Time Domain Music Synthesis with Sample-GAN

Michael Wisnewski
University of Iowa
Iowa City, IA 52242
mwisnewski@uiowa.edu

Abstract

In this paper I propose a novel model for sample level audio synthesis. I combine the two approaches of LSTMs and GANs into one model. I believe that the joining of these two often separate concepts may yield realistic and recognizable audio. Though I wasn't able to generate the results I had hoped for, I believe that there may be promise with this approach.

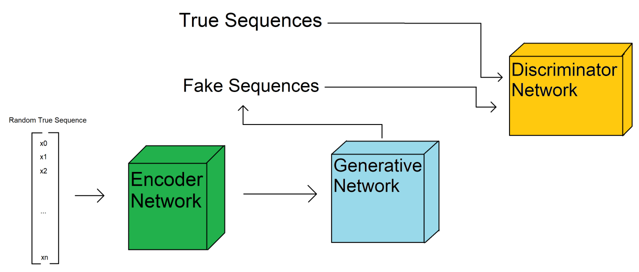

Figure 1. Generalized architecture

1. Introduction

Music is one of the few remaining artistic fields in which deep learning continues to struggle. Two paradigms dominate current research today: training GANs to generate MIDI (Musical Instrument Digital Interface) sequences, and training RNNs to generate individual samples. The MIDI approach has shown promise for generating simple melodies. Due to MIDI's symbolic nature, it is not possible to generate results that are not recognizable as music. It is significantly harder to generate samples that sound cohesive when played back. With recent advances in models like SampleRNN, the time domain has become closer to be being mastered. The last hurdle that remains is the presence of significant amounts of static that break the immersion.

Generating time domain music comes with many techincal problems that must be accounted for, primarily the sheer size of the dataset. Modern recording samples at 44,100 Hz, meaning that one second of audio contains 44,100 individual samples. To generate even a tenth of a second of audio requires 4,100 samples. The high dimensionality of the data demands larger networks that require more memory and computing resources. This isn't inherently a problem, but it does result in longer training times. SampleRNN avoided this by encoding the generator output, then decoding it to playable waveform. I chose to avoid this complexity and see if a moderately larger network could handle this data.

In this work I investigate using an approach similar in concept to SampleRNN, but training it with a GAN architecture and seeding the generator with encoded audio rather than random noise. In order to preserve the time dependant patterns, I implemented LSTMs into the generator and discriminator networks in the architecture, however I use a U-Net like structure for encoding. I avoid the attention mechanisms present in SampleRNN for simplicity sake, and to see if they are even necessary when working with GANs. In GAN image generation, Huang *et al.* have shown in their 2018 paper *IntroVAE* that seeding the generator with encoded audio drastically increases the photo-realism of the output images. It tends to replace the blurry space surrounding the focus point with high resolution backgrounds. I theorize that this blur is equivalent to the static present in state of the art audio generation. It is possible that performing the audio analog of this technique can remove unwanted noise.

The goal for this model is that it is trainable on a wide variety of music. The model should be able to learn the patterns present in the training set and recreate them, regardless of the type of music supplied. This may be able to give us insight into song structure and *"what works"* in music that may not necessarily be articulated yet.

2. Sample-GAN Model

First, the auto encoder network is trained. This network should be entirely unaffected by the GAN training and should always produce as high quality encoding as possible. The encoder is passed the training set and forced to reduce the dimensionality. This is then decoded by the decoder and compared to the source data. This process is entirely self supervising and could be accomplished relatively quickly.

When training, a random segment of audio is selected and encoded. This encoding is then sent to the generator to seed the generation. This encoded vector replaces the typical random noise vector used for seeding. The generator holds the last *n* samples and generates the next sample based on this array. As samples are generated they are shifted into this array and the most distant is shifted out. The seed is initially placed into this array.

The generator generates *m* sequences of *n* length. These sequences, along with *m* random sequences from the training set are sent to the discriminator. The discriminator predicts if a given sample is real or fake. Based on the discriminators accuracy the weights of both the discriminator and the generator are updated accordingly.

2.1. Dataset Generation

The data set was generated by splitting up the input songs into segments of *n* samples. I had ease of use in mind, so I included scripts that allow a user to import their own songs. It accepts either mp3 or wav files, converts to wav if necessary, then performs the splits to generate the dataset. The dataset generation tool assumes that input files are 44.1 kHz sample rate and 16 bit audio, but will convert to mono as necessary. There is a stride of *n* as the splitter moves across each song, meaning that it will generate sequences of [1, 2, ..., n], [n+1, n+2, ..., n+n], etc. This was done purely to reduce the amount of data generated, as a stride of 1 simply produces too much to store in RAM. This also allows for easy implementation of stateful LSTMs. For simplicity sake, the current LSTMs are completely statless, but this dataset structure would work perfectly with a stateful model as well.

2.2. Data Normalization

Wav files are comprised of 16 bit signed values, so they can range from -32,768 to +32,768. Normalizing the data is as easy as dividing the raw data by 32,768. This produces segments with a range of -1 to +1. The generator produces samples between -1 and +1 that can then be upscaled by 32,768 to produce a playable Wav file.

2.3. Python Libraries

The model was first developed in pure tensorflow, but after encountering difficulties, I switched to Keras. Keras allowed for easy and quick changes to the model during my testing phase. It was much easier to change hyper-parameters, as well as train the models using their built in ”fit” function. While Keras is not production ready, it was certainly worth the trade off to allow for this fast and dynamic prototyping. Keras also allowed for easy implementation of the cuDNNLSTM layer. This is identical to the standard LSTM layer, but it is executed on the GPU. This drastically reduced training time.

The Pydub library was used to load mp3s from disk and save them as wavs. Pydubs AudioSegment class is extremely robust. mp3s were loaded here, converted to mono, then saved as wav files. Scipy was a little more efficient and easy to split the files, so these converted wavs are then loaded back in and split into segments. Scipy is also used to write the generated segments back into playable wav files.

3. Methodology

Purely for consistency and convenience, every iteration of the model was trained on the same album. The only mp3 album I had access to was an obscure progressive metal album, but in retrospect it may have been wise to use more simple music, perhaps classical guitar or singer-songwriter music. The patterns present in these genres of music are more consistent than those present in progressive music.

The *n* value mentioned several times previously refers to the size of the segments. In my testing, I largely used $n = 500$, but I also experimented with $n = 1000$ and $n = 4410$. Mehri *et al.* (2017) found in their SampleRNN paper that increasing *n* will decrease convergence time and increase quality, but requires significantly more memory. Their ”sweet spot” was $n = 512$, or about 32 ms of audio. Their source audio is 16 kHz sample rate, so they need less samples to fill more time. This certainly makes the data more manageable and is something I will consider doing in the future. There may simply be too much information present in 44.1 kHz audio.

Further testing is certainly needed to fine tune these parameters. A 440 Hz tone sampled at 44.1 kHz has a period of about 100 samples. This means 5 wavelengths are included just to determine the next sample. Music contains a wide range of frequencies, anywhere from about 80 Hz to nearly 10 kHz. This means it is likely that an enormous amount of data from ”old” tones is being accounted for when predicting the next. It is hard to say whether or not this is harmful, but it is certainly worth further research.

3.1. Network Design

Both the discriminator and generator networks consist of two LSTM layers stacked on top of one another. This allows retention of time-dependent patterns when learning patterns that occur over large time periods. I also experimented with performing minor compression on the input data before passing it into the LSTMs, by performing a 1D convolution and max pooling operation. This appeared to have little to no effect on the results, so I removed them to reduce the number of trainable parameters. Each network culminates in a dense feed-forward layer that determines either the validity of the segment or the value of the generated sample.

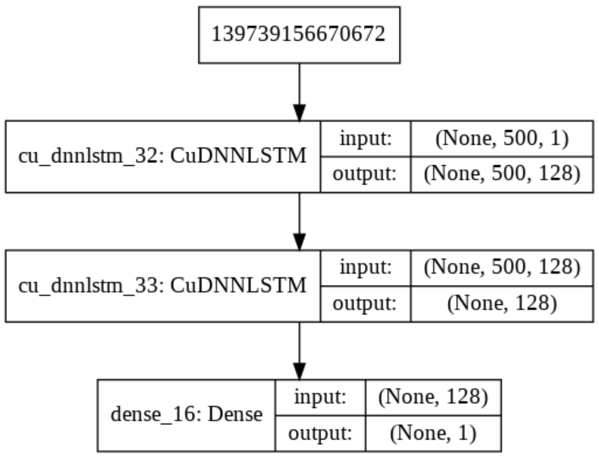

Figure 2. Discriminator Architecture

Since the autoencoder receives an input vector of constant size, it was unnecessary to use LSTMs. I instead compressed the data with a series of convolution and max pooling operations, similar to a U-Net but on a smaller scale. The decoder is used just for training the encoder network, so only the encoder shall be shown.

4. Results

I experienced first hand just how difficult GAN architectures can be to train. The training process for an entire album took about an two hours, and generating one second of audio took about 30 minutes. These long train and test times made it very difficult to see if the model was truly

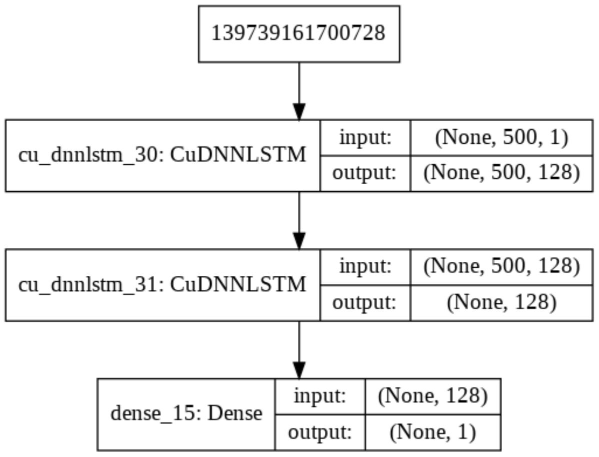

Figure 3. Generator Architecture

learning. The process of training, generating, and testing new structures generally took a little over two hours. In the end, I discovered that it was unlikely the model was learning anything of meaning.

The early results were completely silent, 0-filled wav files. They produced no sound and contained no amplitudes. This was caused by a simple bug that was easily fixed. My next round of generation was not much better. The model produced a small transient ”click” at the beginning of the track, then would converge to a positive value and cease all movement. This also produces a largely silent track. Examples are shown below in Figure X.

What is interesting to me is that the direction of the transient and the convergence value are different across all generations. This leads me to believe that the model is in fact generating unique points from the noise, but that there may be a bug in the generation that forces it towards some value. I searched for what could be causing this, but I never managed to find anything. It is possible that the use of two separate libraries for handling the audio data may result in bad test data, but I never found anything promising with this theory. It also seems that the convergence is complete once the generator’s starting noise vector has been entirely shifted out of its memory. It certainly warrants further investigation to discover what is causing this. It is entirely possible that the model is functioning but a bug is limiting the output.

5. Related Work

This research is largely based on the Mehri *et al.* (2017) paper: *SampleRNN*, and Olaf Mogren’s 2016 paper: *C-RNN-GAN*. SampleRNN is currently the most advanced au-

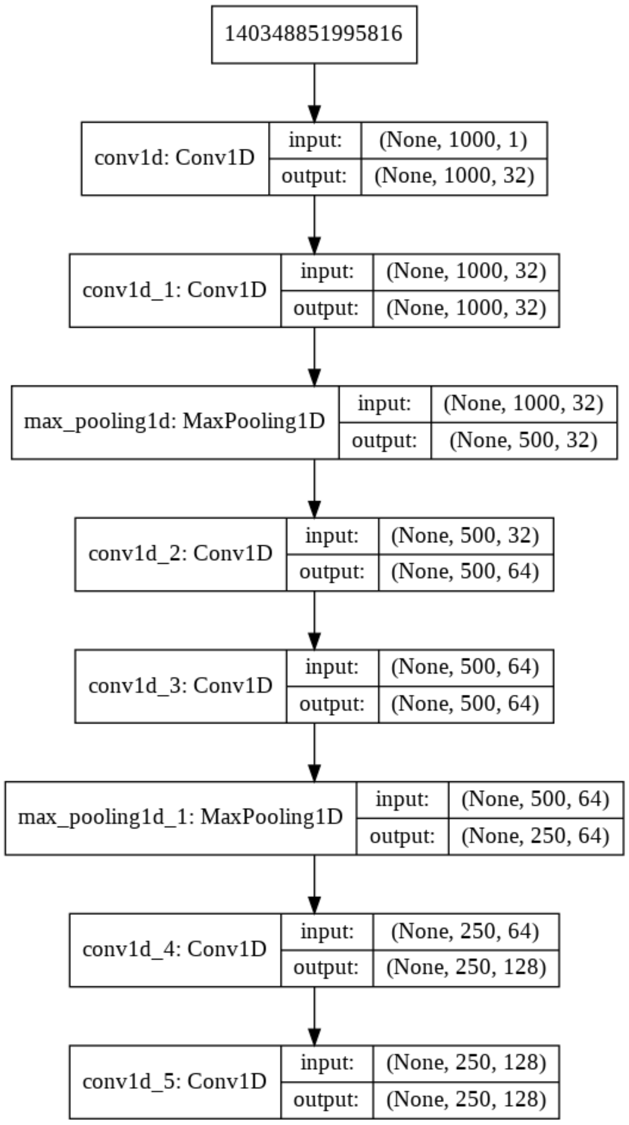

Figure 4. Encoder Architecture

dio synthesis model known, and it uses RNNs exclusively to achieve this. It produces very convincing audio when trained on human speech and classical piano pieces. Cj Carr and Zack Zukowski set to test the limits of SampleRNN by exposing it to ”outliers in human music” in their 2017 paper *Generating Black Metal and Math Rock*. They discovered that, when exposed to complex modern music, SampleRNN begins to break down slightly. The results were still fascinating and truly impressive, but static and noise began to overwhelm the intended music.

Mogren experimented training RNNs with GAN to produce MIDI sequences. I based my architecture on what he found in his research, but generated and trained on samples rather than MIDI. His results were groundbreaking at the time but have since been surpassed. The output samples he supplies contain little to no ”musical direction” or melody. This paper showed that there is potential for these types of models, so I strived to expand his work into the time domain.

Huang *et al*. (2018) *IntroVAE* found that seeding generative networks with encoded vectors rather than noise lead to significantly higher resolution image synthesis, to the point of being photo-realistic. Many models at the time that were trying to generate human faces contained very clear faces surrounded by blurry artifacts. This method removed this blur and added further detail to faces. As I stated earlier, I hypothesize that the blur surrounding these images is the pixel equivalent of static/noise in audio. By seeding the generator with encoded audio, it may be possible to produce extremely realistic audio.

6. Conclusion

I have introduced a novel model for audio synthesis: Sample-GAN. Sample-GAN’s aim is to produce high quality audio similar to that on which it is trained. It contains a generator and discriminator that play a minimax game, slowly improving one another until the generation is indiscernible from real music. While Sample-GAN did not achieve what I set out for it, I still believe that this model is capable of generating recognizable music. I have learned a significant amount about training GANs and believe that with further research this model could be optimized or restructured to achieve the desired goal. This model may easily be extended with stateful LSTMs, which may lead to significant increases in learning and yield realistic output.

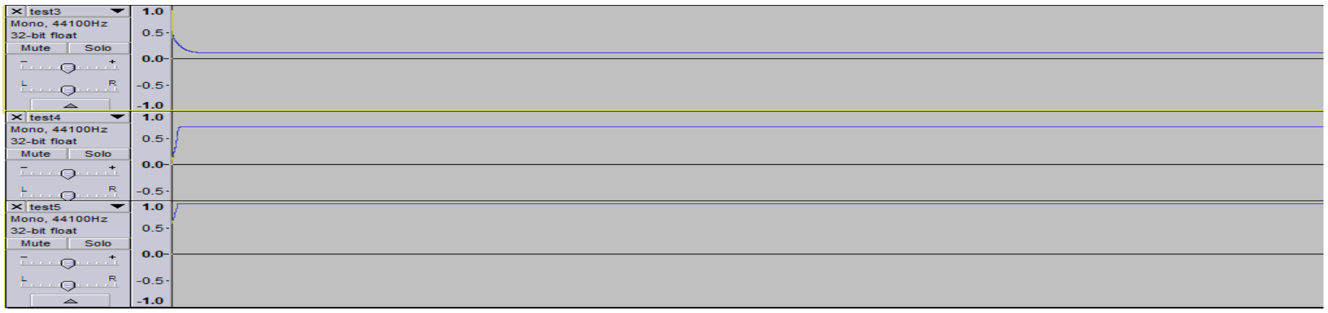

Figure 5. Sample of Generated Outputs

References

Soroush Mehri, Kundan Kumar, Ishaan Gulrajani *et al*. SampleRNN: An Unconditional End-To-End Neural Audio Generation Model. *arXiv:1612.07837v2 [cs.SD]*. 11 Feb 2017

Olof Mogren. C-RNN-GAN: Continuous Recurrent Neural Networks with Adversarial Training. *arXiv:1611.09904v1 [cs.AI]*. 29 Nov 2016

Huaibo Huang, Zhihang Li, Ran He, Zhenan Sun, Tieniu Tan. IntroVAE: Introspective Variation Autoencoders for Photographic Image Synthesis. *arXiv:1807.06358v2 [cs.LG]*. 27 Oct 2018

Zack Zukowski, Cj Carr. Generating Black Metal and Math Rock: Beyond Bach, Beethoven, and Beatles. NIPS 2017

Chris Donahue, Julian McAuley, Miller Puckette. Adversarial Audio Synthesis. *arXiv:1802.04208v3 [cs.SD]*. 9 Feb 2019

Jongpil Lee, Jiyoung Park, Keunhyoung Luke Kim, Juhan Nam. Sample-Level Deep Convolution Neural Networks for Music Auto-Tagging Using Raw Waveforms. *arXiv:1703.01789v2 [cs.SD]*. 22 May 2017

Aaron van den Oord, Sander Dieleman, Heiga Zen *et al*. WaveNet: A Generative Model for Raw Audio. *arXiv:1609.03499v2 [cs.SD]*. 19 Sep 2016