Abstract—Since the machine generated music became more popular, our group plan to train a more effective model and get better music performance. We propose to use a model combination called C-RNN-GAN. This model is based on Generative Adversarial Network (GAN) model, with using convolutional Recurrent Neural Network (c-RNN) inside the GAN. This model is trained with classical and jazz MIDI dataset. The result will be demonstrated with generated music.

Keywords—music generation, generative adversarial network, recurrent neural network

I. INTRODUCTION

The music industry nowadays heavily relies on the machine generated contents to give producers inspiration. Beats, rhythms, and melodies are the main components of songs that can be reproduced by machine now. Music is the most common continuous signal model. In the past, there have been many studies on the simulation and analysis of this continuous signal, but how to recreate it by machine has not yet been explored, so it has great academic interest. In addition, music is also a huge commercial product. Successful hits will bring huge economic benefits.

In our project, we aim to use machine learning to train a model and use it to generate some unique pieces of music. To proceed on this generation problem, we first focus on GAN algorithm. We searched for some music generation related research and found one using C-RNN-GAN to train model and generate music [1]. In this research, the authors combine the GAN and RNN together. The main structure is GAN. The generator and discriminator are built by using RNN structure for processing continuous data. We based on the code from this research and modify it to try to get better generation performance. The input to our algorithm is midi datasets of classical and jazz music. We then use a GAN to output a midi generative music.

II. RELATED WORK

A. Using neural network model

In 2002, Eck and Schmidhuber used recurrent neural network with long short-term structure (LSTM) to analyze and reproduce blue music [2]. The model they used is similar to the one we used except that they didn’t use GAN to train the model. The main feature they used to train are melody and chord only. However, though they proved the LSTM structure worked well with RNN, they didn’t compare it with other methods. Besides, their result is difficult to evaluate objectively. But the idea of using RNN with LSTM is useful for later research. In 2007, Adiologlu and Alpaslan used a feed-forward artificial neural network to train a model for generating two-voice counterpoint music [3]. In their research, they successfully trained a model which can compose two-voice counterpoint based on a piece of music. However, it is hard to be said as music generation. It only learns the way to do the composition with counterpoint method. But the goal of their research was to propose a new schema for representing notes, melodies, and time. In 2016, Olof Mogren and his team tried to combine RNN and GAN method to train a model for regenerating pieces of music. This research is the base of our project. In 2019, Hawthorne and his team used autoencoder to train a model which can produce a factorized approach to musical audio modeling capable of generating about one minute of coherent piano music [4]. They used a structure called Wave2Midi2Wave. The input is wave signal of music, and the encoder will encode it to midi type data, and finally the decoder will decode it back to wave type data. They used listening test to show that their model can reconstruct realistic music. However, it is not a typically music generation model that can be used to create realistic music.

B. Using other methods

In 2014, Roig and his team separated music to different pattern and used big dataset to train a probabilistic model, which was used to generate melody according to what pattern was chosen [5]. In this research, the basic model they used is the probabilistic model. They separated the music based on time signature, rhythm pattern, and pitch contours and calculated the probability of each kind of pattern. Based on the information given to the model, it can successfully generate a new melody. It is a brand-new direction to generate music, but, however, it
required some specific information to generate. In 2018, Huang, Chen-Zhi Anna team tried to improve the model. They used long-term structure to help to train the model. They implemented a transformer decoder with recurrent and convolutional neural network instead of pre-position input representations. As the result, this new transformer showed higher win rate in listening test, which proves the RNN with LSTM is useful in continuous signal data processing. As those researches we found, we are confident about using this C-RNN-GAN to generate music.

III. DATASET AND FEATURES

We are using Musical Instrument Digital Interface (MIDI) dataset including jazz music data and classical music data. Midi files can have up to 16 channels with different instruments. Thus, we can separate the music files into different instrument category and train the model based on that. For classical music dataset, we have 3157 music samples with 127 composers and for jazz dataset, we have 465 music samples with 74 composers. We have 20 percent of the data for validation, 20 percent of the data for test, and 60 percent of the data for training. Thus, for the classical dataset, we have 1894 train samples, 631 test samples, and 631 validation samples; for the jazz dataset, we have 279 train samples, 93 validation samples, and 93 test samples.

In order to preprocess the music data, we used the python midi package. It has the functionality to separate each note into different channels (instruments) by the note-on event and note-off event methods. In this way, we can separate notes from the music files as our main components. Each note will have the information of the tone length, frequency, velocity, and timing. These will be main feature extracted from our note events. Because we have time-series data, this also helps us to discretize the music data. We used the midi_pattern.resolution and midi.events.SetTempoEvent to get the resolution and the beat information. The maximum song length is set to be 500, which mean the maximum number of input events is 500 to make sure the generator and discriminator have the same range of the input.

IV. MACHINE LEARNING MODEL

The model we used for this music generation project is continuous-Recurrent Neural Network Generative Adversarial Network (C-RNN-GAN). The main idea is to have C-RNN model nested in the GAN model in order to generate more realistic sequential music data. RNN models are usually used for sequential data and GAN helps to produce the highly realistic data more efficiently. The following is a schematic of the C-RNN-GAN model:

![Fig. 1 Schematics of the C-RNN-GAN model[1]](image)

A. Recurrent Neural Network (RNN)

RNNs are usually used to model sequential data. In the structure of RNN, each output of the previous cell is fed into the next cell as the input. The advantage for using RNN is that it takes account of the historical data used in the training process. As we see in the following figure, each intermediate output of a cell is taken to the next cell. The RNN usually returns the probability as maximum likelihood of the next cell given the preceding probabilities of the previous cell.

![Fig. 2 The structure of traditional RNN. [7]](image)

![Fig.3 A typical RNN cell. [7]](image)

The each intermediate cell is created as the sum between the linear combination between the weight to the sample $W_{ax}$ and weight to the previous cell $W_{aa}$ and a total bias. There is also an activation function $g1$

$$a^t = g1(W_{aa}a^{t-1} + W_{ax}x^t + b_a)$$  \hspace{1cm} (1)$$

The output of each cell is a linear combination with weights $W_{ya}$ and bias $b_y$, and the activation function $g2$.  

Here we use Long short-term memory (LSTM) networks as the cell inside the network. LSTM has gates that overcome the vanishing gradient problem and learn longer dependencies. The vanishing gradient problem has is specific to the use of the RNNs, because as the layers increases in the RNN, the use of the activation function will cause an exponential increasing and decreasing of gradients. When the back prop begins, it is hard for the system to grasp the gradients for the long term structures as the number is either too large or too small. The LSTM accounts for the vanishing gradient problem. The LSTM cell has the following structure:

\[ y^t = g_2(W_\gamma a^t + b_\gamma) \]  

\[ \Gamma = \sigma(W x^t + U a^{t-1} + b) \]  

Where \( \sigma() \) is the sigmoid function with the clip setting. In the figure, four of these gates are used in element wise multiplication fashion.

\( \Gamma_f \): is the forget gate
\( \Gamma_u \): is the update gate
\( \Gamma_r \): is the relevance gate
\( \Gamma_o \): is the output gate.

By default, the forget gate is turned off, and the output is printed as the following:

\[ c^t = \Gamma_u \cdot \hat{c}^t + \Gamma_f \cdot c^{t-1} \]  

\[ a^t = \Gamma_r \cdot c^t \]  

\( \cdot \) specifies the element wise multiplication

On the Discriminator architecture, we also used bidirectional RNN, the bidirectional RNN also uses the LSTM as their cells.

In the TensorFlow 1.15, the following functions are used for construction of the LSTM cells

LSTM has feedback connections that are suitable for sequential data such as video and music. Here, we have LTSM networks nested in the GAN model. In the generator, we have unidirectional LTSM networks. In the discriminator, we have bidirectional LTSM networks.

The LTSM networks composed of 100 LTSM cells that concatenate the input from the random generator and also the input from the previous LTSM cell to have the output for the generated features. Note that for the unidirectional LTSM networks in the generator model, each LTSM cell will take the input from the previous cell. But for the bidirectional LTSM networks, each cell will concatenate the input from both previous and next cells to have more accurate decision.

**B. Generative Adversarial Network (GAN)**

GAN model usually has two components: a generator and a discriminator. The goal of the generator is to produce fake music sample from random generated input. The purpose of the discriminator is to identify the generated sample from the real data input. Generator and discriminator have the competitive relationship. The generator wants to fool the discriminator with the generator data very close to real data, and the discriminator tries to decide to determine whether the input is fake or real. This framework is efficient to train deep generative models without many normalization constants which can save a lot of time.

The loss function of both generation and discriminator are given by

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

\[ L_D = \frac{1}{m} \sum_{i=1}^{m} \left( -\log D(x^{(i)}) - \log (1 - D(G(z^{(i)}))) \right) \]  

Due to the competitive relationship of the between the generator and the discriminator, they want to maximize the loss of each other. The two networks will be train at the same time in order to have the generated sample unable to be identified by the discriminator.
V. EXPERIMENTS/RESULTS/DISCUSSION

A. Experiment

The hyperparameter we used here are Validation/test ratio = 20% Learning rate = 0.1 Hidden layer size for Generator(G) and Discriminator (D) = 100

The random generator in G has the input size of \( x \in (20 \times 4) \) matrix, and the output for G has \( 100 \times (20 \times 4) \) size. There are in total 100 LSTM cells inside.

The depth for each LSTM cell is 2, and the hidden layer size for LSTM is 100 units, as opposed to the 350 used in the literature [1]. This setting for LSTM is used for both G and D unidirectional RNNs and bidirectional RNNs.

For each song midi input, it is split into \( 20 \times \text{songlength} \times 4 \) as 20 is the batch size, 4 is the numbers of the features, 500 is the max song piece length. The song length for each epoch could range from 10 to 50. The discriminator D has same 100 LSTM bidirectional cells inside the discriminator structure. Discriminator has two outputs, one is the decision, which is 20 numbers to match the batch size, and the D generated features of dimension of shape \( (200 \times 20 \times 40) \)

The input layer for G uses relu() function as the activation function. The hidden layers of LSTM used the sigmoid function as the activation function as described in the previous section. The output layer has no activation functions for G as the data point from this layer is treated as the generated music. The output layer for D also uses the sigmoid for getting the decisions of each batch.

Sounds are pieced together through the generator. The pieces generated from the previous epochs are also reused through the next epoch of training.

Feature matching techniques are employed during training except for the loss. The idea of feature matching is to control overfitting of the D and to allow larger variance allocated to G to create errors with D.

B. Results and Discussion

As we see in the loss of the training and validation data, the loss as specified via equation (6) and (7) are dropping for the evolution of the epochs. Since using RNN, it takes a long time for one epoch to train. We will terminate training after some epochs and restart. This is action results the discontinuities presenting in the loss figure. Another observed performance hindering result is that sometimes an empty song would be generated in an epoch. The result for this phenomenon might be that the discriminator being too strong against the generator. The overall returning gradients coming from the discriminator is too high so that the generator has to adjust the weight violently. So some of the feature such as the tone length could be adjusted to 0. One way to amend this is to employ a technique called pretraining. Pretraining on the generator and discriminator often get the rate of convergence faster across epochs and drop the occurrences of the empty song. One example is that around 30 epochs, there is an empty song generated, this shows the existence of the empty song.

The discontinuity is resulted from turning off training and restart, some times if we allow the memory growth, the model will run out of memories because we use previously generated data. When restart, we essentially retrain the models.

In Fig 7. There are four mainly used evaluation criterions: (1) Scale consistency, this is the fraction between tones and a piece of standard scale, this is a variance indicator of the song. If the consistency is too high, it is either an empty song or the variation of the song is too low. (2) Tone span, this is the measure between the highest tone and the lowest tone in the sample. (3) Polyphony, this is the measure of the frequency of the two tones appearing simultaneously, as we hear many chords in the piano play, the C,F,Em all has the these polyphony. The result show that very little of the polyphony appeared during the training, suggesting that our network has not yet learned to play chords. (4) Repetitions: It measures how often a short sequence is repeated in the song. Three tone repetition gives the indication of how three tone structures repeats.

For results, we also have the generated music data. We did the experiments for both jazz and classical music dataset. The music results are contained the zip files.
VI. CONCLUSIONS/FUTURE WORK

In conclusion, we have shown C-RNN-GAN model is one of the options for music generation based on the loss and the feature matching results and also from the generated music samples. The results are similar to the real data. For further improvements, it will be interesting to implement larger datasets and different music genres to see more possibilities. Also, this model can be an extension for existing music editing software such as Ableton to give producers inspirations and ideas.

REFERENCES


Individual contribution

● Qi Yue:

He mainly adjusted the code base to fit python 3.0 environment and machine learning models and tuned the parameters for the model structure. He also generated the main results using the available models. He tried on different features that could be comprehended using the midi loader and tried to compare with the results without these features, but the performance of the network was not robust enough.

● Huiyan Li:

She reviewed and searched the background topic and relative information related to the topic. She helped build the suitable environment and the packages to run the model. She analyzed and tried to improve the training model. She helped to implement the jazz dataset for the training model and analyzed the results.

● Zihao Mo:

He conducted relevant research references for the content of this project. He built an environment and installed necessary dataset for the training of the model on Datahub. He successfully implemented the model training on Datahub. He helped analyzed the model and tried to improve it. He helped trained the classical midi music dataset. He helped prepare the presentation and write the final report.
Replies to critical review

Critical review from group 51:

- Really great model architecture slides.
  Thanks.
- Nice custom codebase for this project.
  Thanks.
- On the feature matching slide, can you explain more how “stats” y-axis are calculated?
  I think you asked about the evaluation criteria, please see the result section, which has an elaboration of how each of them is calculated.
- Why is feature matching important for the overall project objective?
  The feature matching gives another way to evaluate the performance of the generative audio is good or not. Although we mainly used score of each feature to evaluate the performance of the audio, it doesn’t sound real when the score is high. So, we use feature matching to make the generative audio sounds more real and comfortable.
- Can you better explain discontinuities in the loss plot? Why does restarting increase the loss so much? Are there possible implications?
  See p4 result section. The discontinuity is because the model lost it’s weights, essentially retrained the generator and discriminator. We tried limiting the memory use of the GPU, and then the memory should not grow too large when we train the model in future use.
- Why did you choose the R-CNN-GAN?
  It should be C-RNN-GAN. We did some research about the generation music. One research conducted in 2002 proved that C-RNN structure can perform well with training the continuous data. We also referred to another research conducted in 2016 argued that GAN model could give better performance using the generator.
- How did you make choices for model architecture?
We mainly based on the model architecture in Ref. [1]. After several times of training, we decided to continue use this architecture because it is efficient and straightforward.

- Could you please explain what “notes” imply?

Notes in music is a symbol. In our dataset, a note means a value point in specific time. It denotes the pitch and duration of the music. More detail can be found in Section 3.

Critical review from group 81:

- It is a bit unclear what the network is trained toward. It is understandable that the generator network should train to produce realistic music, and the discriminator should train to identify whether its input is genuine or generated music. However, it is a bit unclear how the generator knows to produce its output. Does the generator take real music as its input or is the input random input to generate “random” music?

  The generator does not use the real music as it’s input, but it’s goal is to maximize the loss in the discriminator. The feature matching also changes the weights in the G such that it is several music quality indicators. (see result section on p4)

- Were any other activation functions other than ReLU explored?

  Activation functions such as sigmoid and tanh are also explored, but the relu function is used mainly to confine the values between 0 and 1 to make valid probabilities or log likelihood definition in the RNN (see p3 on the explanation in RNN)

- In Slide 4, it is mentioned that better music has the potential to bring more economic success. However, music is a subjective art form. Is there any way to quantify the relative quality of the generated music by using some common metrics/features found in existing “good/popular” music?

  To be honest, it is hard to quantify the music. People’s judgement on music is very subjective. The most intuitive judgement must be the feeling when listening to it. In addition, another way is to use some music theory to analyze the music and use experience to judge whether it is feasible. For now, there is no such a standard of metrics/features accepted by public to quantify the quality of the generated music.
Critical review from group 82:

- Great job, it seems like you guys did a lot of research into the previous approaches at this issue and highlighted them well in your PowerPoint and video. Your code review was very in depth and your group did well at explaining the model summaries. I appreciated the graphs you guys provided and how you also showed both training and validation loss, it provided insight into how well your model performed at the given task. The feature extraction slide was very intriguing since it essentially highlights what makes a song. It would be interesting to compare how the different features vary between music genres.

  Thanks.

- In terms of criticisms, I think it would be better to play the full results, it’s an interesting topic and the presentation would’ve been more captivating by playing the full audio file.

  Thanks for the suggestion. However, the audio we played in the presentation was the full length version.

- Furthermore, it would’ve been interesting to talk more about the other approaches and explain why they aren’t as promising, maybe even experiment with one and show the results of it.

  Good suggestion. But as we didn’t have enough time to experiment with other approaches, we only can knew the result and backward in other research. More detail can be found in Section 2. Related worked.

- Elaborating on that a little, giving insight into the issues your group ran into would be cool as well. Lastly, as I briefly mentioned earlier, I think it’d be very interesting to highlight what makes the genres different in terms of the feature extraction and maybe provide a visual.

  Good suggestion. We will try to provide a more intuitive plot in our report. See result section on how these issues were