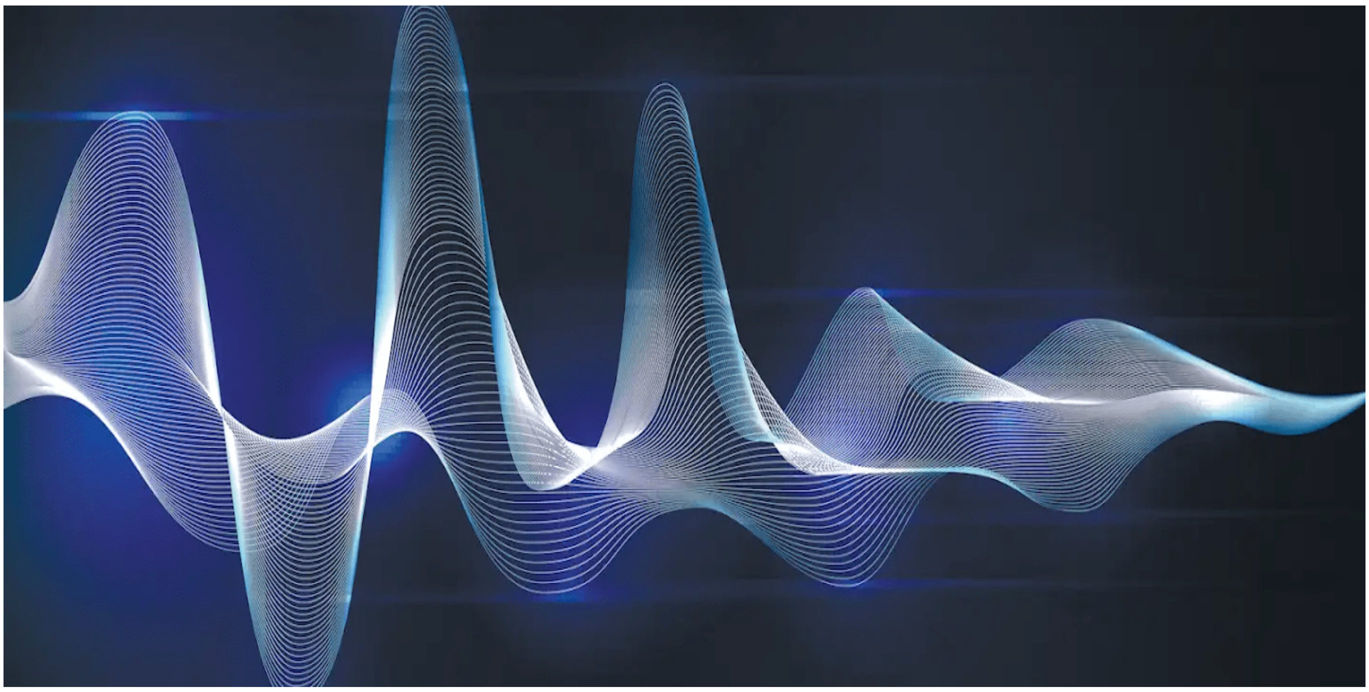
Fall Semester
December 2019

Project in AI-based Time Series Analysis
Raw audio generation using GANs



Student : David Albert
Professor : Jaesik Choi

Contents

Introduction

In this project, we explore methods relative to deep learning for sequential generation. More especially, we will be focused on the paper *Wavenet: A generative model for raw audio* [1] published in 2016.

Directly generating raw audio waveforms is complexe since this kind of signals have very high temporal resolution (at least 16 000 samples per second). As a consequence, in order to generate sequences which are consistent, we need to design a model allowing generate samples that depend on many previous ones. WaveNet model allows this by introducing dilated convolutional layers. The model is an autoregressive generative model whose main idea is similar to PixelCNN [2] but works for audio.

In this report, we introduce a way to combine autoregressive generative models and generative adversarial networks (GAN) in order to improve the samples quality. The idea is simple, we will add a loss term (an adversarial loss) to help generating realistic audio. Combining generative methods is not new idea, some works have already studied a way to combine flow-based models and autoregressive models as done in the papers regarding Inverse Autoregressive Flow [7] and Masked Autoregressive Flow [8].

We will first present the methods that have inspired our model. Then we will theoritically introduce our model. Finally, we will show some experiments made and conclude.

Part I

Background

The model introduce in this project is higly based on two recent advances in generative models.

1 Generative Adversarial Networks

Generative Adversarial Networks (GAN) [4] were introduced by Ian Goodfellow in 2014 and since, the model have been strongly studied.

The principle is that we will build two neural networks which will be trained with concurrent losses. The first network, called generator, allows to generate fake samples (following the distribution p_g) that we intend to be looking as similar as real samples from p_{data}. As for the second network, called discriminator, it allows to distinguish samples from $p_{data}(x)$ and samples from $p_g(x)$.

The training process will take the form of a 2-players zero-sum game where generator and discriminator networks are players and both try to maximize its reward by minimizing opponent reward.

Goal for each "Player":

- **Player 1 = Discriminator network** : Try to distinguish real and fake samples from the generator.
- **Player 2 = Generator network** : Try to fool the first player by generating samples similar to data (real-looking samples).

The decision rule for this game is the minmax rule, which can be rewrite as follow :

$$\min_{\theta_g} \max_{\theta_d} \left[E_{x \sim P_{data}} log(D_{\theta_d}(x)) + E_{z \sim P_z} log(1 - D_{\theta_d}(G_{\theta_g}(z)) \right]$$

- $D_{\theta_d}(y)$: discriminator output for y. It is the likelihood that y is real.
- x : real data (from the training data set)
- z : generator input (usually a random vector from Uniform or Gaussian distribution)
- $G_{\theta_g}(z)$: generated data (fake data)

NB : The drawback of this model is that it trains two networks in the same time and thus it can be unstable.

1.1 Training procedure

To train a GAN model we alternate between the two following phases (the algorithm can be find in Appendix 1):
1) Gradient ascent on discriminator to maximise :

$$C_1(\theta_d) = E_{x \sim P_{data}} log(D_{\theta_d}(x)) + E_{z \sim P_z} log(1 - D_{\theta_d}(G_{\theta_g}(z))$$

2) Gradient ascent on generator to maximise :

$$C_2(\theta_g) = E_{z \sim P_z} log(D_{\theta_d}(G_{\theta_g}(z))$$

The reason of the success of GANs come from the fact that GANs have demonstrated to be able to generate more realistic images than generative models like autoregressive models, flow-based methods or variational auto-encoders. This improvement can be interpreted by the fact that that GANs loss minimizes the Jensen-Shannon divergence (D_{JS}) instead of the Kullback-Leibler divergence (D_{KL}) which is symmetrized and smoothed version of the Kullback-Leibler divergence.

$$D_{KL}\left(p_{data}\|p_g\right) = \mathbb{E}_{x\sim p_{data}(x)}\left(log\left(\frac{p_{data}(x)}{p_g(x)}\right)\right) \tag{I.1}$$

And

$$D_{JS}\left(p_{data}\|p_g\right) = D_{KL}\left(p_{data}\|\frac{p_{data}+p_g}{2}\right) + D_{KL}\left(p_g\|\frac{p_{data}+p_g}{2}\right) \tag{I.2}$$

The fact that GAN loss help making samples more realistic was the main motivation for this work. In fact, WaveNet may have shown to be able of producing audio samples with high resolution but those results are really difficult to be achieved. When training WaveNet model for music generation, one recurrent problem is that generated samples are not consistent we sometimes very huge gap between two chunks. We even can use another formulation of GAN which is Least-Squares GAN [5]. This formulation sometimes leads more stability while training and was tried during experiments.

2 WaveNet : Generating waveform data

In order to generate waveform data which are consistent, we need to design a model allowing generate samples that depend on many previous samples. WaveNet is an autoregressive generative model whose main idea is similar to PixelCNN [2] but have been adapted to work for waveform audio generation. The model allows this by introducing dilated convolutional layers.

2.1 Dilated causal convolutions

Causal convolutions are the main ingredient of WaveNet. By using causal convolutions, the model cannot the violate the ordering in which the data are modeled. The main advantage of causal convolution over recurrent connections is that they are faster to train that recurrent neural networks. However, the problem is that the only way to increase receptive field is by adding layers. As a consequence, for modeling causal dependencies in audio data with high resolution, the authors introduced dilated convolutions.

A dilated convolution is a convolution where filter is applied over an area larger than its length by skipping input values with a certain step. The schema below illustrates a dilated causal convolution.

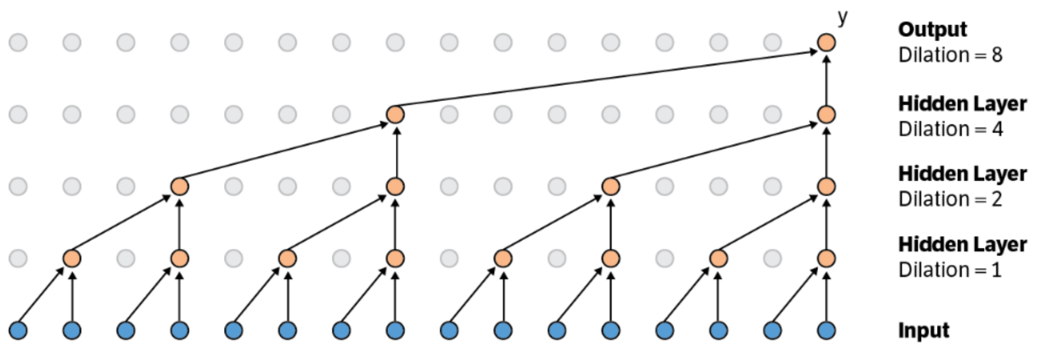


Figure I.1 – Dilated causal convolution

2.2 Other tricks

WaveNet use other tricks to deal with some issues about waveform data generation. For instance, in order to produce a better reconstruction than a simple linear quantization scheme, they use a non-linear quantization $f(x_t) = sign(x_t)\frac{ln(1+\mu|x_t|)}{ln(1+\mu)}$. Also, they use both residual and parameterised skip connections to speed up convergence and enable training of much deeper models.

Part II

Raw audio generation using GANs

We introduce here a way to learn to generate data by combining autoregressive generative models and generative adversarial networks.

1 General formulation

Consider a set of N independant sequences $\mathbf{x} = \left\{\mathbf{x}^{(i)}\right\}_{i=1,..,N}$ of the same distribution $p_{data}(x)$ And $\mathbf{x}^{(i)} \in E^T$ where E is a finite set of values, e.g.. {0,1,...,256}. Each sequence $\mathbf{x}^{(i)} = (x_0^{(i)}, x_1^{(i)}, ..., x_T^{(i)})$ is fully correlated, i.e. $x_k^{(i)}$ and $x_j^{(i)}$ are correlated for any i, j, k.

The joint probability of $x^{(i)}$ (we will now write x for simplicity) is then given by :

$$p(x) = \prod_{t=0}^{T} p(x_t|x_1, x_2, ..., x_{t-1}) = \prod_{t=0}^{T} p(x_t|x_{<t})$$

It means that each sample of the sequence data is conditioned on the samples at all previous timesteps. We model the conditional probability by using a neural networks $p_\theta(x_t|x_{<t})$. For raw audio generation, p_θ is nothing but the model proposed in wavenet paper [1].

Thus, given the first part of a sequence x_0 we can generate the entire sequence by iteratively sampling from p_θ. This sequence will represent the fake output of our GAN model. Let's rewrite the GAN objective function presented in part 1 as follow:

$$\min_\theta \max_\mu \left[\mathbb{E}_{x\sim p_{data}} \left\{logD_\mu\left(x\right) + log\left(1 - D_\mu\left(G_\theta(x_0)\right)\right)\right\}\right] \tag{II.1}$$

where

- D_μ is the discriminator
- G_θ is the generative process
- x_0 is the first value of the sequence x

The full model is given next page in the figure II.1.

As every GAN model, to train this model, we have to train alternatively discriminator and generator.
Discriminator loss:

$$\mathcal{L}_D = -\mathbb{E}_{x\sim p_{data}} \left\{logD_\mu\left(x\right) + log\left(1 - D_\mu\left(G_\theta(x_0)\right)\right)\right\}$$

Generator loss:

$$\mathcal{L}_G = \lambda_{adv}\mathbb{E}_{x\sim p_{data}} \left\{log\left(D_\mu\left(G_\theta(x_0)\right)\right)\right\} + \lambda_{match}\mathbb{E}_{x\sim p_{data}} \left\{\|x - G_\theta(x_0)\|_2\right\}$$

where λ_{adv} and λ_{match} are ratio relating to the importance of fooling the discriminator and matching like real data. We can choose any norm $\|.\|_p$ but a good choice is the norm L_2.

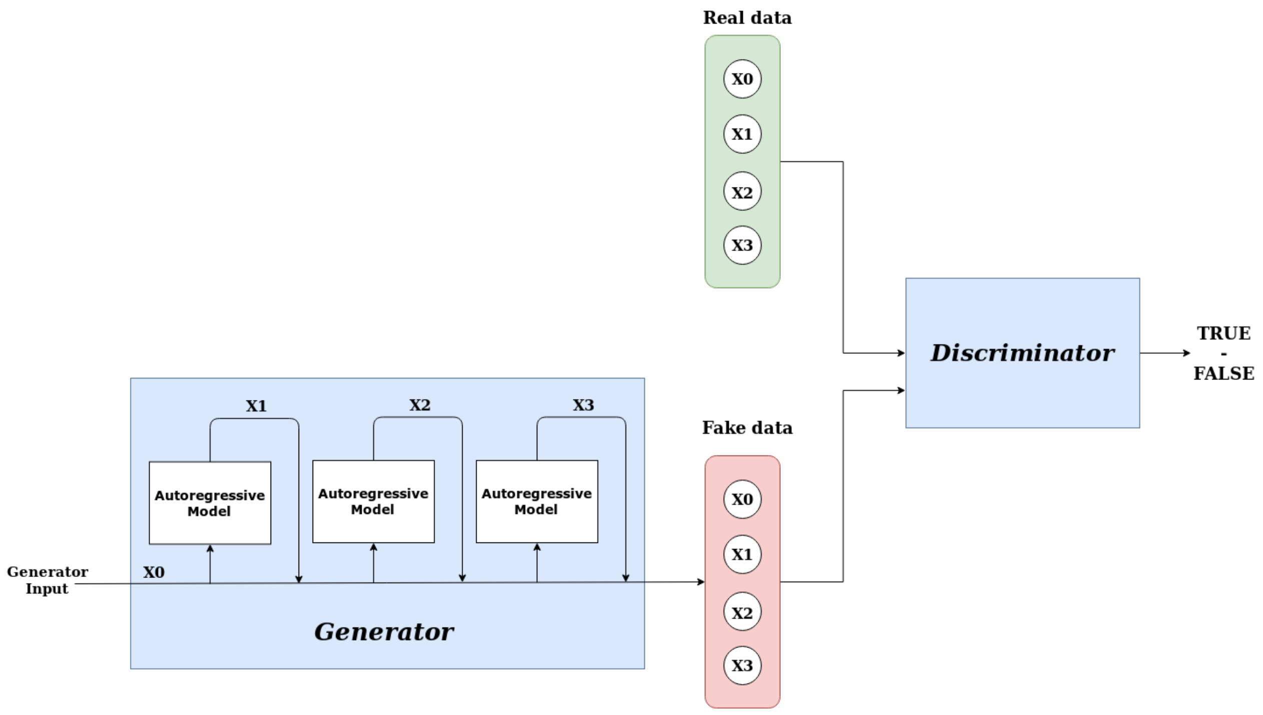


Figure II.1 – Full model of the mix AR/GAN model

2 Raw audio generation with WaveNet and GANs

In the previous section, we gave a general formulation of the method without precise neither the autoregressive model nor the discriminator architecture to use. We will be focusing on raw audio generation. As a consequence, we need to design both generator and dicriminator such that they can deal with audio data with long term dependencies.

Generator and Discriminator architectures

Given the previous formulation, to design the generator we only need to design p_θ. A good choice is to model by the well known WaveNet architecture presented in section 2.

Whereas for the discriminator architecture, we choose to adapt the PatchGAN architecture used by Isola et al. in [6]. The architecture was especially designed to be used with images. In our experiments, we used the same architecture but we change only the convolutional layers. Because we were manipulating audio data, it doesn't make sense to use two-dimensional convolution layers. Instead of this, we used one-dimensional convolution layers. The architecture is shown below.

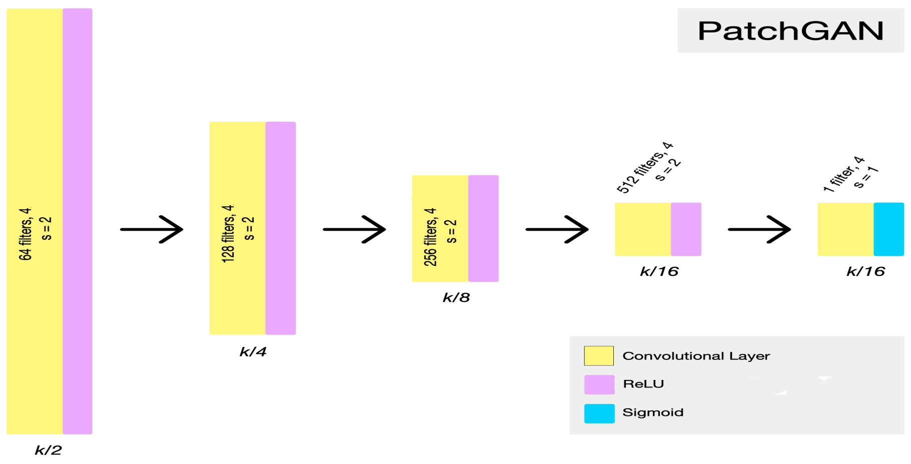


Figure II.2 – PatchGAN Architecture for audio data

3 Experiments

For the experiments, we didn't have implemented all the code from "scratch". We have been strongly inspired by the implementation [3]. We adapted the class *WavenetTrainer* as *WaveGANTrainer* which change from the original by using a discriminator model and so by adding the adversarial loss for both generator and discriminator.

Because our model, as any generative model, requires that data have structural similarities, we have decided to work with a dataset containing only violin musics of Bach. The goal of our experiments was to train both Wavenet with and without the adversarial loss and compare the results. We used *AdamOptimizer* to minimise each generator and discriminator losses with a learning rate of 0.001. We used size of batch of 16 and trained the model for 10 epochs.

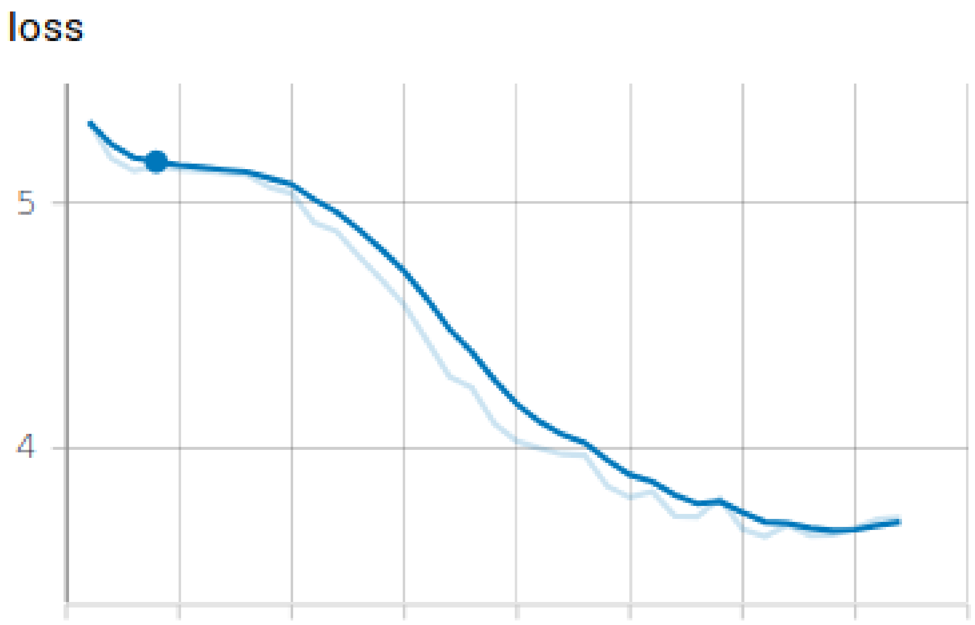


Figure II.3 – Loss for training the model on colab GPU for 6 hours

Conclusion

To conclude on this project, we proposed to mix two popular approaches for data generation which are the autoregressive generative models and the generative adversarial networks. The main purpose was to apply this new model to sequential data and so we chose to be focused on waveform data. For being able to generate accuratly waveform data we used Wavenet as a method for generating songs. We quickly understood that WaveNet is a very powerful model but it requires a huge amount of computational resources to be trained. So that it is difficult to get good results without running the model during several days. The duration of training process combined to the work in a tight deadline didn't help leading to interesting results. For instance, we failed to prove that our approach can improve the quality of audio and the consistency between fragments of the samples generated using only the original WaveNet model.

However, as put forward by the authors in [6], to speed up training, we can use λ_{adv} low compared to λ_{match} to first learn to match accuratly with the target and by minimizing the matching loss and only try to fool the discriminator when as additional objective when generated samples are already accurate. We used this approach for our experiments but due to long training duration, we didn't find significant improvement. The consequence is that we were not able to have a objective opinion on the importance of our approach.

For futur work, we should reuse this model and test it in simpler case with simpler models to train. We think that this idea to combine several generative methods to improve performance can efficiently improve researches about generative models.

Bibliography

[1] *WaveNet: A Generative Model for Raw Audio* [Oord et al., 2016]
https://arxiv.org/abs/1609.03499

[2] *Conditional Image Generation with PixelCNN Decoders* [Oord et al., 2016]
https://papers.nips.cc/paper/6527-conditional-image-generation-with-pixelcnn-decoders.pdf

[3] *WaveNet implementation* [vincentherrmann]
https://github.com/vincentherrmann/pytorch-wavenet

[4] *Generative Adversarial Networks* [Goodfellow et al., 2015]
https://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

[5] *Least Squares Generative Adversarial Networks* [Mao et al., 2016]
https://arxiv.org/abs/1611.04076

[6] *Image-to-Image Translation with Conditional Adversarial Networks* [Isola et al., 2016]
https://arxiv.org/abs/1611.07004

[7] *Improving Variational Inference with Inverse Autoregressive Flow* [Kingma et al., 2016]
https://arxiv.org/abs/1606.04934

[8] *Masked Autoregressive Flow for Density Estimation* [Papamakarios et al., 2017]
https://arxiv.org/abs/1705.07057

Appendix

Algorithm 1 Training GAN

```
1: procedure TRAIN-GAN($number\_epochs, steps, \epsilon, \alpha$)
2:     for number of epochs do
3:         for k steps do
4:             •Sample batch of m noise sample $\{z^{(1)}, z^{(2)}, ..., z^{(m)}\}$ from $p_z$, the latent space.
5:             •Sample batch of m real data $\{x^{(1)}, x^{(2)}, ..., x^{(m)}\}$ from $p_{data}$.
6:             •Update the discriminator by ascending its stochastic gradient:
                   $$\nabla_{\theta_d} \frac{1}{m} \sum^{m} [logD(x^i) + log(1 - D(G(z^i))]$$
         end for
7:         •Sample batch of m noise sample $\{z^{(1)}, z^{(2)}, ..., z^{(m)}\}$ from $p_z$, the latent space.
8:         •Update the generator by ascending its stochastic gradient:
               $$\nabla_{\theta_g} \frac{1}{m} \sum^{m} log(D(G(z^i))]$$
     end for
```