C-RNN-GAN++: Generating Continuous Sequential Data

Danling Jiang
Jacobs School of Engineering
University of California, San Diego
San Diego, CA 92093
`dljiang@ucsd.edu`

Zirui Wang
Halıcıoğlu Data Science Institute
University of California, San Diego
San Diego, CA 92093
`zwcolin@ucsd.edu`

Yiming Zhao
Jacobs School of Engineering
University of California, San Diego
San Diego, CA 92093
`yiz048@ucsd.edu`

Abstract

Generative adversarial networks have been proposed as a good way to generate music and a network called C-RNN-GAN has been proposed to generate continuous sequential data, such as music. In this paper, we propose an improved version of the network, called C-RNN-GAN++. It can generate continuous sequential data similar to the input and we use to generate beautiful music. we will leave the generated songs to readers to judge.

1 Introduction

1.1 Generative Adversarial Network

Generative adversarial network (GAN) is an efficient neural network that is used to produce realistic data similar to the input. It is composed of a generator that is trying to produce data as realistic as possible, and a discriminator that is trying to find out those generated fake data. Generator and discriminator are competing with each other in the sense of game theory. The Nash equilibrium is when the generator good images that the discriminator cannot tell. In this way, both generator and discriminator forces each other to improve, and after some training process, we can use generator to produce realistic data. It has been proved to produce highly realistic data.

1.2 Recurrent Neural Network

Recurrent neural network (RNN) is an artificial neural network that has been proposed to model sequential data. In order to model sequential data, RNN usually takes the output from the previous step as an input to the current step. It usually uses maximum likelihood and is usually trained to predict the next token given a certain sequence of tokens. Thus, it represents a conditional distribution of the token and by sampling tokens from the distribution, we can produce very realistic sequences of tokens.

1.3 Generative Adversarial Network together with Recurrent neural network

Generative adversarial network (GAN) is good for producing realistic data while recurrent neural network (RNN) is good for getting the distribution of the data and making predictions. Thus it is a natural idea to combine GAN and RNN to produce realistic sequential data, such as music. This is exactly what we are doing in this paper.

2 Related Work

There are practices in combining generative adversarial network and recurrent neural network, such as Continuous-RNN-GAN (C-RNN-GAN) proposed by Mogren [Mogren, 2016]. It uses a quadruplets of frequency, length, intensity, timing in the network and it combines RNN into GAN. By making the entire model end-to-end, it can train the network by using standard backpropogation. It takes MIDI files as input and generates some similar music. Based on C-RNN-GAN, we develop an improved version of C-RNN-GAN, called C-RNN-GAN++.

3 Model

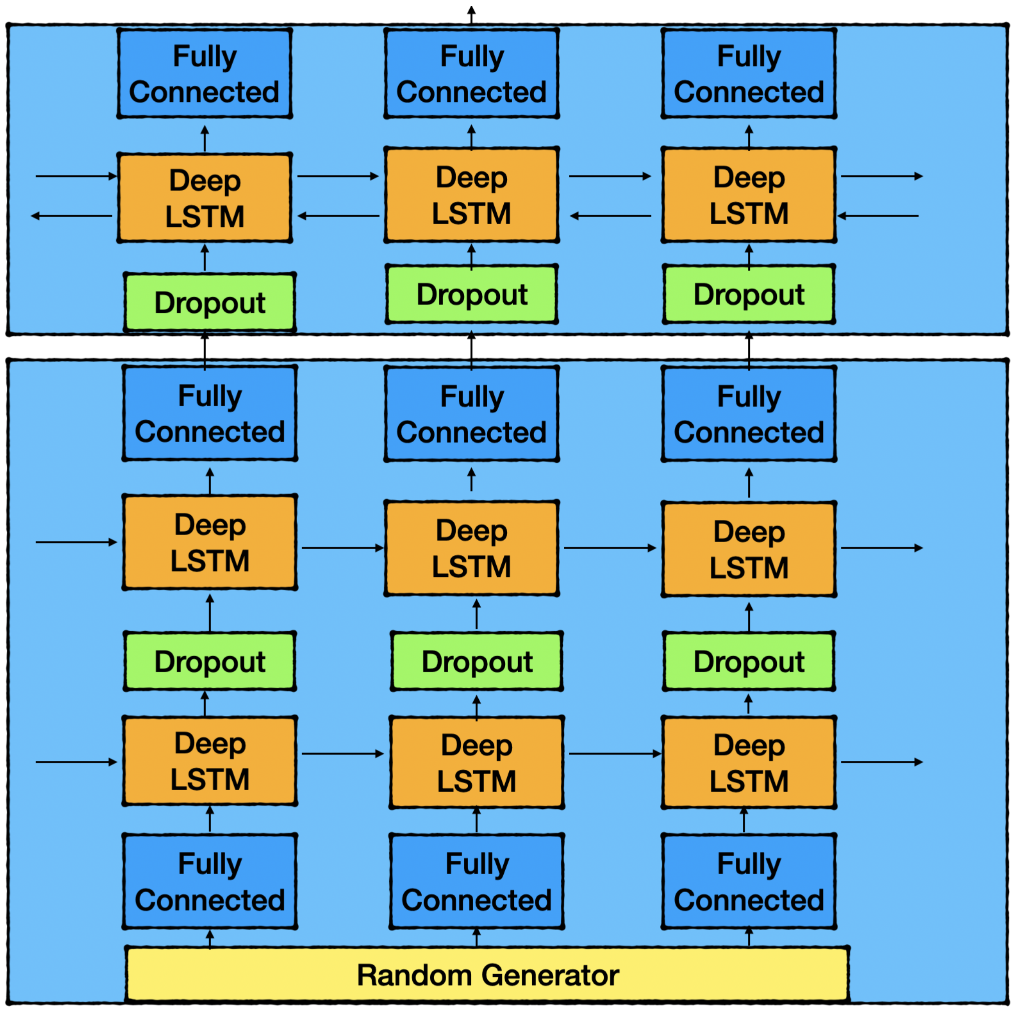

Figure 3 shows a diagram of our neural network

3.1 Architecture

Our proposed model C-RNN-GAN++ is based on the model C-RNN-GAN. It uses generative adversarial network as a basic model. In the generator (G) and the discriminator (D), we use Long short-term memory (LSTM) as our recurrent net. The generator G, which is trying to produce realistic data, is composed of a dropout layer, 2 LSTM layers and 2 fully conneted layer while the discriminator D, which is trying to discriminate whether the data is real or fake, is composed of one dropout layer, one LSTM layer and a fully connect layer.

3.2 Losses

We define loss funtions L_G for generator and loss function L_D for discriminator as following:

$$L_G = \frac{1}{m}\sum_{i=1}^{m} log(1 - D(G(z^{(i)}))) \tag{1}$$

$$L_D = \frac{1}{m}\sum_{i=1}^{m}[-logD(x^{(i)}) - (log(1 - D(G(z^{(i)}))))] \tag{2}$$

In the loss functions above, $z^{(i)}$ is a sequence of uniform random vector in $[0,1]^k$, and $x^{(i)}$ is a sequence from the training data. k is the dimensionality of the data in the random sequence. For the generator G, the random vector is first put into a fully connected layer. The output is then taken into the LSTM cell, concatenated with the output of previous cell. The output of the first LSTM layer will go through a dropout layer. Similarly, the output of the dropout layer will be put into the next LSTM layer, with the same concatenation with the output of the previous cell. Finally, it will go through another fully connected layer and becomes the output of the generator.

For the discriminator D, a sequence will first go through a dropout layer, and then a bidirectional LSTM layer, and finally a fully connected layer to output the prediction of the discriminator.

3.3 Modelling music

In C-RNN-GAN++, we model music in the same way as in C-RNN-GAN. Our model takes music of MIDI format as input. At each data point, we take the values of the tone length, frequency, intensity and the time elapsed since the previous tone. Modeling music in this way enables our network to generate continuous sequential data and thus we can generate beautiful music.

4 Experiment Setup

4.1 basic setup

For the generator G, there are two fully connected layer. The first fully connected layer uses Rectified Linear Unit (ReLU) as its activation function and it takes random vectors as input. The last fully connected layer does not have an activation function and it simply outputs the generated sequence of the generator G. There are also two basic LSTM layers with a dropout layer in the middle. We use a dropout rate of 0.6. For the discriminator D, we simply use a dropout layer with a dropout rate of 0.6, a normal LSTM layer and a fully connected layer with a sigmoid function as its activation. We mainly use sigmoid function to decide whether the input data is true or fake.

4.2 Dataset

We collected our training data from the web in the form of music files in MIDI format, mainly containing video game musics. We record the duration, intensity, tone and the time spent since the previous tone. Our dataset only contains 200 MIDI files, including video game songs from Super Mario, Kingdom Hearts, Pokemon and The Legend of Zelda. Due to our limited computation resources (such as GPU resources), our dataset is relatively small compared with 3697 MIDI files for C-RNN-GAN. We believe that this may be a shortcoming of our project and we may further improve the performance of our model with bigger dataset in the future.

4.3 Training

In training both the generator and the discriminator, we use an Adam Optimizer to optimize the cost of the neural network. We set the max sequence length to 8 so that our model will focus on local pattern while still maintaining some bigger pattern in the music. We use a learning rate of 0.001 and we also do 5 epochs of pre-training for both our generator and our discriminator. We find that training our model does not lead to a very stable convergence. We trained our model for 2000 epochs and it is not very stable. We find that the accuracy of the discriminator always swing from one side to the other. We can generate some very good music from earlier epochs while we may find some music that is not very similar to the input data in the later epochs. We believe there is some improvement to do here but still we do generate some good music that is similar to the input music as you can see later.

5 Result

5.1 Generalization

The main metric we use to measure our model is the similarity between our input music data and our generated music data. As you can see from below, indeed our model can generate data similar to the input data.

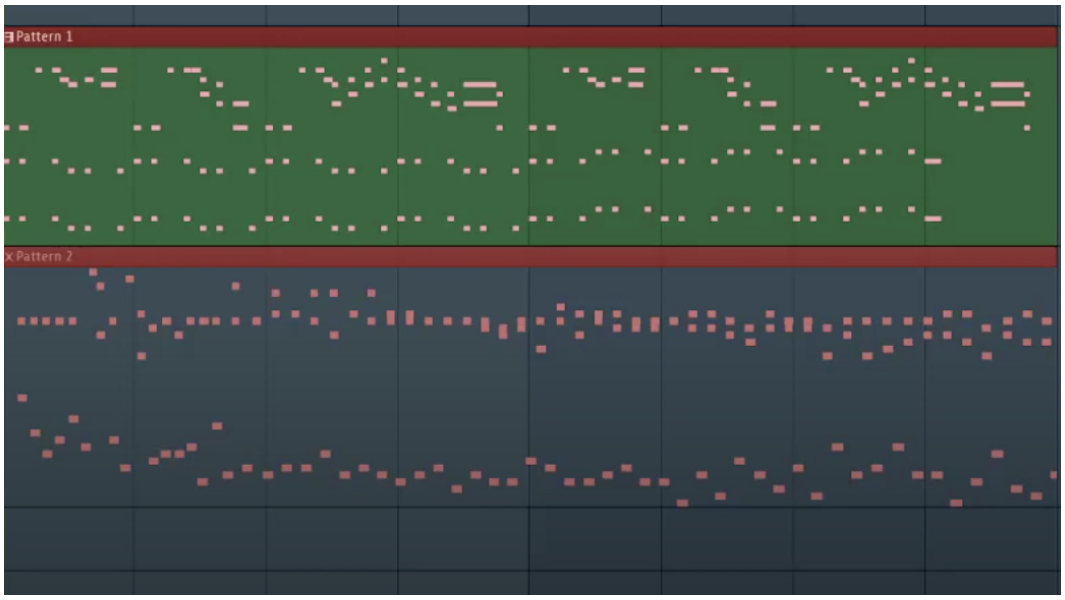

Figure 5.1 shows a comparison between input and output of our model

As you can see from the picture above, the output data is similar to the input data in ways such as the tone length and the range of the pitches. This is a sign of how successful our model is.

5.2 Statistics

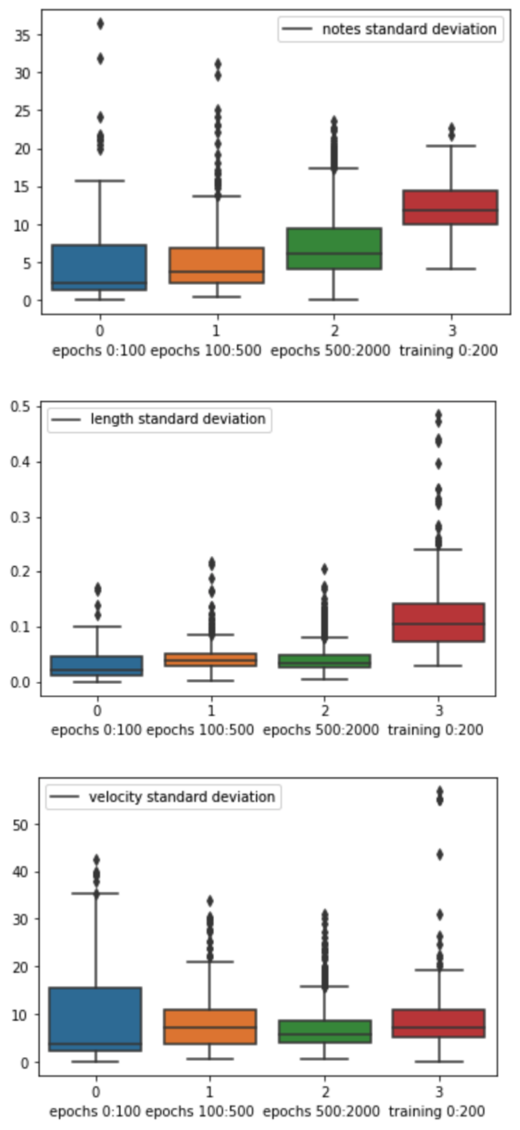

Figure 5.2 the boxplot analysis among of the generation's different range epochs and training set

GAN is a dynamic training model. Therefore, it does not converge. As mentioned in the previous section, we generate data at every training epoch and pick the ones that are more interesting to present. However, not every generation is usable. Therefore we want to analyze the data's distribution in terms of notes, length, tempo, velocity in a macro perspective. We made side-by-side boxplots and see how the generated set is evolving and fitting the statistics of the training set to achieve better accuracy. This not only demonstrates the validity of this model, but helps us to denote insights on the tuning of the model.

According to our analysis, our data set is gradually approaching the ideal training pattern. In general, as the number of epochs increased, the closer the generated statistics fitted the training set in a macro perspective. This is an indication that our generator is learning to generarte music in a more "humanized" way.

6 Discussion

By experimenting and modifying with the existing model, we made significant improvements on pattern recognition and generation. However, there are still existing problems in the aesthetic aspects of the music generated by this model.

6.1 Melody Generation

This model does not work properly for generating continuous melody, and it cannot generate chords. Even though the music data is preprocessed with marks on certain chords, the transformed data fed into the generator neural network did not explicitly show the property of chords as well as chords' transition. Furthermore, as the midi data is directly transformed into the matrix, the model cannot differentiate between melodies and chords that are played in a synchronized approach. Specifically, there were two potential improvements that could solve this problem.

The pytorch implementation for reading tones and chords is oversimplified. While the author has considered 12 base tones as well as their major/minor chords, one can further improve this model by adding an additional feature representing their relative relationship. For instance, transitions C -> Am -> F -> G and D -> Bm -> G -> A are different based on the chords, but are the same based on the relative distance between chords. By adding this additional information, this model can better generalize certain chord progressions that most humans use.

Mixing melodic and chord features with pattern features (Tempo, velocity, timestemp) into one feature matrix may impact the performance of the RNN generator. Since our result showed that this model excelled in generating certain patterns but failed in generating melodys, one way to solve this problem is to use a separate Char-RNN to learn from melodic data while using the C-RNN to learn from patterns. Then, one can concatenate these two separate paths into a generator, and train with a new discriminator that implements 6.1.1 to form new generative adversarial networks.

6.2 Timestamp

This model learned the original data's timestamp, which allowed the generator to generate music that has a rhythmic pattern. However, this pattern can be defective if a few notes that should've formed a chord are generated with slightly different timestamps (for instance, say, 50ms). Since the model is unsupervised and timestamp is not synchronized with the midi's tempo, the above situation could happen if two midi files have slight differences in tempo but the same rhythmic pattern. To resolve this problem, I would suggest transforming timestamp into bars and beats and their relative timestamp (normalized by tempo), and to modify the corresponding generator to make fewer mutations when generating continuous rhythmic patterns.

7 Conclusion

In this paper, we have improved the model from the original paper for C-RNN-GAN into one that can generate different genres and detect more rhythmic and velocity patterns. While more improvements are necessary, we have generated some prototype of chiptune music that resemble the structural patterns of many tracks, such as music in Mario, Zelda, and Pokemon. We believe C-RNN-GAN++ is a successful model in producing realistic music.

Reference

Olof Mogren. C-RNN-GAN: Continuous recurrent neural networks with adversarial training. Conference on Neural Information Processing Systems, 2016.

C-RNN-GAN Paper Arxiv Website:

`https://arxiv.org/abs/1611.09904`

C-RNN-GAN Implementation In Tensorflow:

`https://github.com/olofmogren/c-rnn-gan`

C-RNN-GAN Implementation In Pytorch:

`https://github.com/cjbayron/c-rnn-gan.pytorch`

Supplementary Links

C-RNN-GAN++: Modified PyTorch Implementation of C-RNN-GAN:

`https://github.com/zwcolin/cse190_ml_audio_project`

Visualization of Generated Music:

`https://www.youtube.com/playlist?list=PLJPkbw7JjPIusx1hqPVYCBACcK9cK9VXT`

Generated Midi Files (2,000 epochs) and Selected Music (as .mp3):

`hhttps://github.com/zwcolin/cse190_ml_audio_project/blob/master/Generated%20Set%20and%20Selected%20Music/Generated%20Set%20and%20Selected%20Music.zip`

CSE 190 Project Group Documents Repo:

`https://github.com/zwcolin/cse190_ml_audio_project/tree/master/Documents`