Adversarial Audio Synthesis using WaveGAN Architecture

Anirudha Brahma
Dept. Of Physics, IIT Kanpur
anirbrhm@iitk.ac.in

Ananyae Bhartari
Dept. Of Mathematics, IIT Kanpur
ananyae@iitk.ac.in

***Abstract*—Generative adversarial networks (GANs) have seen wide success at generating realistic high resolution images, but they have seen little application to the domain of audio generation. In this work we implement the WaveGAN architecture to try to generate audio using Unsupervised GAN learning techniques. We also improve the training process of the GAN model by using advanced learning algorithms. In the end we present our results and also suggest pathways for future research.**

I. Introduction

Creating new Beethoven like music from his old music clips, or want to create a text to speech software ? Audio synthesis is probably the way to go. Audio synthesis using Machine Learning is a fascinating problem in the audio signals domain. The idea is to generate new audio clips given examples of the kind of audio that we need. This type of problem has varied uses in music and film industry where it is far easier to generate a piece of audio that is needed instead of painstakingly searching through millions of old records to find the perfect match. We model the problem of Audio synthesis as a generative machine learning problem and we are going to tackle the issue using GANs(General Adversarial Network's). Our experiments in the computer vision domain shows that GANs are capable of mapping lower dimensional latent vectors to high dimensional realistic data. In this paper We are going to apply GANs directly to raw audio form, to create an end to end model capable of generating audio given background context.

II. Audio v/s Images

Naively speaking, Images and Audios are just a bunch of tensors comprising of many numbers. Closely analyzing the principal components of an image and audio depicts the subtle differences in them. The principal components of an Image generally capture intensity, gradient, and edge characteristics, and those from audio form a periodic basis that decompose the audio into constituent frequency bands. We understand that correlations across large windows are very common and that is why filters with large receptive fields are required to process raw audio data. We can understand the audio data to be a time series data of a fixed dimension. We represent our audio data using 1 dimensional vectors, which we obtain by sampling the audio at 16000Hz.

III. WaveGAN Model and Implementation

A. Basic Architecture

The WaveGAN model is based on the DCGAN model which is popularly used for the task of realistic image generation. The DCGAN model is just like a GAN model, which consists of a Generator and a Discriminator. The DCGAN used the idea of Transposed Convolutions in the Generator to iteratively up-sample low-resolution feature maps into a high-resolution image. We discuss the concept of the Transposed Convolutions and Phase Shuffles in the next subsections and how we adapted the idea for the case of Audio. The rest of the model is summarized below :

Table 3. WaveGAN generator architecture

Operation	Kernel Size	Output Shape
Input $\boldsymbol{z} \sim \text{Uniform}(-1, 1)$		$(n, 100)$
Dense 1	$(100, 256d)$	$(n, 256d)$
Reshape		$(n, 16, 16d)$
ReLU		$(n, 16, 16d)$
Trans Conv1D (Stride=4)	$(25, 16d, 8d)$	$(n, 64, 8d)$
ReLU		$(n, 64, 8d)$
Trans Conv1D (Stride=4)	$(25, 8d, 4d)$	$(n, 256, 4d)$
ReLU		$(n, 256, 4d)$
Trans Conv1D (Stride=4)	$(25, 4d, 2d)$	$(n, 1024, 2d)$
ReLU		$(n, 1024, 2d)$
Trans Conv1D (Stride=4)	$(25, 2d, d)$	$(n, 4096, d)$
ReLU		$(n, 4096, d)$
Trans Conv1D (Stride=4)	$(25, d, c)$	$(n, 16384, c)$
Tanh		$(n, 16384, c)$

Table 4. WaveGAN discriminator architecture

Operation	Kernel Size	Output Shape
Input $\boldsymbol{x}$ or $G(\boldsymbol{z})$		$(n, 16384, c)$
Conv1D (Stride=4)	$(25, c, d)$	$(n, 4096, d)$
LReLU ($\alpha = 0.2$)		$(n, 4096, d)$
Phase Shuffle ($n = 2$)		$(n, 4096, d)$
Conv1D (Stride=4)	$(25, d, 2d)$	$(n, 1024, 2d)$
LReLU ($\alpha = 0.2$)		$(n, 1024, 2d)$
Phase Shuffle ($n = 2$)		$(n, 1024, 2d)$
Conv1D (Stride=4)	$(25, 2d, 4d)$	$(n, 256, 4d)$
LReLU ($\alpha = 0.2$)		$(n, 256, 4d)$
Phase Shuffle ($n = 2$)		$(n, 256, 4d)$
Conv1D (Stride=4)	$(25, 4d, 8d)$	$(n, 64, 8d)$
LReLU ($\alpha = 0.2$)		$(n, 64, 8d)$
Phase Shuffle ($n = 2$)		$(n, 64, 8d)$
Conv1D (Stride=4)	$(25, 8d, 16d)$	$(n, 16, 16d)$
LReLU ($\alpha = 0.2$)		$(n, 16, 16d)$
Reshape		$(n, 256d)$
Dense	$(256d, 1)$	$(n, 1)$

Fig. 1. WaveGAN Architecture

B. Transposed Convolutions

A transposed convolution layer, is usually carried out for up-sampling i.e. to generate an output feature map that has a

spatial dimension greater than that of the input feature map. Just like the standard convolution layer, the transposed convolution layer is also defined by the padding and stride. These values of padding and stride are the one that hypothetically was carried out on the output to generate the input. i.e. if you take the output, and carry out a standard convolution with stride and padding defined, it will generate the spatial dimension same as that of the input.

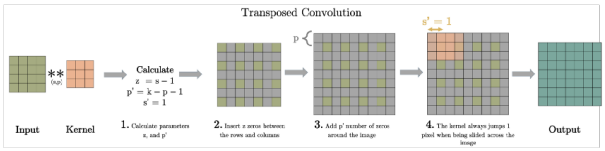

Fig. 2. Transposed convolutions

The DCGAN generator uses the transposed convolution operation to up-sample the low resolution feature map to a higher dimensional vector. Suppose that we want to up-sample a input 2x2 to 4x4output then we simply take a kernel of (another matrix of 2x2) and each entry of the 2x2 input is used to scale the kernel then extend this 2x2 matrix to get a 4x4 matrix by padding with zeros in specific way. we just keep shifting our kernel over our input to generate a 4x4 matrix finally taking the sum of all the 4x4 matrix generated.

This idea is replicated to audio domain for 1 dimensional convolutions :

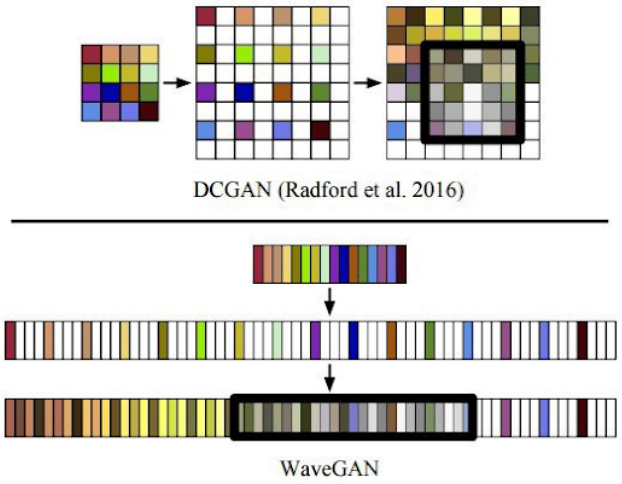

Fig. 3. Transposed convolutions

The DCGANS upsample till we get a 64x64 image or 4096 1D vector we are going to upsample it once more by a factor of 4 to 16384 and going to drop the last 384 bits to get our 16000 length vector back. We are going to use a increasing stride of 4 (not 2 like DCGAN) for each layer.

1) Problems: Using Transposed Convolutions to upscale the audio from noise to the 16000 long vector will impact on the synthesized audio, as they notoriously produce checkerboard artifacts in images like the image shown below and the equivalent in audio signals. The Discriminator could easily learn to spot fake audio based on this artifact alone, deteriorating the whole training process.

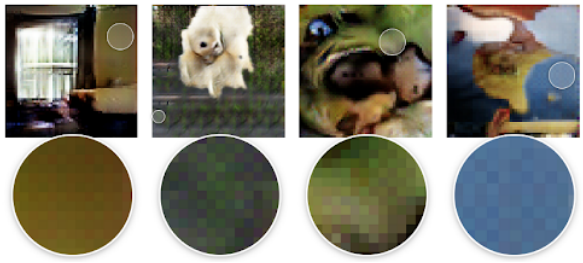

Fig. 4. Checkerboard Artifacts

The main cause of this is uneven overlap at some parts of the image causing artifacts. This paper tackles this problem using Phase Shuffle in this paper.

C. Phase Shuffle

Phase Shuffle is a technique for removing pitched noise artifacts that come from using transposed convolutions in audio generation models. Phase shuffle is an operation with hyper-parameter n. It randomly perturbs the phase of each layer's activation by n to n samples before input to the next layer. In the original application in WaveGAN, the authors only apply phase shuffle to the discriminator. The idea is that the Discriminator should not learn that certain periodic elements are bad and discard the sound on the basis of certain periodic aspects because unlike images sounds have periodicity. Therefore discriminators job becomes more challenging by requiring in-variance to the phase of input wave form.
We choose a k $\in \{-n, -n+1, ...0, ...n\}$ and shift our wave form by k steps to the right, the part which is shifted out of the audio is then reflected and added to the front of the audio.

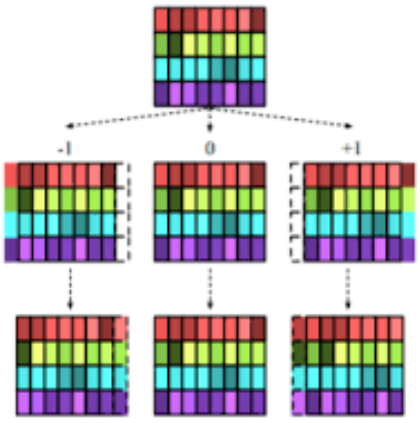

Fig. 5. Phase Shuffle with n = 1

IV. Training

Traditionally GANs use a value function which is defined below, G is trained to minimize the following value function, while D is trained to maximize it:

$$V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim P_X}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim P_Z}[\log(1 - D(G(\boldsymbol{z})))]$$

Fig. 6. GAN Training

The reason GANs are difficult to train is that the architecture involves the simultaneous training of a generator and a discriminator model in a zero-sum game. Stable training requires finding and maintaining an equilibrium between the capabilities of the two models.

Instead of using a discriminator to classify or predict the probability of generated images as being real or fake, the WGAN changes or replaces the discriminator model with a critic that scores the realness or fakeness of a given image. This change is motivated by a mathematical argument that training the generator should seek a minimization of the distance between the distribution of the data observed in the training dataset and the distribution observed in generated examples.
The critic neural network can be trained to approximate the Wasserstein distance, and, in turn, used to effectively train a generator model. Importantly, the Wasserstein distance has the properties that it is continuous and differentiable and continues to provide a linear gradient, even after the critic is well trained. This is unlike the discriminator model that, once trained, may fail to provide useful gradient information for updating the generator model.

To minimize Wasserstein distance, they suggest a GAN training algorithm (WGAN) for the following value function:

$$V_{\text{WGAN}}(D_w, G) = \mathbb{E}_{\boldsymbol{x}\sim P_X}[D_w(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{z}\sim P_Z}[D_w(G(\boldsymbol{z}))]$$

Fig. 7. WGAN Training

Implementing this we use :

- Critic Loss = Average critic score on real images – Average critic score on fake images
- Generator Loss = -Average critic score on fake images

Weight clipping is a means of enforcing that D_w is 1-Lipschitz which is a necessity for the convergence of this algorithm. As an alternative strategy we can replace weight clipping with a gradient penalty (WGAN-GP) that also enforces the constraint. WGAN-GP strategy can successfully train a variety of model configurations where other GAN losses fail.
So the loss for the critic now becomes :

A. Hyperparameters

The paper suggested a set of hyper-parameters without mentioning how the hyperparameter tuning was done. We chose to use exactly those hyperparameters because the training takes a long long time and resources and we did not have the resources to do hyper-parameter tuning for our model ourselves.
The hyperparameters used for training were :

$$L = \mathbb{E}_{\tilde{\mathbf{x}}\sim\mathbb{P}_g}[D(\tilde{\mathbf{x}})] - \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_r}[D(\mathbf{x})] + \lambda\mathbb{E}_{\hat{\mathbf{x}}\sim\mathbb{P}_{\hat{\mathbf{x}}}}\left[(||\nabla_{\hat{\mathbf{x}}}D(\tilde{\mathbf{x}})||_2 - 1)^2\right]$$

Fig. 8. WGAN GP Training Loss Function

Table 6: WaveGAN hyperparameters

Name	Value
Input data type	16-bit PCM (requantized to 32-bit float)
Model data type	32-bit floating point
Num channels (c)	1
Batch size (b)	64
Model dimensionality (d)	64
Phase shuffle (WaveGAN)	2
Phase shuffle (SpecGAN)	0
Loss	WGAN-GP (Gulrajani et al., 2017)
WGAN-GP λ	10
D updates per G update	5
Optimizer	Adam ($\alpha = 1e{-}4$, $\beta_1 = 0.5$, $\beta_2 = 0.9$)

Fig. 9. Hyper-parameters

V. Results

According to the original paper the training took close to 2 days on 4 GPU's parallely to achieve satisfactory results on the SC09 dataset. We trained our model on only the number '8' for a total of 2 hours. We observe a gradual improvement in the samples after each epoch. The final after the last epoch turns out to be much better than the samples generated after the initial epochs, but it was far away from the sound of '8' number.

Acknowledgment

This course project would not have been possible without the support and teachings of Prof. Vipul Arora who have been very helpful in teaching the necessary concepts needed to solve this problem and encouraging us even further to actually understand each and every concept used in the paper. We would also like to thank the Teaching Assistants who helped us with the logistics of this project.
Last but not the least we would like to thank our peers from this course with whom we have had lots of fruitful conversations regarding how to approach this project.

References

[1] Adversarial Audio Synthesis Chris Donahue, Julian McAuley, Miller Puckette
[2] Wasserstein GAN Martin Arjovsky, Soumith Chintala, Léon Bottou
[3] Improved Training of Wasserstein GANs Ishaan Gulrajani1, Faruk Ahmed1, Martin Arjovsky2, Vincent Dumoulin1, Aaron Courville1
[4] Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks Alec Radford, Luke Metz, Soumith Chintala
[5] https://machinelearningmastery.com/how-to-implement-wasserstein-loss-for-generative-adversarial-networks/
[6] https://paperswithcode.com/method/wgan-gp-loss
[7] https://towardsdatascience.com/what-is-transposed-convolutional-layer-40e5e6e31c11: :text=Transposed