ReverbSynth: Applying GANSynth to Audio DSP

https://github.com/ghanashyamvtatti/reverbsynth

Ghanashyam Vinod Tatti
gvt217@nyu.edu

William Bresee
wb630@nyu.edu

December 2020

1 Introduction

The problem of producing commercially viable or artistically satisfying music with either traditional artificial intelligence tools or cutting edge machine learning techniques is an open one, composed of many significantly challenging subproblems. Paramount among these subproblems is the puzzle of generating the audio samples that comprise modern music, along with replicating signal processing techniques typically associated with modern music production. It is this particular domain that we chose to explore for our deep learning research project, defined specifically as the problem of generating musically viable audio samples and replicating traditional digital signal processing techniques with machine learning.

Recently, autoregressive models, such as WaveNet[11], have been successful at generating high quality audio samples. These models generate audio at a sample-by-sample level. As a result, this genre of generative model is slow and resource-intensive. Concurrently, monumental progress has been made in the comparable problem of generating images with Generative Adversarial Networks, known as GANs. The structure of images, and in particular the relationships between pixels, enable GANs[6] to employ convolutional layers to great success. This method of generating media is significantly faster, as it allows for the generating of media in parallel, rather than generating individual samples in serial.

While researchers have successfully used GANs to generate comprehensible, realistic images, few have had success generating audio. The periodic nature of pitched sound means that common strategies for generating images don't translate to the audio domain easily. Google's Magenta team, however, has demonstrated success in synthesizing musically viable audio with GANs, as documented by the GANSynth paper[4]. Specifically, the Magenta team was able to achieve best results by converting audio signals into mel spectrogram representations of frequency and amplitude axes. As a result, machine learning architectures which lend themselves to image generation, specifically GANs employing convolutional layers, could be employed to generate audio signals in this less-traditional format. As a result, the GANSynth model is able to generate audio with the efficiency associated with GANs-based image generation models, while avoiding the pitfalls inherent in trying to generate a more traditional periodic amplitude-over-time audio signal in parallel.

Our goal with this project was to extend the innovations of the GANSynth experiment to digital signal processing. As discussed in the GANSynth paper for ICLR 2019, "GAN researchers have made rapid progress in image modeling by evaluating models on focused datasets with limited degrees of freedom, and gradually stepping up to less constrained domains." We intend for this project and paper to represent a gradual step to a new, less constrained domain – namely, the broader domain of both processed and unprocessed audio. Specifically, we explore training the GANSynth network architecture on a dataset which includes processed and unprocessed audio, with the intention of training the model to replicate processed, or affected, audio signals.

2 Literature Survey

Significant research has been conducted with the aim of training models to generate original music. Of course, this domain is benefitted implicitly by advances in the parallel domain of generating nonmusical audio. In 2016, DeepMind researchers unveiled WaveNet[11], a model capable of generating raw audio waveforms which effectively mimic the human voice. WaveNet is an autoregressive, fully convolutional neural network. As a result, it very effectively captures and uses the wide range of timescales inherent to raw audio. As previously discussed, however, this approach only allows for the generation of one sample at a time. As a result, while effective, WaveNet is extremely slow and computationally expensive. As a result, some researchers, including yours truly, have turned to GANs, seeking a faster and more efficient tool for audio generation.

As early as 2016, researchers were exploring the utility of GANs for music composition. In his 2016 paper, entitled C-RNN-GAN: Continuous recurrent neural networks with adversarial training[9], Olof Mogren describes the architecture and training of generative adversarial networks with MIDI data. We believe this paper to be an important foray into GANs-generated music because it establishes that GANs are capable of improving when dealing with sequential data such as music. Additionally, it establishes that music composition with GANs is an open and difficult problem space. MIDI data is arguably an ideal form of musical data for machine learning purposes; MIDI contains no figurative or literal noise, exists in an easily comprehensible numeric format, and, while sequential, is not periodic in the same way that audio waveforms are. For all of these reasons, MIDI data should also be far more easily generated than waveform data; it is not, however, easily generated, as evidenced by the quality of the musical compositions resulting from this experiment. Thus, generating music is a challenging and arguably unsolved task, even when simplified to the domain of MIDI data. This difficulty justifies the separation of music composition into many smaller subproblems, including that which we deal with in this paper: the problem of generating short, processed audio samples.

Before addressing within the context of this section the success of the GANSynth experiment by Magenta, we must acknowledge the relative success of earlier attempts to generate audio data. Notably, Chris Donahue, Julian McAuley, and Miller Puckette produced a ICLR 2019 paper entitled Adversarial Audio Synthesis[3], which immediately preceded the GANSynth paper and defined both GANs architectures for handling both waveform and spectrogram audio data. This paper deals with a variety of data, including nonmusical data, and the results of this paper are nonmusical; specifically, the paper establishes that, trained on a dataset of spoken numerical digits, both models are capable of generating audible, comprehensible audio data simulating spoken numerical digits. Thus, audio, and not mere MIDI data, can be synthesized with GANs. Though the specific application of the tools documented in this paper is nonmusical, the success naturally implies that comparable success in musical applications is probably attainable. Interestingly, the researchers behind this paper, upon investigation, found an anecdotal preference for the audio generated by the model trained on waveform data, rather than that generated by the model trained on spectrogram data. This anecdotal preference is somewhat in conflict with the findings of the GANSynth paper, namely that mel spectrograms produced the best results.

Finally, we must herein formally acknowledge the paper upon which our research has been based, thus far referred to as the GANSynth paper. Jesse Engel, Kumar Krishna Agrawal, Shuo Chen, Ishaan Gulrajani, Chris Donahue, and Adam Roberts produced the ICLR 2019 paper entitled GANSynth: Adversarial Neural Audio Synthesis[4]. This paper represents a musical parallel to the previously mentioned nonmusical paper; indeed, we suspect extensive conversation between the discoveries documented within the two papers, particularly as the papers have an author in common. This paper describes a GANs architecture, dataset, and training process which produced a model capable of generating musically pleasing audio samples. The model was trained on the Nsynth dataset of four second recordings of individual notes from a wide range of instruments, and is capable of producing comparable four second musical notes. The authors extended the discoveries of this paper to produce a system capable of generating audio realizations of MIDI compositions. We leaned heavily on the architecture described within this paper, and the associated code, as we conducted our own experiments, with the intention of extending the success of this paper to processed and effected audio samples.

We must also herein acknowledge the foundation of our dataset, Magenta's NSynth dataset[5]. Research

in the image recognition and generation domain has benefited greatly from standardized, readily available datasets, often including labels. Magenta recognized the need for a comparable tool in the musical audio domain, and unveiled the NSynth dataset in 2017. It is with this dataset that the original GANSynth model was trained, and our research has relied heavily on processed and unprocessed versions of a subset of this data.

3 Technical Details

3.1 Dataset

We use a subset of the NSynth dataset, both directly and after processing samples via SuperCollider. The NSynth dataset provides data in tfrecords and wav + json formats. While the original NSynth dataset has many features, the only ones used by GANSynth[4] are the following:

Feature	Type	Description
audio	[float]	A list of audio samples represented as floating point values in the range [-1, 1]
pitch	int64	The 0-based MIDI pitch in the range [0, 127]

Table 1: Dataset features

For our project in particular, we work with the subset of the NSynth training dataset used by GANsynth. This subset includes samples of a variety of musical instruments, restricted to a limited pitch range. We have slimmed this dataset down to a single instrument group: bass, and have augmented it by creating a version of every sample which has been processed by a SuperCollider-based[2] reverb.

Number of audio files	400
Number of bass acoustic samples	200
Number of bass reverb samples	200
Length of each sample	4s
Sample Rate	16kHz
Dimenstions	64,000

Table 2: Dataset details

The dataset is finally converted into tfrecords before it is used as input for training the model.

3.2 Model

We will be using the GANSynth[1] model using our modified dataset. It adapts progressive training methods[8] to generate audio spectra.

In brief, the model samples a random latent vector from a spherical Gaussian and runs it through a group of transposed convolutions to upsample and generate the output data. This data is fed into a discriminator network whose architecture mirrors the generator's to measure the divergence between the real and generated distributions. A gradient penalty[7] is used to promote Lipschitz continuity.

The audio is encoded as spectral representations using STFT magnitudes and phase angles. The "instantaneous frequencies" (IF) of these along with their log magnitudes are transformed to a mel frequency scale ("IF-Mel"). While the model does support other spectral representations, it was found that IF-Mel is

the least lossy representation.

The generator and discriminator both use Wasserstein loss[7].

In addition to the audio, a one-hot encoded representation of the musical pitch is appended to the latent vector to achieve tweaking of both pitch and timbre. An auxiliary classification loss[10] is added to the discriminator that tries to predict the pitch label.

Tables 3 and 4 describe the layers in the generator and discriminator architectures respectively.

Layer	**Output Size**	**Kernel Width**	**Kernel Height**	**Kernel Filters**	**Nonlinearity**
concat(Z, Pitch)	(1, 1, 317)	-	-	-	-
conv2d	(2, 16, 256)	2	16	256	PN(LReLU)
conv2d	(2, 16, 256)	3	3	256	PN(LReLU)
upsample 2x2	(4, 32, 256)	-	-	-	-
conv2d	(4, 32, 256)	3	3	256	PN(LReLU)
conv2d	(4, 32, 256)	3	3	256	PN(LReLU)
upsample 2x2	(8, 64, 256)	-	-	-	-
conv2d	(8, 64, 256)	3	3	256	PN(LReLU)
conv2d	(8, 64, 256)	3	3	256	PN(LReLU)
upsample 2x2	(16, 128, 256)	-	-	-	-
conv2d	(16, 128, 256)	3	3	256	PN(LReLU)
conv2d	(16, 128, 256)	3	3	256	PN(LReLU)
upsample 2x2	(32, 256, 256)	-	-	-	-
conv2d	(32, 256, 128)	3	3	128	PN(LReLU)
conv2d	(32, 256, 128)	3	3	128	PN(LReLU)
upsample 2x2	(64, 512, 128)	-	-	-	-
conv2d	(64, 512, 64)	3	3	64	PN(LReLU)
conv2d	(64, 512, 64)	3	3	64	PN(LReLU)
upsample 2x2	(128, 1024, 64)	-	-	-	-
conv2d	(128, 1024, 32)	3	3	32	PN(LReLU)
conv2d	(128, 1024, 32)	3	3	32	PN(LReLU)
output	(128, 1024, 2)	1	1	2	Tanh

Table 3: Generator Architecture

Layer	Output Size	Kernel Width	Kernel Height	Kernel Filters	Nonlinearity
image	(128, 1024, 2)	-	-	-	-
conv2d	(128, 1024, 32)	1	1	32	-
conv2d	(128, 1024, 32)	3	3	32	PN(LReLU)
conv2d	(128, 1024, 32)	3	3	32	PN(LReLU)
downsample 2x2	(64, 512, 32)	-	-	-	-
conv2d	(64, 512, 64)	3	3	64	PN(LReLU)
conv2d	(64, 512, 64)	3	3	64	PN(LReLU)
downsample 2x2	(32, 256, 64)	-	-	-	-
conv2d	(32, 256, 128)	3	3	128	PN(LReLU)
conv2d	(32, 256, 128)	3	3	128	PN(LReLU)
downsample 2x2	(16, 128, 128)	-	-	-	-
conv2d	(16, 128, 256)	3	3	256	PN(LReLU)
conv2d	(16, 128, 256)	3	3	256	PN(LReLU)
downsample 2x2	(8, 64, 256)	-	-	-	-
conv2d	(8, 64, 256)	3	3	256	PN(LReLU)
conv2d	(8, 64, 256)	3	3	256	PN(LReLU)
downsample 2x2	(4, 32, 256)	-	-	-	-
conv2d	(4, 32, 256)	3	3	256	PN(LReLU)
conv2d	(4, 32, 256)	3	3	256	PN(LReLU)
downsample 2x2	(2, 16, 256)	-	-	-	-
concat(x, minibatch std.)	(2, 16, 257)	-	-	-	-

Table 4: Discriminator Architecture

3.3 Training Details

The architecture was directly adapted from the Tensorflow implementation of GANSynth in Magenta. The model was trained with the ADAM optimizer. Both, the generator and discriminator use box upscaling/downscaling. The generators use pixel normalization. Since we had to deal with a significantly smaller dataset, we ran through only 8 of the 12 stages of the progressive GAN training.

For tuning the hyperparameters, the authors of the original paper swept over learning rates (2e-4, 4e-4, 8e-4) and weights of the auxiliary classifier loss (0.1, 1.0, 10), and found that a learning rate of 8e-4 and classifier loss of 10 performs the best.

Hyperparameter	Value	Description
data_type	mel	Data is encoded into a Mel spectrogram
total_num_images	11,000,000	# data samples to use
stable_stage_num_images	800,000	# data samples to use at stable stages
transition_stage_num_images	10,000	# data samples to use at transition stages
batch_size_schedule	64	Batch size for each iteration
latent_vector_size	256	samples from spherical gaussian
kernel_size	3	For the conv2d layers
high_frequency_resolution	True	uses a start height of 2 and start width of 16

Table 5: Hyperparameters

The model was trained on a single V100 GPU, with a batch size of 64 for 3 days. It trained for a total of 9 stages.

4 Results

4.1 Outputs

Since our model generates audio, we depict the outputs as images in this paper and do a stage-wise (stages 5 to 8) comparison of real vs generated IF-Mel and pitch data. You will also find a sample generated output here: Pink Panther with reverb

4.1.1 Mel

Figure 1: Stage 5 Real

Figure 2: Stage 5 Fake

Figure 3: Stage 6 Real

Figure 4: Stage 6 Fake

Figure 5: Stage 7 Real

Figure 6: Stage 7 Fake

Figure 7: Stage 8 Real

Figure 8: Stage 8 Fake

4.1.2 Pitch

Figure 9: Stage 5 Real

Figure 10: Stage 5 Fake

Figure 11: Stage 6 Real

Figure 12: Stage 6 Fake

Figure 13: Stage 7 Real

Figure 14: Stage 7 Fake

Figure 15: Stage 8 Real

Figure 16: Stage 8 Fake

4.2 Losses

We tracked the Wasserstein losses of the generator and discriminator along with the a few others.

Auxiliary Conditioning Loss (ac loss) - This is the loss related to the introduction of auxiliary conditioning of pitch in the generation.

G&L Consistency Loss (gl loss) - There's an inherent loss in transforming STFTs to a Mel frequency scale and back. G&L Consistency tracks this loss.

Figure 17: Legend

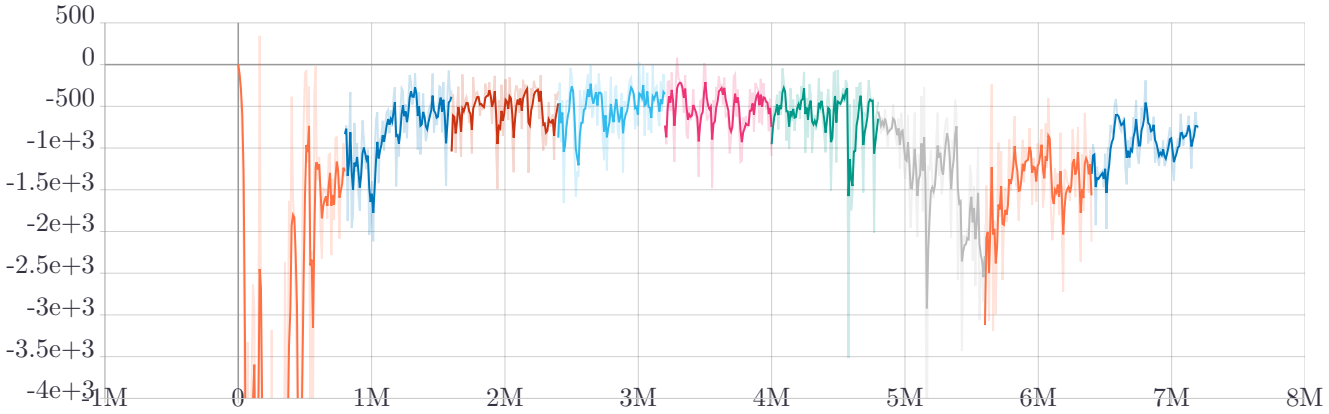

Figure 18: Discriminator Wasserstein Loss

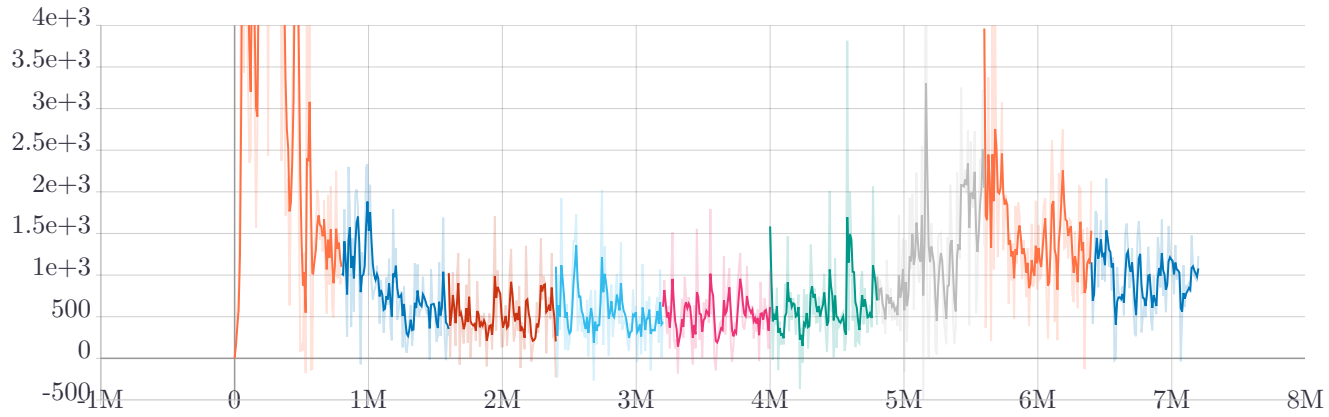

Figure 19: Generator Wasserstein Loss

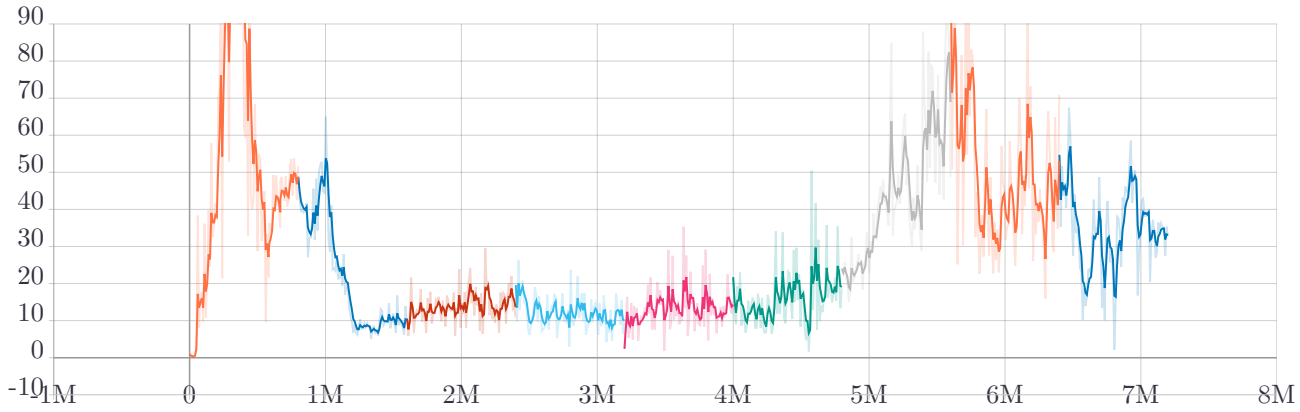

Figure 20: Wasserstein Gradient Penalty

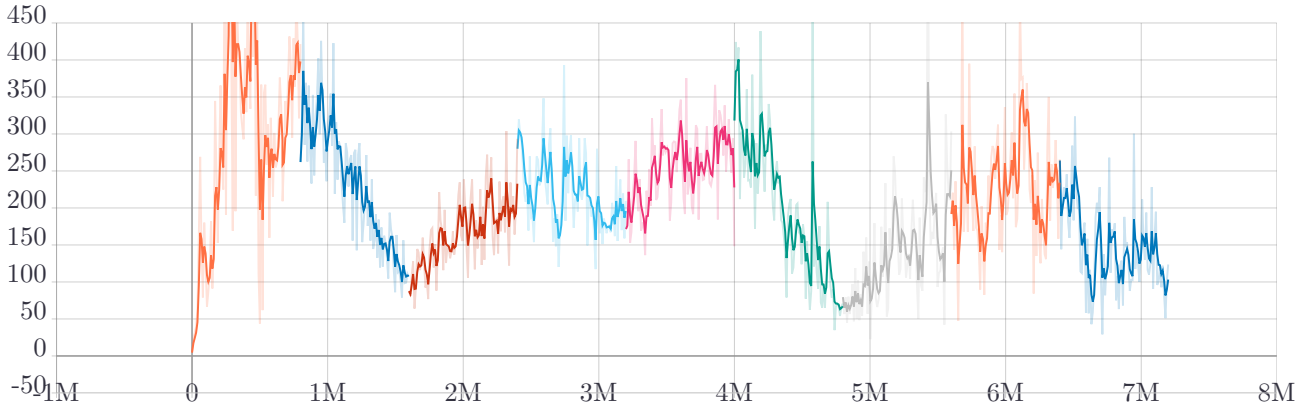

Figure 21: Fake AC Loss

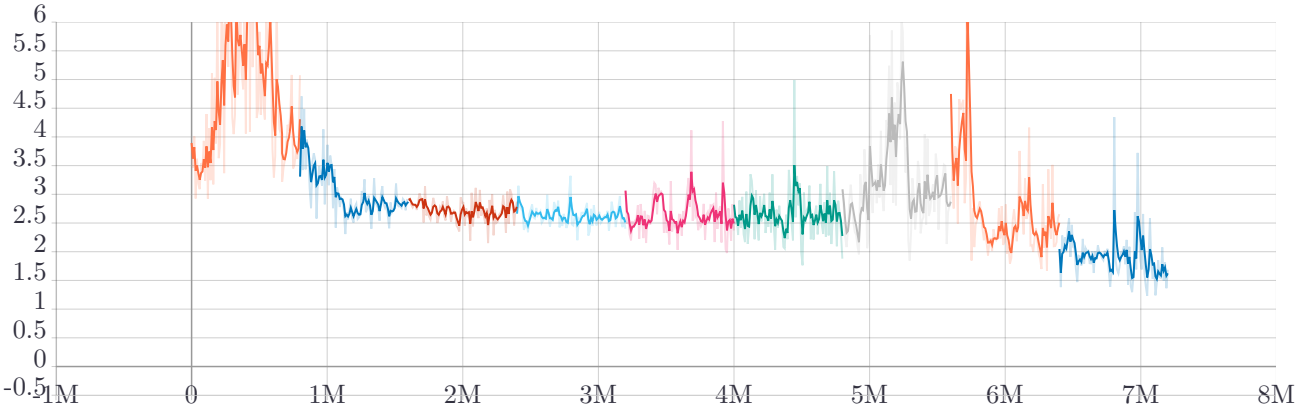

Figure 22: Real AC Loss

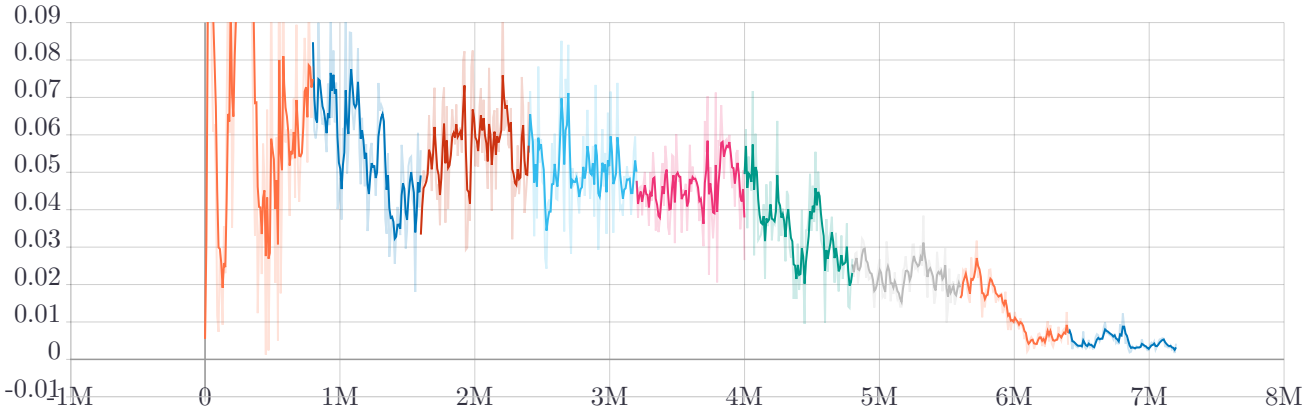

Figure 23: Fake GL Loss

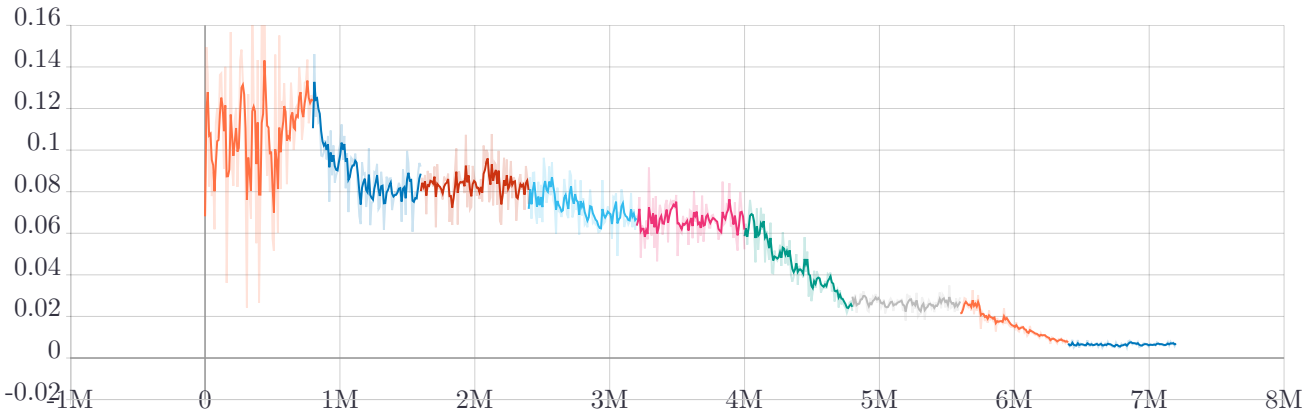

Figure 24: Real GL Loss

5 Conclusion

Our attempt to generate short, musical audio samples representative of the audio one might expect from an affected or processed sample was successful. Overall, loss rates decreased almost across the board, and anecdotally, the audio we generated sounded like our audio dataset with reverb. Of course, this first model also lays bare the limits of our strategy, and makes clear several pathways for further research. First, of course, our dataset was very limited. Specifically, we used only processed and unprocessed samples from a single instrument, and we can reasonably expect our model, given far more time and memory, to handle many instruments, as the original GANSynth model did. Additionally, we see no reason why multiple forms of processing could not be handled by the same model, and this would be an exciting angle to explore. Additionally, we would like to modify the original model to accept more labels, including labels for multiple forms of signal processing. In this way, it may be possible to handle more forms of signal processing and more effectively control what sounds are actually produced by the model. Furthermore, it would be exciting to conduct parallel research with a model capable of accepting a representation of a full audio file as input. We hope that this format of model would be capable of effectively learning effects and processing existing audio with those effects. In summary, we believe that our project was successful in that it enabled us to generate processed audio, but our project also made clear the limits of our approach and provided insight into possible directions for future research.

References

[1] magenta/magenta.

[2] supercollider/supercollider, December 2020. original-date: 2012-05-04T19:24:51Z.

[3] Chris Donahue, Julian McAuley, and Miller Puckette. Adversarial audio synthesis. In *International Conference on Learning Representations*, 2019.

[4] 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.

[5] Jesse H. Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Douglas Eck, Karen Simonyan, and Mohammad Norouzi. Neural audio synthesis of musical notes with wavenet autoencoders. *CoRR*, abs/1704.01279, 2017.

[6] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[7] Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, and Aaron C. Courville. Improved training of wasserstein gans. *CoRR*, abs/1704.00028, 2017.

[8] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *CoRR*, abs/1710.10196, 2017.

[9] Olof Mogren. C-RNN-GAN: continuous recurrent neural networks with adversarial training. *CoRR*, abs/1611.09904, 2016.

[10] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans, 2017.

[11] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016.