ATIAM 2021 - ML Project
Music compression and generation

Constance Douwes, Tristan Carsault, Philippe Esling

Abstract

In most scientific domains, the deep learning community has largely focused on the quality of deep generative models, resulting in highly accurate and successful solutions. However, this race for quality comes at a tremendous computational cost, which incurs vast energy consumption.

In this project, we aim to produce long high-quality audio samples based on a recent approach proposed by Korneel van den Broek, called MP3net [1]. The model is a deep convolutional GAN that uses MP3/Vorbis audio compression algorithms that reduce the computational power required to generate raw audio samples by exploiting psychoacoustic properties of the audio signals.

We propose to implement the model iteratively, starting with a simple GAN model on the well-known MNIST dataset. Then, the two data processing , the MCDT and the psychoacoustic filter, will be implemented. Finally, the whole model will be coded in Pytorch and trained on the musical style of your choice.

1 Preliminary work

In this project we will use PyTorch, a versatile machine-learning Python framework that you will use to code your first Generative Adversarial Network (GAN).

Exercise 0: Familiarize yourself with PyTorch and other useful libraries

This exercise only includes basic programming skills that you will need for the following work. If we already have enough expertise, you can skip it but I'd rather recommend you to have a look.

1. Install PyTorch on your device if it isn't already done.
2. Follow the basic tutorials and learn how it works `https://pytorch.org/tutorials/`.
3. Learn how to implement basic models, therefore use `torch.nn` library.
4. Install and read about relevant libraries for audio signal processing such as `librosa`[1] or `torchaudio`[2]

Exercice 2: Implement your first GAN

As Generative Adversarial Networks [2] are difficult to implement and train, you will start with a simple problem: adversarial digits generation. Therefore, you will use the MNIST dataset (composed of 70,000 examples of hand written digits) and implement a simple convolutional GAN composed of a *discriminator* and a *generator*. As the main article is quite cryptic on how to code this, you can rely on the tutorial `https://bit.ly/3wuLAwS` (that is in Keras otherwise it's too simple) but will give you a good intuition of the system.

[1] `https://librosa.org/doc/latest/index.html`
[2] `https://pytorch.org/audio/stable/index.html`

1. Based on the tutorial or another source you will find, develop your very own GAN.
2. Train your model and compare the results from state-of-the-art.

2 Data preprocessing

Exercise 3: Modified Discrete Cosine Transform (MCDT)

The model uses the MDCT data representation directly, which includes all phase information. Phase generation is hence integral part of the model.

1. Load a 96s signal and apply the MDCT transorfm. You can either code your own implementation, but we recommand the use of a python package such as `mdct`[3].
2. Plot the MDCT amplitude $A_k(m)$ along with the MDCT spectrogram $A_k^2(m)$ of that same music sample.

Exercise 4: The Psycho-Acoustic Filter

The human ear is not able to distinguish between all the different possible raw audio signals. This phenomena is known as the psychoacoustic filter [3].

1. Based on the online source code you will find here [4], propose a PyTorch compatible version.
2. Exhibit a compression ratio resulting from this transformation.
3. Implement a gaussian noise to account for the psychoacoustic quantization error.

3 Music generation

Exercise 5 : Make the audio dataset

In the original paper, authors use a subset of the MAESTRO dataset that is composed of 200 hours of piano tracks. You will have to build your own dataset of your favorite music style (it could be classic, electronic, jazz...) by collecting at least 10 hours of stereo tracks in .wav format.

Exercise 6: Code and train the MP3Net

Following the original paper [1], we propose to re-implement the model within the Pytorch framework. The training time in the paper is estimated at around 120h. Therefore, all tests must be done beforehand.

1. Define and implement the different layers in the Pytorch formalism.
2. Propose a model architecture and adapt parameters (number of units/layer/...) in order to obtain a total number of parameters equal to or less than that of the article.
3. Establish the training and testing procedure and ask your tutors to use GPUs when necessary :)
4. (Bonus) Evaluate the energy cost of the training and the infererence phases.

Advices

- As you count several members in your team, to parallelize the work to save time. For example, section 1 and 3 can be made in parallel of sections 2. You will move on much faster if you distribute and schedule these tasks among you!
- Make recurrent updates between all of you of your individual advances, such that everybody stay in touch and get ready to help another member on one task.

[3] https://mdct.readthedocs.io/en/latest/

[4] `https://bit.ly/3FsE6hc`

- The project is quite ambitious, but you will be evaluated more on your workflow and the developments / conclusion in the report than on the final results. We are here to help you, do not hesitate!
- **[Important]** we shall run your models on GPU, that can be then CUDA compatible. Do not forget to check the CUDA semantics here[5] in order to make your model CUDA compatible. Also, make your code as elegant as possible (using all the power of the `torch.nn` library), so that we are able to fix your code in case.

References

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

[2] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[3] Udo Zölzer. *Digital audio signal processing*. John Wiley & Sons, 2008.

[5] `https://pytorch.org/docs/stable/notes/cuda.html`