MUSIC GENERATION WITH GANS: CONVOLUTIONAL LSTM GENERATIVE ADVERSARIAL NETWORKS FOR POLYPHONIC MUSIC GENERATION

Yuxuan Liu
yliu334@jh.edu

Xiaodong Huo
Xhuo2@jhu.edu

Abstract

In this paper, we propose two music generation model using the frameworks under generative adversarial networks(GANs). The first model is implemented with bi-LSTM generator and bi-LSTM discriminator. The second model is implemented with CNN generator and bi-LSTM discriminator. We train the proposed models on Lakh Pianoroll Dataset of over one hundred thousand bars of rock music. The final generated music is a single-track piano roll formatted piano track. The dataset is acquired from Dong. H.W et al., [2] We show our trained model can generate realistic, coherent, and aesthetic music with no need for data or hand-crafted features. Though the generate music quality fall short compares with human composition, future work will be done to add music tracks and improves the naturalness of generated music. The audio results are available here: https://drive.google.com/drive/folders/1GBQEZpyDOV0gwx4AfNWdOu8BL4GsUYHU?usp=sharing

1. Introduction

Deep learning is a fast-growing domain since 2012, and it has been used substantially in image recognition, speech recognition, and text to speech translation. One notable music generation model using DNN was first introduced by WaveNet in 2016 with a dilated CNN [8]. In recent years, generative adversarial network (GANs)[4] and Variational Autoencoder(VAEs)[5] are successful in generating fake images, videos and texts. The generative networks also make efforts towards music generation. This paper takes the approach of exploring existing architectures and find the optimal solution. The objective of "Generate realistic and aesthetic music" can be broken down into optimize 3 dimensions of analysis, this methodology is proposed by Jean-P.B.[1]

- Objective
  - *What music content is to be generated?*
  - Choices: melody, polyphony, accompaniment,or counterpoint.
- Representation
  - *Music format choices?*
  -MIDI, piano roll, text, ABC note.
  - *How to encode the representations?*
  - Choices: one-hot, many-hot or scalar.
- Architecture
  - *What DNN architecture is applied for construction?*
  - Choices: feedforward network, recurrent network, autoencoder, or generative adversarial networks.

By thoroughly comparing the existing implementations for music generation. We pick the flowing combinations of dimension:

- Objective: The DNN for music generation should be able to manipulate **melody, polyphony, and accompaniment**.
- Representation:
  1. The final produced form of music should be **waveform file type** that can be played across various devices. Piano roll format
  2. **Piano roll** music format is used for both training and predicting.
  3. The **many-hot** encoding representation is used for this Model.
- Representation: Use GANs with LSTM and CNN as our architecture.

To summarize, our objective is to build a DNN that generate polyphonic music. We propose a Generative Adversarial Network(GAN) model, with CNN as Generator and bi-LSTM as Discriminator. The music format representation we decided to use is Piano Rolls.

1.1. Music Representations

For the dataset, we have two options with piano rolls and MIDI events/ABC notes. Piano rolls are binary files represents the pitch notes for time intervals. Piano rolls are also using bars instead of notes as the basic element. Midi files are text-like events, formatted as logging files. The piano roll files, on the other hand, look like images. Thus, we have additional benefits of using CNNs on the piano roll files. We switch from using MIDI files representation to the Piano roll binary representation is to preserve the local and translation invariant patterns. Also, piano roll files are more convenient to perform convolution for its image like nature.

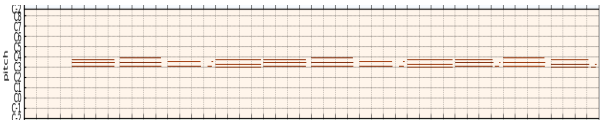

Fig 3. Piano roll representation

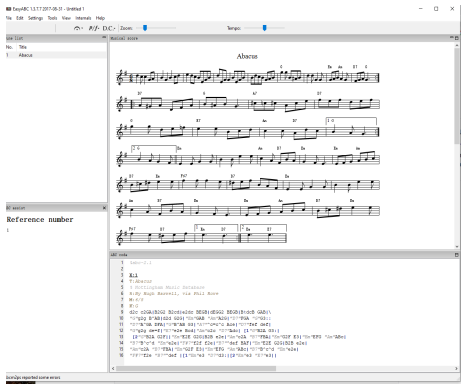

Fig 2. ABC-note-LSTM generated music

1.2. Motivations

Our motivations for building an automatic music generating system are: 1) create tools to help music creation while giving artist inspirations about composition and melody; 2) empower non-musical people to make music; 3) create copyright issue free music for content video and game creators. Why choose deep learning over grammar-based, rule-based music generation systems, automata models, and Markov models? The motivation for using deep learning (and more generally machine learning techniques) to generate musical content is its generality. Compares with the hand-crafted features and deep learning, the learned features are more adaptive as it can be applied to various musical genres. Moreover, as opposed to using handcrafted models, deep learning is better for processing raw and unstructured data. This attributes to the nature of deep learning in extracting features and high-level representations.

1.3. Challenges

Even though many approaches have been used for this problem [9], the tasks remain challenging for the following reasons. Local features play an important role in music generation. Music has a highly hierarchical structure, with the high-level building blocks containing low-level features such as paragraph to phase and to bar. People pay a lot of attention to patterning, rhythm, flow, and coherence. Thus, music pieces have high time dependencies, and local features and patterning are important.

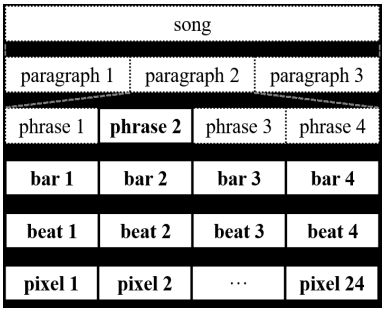

Fig 1. Hierarchical structure of a music piece

Also, musical notes are modulated. Time sequence modeling is not naturally suitable for polyphonic music. Thus, we can't use the generalized fine-tuned model from language generation and monophonic music generation for this project.

2. Related work

GANs with CNNs and bi-LSTMs are used for this project, the following section is the general reviews for GANs and LSTMs.

2.1. Generative Adversarial Networks

. [4] The generative Adversarial Networks consists of two models, a generator model **G**, and a discriminator model **D**. The objective of the generator is to generate fake samples that can fool the discriminator. the objective for the discriminator is to distinguish the fake samples from the real ones.

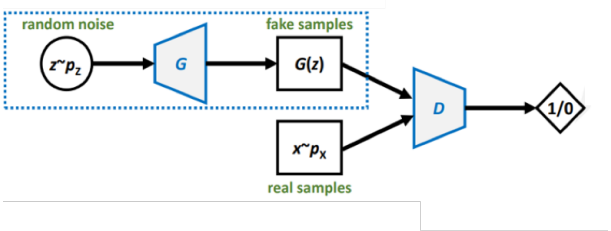

Fig 4. Generative Adversarial Networks

Before describing the mathematical models, we define the following parameters:

Symbol	Parameter
D	Discriminator
G	Generator
θ_d	Parameters of discriminator
θ_g	Parameters of generator
$P_z(z)$	Input noise distribution
$P_{data}(x)$	Original data distribution
$P_g(x)$	Generated distribution

2.1.1 Loss function

The loss function for GANs are derived from the binary-cross entropy loss where y is the original data and is the reconstructed data:

$$L(\hat{y}, y = [y.log\hat{y} + (1-y).log(1-\hat{y})])$$

The Discriminator loss for real data can be obtained from setting $P(data)(x)$ as $y = 1$ and $\hat{y} = D(x)$:

$$L(D(x), 1) = log(D(x)) \tag{1}$$

On the other hand, if the data is fake , the label becomes $y = 0$, and $\hat{y} = (D(G(z))$

$$L(D(G(z)), 0) = log(1 - D(G(z))) \tag{2}$$

For discriminator to correctly distinguish the fake and real images, the loss functions should be maximized:

$$L^{(D)} = max[log(D(x)) + log(1 - D(G(z)))] \tag{3}$$

The generator's objective is exactly opposite to the discriminator. Thus, equation (3) is minimized.

$$L^{(G)} = min[log(D(x)) + log(1 - D(G(z)))] \tag{4}$$

The combined equation (3) and (4) becomes:

$$L = \min_G \max_D [log(D(x)) + log(1 - D(G(z)))] \tag{5}$$

For the entire data set, the loss function becomes:

$$\min_G \max_D V(D, G) = \min_G \max_D ($$
$$(E_{x\ P_{data}(x)}[log(D(x)) + E_{z\ P_z(z)} log(1 - D(G(z)))] \tag{6}$$

The following graph is an example of back propagate the loss function.

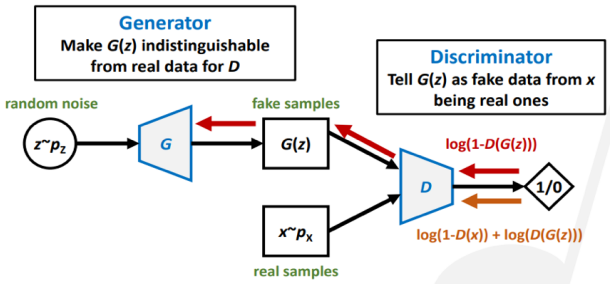

Fig 5. Loss Function and backpropagation

2.1.2 Problems and solutions with GANs

There are some problems we experienced with GANs. One problem was the unbalanced Generator and Discriminator. Unlike a feed-forward neural network, GANs are trained with two blocks. Initially, the Generator has too many parameters than the Discriminator, the resultant training time are drastically different. We choose an updating ratio of 1:10 for Generator and Discriminator to compensate for this problem. Another problem of near-zero loss function updates was solved by randomizing the initialization neuron weights.

2.2. RNN | LSTM

.

[7]Unlike feed-forward neural networks, LSTM has feed-back connections that are capable of processing sequences of data. Therefore, LSTM is suitable for finding time depended on features, and it is suitable for music generation tasks. Long Short Term Memory networks are a special kind of RNN, capable of lean ring long-term dependencies. LSTM replaces the simple neural network inside the RNN block with four networks.

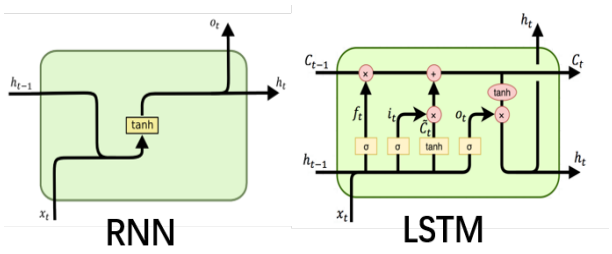

Fig 6. RNN-LSTM

LSTM models have 3 gates: "forget gate layer", "input layer gates" and "output gate layer". The cell state is the line running through the top of the diagram. The cell state is where information runs through the chain. The forget gate layer is on the left most of the diagram, it decides whether to keep or discard certain information.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

Next step is to decide what new information we are going to store in the cell state, it consists of two parts, the "input layer gate" and tanh layer$\hat{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

Finally, we need to decide the filtered version of output which we are going to output:

$$o_t = \sigma(W_o \cdot [C_{t-1}, h_{t-1}, x_t] + b_f)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \hat{h}_t$$

$$C_t = f_t * C_{t-1} + (1 - f_t) * \hat{C}_t$$

The internal states value can be written as : $z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$; $r_t = \sigma(W_r \cdot [g_{t-1}, x_t])$; $\hat{h}_t = tanh(W \cdot [r_t * h_{t-1}, x_t])$

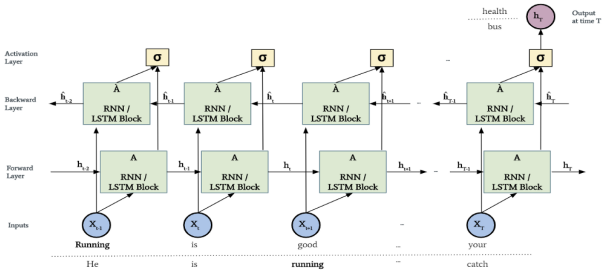

Fig 7. Bidirectional LSTM

Bidirectional LSTMs are an extension of the traditional LSTMs. Bidirectional LSTM improves the model performance by training two LSTMs. In bidirectional LSTM, the second LSTM is simply a reversed copy of the original LSTM. As shown in the figure above, the model consists of two layers side-by-side. The input sequence and the reversed input sequence are feed-ed into both networks. In this project, we chose to use bidirectional LSTM to improve the performance of the model without extra training data.

2.3. State of art models for Music generation

The following sections are recent successful approaches for music generation: Magenta, MuseGAN, Wavenent and MsueNet.

2.3.1 Magenta — 2016

. https://magenta.tensorflow.org/

Magenta project was developed by Google Brain in 2016. They developed with an LSTM model tuned with Reinforcement Learning. The idea of Reinforcement learning is to set rules and metrics to teach the model to follow certain rules while still retain some information learned from data. For example, it can reduce the pause when it's too long. The Magenta performance is pretty promising compares with LSTM.

2.3.2 MuseGAN — 2017

. https://salu133445.github.io/musegan/ [3][2]

MuseGAN project can generate multiple tracks music up to seventeen different instruments. The implementation is using CNNs with GANs. The basic element this network for training train is also switch from notes to bars. Thus, local and translation invariant patterns are preserved. They also developed evaluation metrics to evaluate the performance of the generated music which can be useful for us.

2.3.3 Wavenet — 2016

. https://deepmind.com/blog/article/wavenet-generative-model-raw-audio [8]

Wavenet is popular in text to speech translations, the network is developed with dilated CNNs also preserves a long time dependencies.

2.3.4 MuseNet — 2019

https://openai.com/blog/musenet/

MuseNet uses NLP large-scale transformer model to predict the next token in a sequence and proves to have state of art performances.

3. Method

The core concept of GANs is to achieve adversarial learning by constructing two networks: the generator and the discriminator. The generator maps a random noise z sampled from a prior distribution to the data space. The discriminator is trained to distinguish real data from those generated by the generator. The generator is trained to fool the discriminator. In our application, the generator will generate the music piano roll from the random vector z. And the discriminator is designed to distinguish the real music from the data set and the generated music.

1) Bi-LSTM GAN model

Since LSTM is good at dealing with the time-sequence signal, our first GAN model is a bi-LSTM GAN model. Both the generator and discriminator parts are LSTM based model. The structure of the model is shown in figure 4 and figure 5. The generator contains three LSTM layers. The first LSTM layer has 512 nodes. The second layer is a bidirectional LSTM layer with 1024 nodes in total. Bidirectional LSTM layer here is used to capture the relationship between the previous information as well as future information. The third layer is also a bidirectional LSTM layer

with 1024 nodes. And then, two dense layers are followed to give the piano roll output. Dropout is applied in every layer.

The discriminator part starts with two LSTM layers. The second layer is a bidirectional LSTM layer. Then three dense layers are followed. For the first two dense layers, the activation function is leaky ReLU function. The activation function of the last layer is the sigmoid function.

Layer (type)	Output Shape	Param #
lstm_19 (LSTM)	(None, 100, 512)	1052672
dropout_9 (Dropout)	(None, 100, 512)	0
bidirectional_7 (Bidirection	(None, 100, 1024)	4198400
dropout_10 (Dropout)	(None, 100, 1024)	0
bidirectional_8 (Bidirection	(None, 1024)	6295552
dense_14 (Dense)	(None, 1024)	1049600
leaky_re_lu_8 (LeakyReLU)	(None, 1024)	0
batch_normalization_1 (Batch	(None, 1024)	4096
dense_15 (Dense)	(None, 100)	102500
reshape_1 (Reshape)	(None, 100, 1)	0

Fig 4. Bi-LSTM GAN model Generator

Layer (type)	Output Shape	Param #
lstm_17 (LSTM)	(None, 100, 512)	1052672
bidirectional_6 (Bidirection	(None, 1024)	4198400
dense_11 (Dense)	(None, 512)	524800
leaky_re_lu_6 (LeakyReLU)	(None, 512)	0
dense_12 (Dense)	(None, 256)	131328
leaky_re_lu_7 (LeakyReLU)	(None, 256)	0
dense_13 (Dense)	(None, 1)	257

Fig 5. Bi-LSTM GAN model discriminator

The loss function of the discriminator is the cross-entropy function. Mini-batch stochastic gradient descent is applied to train the model. The beginning learning rate is 0.01 and decreases in every epoch. However, while training, the discriminator is much easier to train compared to the generator. Discriminator accuracy tends to improve every fast. The loss of the discriminator decreases very fast, but the loss of the generator decreases very slow. This means that our discriminator is very good at distinguishing the generated music from real music. But our generator can't generate good enough music. To solve this problem, we applied several regulation methods, such as pretraining the generator model with unsupervised learning, training generator several times while training the discriminator only once. These methods do somehow improve the performance of the generator model, but the loss of the generator and the discriminator still can't converge or oscillated as we expected.

2) CNN-LSTM GAN model

To overcome the training problem mentioned before, we improve our model by a CNN-LSTM model. Borrowing the idea from the MuseGAN network[2], we used a CNN network to generate piano roll data. This is reasonable because different from the truly time-sequence dataset, the piano roll is a more figure-like file. For one bar of music, the length is always the same. This acts as the length of the picture. The width of the picture is corresponding to the notes. The piano roll data actually looks more like image patterns.

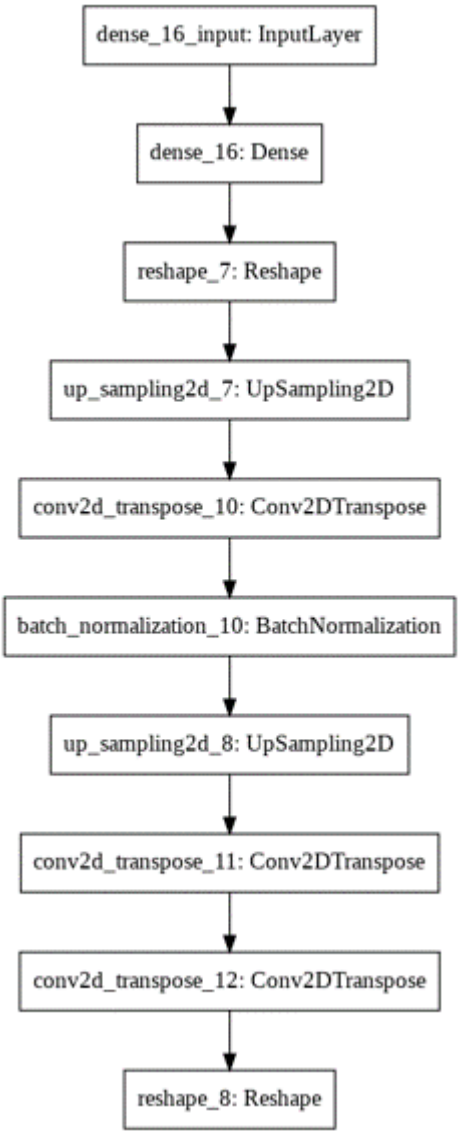

Fig 6. CNN-LSTM GAN model Generator

The structure of the generator and the discriminator are shown in figure 6 and figure 7. The generator model is a full convolutional neural network. The structure detail is to mimic the famous DCGAN network[6]. The model is composed of deconvolutional layers and upsampling layers. The input of the model is still a 100 dimensions random vector. The model first composed of a dense layer followed by a reshape layer to generate a 2D matrix. The Upsampling layer is used to increase the size of the matrix. Deconvolutional layers act as a transpose process of the convolution layer. The function of the convolutional layer is to extract features from the pattern. The deconvolutional layer is trained to generate patterns from the feature. The upsampling layer and deconvolutional layer act as a subunit in the model. In our model, in order to decrease the size of the parameters, the model has two subunits. A batch normalization layer is attached after the first subunit. And a leaky ReLU activation function is applied at the end of each subunit. After these two subunits, the model has another deconvolutional layer. The output of the deconvolutional layer will then be reshaped to the piano roll file.

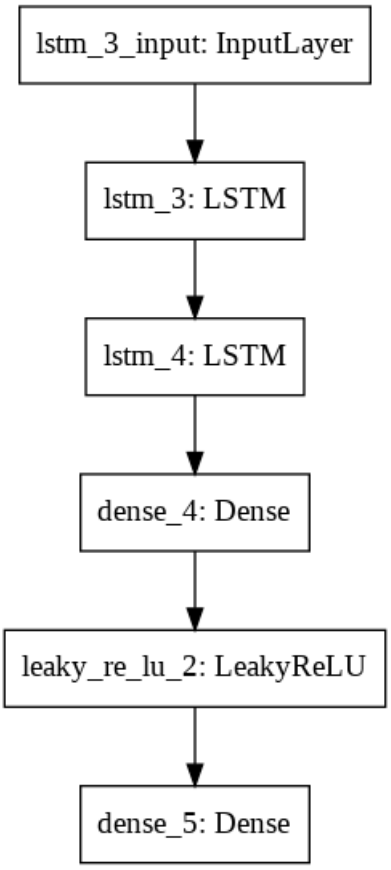

Fig 7. CNN-LSTM GAN model Discriminator

Since in the previous Bi-LSTM model, our discriminator model performance well in the distinguish task. We still use the LSTM model as the discriminator. In this model, Adam optimizer is used instead of the stochastic gradient descent. All other learning parameters are same as the previous model. As shown in the figure 8, this time the loss of the discriminator and the generator osciatalted as we expected after 60 epochs.

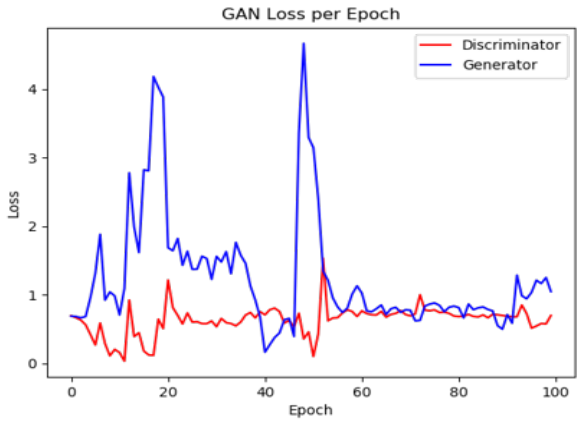

Fig 8. CNN-LSTM GAN model loss

4. Result

The example piano roll result for each models are shown in the figure 1.

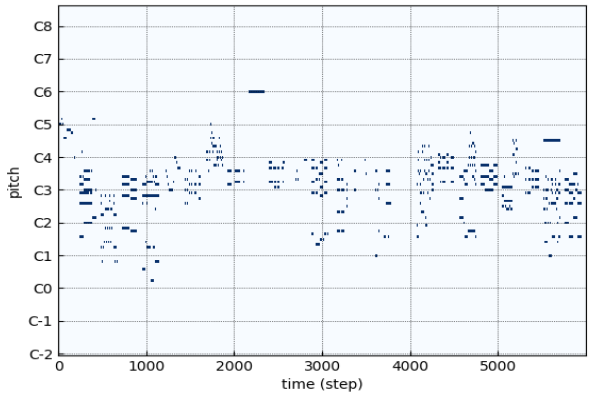

Fig 9. Example generative results for the CNN-LSTM GAN model

To evaluate our models[2], we used several metrics that can be computed for both the real and the generated data.

1) EB: ratio of empty bars (in %)

2) UPC: number of used pitch classes per bar (from 0 to 12)

	Empty bars (EB; %)	Used pitch classes (UPC)	Qualified notes (QN; %)
Training data	24.8	3.28	88.4
Naïve model	40.6	8.92	30.4
Bi-LSTM model	39.5	7.73	48.7
CNN-LSTM model	27.7	5.24	67.4
MuseGAN model	22.6	4.13	62.2

Table 1. Evaluation result for different models

3) QN: ratio of qualified notes (in %). We consider a note no shorter than three times steps as a qualified not. QN shows if the music is overly fragmented.

We generate 2000 bars with each model and evaluate them in terms of the proposed objective metrics. The result is shown in Table 1.

We computed all these metrics to the training data as a standard. To generate good fake music, values of the models should be closer to the values of the training data. We also showed the result from MuseGan network as a reference to compare our result with the cutting-edge result. From the metrics, we see that the CNN-LSTM model tends to perform the best in all our three models. For EB, all three models have more bars then the training data. This implies there are less chords in our generated models. From UPC, we see all three models tend to use more pitch classes. Also from QN, our models tend to generate less qualified notes. These two metrics indicate that our models are more likely to generate segmented notes and might introduced some random noise in the piano roll data. Comparing our best model to the MuseGAN model, we have better QN rate. But worse EB and UPC scores.

5. Discussion

As we mentioned before, we think the LSTM network is better to model time-sequence data. So we first tried a bi-LSTM model to use LSTM layers to constitute our generator and discriminator. But the loss of the generator and the discriminator can't converge. We think the main reason is that the trainable parameters in the generator are much more than the trainable parameters in the discriminator. The generators also have two more layers comparing to the discriminator. This makes the generator hard to train. More works need to be done to modify the model to balance the generator and the discriminator.

Comparing our best model with the cutting-edge models, we still found some distance. We think this distance may result in two reasons. The first reason is the size of the training dataset. Although we use the same dataset, we only randomly select ten percent of the dataset. So the training time is much shorter than the MuseGAN model.

We used about 6 hours to train a single model in our machine. The second reason is the choice of the hyperparameters. In our GAN model, there are a lot of hyperparameters, such as the size of each layer, learning rates, dropout rates.

Because of time, we did not use a validation dataset to find out how these parameters will change our results. We think these two reasons could be the direction to improve our model.

In our model, we only realized a single track (one instrument) music. Our dataset enables us to generate a multi-track model. However, we only use a single track to train the networks. Related works proposed several methods to generate multi-track music. The relationship between different tracks needs to be considered. One way is to use a single random vector to generate all tracks. Another way is to introduce another intra-tract random vector to control the different tracks. We think all these models are reasonable and can be tried in future work.

6. Conclusion

In this work, we have proposed two novel music generation models under the framework of the GANs. The first model is implemented with a LSTM generator and a LSTM discriminator. The second model is implemented with a CNN generator and a LSTM discriminator. We used several objective metrics to evaluate our models. The objective metrics show that our models can somehow learn some structure of music. Although they still fall behind human music, the proposed models have some desirable properties. We hope our future work can further improve the model.

.

References

[1] Jean-Pierre Briot, Gaëtan Hadjeres, and François-David Pachet. Deep learning techniques for music generation – a survey, 2017. 1

[2] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment, 2017. 1, 4, 5, 6

[3] Hao-Wen Dong and Yi-Hsuan Yang. Convolutional generative adversarial networks with binary neurons for polyphonic music generation, 2018. 4

[4] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014. 1, 2

[5] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013. 1

[6] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015. 6

[7] Ralf C. Staudemeyer and Eric Rothstein Morris. Understanding lstm – a tutorial into long short-term memory recurrent neural networks, 2019. 3

[8] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio, 2016. 1, 4

[9] H. A. Wibowo. Generate piano instrumental music by using deep learning. *https://towardsdatascience.com/generate-piano-instrumental-music-by-using-deep-learning-80ac35cdbd2e*, 1(1):1, 2019. 2