Generating Electronic Music with Adversarial Networks

Ronak Malde
Stanford University
rmalde@stanford.edu

Abstract

Music generation using machine learning techniques has seen success when generating discrete sequences of notes (MIDI files) for a limited number of instruments, primarily using encoder-decoder models to generate notes. It is impossible, however, to generate imitations of electronic music, however, where there are hundreds of instruments and effects layered on top of one another. For this task, one must synthesize the actual audio signal of a song rather than generating discrete notes of a limited number of instruments. The WaveGAN architecture, a Deep Convolutional GAN for audio, has seen success in synthesizing the audio signal of human speech. In this paper we extend upon WaveGAN's architecture to generate short clips of electronic music. We also introduce elements from the StyleGAN paper into our architecture, such as AdaIN and progressive growing, and also experiment with a Transformer-based model and see an improvement in results. While the results are still far from human-produced music, these approaches seem to be promising.

1. Introduction

GANs have had large success in the domain of images, generating anything from objects to portraits. They have not, however, been explored equally in the space of audio, and particularly music generation. Previous success for music generation has been accomplished using sequence encoder-decoder models, such as with LSTM's and with Transformers (Music Transformer: Generating Music with Long-Term Structure (tensorflow.org)). The main limitation with this approach, however, it is generated MIDI outputs - an encoding of notes and durations - upon which an instrument (like a piano) can be played and layered with other distinct instruments. In modern music production, however, songs are composed of hundreds of different instruments and effects which are not easily decomposable. In order to generate realistic samples of music with multiple instruments, one must generate the actual sound data - something a sequence model is ill suited for. There has been one major work in this area, WaveGAN [1], that uses a Deep Convolutional GAN that achieved reasonable success in generating single sounds, like a one-word human speech, a drum kick, or a bird sound. This project aims to extend WaveGAN to generating longer pieces of music by improving the model architecture, using techniques that have found success in the domain of generating images.

1.1. Similar Work

Google's Magenta lab has several pieces of work on music generation using various techniques in deep learning. Their best model in 2019, called GANSynth [2], generated discrete music tokens for which any instrument of any timbre can be applied to. Thus, they were able to construct a polyphonic generation from these several distinct generations. There have also been other similar approaches to creating polyphonic music, such as by using transformers, as shown by OpenAI's MuseNet [6]. Unfortunately, this approach limits the generation to a discrete set of instruments for which the timbre must be applies. This project aims to also use GANs, but to generate the actual raw audio file instead.

The paper MP3Net [7] attempts to generate longer pieces of music, at a minute-long with longer time dependencies. It converts the audio into a spectrogram, a 2d representation in the frequency domain. While spectrograms have been useful for classification tasks, they lose a lot of information in the conversion process from the raw audio, and so it is difficult to reproduce audio from just the spectrogram. I think that a combination of these two approaches could be useful, where a DCGAN is used to generate the raw audio, but spectrograms are used to encode long term information.

Amazon released a paper this year using Transformers and GANs in conjunction to generate music[4]. The paper actually uses a pretrained BERT module, which is used for NLP tasks, on the task of music generation. Transformers are generally better for encoding time dependencies, so I plan on experimenting with a similar architecture, but generating the raw audio samples.

Finally, there is a paper that uses progressive growing in GANs to generate cleaner audio [5]. Their approach still

generates in the pitch domain, but at a much higher fidelity that could potentially imitate music. Our project aims to accomplish the task with the raw waveform data rather than in the pitch domain, but because of the success of progressive growing in this paper, that is something to consider to improve our model.

2. Data

The dataset is the Beatport EDM Key Dataset [**?**], made by the company Beatport, an American online store for distributing music. Electronic dance music (EDM) songs were chosen for the project because the genre contain layerings of many different instruments, oftentimes synths that can be chained together and indistinguishable. As a result, the model will have to generate the entire sample rather than individual instruments. The dataset contains 1,486 two-minute sound excerpts from EDM songs in mp3 format with a sample rate of 16000 Hz. The data is free to download as a zip file.

2.1. Dataset Preprocessing

The data was resampled at a standard 16kHz (in case there were clips from a different sampling rate), meaning that each second of audio will have 16,000 data points. I then randomly sampled 4 clips from each song to build the dataset. Each clip was approximately 4 seconds long, more specifically 65536 data points long. This number was chosen because it is 2^{16}, which will allow for easy upsampling using transpose convolutions.

The data was then split 80/10/10 into train, validation, test respectively.

3. Approach

3.1. Baseline Model

As a baseline model, I implemented the model from the WaveGAN paper discussed above, because it had achieved success in the domain of sound generation. The limitation of WaveGAN is that it only had success with generating single sounds, such as one word of human speech, or a single bird sound. WaveGAN uses a very similary architecture as the original DCGAN, but with some modifications to apply to the 1-dimensional audio domain, rather than for iamges.

The generator of WaveGAN starts off with a fully connected layer that has the noise vector (of length 100 in my implementation) as an input. Then there are 6 1-dimensional transposed convolution layers, which each upsample at a factor of 4, which then finally produces at 65536-length sequence.

The discriminator matches the generator's architecture with 6 1d convolutions.

For the loss, I used BCE with Logits Loss and Adam optimizer for both the discriminator and the generator. I started with the discriminator having a 3x larger learning rate than the generator, as did the WaveGAN paper, but I tuned this on the dev dataset during future experiments.

3.2. Improvements to Baseline

In preliminary training, the baseline model's generator suffered from mode collapse, and many of the generated outputs sounded similar. To mitigate this, I altered the baseline model in three ways. First, I introduced Spectral Normalization to the discriminator, which limits the Lipschitz constant of the discriminator and can help prevent mode collapse. Second, I used Wasserstein loss with gradient clipping rather than BCE loss to keep giving the generator useful information even if the discriminator learned much faster. Third, I added a learning rate warmup to both the generator and the discriminator, to prevent mode collapse early in training.

3.3. Adding StyleGAN architecture

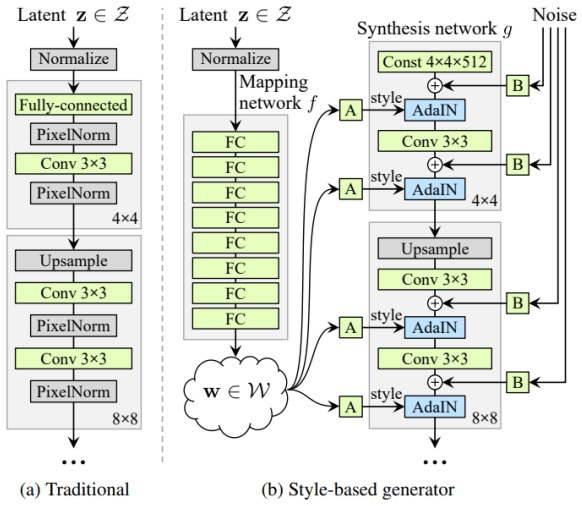

Figure 1. StyleGAN architecture

One major fallback of the baseline was the noisy-ness of the output. To a human ear, the generated music does not sound "clean" like normal produced music, because there are random artifacts and noise that muddies the output. To solve this, I implemented a similar approach to StyleGAN's AdaIN layers. I constructed a fully connected network of 6 layers, where the output of each layer is fed into the transpose convolution of the generator at each step, along with an additional noise vector.

3.4. Risky Experiment: Transformers

Transformers have seen success in almost every area of machine learning, so I wanted to experiment and see if I

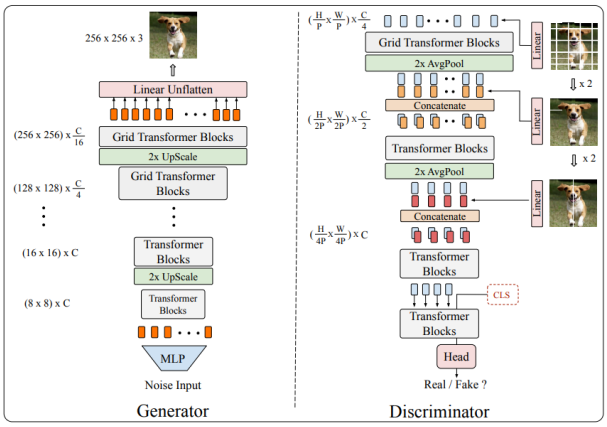

Figure 2. TransGAN architecture

could obtain meaningful results incorporating Transformers to generate music. TransGAN [3] found impressive results constructing a GAN using Transformer encoder layers applied to image generation. For this experiment, I adapted the TransGAN architecture to work for 1-dimensional audio data rather than 2d image data. My model is constructed of a MLP which feeds into 6 transformer encoder layers which are upsampled using interpolation.

I also attempted to combine the Transformer layers with my existing DCGAN architecture by making the final two layers of the DCGAN Transformer layers instead, but this led to instant mode collapse during training and I did not receive meaningful results from this architecture.

4. Experiments

4.1. Training

All models were run for 100 epochs on the train dataset, and the final metrics were measured on the test dataset. The val dataset was used for intermediate evaluation for hyperparameter tuning, such as adjusting learning rate and warmup steps. An example loss curve during training is shown in Figure 1. Note that in Milestone 2, I gathered results from training for 20 epochs, but I found that a much longer training time yielded far better results, so I retrained for 100 epochs for all models.

4.2. Evaluation methods

Pitch Accuracy and Pitch Spread It is difficult to quantitatively measure the effectiveness of a generator, particularly on a domain like music. The paper GANSynth [2] Google Magenta introduced a measure of Pitch Accuracy and Pitch Spread. Ideally, a piece music will have a mean centered in the range of 800 to 8000Hz, as is with most EDM music. Additionally, electronic dance music typically has a high spread of instruments with different pitches. Thus, we need a measurement that rewards the model for having a high range in pitches generated.

To accomplish this, I used the Crepe python library, that uses a CNN model to output the pitch of a song (in Hz) at each millisecond. I took the logarithm of these pitches (because humans perceive pitch logarithmically),and took the mean and standard deviation of 1000 generated pieces for each experiment.

Human scoring At the end of the day, generated music is for the appreciation of the listeners, and so the best judge of generated music would be a human judge. I set up an evaluation pipelines that generated 100 samples for each model, and I assigned a score of 0 to 5 on the quality of music. The criteria of the "quality" of the music were on: 1) how "real" the audio sounded, and 2) how "pleasant" the clip was to listen to.

This score is entirely subjective, but these findings serve as a point of reference for the music. As a note, I also blindly mixed generated and real samples of music, and all the real samples received a score of 5.

All of the metrics for the different models are summarized in Table 1. Note that the StyleGAN model did not include Spectral Norm and Warmup, but it did include W-loss. Different combinations of these two approaches can be explored in future experiments.

4.3. Analysis

The baseline received an extremely low human score, and additionally the mean pitch was extremely low, and the standard deviation was low. This is likely because of the mode collapse that occurred in the baseline model. There were several generated samples that were all noise, and many at lower pitches.

Adding the spectral norm, warmup, and W-Loss seemed to fix a lot of these issues of mode collapse, leading to a better mean pitch, and a higher standard deviation of pitch and there were less samples that sounded identical. These improvements achieved the highest human score, samples sounded decent, however they still had a lot of noise, and sounded as if they were coming from a faraway radio station.

The StyleGAN architecture achieved similar results to the second model, but there was a much higher standard deviation in pitch, which was also noticeable when listening to the samples. There was also far less noise in the StyleGAN model compared to the other models, but a slight decrease in how coherent the music sounded.

Finally, the Transformer model actually achieved decent results, and many of the samples sounded similar to the DCGAN approaches. There were frequent samples generated that were just pure noise, however, which brought down the human score for the Transformer.

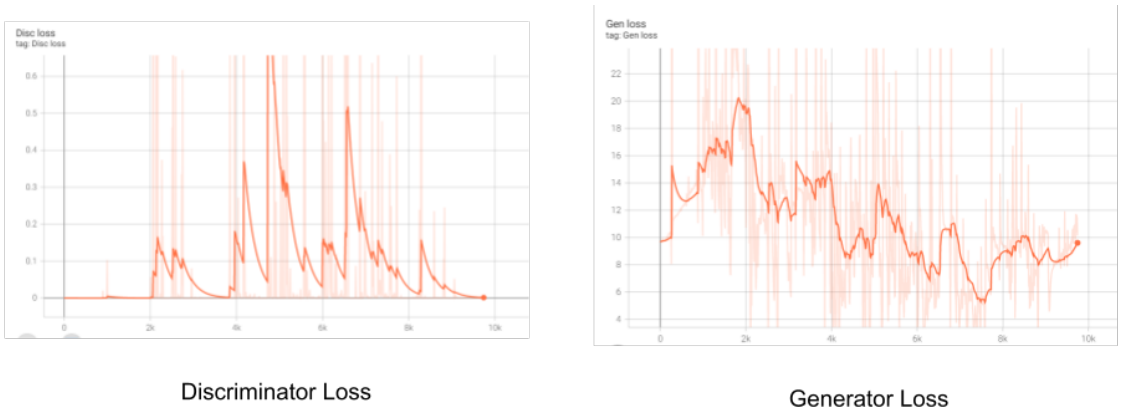

Figure 3. Example Loss Curves

Model	Pitch Mean (Hz)	Pitch std (Hz)	Human score
Baseline	66.21	755.4	1.2
Improved	8264	4410	3.5
Add StyleGAN architecture	5899	6347	3.4
Transformer GAN	6932	4309	2.9

Table 1. Models on evaluation metrics

These results show promise for both the DCGAN and Transformer approaches. Both types of models were extremely volatile with hyperparameters, and sometimes adjusting a few parameters such as learning rate and warmup would be the difference betwee mode collapse and a successful model. Additionally, these models continued to improve at every epoch and I believe that given more time and resources, we could achieve even better results by simply training for more time with exisitng model architectures.

One main drawback for train time is the W-loss with gradient clipping. Calculating the gradient in the loss function is extremely time consuming, and was a large obstacle for training for a longer time.

5. Conclusion

While the model had reasonable success in generating music, it is still far from generating convincing EDM music that could fool a human. Both model categories, of DCGAN's and Transformer GANs produced promising results, but more training time and stability against mode collapse are needed to produce exceptional results, as have been shown with image generation with similar models. This is definitely an exciting area for GANs, however, and there is a lot of unexplored and untapped potential to generate great music.

5.1. Future work

It would be great to explore generated longer lengths of sound clips, rather than just being limited to 4-second clips. While this probably won't result in meaningful results with our current training scheme, it could be possible to also interject transformers at intermediate steps to encode long-term information about the music, perhaps by using spectrograms.

For evaluation, I also would like to have a more robust quantitative measurement for the effectiveness of the outputs. I will consider using an Inception score using a trained musical genre classifier.

Finally, it would be interesting to enforce musical rules upon the generated music, to perhaps teach the model to stay within a certain tempo or scale, as music does. This could perhaps lead to more realistic generation of music if some of these rules are predefined.

References

[1] Chris Donahue, Julian J. McAuley, and Miller S. Puckette. Synthesizing audio with generative adversarial networks. *CoRR*, abs/1802.04208, 2018.

[2] Jesse H. Engel, Kumar Krishna Agrawal, Shuo Chen, Ishaan Gulrajani, Chris Donahue, and Adam Roberts. Gansynth: Adversarial neural audio synthesis. *CoRR*, abs/1902.08710, 2019.

[3] Yifan Jiang, Shiyu Chang, and Zhangyang Wang. Transgan: Two pure transformers can make one strong gan, and that can scale up, 2021.

[4] Aashiq Muhamed, Liang Li, Xingjian Shi, Suri Yaddanapudi, Wayne Chi, Rahul Suresh, Zachary Lipton, and Alex Smola. Transformer-gan: Symbolic music generation using a learned loss.

[5] Manan Oza, Himanshu Vaghela, and Kriti Srivastava. Progressive generative adversarial binary networks for music generation. *CoRR*, abs/1903.04722, 2019.

[6] Christine McLeavey Payne. Musenet, Jun 2021.

[7] Korneel van den Broek. Mp3net: coherent, minute-long music generation from raw audio with a simple convolutional GAN. *CoRR*, abs/2101.04785, 2021.