Classical Piano Music Generation using GAN

Rose Gurung
School of Computing and Information Sciences, University of Maine, 168 College Ave, Orono, Me 04469
rose.gurung@maine.edu

Abstract: Lately, generating realistic images, videos, texts, and aesthetically pleasing paintings using Generative Adversarial Networks (GANs) has been very popular. These tasks have proven the ability of GANs to incorporate creativity into AI. GANs are highly versatile and can be used to generate anything that can be synthesized into images. In this project, I will be using GANs to generate music based on the Musical Instrument Digital Interface (MIDI) files of classical piano pieces. These files are used to train popular GAN models in attempts of generating compelling musical pieces. Different models for both the generator and the discriminator networks comprising the GAN are experimented with and their affects in the generated music is discussed. Finally, some piano pieces (MIDI files) are generated and analysed for different configurations. It is concluded that GANs are incredibly hard to train but there are solutions to avoid such difficulties in the future. © 2022 The Author(s)

1. Introduction

Generative Adversarial Networks (GAN) are an innovative area of deep generative model that has introduced creativity into Artificial Intelligence (AI). It was first proposed in 2014 by Ian Goodfellow [1] and his teammates as an improved competitor to generative models. Since then, it has been continuously developed with new kinds of GAN created by researchers ever so often. From the most popular GAN known as Deep Convolutional GAN (DC-GAN), Conditional GAN (CGAN, to Info GAN, GANs have had a huge progress and since its birth, likely from its experiments with a broad range applications in dealing with Computer Vision, Natural Language Processing, or other domains of AI.

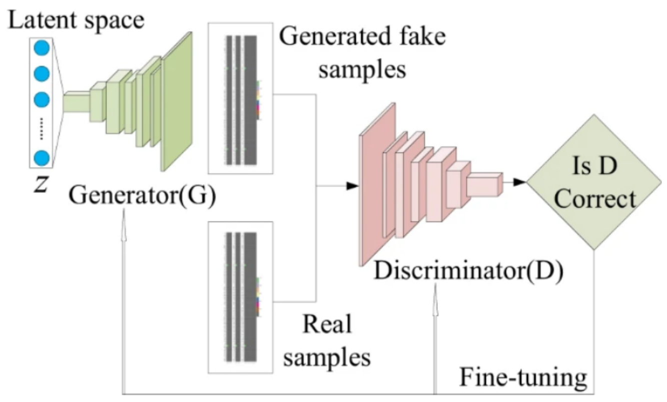

Fig. 1. A GAN Model [2]

The design of GAN is inspired by the theory of game. A diagram of a normal GAN architecture is shown in figure 1. It works with two main neural networks– the generator and the discriminator. During adversarial training, the generator and the discriminator continuously compete with each other until they reach a Nash equilibrium. The core task of the generator is to emulate the probability distribution of real samples and generate fake samples that closely match the patterns of the original sample. It then sends these fake samples to the discriminator. On the other hand, the task of the discriminator is to distinguish between the real samples and the samples generated by the generator. The generator receives the decisions made by the discriminator and generates a different sample in attempts to deceive the discriminator. They constantly try to outperform the other and as a result of this adversarial training, the generator is able to generate samples that fit the original distribution of the real samples.

Fig. 2. GAN generated faces of people [3]

GANs have shown great performance generating realistic looking Images that can even fool the eyes of a human. Figure 2 shows randomly selected images of people's faces generated by NVIDIA's "Style-Based" GAN. GANs are also highly versatile, allowing them to generate realistic samples of anything that can be synthesized into images, like for instance, music. There are several GANs developed for this specific task of generating musical pieces, namely, MuseGANs [11] and GANSynth [13]. Generating music has a few notable differences from generating images or videos. Unlike images and videos, music is an art of time requiring a temporal method. Moreover, music usually requires several tracks or instruments with their own temporal dynamics. Lastly, musical notes are grouped into chords, harmonies or melodies which further require some control as to the kinds of samples that need to be generated. As a result of these added complexities, there is less investigation dedicated to creating musical pieces using GANs compared to generating images, videos, and texts. Furthermore, these qualities of music generation make it challenging for a generative network to produce realistic music with the chords and harmonies of all instruments in their respective chronological orders. Therefore, it is evident that this field requires more work into building a model that can take into account all these challenges and generate realistic musical pieces.

2. Related Work

Recently there has been a surging number of models proposed to generate music. Most earlier works used RNNs and focused only on monophonic music i.e. music with only a single instrument or music with only a single track [4]. This was also adapted to polyphonic music by Hadjeres, et al [5] who used RNNs to generate four-voice chorales. Chu et al. [6] were able to generate music with chords, melody, and drum tracks using hierarchical RNNs to coordinate the three parts. GANs were explored for this task in the form of C-RNN-GAN [7] to generate music as a series of note events by introducing some ordering of notes and using RNNs in both the generator and the discriminator of the GAN. Using this note event representation, SeqGAN [8] combined GANs and reinforcement learning to produce music as sequences of discrete tokens. All these works compose music from scratch. MidiNet [9] used MIDI files with conditional, convolutional GANs to generate melodies that follow a chord sequence given a priori, either from scratch or conditioned on the melody of previous bars. DeepBach [10] took the challenge of generating Bach-like four-part chorales by using a generative model that is steerable in the sense that a user can constrain the generation by imposing constraints such as notes and rhythms. MuseGANs [11] have used GANs to not only generate music from scratch but also to generate a musical track that accompanies a human-made track. More recently, WaveNet [12] and GANSynth [13] has been able to produce raw audio waveforms that include speech as well as music generation and have claimed to generate high-fidelity and locally-coherent audio that are regarded as highly realistic musical fragments even by human evaluation metrics.

3. Methodology

To avoid some of the challenges of generating polyphonic music, this project will focus only on using monophonic music, i.e. a single track with only one instrument. Following this, several monophonic pieces can be combined into a polyphonic musical piece (music with multiple tracks or instruments). Most music generation based on GANs make use of the midi files. These are files that do not contain the actual audio content but contain information about the pitches, timings, and durations of the notes that are played. The MIDI files can then be converted into images depicting with black background and white blocks that represent the pitch, timings and durations of the notes i.e all the information contained in a MIDI audio file. These can be called MIDI images.

3.1. Dataset Preparation

Training a traditional GAN can require from 50,000 to 100,000 images to reproduce a substantial result. Hence, training GANs hinges critically on the availability of a huge dataset with diverse content. Among the plethora of MIDI files that can be found online, it was difficult to find an ample number of samples for a certain composer like Bach or Chopin. Also, there wasn't much variation in the ones that were found until I discovered GiantMIDI-Piano. The GiantMIDI-Piano dataset has 10,855 MIDI files for solo piano pieces composed by 2,786 different composers. Among the 10,855 MIDI files there were 7,127 files that had the surnames of the composers.

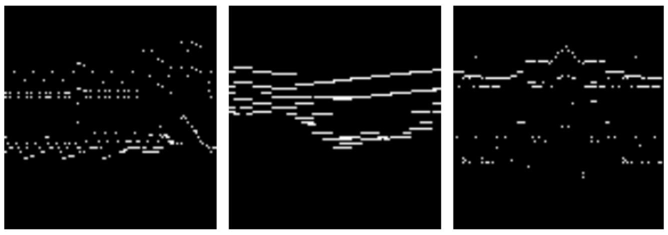

Fig. 3. Original MIDI images

With the help of the music21 library and a supporting script (github reference), the MIDI files were converted into images. Each note can be denoted by a white block where the height of the block will represent the pitch of the note and the width will represent the duration of the note. The images have a fixed size of 106 rows and 100 columns. 106 is the number of different notes (From 21 (A0) to 127 (G9)) that are available in the MIDI file and 100 is the chosen length of the song which will be the same for all MIDI images. The length roughly converts to 13 seconds. If a song is longer than that, the song is split and there will be several midi images created for it, all of length 100. Each pixel represents a note played for 1/4 beats. These images were also resized into 106X106 squares just for its ease of working with convolutions. Three resized MIDI images are shown in the figure 3. I chose to use the subset of GiantMIDI-Piano with surnames because those 7,127 midi files would produce 50,000 MIDI images which should be sufficient to train the GAN.

During extraction and uploading of the dataset to the VSCode environment, some of the data were lost, due to memory issues and I ended up with 4062 MIDI files. These files when converted to MIDI images generated 28587 images after conversion on GPU for 10 hours. I decided to only use these due to the time constraint concerned with converting more files and further training.

3.2. Training

Tensorflow Keras was used to make the layers of the GAN model. The pixels are extracted from the MIDI images and normalized into [-1 , 1]. The buffer size is defined as 30000 so that the data is all shuffled properly and the model is run with all the standard batch sizes of 32 and 128.

The generator uses upsampling layers, the transpose convolutional layers, to produce an image using a random vector of size 100. A dense layer of 53X53X128 takes the random noise vector first and this is upsampled several times until it reaches the size of the output image 106X106. The layers use leaky ReLu for all layers except the last layer which uses tanh. This is because the input has been normalized to [-1, 1]. The discriminator is a standard CNN image classifier. Using binary cross entropy, the model is trained to output a positive value for real images and negative values for fake images(generator images).

During the training loop the generator generates an image and the discriminator makes a decision. Both the models use different Adam optimizers with different learning rates. The losses are calculated for both the generator and the discriminator and the gradients are applied to update both the models. After the final epoch, resulting MIDI images are generated and converted into MIDI files that can be played.

4. Experiments

I experimented with various aspects of the GAN network. The first discriminator and generator architecture I used was the one that works well with the MNIST dataset. The samples in MNIST dataset are similar to the MIDI files generated in that they both have dark backgrounds with the areas of interest that are much lighter. This model was run with varying subsets of MIDI files. The model did get better with bigger datasets at first. I started out with a smaller dataset to verify the size of the dataset that would be appropriate for GAN and because converting a huge dataset into MIDI images would be computationally expensive. Using 1500 MIDI images created a blank image that only played one note in the 13 seconds interval. An example of the generated image with 1500 MIDI images is shown in figure 4 (a).

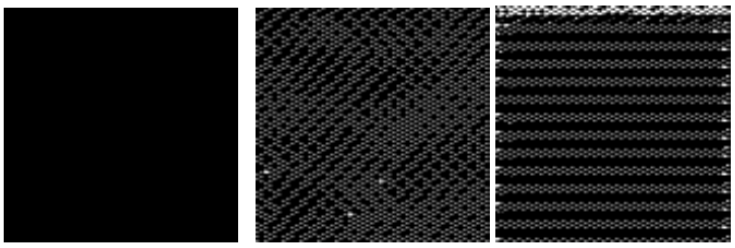

Fig. 4. MIDI images generated by GAN. (Left) With 1500 training images, (Middle) with all images in the dataset, (Right) with new improved model

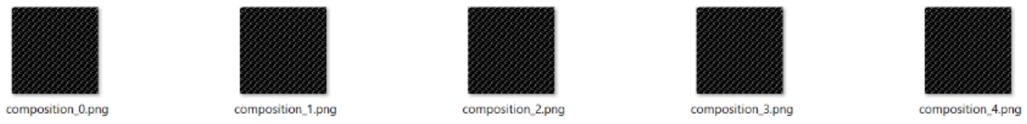

Fig. 5. An event of Mode Collapse when all the images generated look the same

Using 2000 MIDI images created more notes but were still random and didn't look like the original MIDI images. An example is in the figure 4 (b). Further training with varying batch size, epochs, learning rates and varying random noise size generated multiple images that looked and played exactly the same. At this point, I realized the model had encountered mode collapse which is a common issue with GAN models that prevents it from generating diverse images. An example with the four composition files that look exactly the same is shown in figure . To get around this, I tried a different architecture. In an attempt to make the model stronger, I added more dense layers between the convolutional layers to the generator model. I did that because there is no way to know how representative the output of the convolutional layers are going to be of the input images. For the discriminator, I added more dropout layers with higher dropout rates. WIth the new model architecture, the images generated were as seen in figure 4 (c), still with a lot of noise and noise-like white notes. After some research, I tried training with a different set of hyperparameters like smaller batch size 8 and 16 and noise vector of 128. The reasoning was that with a smaller batch size the discriminator won't be able to overpower the generator at the start of the training.

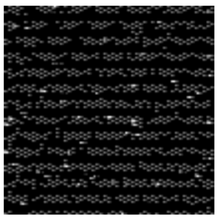

Fig. 6. MIDI image with the updated architecture, full dataset, batch size 16, noise vector size 128 and learning rate 1e-4

Finally, due to unsatisfactory results with the smaller datasets, I used the entire dataset of 28587 images. With this dataset and batch size 8, the models still produced a lot of grayish black patches that still didn't translate to a good MIDI file. Moving on to batch size 16, I was able to get the best result so far with noise vector 128, and learning rates 1e-4 for both the generator and the discriminator. The model was able to produce substantially more playable notes as shown in figure 6. Only the bright white ones are considered notes by the image to MIDI converter. With this configuration, a few images were generated after every 15 epochs.

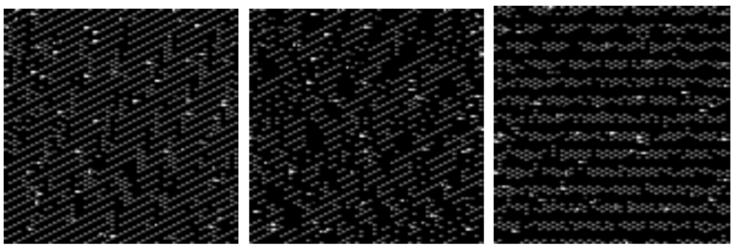

Fig. 7. Three MIDI images generated by the final GAN model

5. Results

The resulting images generated are as seen in figure 7. The background still has a lot of noise in the form of tiny gray blocks/bars. However, these did translate to clear full 13 seconds piano pieces. The notes in the piano pieces as evident by the images sounded very random but with rare instances where the combination of notes did make sense.

I have attached a few samples (both MIDI images and playable files generated by the generator. These can be played in the VLC player with an extension added to support MIDI files. These MIDI files usually start with a few stable starting notes, followed by plenty of random notes and finally end with a higher pitched ending note. The model does seem to have learned the usual starting and ending notes common to a lot of classical piano pieces.

6. Discussion

The model wasn't able to take the advantage of the huge dataset available to it. Considering that the variable dataset sizes and hyperparameter configurations didn't significantly affect the results of the model, the issue must be that the model architecture still wasn't strong enough. One of the most frequent roadblocks I encountered was mode collapse, where all the outputs generated were the exact same. Once I encountered this with the big dataset there was no way to get around it no matter the changes I made. GANs are supposed to be notoriously difficult to train. They are a pretty new concept to deep learning and despite extensive research related to these models, many topics still remain a black box. Many tips that can be found to improve the GAN did not seem to work for this particular purpose of MIDI images. It could be related to the fact that the details observed in the real MIDI images are very subtle white bars. The model does generate white bars/blocks but they are very few compared to the grayish ones which doesn't translate to playable notes.

7. Future Works

One of the things we could try to improve the model outputs is to use the Wasserstein loss function which is used in the WGAN(reference) paper or the RaLS (Relativistic Average Least Square) Loss function that are known to work better for GANs than the binary cross entropy loss. One of the common patterns I noticed with the generated output was the gray bards were arranged in evenly separated horizontal or slanted grids as seen in figure 8

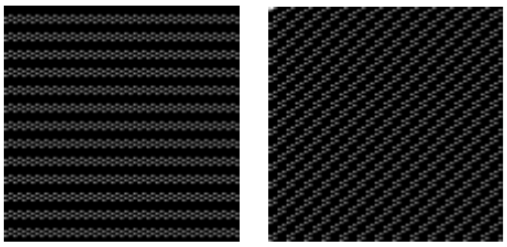

Fig. 8. Grid like structures in MIDI images

One of the solutions to overcome this would be to use the Upsampling layers in place of the Transposed convolutional layers. Transposed convolutions are known to produce grid-like checkerboard patterns while the Upsampling layers would be more appropriate to produce natural looking images. This is known to work with the image datasets like the Stanford Dogs datasets but I'm not sure if it would work with our MIDI images. Another issue with my model was that it was difficult to know when the model would converge. We could use the MiFID (Memorization-informed Fréchet Inception Distance) metric to evaluate the generated samples and use early stopping to get the best results. Lastly, although most of the GANs use ADAM optimizer for training, some GANs tend to work better with Gradient Descent optimizer, so with the combination of other optimizations, it would be worth trying this out too.

8. Conclusion

In this project, a GAN model was studied and dissected in order to generate classical piano pieces. After experimenting with various dataset sizes, hyperparameter configurations and a couple different model architectures, and settling for the ones that gave the best output, the model was able to generate MIDI files with playable notes enough to be considered a part of a song. However, these pieces of music weren't very compelling and sounded rather random.

Training two neural networks in an adversarial way was challenging as it was important and rather difficult to maintain the balance between the two. However, there are several measures to improve the model and there is much research devoted to introducing newer and better techniques for training them.

References

1. Goodfellow, Ian, et al. ”Generative adversarial networks.” Communications of the ACM 63.11 (2020): 139-144.
2. Dan, Yabo, et al. ”Generative adversarial networks (GAN) based efficient sampling of chemical composition space for inverse design of inorganic materials.” npj Computational Materials 6.1 (2020): 1-7. Sturm, Bob L., et al. ”Music transcription modeling and composition using deep learning.” arXiv preprint arXiv:1604.08723 (2016).
3. Karras, Tero, Samuli Laine, and Timo Aila. ”A style-based generator architecture for generative adversarial networks.” Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2019.
4. Sturm, Bob L., et al. ”Music transcription modeling and composition using deep learning.” arXiv preprint arXiv:1604.08723 (2016).
5. Briot, Jean-Pierre, Gaëtan Hadjeres, and François-David Pachet. ”Deep learning techniques for music generation–a survey.” arXiv preprint arXiv:1709.01620 (2017).
6. Chu, Hang, Raquel Urtasun, and Sanja Fidler. ”Song from PI: A musically plausible network for pop music generation.” arXiv preprint arXiv:1611.03477 (2016).
7. Mogren, Olof. ”C-RNN-GAN: Continuous recurrent neural networks with adversarial training.” arXiv preprint arXiv:1611.09904 (2016).
8. Yu, Lantao, et al. ”Seqgan: Sequence generative adversarial nets with policy gradient.” Proceedings of the AAAI conference on artificial intelligence. Vol. 31. No. 1. 2017.
9. Yang, Li-Chia, Szu-Yu Chou, and Yi-Hsuan Yang. ”MidiNet: A convolutional generative adversarial network for symbolic-domain music generation.” arXiv preprint arXiv:1703.10847 (2017).
10. Hadjeres, Gaëtan, François Pachet, and Frank Nielsen. ”Deepbach: a steerable model for bach chorales generation.” International Conference on Machine Learning. PMLR, 2017.
11. Dong, Hao-Wen, et al. ”Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment.” Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 32. No. 1. 2018.
12. Oord, Aaron van den, et al. ”Wavenet: A generative model for raw audio.” arXiv preprint arXiv:1609.03499 (2016).
13. Engel, Jesse, et al. ”Gansynth: Adversarial neural audio synthesis.” arXiv preprint arXiv:1902.08710 (2019).