ML Project ATIAM
Théis Bazin
Constance Douwes
Philippe Esling

Spectrogram Inpainting

Discrete latent representations for lightweight DAFx

2020-11-24
theis.bazin@ircam.fr
constance.douwes@ircam.fr
philippe.esling@ircam.fr

Abstract

Deep learning models of audio offer impressive results and have allowed to steadfastly improve on the state-of-the-art, both for quantitative MIR tasks as well as for creative usages ; yet, with typical sounds sampled at high frequencies, applying deep learning techniques remains computationally difficult. This is even more true for interactive applications, where maintaining a low latency for the proposed systems is essential to providing smooth user-experiences. The recently proposed *Vector Quantized*-VAE (VQ-VAE) framework allows to address this challenge by learning a highly *compact* and *compressed* representation of signals as discrete 2D grids of code indexes that can be used as a convenient, high-level input representation for subsequent modelling and processing.
In this project you'll become familiar with the VQ-VAE paradigm and apply it on audio data (first on the NSynth dataset and optionally on an audio dataset of your choice) to obtain compact representations of sounds. We will then use this representation to explore potential creative uses, in the form of a Gaussian blur-like audio processing effect, akin to image processing. This amounts to learning localized distributions of the code indexes, and you will resort to strong local independence assumptions in order to again keep the computational load low. Finally, you will be tasked with computing the energetic cost of both model training and inference, inviting you to get familiar with rising concerns in the Deep Learning community regarding the environmental impact of the field.

Global instructions

Generalities

Deadline ..... January 2021
Organization ..... Group of 3 to 4 students
Repository location ..... `GitHub`

Project folders

↳ `code/` ..... containing your PEP8-compliant code, organized in modules
↳ `report/` ..... containing your final report in PDF format

Report

Redaction ..... LaTeX using the **provided formatting style**
Language ..... English
Maximum number of pages ..... 8 pages

Evaluation grid

Report - Including content, results and style ..... **7 pts**
Code - Accuracy, evaluation and coding style ..... **13 pts**

For more information on the project visit the ATIAM Machine Learning projects homepage[1].

[1] https://esling.github.io/atiam-ml-project/

1. Introduction

We first introduce the motivations for this project as well as the suggested approach that you will be implementing. This introduction is a bit long, but this is mainly to transmit to you the reasoning behind the proposed approach and clarify the structure of the project for you.

1) Goal

The goal in this project is to help you get familiar with deep learning for the interactive transformation of audio signals. To this end, you will build a rudimentary digital audio effect based on local (in the time-frequency plane) transformations of sounds, akin to the Gaussian blur effect typical in image processing software. This effect will operate by regenerating *local portions* of the sound conditioned on the surrounding audio content (i.e. at neighbouring time steps and frequencies), a process called *inpainting*. Crucially, performing inpainting in a principled, stochastic manner requires building a model that is able to learn a factorization of the probability distribution of sounds in *local* factors of the form (with $X \in \mathbb{R}^{\mathbb{N}\times\mathbb{M}}$ a spectrogram):

$$\phi(X, t, f) = p(X_{t,f} | \{\text{neighbours of } X_{t,f}\})$$

Doing this naively would be intractable[2], since audio data, typically sampled at high frequencies above $16\,\text{kHz}$, tends to be very high-dimensional and the resulting probability distributions would involve an exploding amount of factors.
Furthermore, for interactive, creative uses, it is desirable to provide the user with efficient control mechanisms, requiring a fine balance between fine-grained control, for instance sampled at the audio rate, which would be precise but cumbersome and of little practical use, and only offering very high-level controls, not enabling the user to reach desired results. Ultimately, a proper interactive system should strike the balance between enabling the user to effectively perform changes and actions, and letting them feel like they really are the one controlling the output and it's not just the computer creating stuff on its own[3]. We will therefore have to devise a means of compressing the data to make further processing both computationally tractable and convenient for interaction.

2) Proposed approach

In this assignment, we propose to adopt a two-step approach to address those issues. Instead of directly modeling the audio signal (or a high-resolution spectrogram representation of it), we will learn two complementary systems. The first, based on the recently proposed *Vector Quantized*-VAE [5, 6] deep-learning architecture, will allow you, through both coarse *downsampling* and a *discretization* mechanism, to obtain an intermediate representation of the sounds as *compact* grids of *integer-valued code indexes*. This encoder-decoder architecture is depicted in a simplified fashion on Figure 1. The interest of this approach is twofold: by moving from fine-grained, floating-point valued data to small grids of integers, further modeling will be made much simpler and more efficient[4]. Second, we note that the downsampling step introduced by this approach additionally offers a convenient way to address the "operating-scale" concern presented above: a single element of the resulting grid of integers will describe the local content for a whole region in the time-frequency plane. By choosing the amount of downsampling performed by the model, you will be able to decide the scale of transformations performed by the model, either micro or macro.

[2] Especially on a laptop without a deep learning-capable GPU like the ones we expect you to be working on.
[3] Or at least, that is what we believe!
[4] This idea of learning powerful, compact representations of data in an automated fashion is at the core of deep learning.

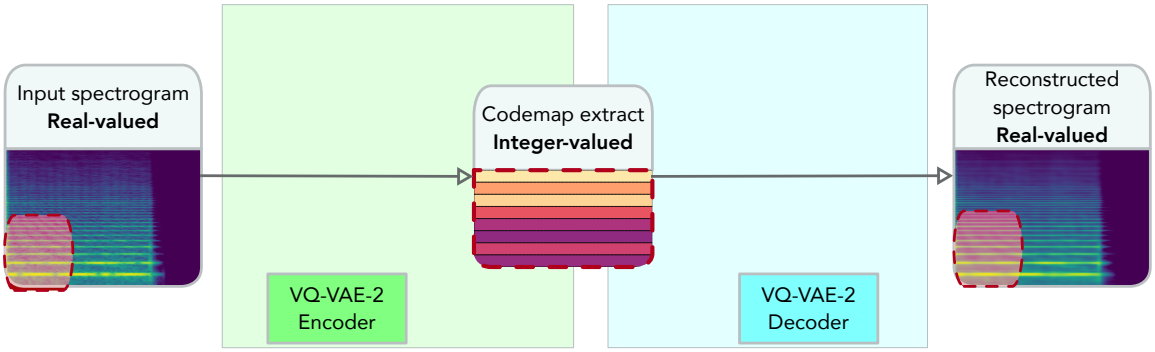

Figure 1: Proposed VQ-VAE-2 architecture for spectrograms. The encoder converts spectrograms into compact integer maps. The decoder is trained to reconstruct the original spectrogram from these discrete codemaps.

Using the resulting representation, your next task will thus be to learn the local probability distributions of these 2D code grids. To this end, you will again have to resort to strong simplifying assumptions in order to make this modeling tractable. Here, one could make use of powerful autoregressive models such as Transformer-based architectures, but we will keep things more simple and accessible for this part and require you only to train a classic, lightweight approach, widely used for instance in Natural Language Processing: a *Naive Bayes* classifier. This model replaces the complex probability distribution with a much sparser one through coarse independence assumptions and approximates the previously described local factors as:

$$p(X_{t,f}|\{\text{neighbours of } X_{t,f}\}) \approx \prod_{(t',f') \text{ neighbours of } (t,f)} p(X_{t,f}|X_{t',f'})$$

This approximates requires only a linear number of factors, making modeling much easier. Interactively transforming a sound then amounts to picking a zone in the sound and resampling it according to the probability distribution computed above, using the values of its neighbours as conditioning information.

2. Preliminary work

Exercise 0: familiarize yourself with PyTorch and other useful libraries. This exercise only includes learning or refreshing basic programming skills that you will need for the following.

1. Install PyTorch and get familiar with it through basic tutorials: `http://pytorch.org/tutorials/`
2. Install and read about relevant libraries for audio signal processing: you might be familiar with `librosa`[5] by now, we nonetheless recommend also checking out the latest release of `torchaudio`[6] for a solution readily integrated into the PyTorch framework (file I/O, spectrograms computation...), this could help you gain some time and clarify your code!

We recommend using `anaconda` or its more lightweight cousin `miniconda`[7] for managing Python environments and dependencies, in order to build a clean and common environment for your project. Furthermore, including an environment.yml or requirements.txt file to your final submission for the reviewers to reproduce your results would certainly be appreciated![8]

[5] `https://librosa.org/doc/latest/index.html`
[6] `https://pytorch.org/audio/stable/index.html`
[7] Both can be found at `https://www.anaconda.com/`.
[8] And is generally considered good practice for your future undertakings.

3. Vector Quantization

In this first part of the work, you are expected to extend on the VAE approach you studied during the class lectures and implement the *Vector Quantization* layer for the VQ-VAE. To begin with, read the two original papers by Deepmind introducing the VQ-VAE: *Neural Discrete Representation Learning* [5] and *Generating Diverse High-Quality Images with VQ-VAE-2* [6]. If you have questions on those, feel free to reach out to us at this point!

1) MNIST VQ-VAE

You will start by prototyping the VQ layer on the simple and easily accessible MNIST [4] dataset of handwritten digits.

Exercise 1: Implementing the VQ-VAE

1. Implement a **simple** Vector-Quantization layer, which takes as input a `FloatTensor` containing a batch of images, projects it onto a dictionary of codewords, and returns both the indexes of the codewords used (an `IntTensor`) and the quantized version (a `FloatTensor`), i.e. the projection of the input onto the dictionary of codewords. Crucially this requires:

   (a) Implementing the stop-gradient operator `sg` needed to properly control the flow of gradients in the VQ-VAE loss expression,

   (b) Implementing an appropriate reshaping and projection operation onto the dictionary of codewords.

2. Integrate this layer into a VAE built for MNIST (using the architecture of your choice, Fully-Convolutional, Fully-Connected, ResNet... it doesn't matter that much, we're only interested in checking if the results are reasonable enough at this point!).

⋆ *Show your code and results to your supervisors* ⋆

Exercise 1*: improving the VQ-VAE (optional) At this point you should have a rudimentary but operational VQ layer, you can (and should!) move on to the following. Remember that the goal of this project is to build the full sound-editing system, even with basic and imperfect building blocks, rather than pushing a single component of this system to perfect results. Nonetheless, if you wish to improve the architecture of your layer, you can look up the following tweaks for the VQ-VAE (you can draw inspiration from the Jukebox paper by OpenAI [1], in which the authors propose an application of VQ-VAEs to audio at the waveform level and list some of those "tricks-of-the-trade"!):

- Replacing the loss expression to use *Exponential Moving Averages* (EMA),
- Implementing codeword restarts, where less frequently "used" codewords are periodically replaced with a vector drawn at random from the layer's input,
- Devising a better-than-just-random way of initializing the codebook.

2) Spectrogram VQ-VAE

Exercise 2: audio data-loading In this second step, you are expected to build a data-loading stack to apply the previously designed model to spectrograms. You can draw inspiration from the approach described in the GANSynth paper [3] to get some ideas of how to do this, but generally any pipeline

which returns batched spectrograms from audio files "on-the-fly" would be a great start. We recommend you perform your experiments on the NSYNTH dataset of instrumental sounds [2], since it is standard and convenient to work with, all sounds having a fixed duration of 4 seconds.
Feel free to experiment around with representation parameters (hop-length, N_{fft} ...) and model hyperparameters (dimension and number of codewords, architecture of the encoder and decoder...) to try and improve the reconstruction quality of your spectrogram VQ-VAE.
⋆ *Show your code and results to your supervisors* ⋆

4. Codemap Inpainting

You now have a way of converting back and forth from spectrograms to compact, discrete codemaps. You will now be building a model to predict probability distribution for position in those codemaps, using conditioning information from the neighbours. In order to keep the relevant distributions tractable, strong independence assumptions have to be made. We recommend first experimenting with a very coarse *Naive Bayes* classifier before trying anything more involved.

Exercise 3: Naive Bayes classification Look up the ideas behind the Naive Bayes model and implement a Naive Bayes classifier for your codemaps. It should take a position and the values of the surrounding cells as input, and return a distribution over the dictionary of codewords for the current position. You can check-up the `scikit-learn` library for a ready-to-use implementation of Naive Bayes.
⋆ *Show your code and results to your supervisors* ⋆

Exercise 3*: Experimenting with other classification methods (optional) If you have made it this far, congrats! You can now try more advanced techniques to predict the desired local distributions of codes. For instance, you could train an autoregressive model such as an LSTM or a Transformer on a linearized version of the code grids. But again, feel free to try things out, this area here is very open!

5. Making the model go

Equipped with the two models built in the previous section, you can now write some scripts to interactively or programatically transform sounds at the command-line. You can also now experiment with various ways of editing sounds, have some fun and propose some ideas!

6. Revisiting the approach: environmental concerns in deep learning

References

[1] Prafulla Dhariwal et al. “Jukebox: A Generative Model for Music”. In: 2020. arXiv: 2005.00341 [eess.AS]. URL: http://arxiv.org/abs/2005.00341.

[2] Jesse Engel et al. “Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders”. In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML’17. Sydney, NSW, Australia: JMLR.org, 2017, 1068–1077.

[3] Jesse H. Engel et al. “GANSynth: Adversarial Neural Audio Synthesis”. In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=H1xQVn09FX.

[4] Yann LeCun and Corinna Cortes. “MNIST handwritten digit database”. In: (2010). URL: http://yann.lecun.com/exdb/mnist/.

[5] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. “Neural Discrete Representation Learning”. In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 6306–6315. URL: http://papers.nips.cc/paper/7210-neural-discrete-representation-learning.pdf.

[6] Ali Razavi, Aäron van den Oord, and Oriol Vinyals. “Generating Diverse High-fidelity Images with VQ-VAE-2”. In: *Advances in Neural Information Processing Systems 32, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*. Ed. by Hanna M. Wallach et al. 2019, pp. 14837–14847. URL: http://papers.nips.cc/paper/9625-generating-diverse-high-fidelity-images-with-vq-vae-2.