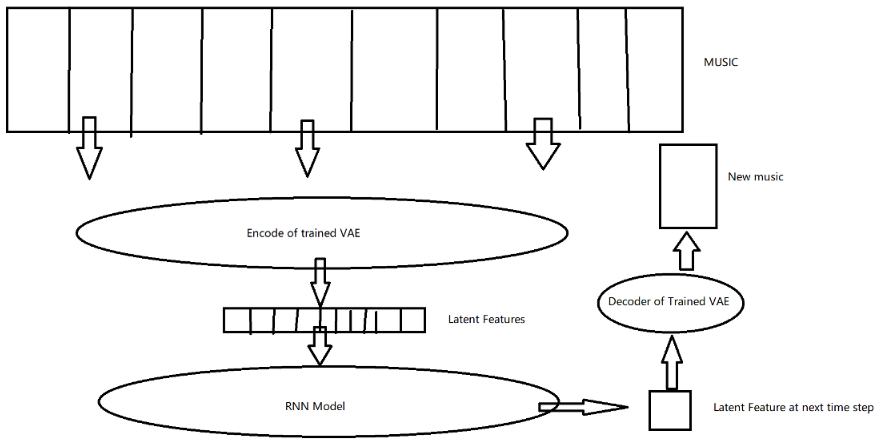


Figure 9: VAE-RNN architecture

The generation process for our model has the following steps:

- Given a new piano track with various time span
- Use the trained piano VAE to encode that track into a series of latent features
- Use the trained piano RNN to predict the latent feature of piano at t+1 given the history of piano latent features up to t
- Use the trained RNNs for other instruments to predict their latent features at t+1 given the history of piano latent features up to t+1

- use the trained VAE for each instrument to decode the latent features and combine the generated piano rolls to a new music

The model architecture is illustrated below in Figure 9:

We use VAE as the feature extraction model because its latent space has nice properties that can be used latter to do style transfer through interpolation. For example, if we want to transfer a piece of music at a given time frame to another style, we could first learn the latent variables for these two time-frames and perform the interpolation.For our architecture we use a 512 latent size and 128x128 window for our VAE.

Using a 128x128 VAE we are able to condense a average pianoroll at 5000 time steps to around 40 time steps which could be fed into a RNN in one pass. This means the RNN is now able to learn the long term development of a average long song in full extent.

5 Results

Our attempt to do music genre classification task and generation using midi data proves to be successful. We achieve a reasonable accuracy for each genre given that our dataset is imbalanced heavily and label definition are ambiguous.The music generated using VAE-RNN is quite satisfactory. The training and testing loss of piano VAE and RNN is shown in Figure 10.

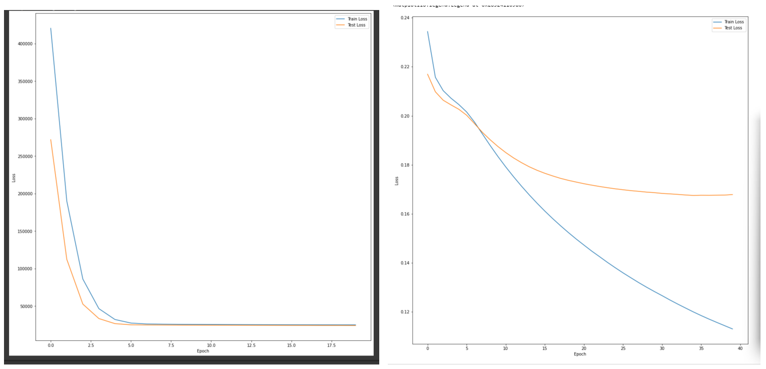


Figure 10: Left: Piano VAE Loss Curve Right: Piano RNN Loss Curve

The pianoroll plot of a piece of music generated using our VAE-RNN is shown in figure 11. It can be seen that the model is able to generate meaningful and please to hear music.And for most of the time and music is in tune. Another significant part of this model is that all other instruments besides piano can be seem as accompaniment instrument because the input they are given is the history of latent features of piano. It can be perceived the music generated has some interesting dynamic and interaction happen between tracks because of this important nature of our model. You can hear all of the music generated using in this paper and more on https://github.com/NayutaQiu?tab=repositories

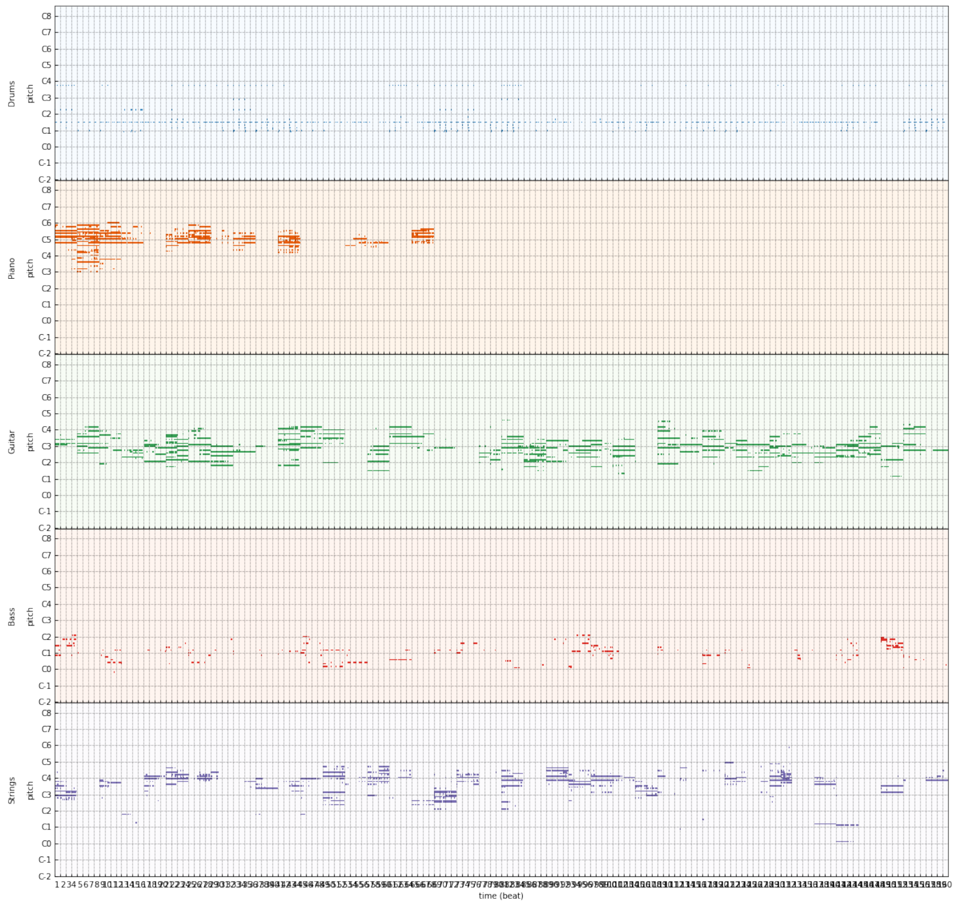


Figure 11: Generated Pianoroll Sample

6 Discussion

6.1 Findings

Although most works done in music generation use either the tokenization approach or GAN, they all failed to capture the long-term development or dependency of music because of the limitation imposed on the RNN, namely the gradient vanishing problem. Lots of attention and effort has been spent to develop RNN that could learn longer dependency, but few prove to be successful. For our project, we are able to overcome this hard problem by transforming our data into a much more compact form and thus aid the RNN in performing actual long-term learning. We believe such a methodology can be applied to other domains involving time-series data.

6.2 Limitations and Ethical Considerations

Our models are far from optimal because our time and computation resources are very limited. Many possible improvements could be made, such as tuning different hyperparameters and adding more layers to it. Also, the music generation process currently requires a great effort of manual tuning in order to filter noise or unwanted notes from the music generated.

Our intention is for our model to be only for recreational uses. People can use it to generate music based on their favorite songs, enjoy the generated music and gain inspiration from it. However, as researchers continue to work on better and better music generation models, how to protect musicians from being harmed by such models is also important. For instance, a singer can generate a piece of music from a very popular song. And the generated music is somewhat different from the original but keeps significant features. The singer can claim the music to be his own creation. And that might harm the originality of musicians.

Another concern is that though our training set contains a large variety of songs, it may not be representative of all music. Some less popular categories might not exist in the data. Thus we might fail to generate good music for those categories.

6.3 Future Research Directions

One of the most significant improvements that could be made to our music generation model is to tune all training music into the same key. However, there is no simple way to identify the key to a piece of music. But we think it is possible to have our model learn to identify the key of a piece. Currently, the music generated, especially the piano part, can be out of tune for a relatively long period of time which is rare to see in pop music. We believe that by tuning all training music into the same key, the model will be able to produce much more pleasing music. Also, we could use a different RNN model to replace the GRU part, including LSTM and transformer, to see which one works better. There is also a need to find right threshold to filter noise from good notes in a more systematic way so user do not have to manually tune the generated music endlessly.

7 Conclusions

We are quite satisfied with the result we are able to get, given that our time and computation resource is so limited. We also solved all kinds of difficulties met in the data pre-processing and model training stages. The final VAE-CNN model we developed comprised 10 models, and it is the most fruitful contribution we made, which we have not seen in the past paper. We are hoping we can publish a paper on this subject and are willing to refine this model in the future.

8 References

References

[Ber+11] Thierry Bertin-Mahieux et al. “The Million Song Dataset.” In: Jan. 2011, pp. 591–596.

[Doe16] Carl Doersch. *Tutorial on Variational Autoencoders.* 2016. DOI: 10.48550/ARXIV.1606.05908. URL: https://arxiv.org/abs/1606.05908.

[Don+17] Hao-Wen Dong et al. “MuseGAN: Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment”. In: (Sept. 2017).

[Lib70] Columbia University Libraries. *Learning-based methods for comparing sequences, with applications to audio-to-MIDI alignment and matching.* Jan. 1970. URL: https://academiccommons.columbia.edu/doi/10.7916/D8N58MHV.

[SMR12] Alexander Schindler, Rudolf Mayer, and Andreas Rauber. “Facilitating Comprehensive Benchmarking Experiments on the Million Song Dataset”. In: Oct. 2012.

[Sha19] Andrew Shaw. *Creating a pop music generator with the Transformer.* Aug. 2019. URL: https://towardsdatascience.com/creating-a-pop-music-generator-with-the-transformer-5867511b382a.

[YCY17] Li-Chia Yang, Szu-Yu Chou, and yi-hsuan Yang. “MidiNet: A Convolutional Generative Adversarial Network for Symbolic-domain Music Generation using 1D and 2D Conditions”. In: (Mar. 2017).