Hierarchical temporal learning for multi-instrument and orchestral audio synthesis

Antoine Caillon

Thèse de doctorat de Sorbonne Université

Spécialité Informatique

ED130 - Ecole doctorale Informatique, Télécommunications et Electronique (Paris)
Sciences et Technologie de la Musique et du Son (UMR 9912)
Institut de Recherche et de Coordination Acoustique Musique
Equipe Représentations musicales.

Hierarchical temporal learning for music neural audio synthesis

Antoine Caillon

Supervised by Philippe Esling

Directed by Jean Bresson

Defended before the following jury:

Simon Colton, Professor (QMUL)	Reviewer
Bob L. T. Sturm, Associate Professor (KTH Stockholm)	Reviewer
Patrick Gallinari, Professor (Sorbonne Université)	Examiner
Mark Sandler, Professor (QMUL)	Examiner
Michèle Sebag, Principle Scientist (CNRS)	Examiner
Jean Bresson, Senior researcher (IRCAM)	Director
Philippe Esling, Associate Professor (CNRS)	Supervisor

Antoine Caillon: *Hierarchical temporal learning for music neural audio synthesis.*

caillon@ircam.fr - caillonantoine.github.io/phd_support

Version: June 20, 2023

Abstract

Recent advances in deep learning have offered new ways to build models addressing a wide variety of tasks through the optimization of a set of parameters based on minimizing a cost function. Amongst these techniques, probabilistic generative models have yielded impressive advances in text, image and sound generation. However, musical audio signal generation remains a challenging problem. This comes from the complexity of audio signals themselves, since a single second of raw audio spans tens of thousands of individual samples. Modeling musical signals is even more challenging as important information are structured across different time scales, from micro (e.g. timbre, transient, phase) to macro (e.g. genre, tempo, structure) information. Modeling every scale at once would require large architectures, precluding the use of resulting models in real time setups for computational complexity reasons.

In this thesis, we study how a hierarchical approach to audio modeling can address the musical signal modeling task, while offering different levels of control to the user. Our main hypothesis is that extracting different representation levels of an audio signal allows to abstract the complexity of lower levels for each modeling stage. This would eventually allow the use of lightweight architectures, each modeling a single audio scale. We start by addressing raw audio modeling by proposing an audio model combining Variational Auto Encoder (VAE) and Generative Adversarial Network (GAN), yielding high-quality 48kHz neural audio synthesis, while being 20 times faster than real time on CPU. Then, we study how autoregressive models can be used to understand the temporal behavior of the representation yielded by this low-level audio model, using optional additional conditioning signals such as acoustic descriptors or tempo. Finally, we propose a method for using all the proposed models directly on audio streams, allowing their use in realtime applications that we developed during this thesis. We conclude by presenting various creative collaborations led in parallel of this work with several composers and musicians, directly integrating the current state of the proposed technologies inside musical pieces.

Résumé

Les progrès récents en matière d'apprentissage automatique ont permis l'émergence de nouveaux types de modèles adaptés à de nombreuses tâches, ce grâce à l'optimisation d'un ensemble de paramètres visant à minimiser une fonction de coût. Parmi ces techniques, les modèles génératifs probabilistes ont permis des avancées notables dans la génération de textes, d'images et de sons. Cependant, la génération de signaux audio musicaux reste un défi. Cela vient de la complexité intrinsèque des signaux audio, une seule seconde d'audio brut comprenant des dizaines de milliers d'échantillons individuels. La modélisation des signaux musicaux est plus difficile encore, étant donné que d'importantes informations sont structurées sur différentes échelles de temps, allant du micro (timbre, transitoires, phase) au macro (genre, tempo, structure). La modélisation simultanée de toutes ces échelles implique l'utilisation de larges architectures de modèles, rendant impossible leur utilisation en temps réel en raison de la complexité de calcul.

Dans cette thèse, nous proposons une approche hiérarchique de la modélisation du signal audio musical, permettant l'utilisation de modèles légers tout en offrant différents niveaux de contrôle à l'utilisateur. Notre hypothèse principale est que l'extraction de différents niveaux de représentation d'un signal audio permet d'abstraire la complexité des niveaux inférieurs pour chaque étape de modélisation. Dans un premier temps, nous proposons un modèle audio combinant Auto Encodeur Variationnel et Réseaux Antagonistes Génératifs, appliqué directement sur la forme d'onde brute et permettant une synthèse audio neuronale de haute qualité à 48 kHz, tout en étant 20 fois plus rapide que le temps réel sur CPU. Nous étudions ensuite l'utilisation d'approches autoregressives pour modéliser le comportement temporel de la représentation produite par ce modèle audio bas niveau, tout en utilisant des signaux de conditionnement supplémentaires tels que des descripteurs acoustiques ou le tempo. Enfin, nous proposons une méthode pour utiliser tous les modèles proposés directement sur des flux audio, ce qui les rend utilisables dans des applications temps réel que nous avons développées au cours de cette thèse. Nous concluons en présentant diverses collaborations créatives menées en parallèle de ce travail avec plusieurs compositeurs et musiciens, intégrant directement l'état actuel des technologies proposées au sein de pièces musicales.

Acknowledgments

First, I would like to express my gratitude to my supervisor Philippe Esling for offering me the possibility to undertake this thesis, not to mention the countless discussions we had during these three years and his support in setting up many scientific, artistic and professional opportunities. I would also like to deeply thank my director Jean Bresson for his precious help and advice throughout this journey. Furthermore, I would like to thank the *MAKIMONO*, *ACTOR* and *DAFNE+* projects which funded this thesis, and Hugues Vinet for allowing me to finish it properly.

Second, I wish to thank Simon Colton and Bob L. T. Sturm for reviewing this manuscript, alongside Patrick Gallinari, Mark Sandler and Michèle Sebag for examining my thesis defense. I would also like to thank the members of my thesis committee Jesse Engel and Edouard Oyallon for their guidance and help during the second half of this thesis.

Even though starting a PhD at the dawn of a worldwide pandemic does not help with building strong scientific relationships, I had the chance to meet, discuss and work with many people over the course of this thesis. Starting with the members of my lab, I would like to thank Axel Chemla–Romeu-Santos for his unfailing support, and for the creation of precious opportunities of making actual music through the collaboration with *Le Cirque Électrique*. Next, I would like to thank Adrien Bitton, Constance Douwes, David Genova, Giovanni Bindi, Maxime Mantovani, Nils Demerlé, Ninon Devis and Sarah Nabi for all the moments we shared, the successes and obstacles we went through as a team where we started as colleagues, and ended up friends.

Additionally, I would like to thank Maxime Mantovani again, but also Alexander Schubert, as they both gave a proper musical dimension to this thesis, and at the same time helped me find purpose to this work. I would also like to thank Neil Zeghidour, Jesse Engel and Chris Donahue for their time and the invaluable insights and advice they shared over the course of many discussions. Many thanks to Ben Hayes, Georg Hajdu and Espen Sommer

Eide for their trust as they gave me the opportunity to reach many artists and scientists through several masterclasses, each giving new dimensions to this work.

Finally, I would like to thank my parents Françoise and Régis, my brother Paul, Pierre, Anaïs and Connor for their support before, during and (without a doubt) after this thesis.

Last and foremost, I would like to address my deepest thanks to my fiancé Adèle, who supported me during the entirety of my academic journey and helped me get through all the different stages of excitement, happiness, anxiety and despair encountered along the way. While many might say that nothing is ever certain, I know for a fact that none of this would have been possible without her.

Contents

List of Figures

List of Tables

Acronyms

ACIDS Azimuth Conjunction in Declining State. 115

ALiBi Attention with Linear Biases. 40, 79, 80

BERT Bidirectional Encoder Representations from Transformers. 41–43

CNN Convolutional Neural Network. 24, 45, 119

CQT Constant-Q Transform. 16

DAW Digital Audio Workstation. 2, 5, 8, 102

DDSP Differentiable Digital Signal Processing. 50, 51, 91

DFT Discrete Fourier Transform. 13, 15

DSP Digital Signal Processing. 12, 102, 103

ELBO Evidence Lower BOund. 30, 31, 55, 56, 67, 69

EMA Exponential Moving Average. 68

FIR Finite Impulse Response. 15

FM Frequency Modulation. 1

GAN Generative Adversarial Network. v, 28, 29, 32–35, 62, 110

GL Griffin-Lim. 48, 49

GP Gradient Penalty. 32

GRU Gated Recurrent Unit. 26, 80, 84

iDFT Inverse Discrete Fourier Transform. 14

JND Just Noticeable Difference. 62

KL Kullback-Leibler. 29, 30, 66, 67, 69, 70, 108

KS Karplus-Strong. 1

LSTM Long Short-Term Memory. 26

MLM Masked Language Modeling. 41, 42

MLP Multi-Layer Perceptron. 80

MMD Maximum Mean Discrepancy. 67, 69, 70

MPS Metal Perfomance Shader. 104

NF Normalizing Flow. 33–35, 47, 62

NLP Natural Language Processing. 36, 72, 118, 119

PQMF Pseudo Quadrature Mirror Filters. 17

RAVE Realtime Audio Variational autoEncoder. 8, 58, 67, 68, 70–73, 82, 84–86, 91, 92, 97–100, 103–106, 108, 109, 111–113, 115, 118–120

RBF Radial Basis Function. 67

RNN Recurrent Neural Network. 26, 36, 48

RoPE Rotary Positional Encoder. 40

RTF Real-Time Factor. 85, 88, 89, 97

RVQ Residual Vector Quantization. 68–70, 118

SGD Stochastic Gradient Descent. 19, 20

SNR Signal-To-Noise Ratio. 12, 13, 45

SPE Sinusoidal Positional Encoding. 39

STFT Short-Time Fourier Transform. 15, 16

SVD Singular Value Decomposition. 57

TTS Text-To-Speech. 45, 46, 49

VAE Variational Auto Encoder. v, 29–31, 33, 34, 48, 50, 54, 63, 65, 67, 68, 86, 108, 110, 118

VQ-VAE Vector Quantized Variational Auto Encoder. 68

WN Weight Norm. 28

If you don't know what to do, there is actually a chance of doing something new. As long as you know what you are doing, nothing much of interest is going to happen.

Philip Glass

Chapter 1

Introduction

On music and technology

Throughout history, the evolution of music has been closely linked to technological advances. From the invention of the first musical instrument, to the development of electronic synthesizers and computer music, technological innovations have constantly opened up new possibilities for musical expression.

As an example, the *piano forte* and *modern piano* introduced in the 18^{th} century gradually replaced the *harpsichord*, paving the way towards a new type of musical expression with an improved control over the volume level of the instrument. In the 20^{th} century, the development of analog synthesizers and other electronic instruments allowed the exploration of new techniques for musical creation and sound design, leading to the emergence of entirely new music genres, from techno and house to movie scores and *avant-garde* music.

The advent of computer-generated music in the late 20^{th} century revolutionized the way music was created, as artists were no longer limited by the constraints of physical hardware synthesizers. New synthesis methods were invented, from Frequency Modulation (FM) to the Karplus-Strong (KS) algorithm. Sample-based methods were also introduced, such as *wavetable* or *granular* synthesis. In addition to new soundscape possibilities, the development of tools such as Max/MSP or PureData has further expanded the potential of computer-generated music. These programs allow musicians and composers to create music using algorithms and other mathematical models, allowing the creation of complex and intricate musical structures that would be difficult or impossible to achieve using traditional musical instruments or

synthesizers. This has led to the creation of new sub-genres of music, such as algorithmic composition and generative music, as well as the incorporation of algorithmic techniques into more traditional forms of music. Thanks to their modular nature, which allows for fast prototyping, these tools are still opening up new avenues for musical experimentation and exploration.

It is important to consider that the relationship between technological advances and musical evolution is inherently reciprocal. If technology has greatly influenced the development of many musical genres, there are countless instances in history where a particular artistic direction has driven the research and development of new tools or interfaces for music creation. One such example is the design of the *piano-forte*, which originated from the need to add nuances to the sound of the harpsichord. Similarly, pitch correction techniques gained popularity through their use as a musical medium in the late 1990s, leading to a significant amount of work on improving these tools. As shown in Figure 1.1, music genre exploration and technological innovations are intertwined, constantly influencing and shaping each other towards a shared creative goal.

Regardless of technical or artistic innovations, a central part of music creation lies within the physical constraints of the artist itself. While the introduction of *sequencers*, *loopers*, or Digital Audio Workstations (DAWs) allowed the user to manipulate different streams at the same time, one might eventually reach a maximum number of parameters it can control simultaneously. This problem is representative of the compromise between the controllability of an instrument and the diversity of its potential sounds. Many solutions to this problem have been proposed over the years, from the creation of expressive interfaces (e.g. Theremin, Touché controller, ROLI seaboard or Leap Motion) to the simplification of control parameters towards high level concepts (e.g. brightness or flatness).

Today, the continued exploration of new techniques for musical creation and sound design is essential in driving the evolution of music forward and allowing artists to continue to push the boundaries of what is possible. By delving into new ways of interacting with the design process, artists can open up new horizons in terms of musical creativity and continue the long history of innovation in the field.

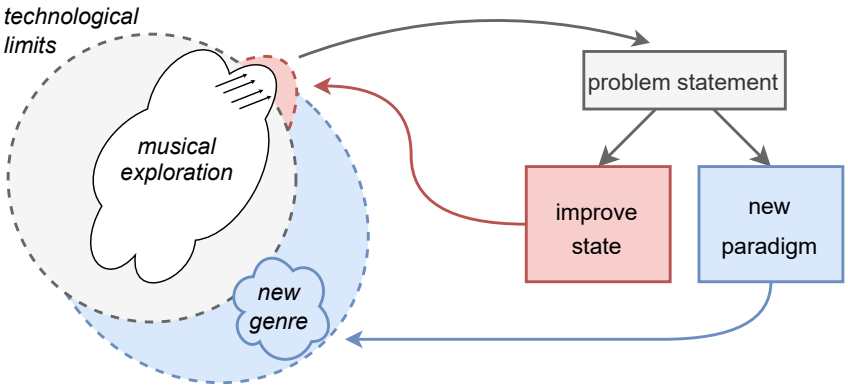


Figure 1.1: Relationship between musical exploration and technological innovations. Musical genre exploration pushes the current technological limits, steering the research and development of new tools and interfaces. Reciprocally, the introduction of new paradigms for musical creation is at the source of novel genres.

Deep learning as the next musical evolution

Complex synthesis techniques often involves a trade-off between an increase in the number of controllable parameters (e.g. large modular systems) and a lack of intuitive interpretation for their effects. This can be especially challenging for musicians who are not trained in the technical aspects of synthesis, as they may not have a strong foundation in the underlying principles and mechanisms of the synthesis process. This lack of understanding can further hinder the ability of the musicians to use their intuition to create a specific sound. While this might be less of a problem for skilled musicians eventually able to reach a set of parameters producing the desired sound, the time spent on this low-level task might hamper their creative flow. Unfortunately, it is not clear if the trade-off between synthesis techniques complexity and produced sounds variety can be avoided at all using classical techniques.

One of the many appeals of using machine learning for musical creation is the paradigm shift from explicit hand-crafted models to example based learning. Indeed, classical synthesis techniques often rely on a preconceived understanding of a given phenomenon to build the processing chain eventually producing a sound. Whether the resulting model is based on a physical study of an existing vibrating system or designed from scratch, it is usually

possible to interact with it through the modification of low-level parameters, directly interfering with core elements of the sound synthesis system. However, building a model encompassing all the parameters involved in the creation of a sound results in extremely complex synthesis methods, when such a model can be found at all.

Applied to audio signals, machine learning techniques instead propose to *learn* a set of parameters associated with a function to perform classification, transformation or generation tasks based on ground truth examples. Interacting with the resulting system usually does not imply a direct transformation of its parameters, but can be achieved through the modification or definition of higher-level concepts depending on the nature of the model. Many existing audio related research fields benefit from using machine learning to solve problems, from source separation [1] to transcription [2] tasks. Additional tasks have been introduced thanks to the potential of *deep learning*, which is the application of machine learning techniques to complex functions leveraging thousands to billions of parameters. Such tasks include generative systems [3, 4] synthesizing audio based on a large scale set of examples, either unconditionally (i.e. autonomously) or based on high-level inputs (e.g. text description, style or lyrics).

Even though the use of deep learning techniques has improved the state-of-art in many research fields, it does not come without a cost. Indeed, deep learning systems are characterized by their high computational complexity and extensive data requirements. This implies the use of powerful computational resources and large amounts of training data for effective performance. The design and training of these methods can be challenging, and their complex nature can hinder understanding or interpretation of their predictions. Finally, the quality of the training dataset is crucial to achieve reasonable performances. Gathering a good enough dataset for a given task can be expensive or in some cases even impossible.

Regarding audio applications, the integration of deep learning methods for general-purpose audio processing (i.e. transformation or generation) is far from trivial, mainly because of the need for *realtime enabled systems*, able to process audio signals based on fixed size buffers while doing it faster than realtime. This yields several problems, as the current state-of-the-art in deep learning applied to audio processing is mostly not streamable (i.e. they cannot process audio *streams* and are limited to finite audio files where the full signal is readily available). In addition to this, most existing techniques for high-quality audio synthesis and processing are significantly slower than realtime. Furthermore, the integration of deep learning methods inside regu-

lar DAWs is relatively unexplored, leaving artists with offline command line utilities as the main interaction with state-of-the-art techniques. Overall, processing high-quality audio signals with deep learning methods remains challenging for many reasons, most of them issued from the structural complexity of audio signals themselves.

Nonetheless, the potential of deep learning for musical creativity, and even art in general is huge. Recent advances in data-driven generative modeling have yielded impressive results in terms of generation quality, control and diversity for various tasks including text, image, sound and video synthesis.

Temporal scales of musical audio signals

Digital audio signals are complex to analyze or synthesize, as they are information streams containing thousands of individual samples per second. Usual low-fidelity signals are composed of 8000 samples per second, which is at least 5 times less than audio signals considered as "high quality". Furthermore, audio signals are structured according to multiple *scales*, where different levels of information are scattered across the signal. Building a proper understanding of the behavior of an audio signal thus requires to process all scales at once, which is a particularly complex problem.

(a) 20 milliseconds (b) 100 milliseconds (c) 3 seconds

Figure 1.2: Different waveform scales.

At the lowest level of representation, audio signals can be thought of as the concatenation of a few individual audio samples (e.g. 5ms to 20ms, see Figure 1.2(a)), whose temporal organization and amplitude correlate with fine details of the signal, such as its pitch, timbre, and other spectral characteristics. Studying the organization of multiple groups of those samples can lead to the understanding of higher-level features about the signal such as *phonemes* for speech applications (e.g. 20ms to 100ms, see Figure 1.2(b)). Considering higher-level scales yield features that depend on the type of analyzed signal. Continuing with the speech example, modeling the relationship between phonemes gives access to information about words and sentences.

Going even higher level might result in insights about the overall *context* from which the previous sentences, words, pitches and timbres are extracted.

Accurately modeling musical audio signals thus requires considering all scales simultaneously, as low-level features such as pitch and timbre are just as crucial as the context from which they originate.

A multi-scales approach to audio modeling

Regarding the analysis and synthesis of musical audio signals, our main hypothesis is that a suited method for modeling multi-scales signals lies in the design of a hierarchical approach, addressing each scale separately. In contrast with the use of a single system processing all scales at once, such a multi-scales system would be beneficial for several reasons. Being able to access multiple levels of predictions would help with the interpretability of predictions, while granting the user more control over the generation as shown in Figure 1.3.

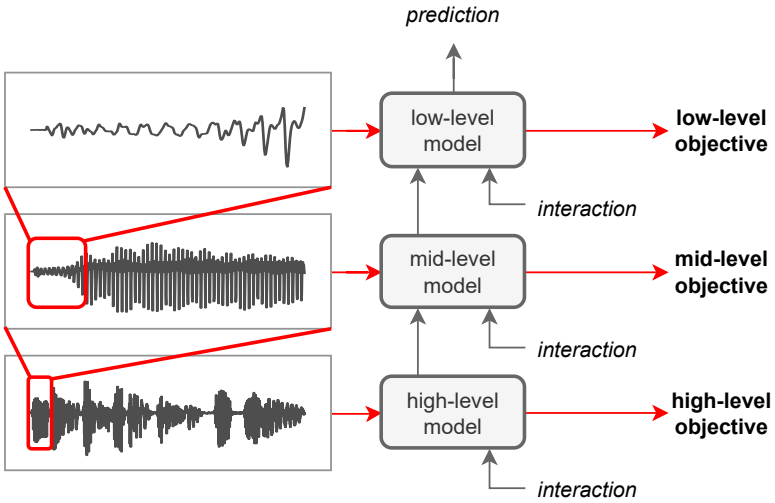


Figure 1.3: Multi-scales modeling of an audio signal. Different scales of audio are fed to several models with separate objectives. The optimization of all models is performed individually (red arrows), however models are combined in order to produce a prediction encompassing all scales (gray arrows). Interacting with the intermediate states of this combined model gives the user a multi-scales control over the generation.

Furthermore, we believe that a system composed of several sub-models tailored to process a single audio scale may help with reducing the computational complexity of the overall model. This is fundamental to obtain models usable in real-time, which is one of the main targets of this thesis.

Outline

In this thesis, we propose a deep learning framework for modeling musical audio signals using a hierarchical approach. Since all of our contributions leverage many tools from the machine learning field, we start by introducing important concepts about machine learning model definition and optimization in Chapter 2, alongside relevant works from the deep learning literature. This includes general purpose modeling techniques in addition to audio-oriented methods to analyze and synthesize musical signals.

Then, we present our work on *representation learning* applied to audio signals in Chapter 3, intended as the first building block of our hierarchical approach. Indeed, representation learning models aim at extracting salient features about an input dataset. Therefore, it can be used to provide a compact and higher-level representation of an audio signal that can be later used by a higher-level model. Therefore, we propose the Realtime Audio Variational autoEncoder (RAVE) model, which in essence is a representation learning model analyzing and synthesizing high-quality raw audio waveforms. With the full hierarchical objective in mind, we introduce several techniques to reach perceptually high-quality synthesis, while enforcing the model to learn the most compact representation possible. As we eventually target the use of the resulting techniques in real-time, we optimize the architecture of the model to make it 20 to 80 times faster than real-time without leveraging expensive hardware accelerators.

In Chapter 4, we address the task of learning the temporal behavior of the previously learned compact representation. We frame the problem as an *autoregressive* task, where our proposed system is optimized to predict the probability distribution of a single latent time-step given the previous ones. We study the performances of several architecture types, and propose the use of different probabilistic graphical models to reach the best accuracies while staying faster than real-time during generation. A comparison with previous works shows that our methods outperform all baselines in terms of accuracy while being faster during both training and generation.

In order to make our work compatible with real-time processing, we introduce in Chapter 5 several techniques to convert a pretrained deep learning model into a *streaming* model, effectively allowing it to process live audio streams. We adapt this method to our RAVE model, and show that the resulting streaming model behaves exactly as the original, while gaining access to buffer-based processing. We additionally introduce a set of applications designed to interface pretrained streaming models with regular DAW, such

as Max/MSP and PureData.

Designing artistically relevant tools can be complex without the insights from actual artists. Therefore, we collaborate with several professional artists to identify the most important features that models and interfaces should integrate. We present in Chapter 6 the result of these collaborations, both in terms of artistic outcome and scientific insights. Finally, we conclude this thesis in Chapter 7 by reflecting on the different methods proposed, alongside potential future developments.

We strongly encourage the reader to visit our supporting webpage[1] while reading to get additional information, audio and video examples and interactive demonstrations of our work.

[1] See `caillonantoine.github.io/phd_support`

As individuals, each of us is extremely isolated, while all linked by a prototypical memory.

Haruki Murakami

Chapter 2

State-of-the-art

In this chapter, we present the main concepts and ideas needed to understand the remainder of this thesis. First, we present fundamental notions in Digital Signal Processing (DSP), spectral analysis and filtering in Section 2.1. Then, we introduce the theory of machine learning and model optimization in Section 2.2. This allows us to present the current state-of-the-art in deep generative modeling in Section 2.3, sequence modeling in Section 2.4 and details about deep audio modeling in Section 2.5.

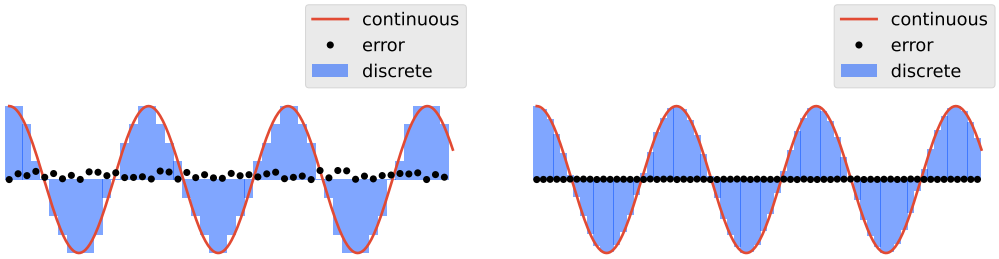
2.1 Digital audio synthesis

Digital audio signals represent continuous electric voltages originally produced by various transducers (e.g. microphones, sensors). Over the past decades, audio signals have started to be artificially crafted, or *synthesized* using electronic circuits emulating oscillations of the air pressure. While many types of synthesizer exist, most of them are built around a simple model of sound production, usually describing high level features about the target signal as well as ad-hoc control parameters over it. As an example, additive synthesizers model audio signals as a sum of sinusoidal components whose individual amplitudes can be modified to shape perceptual attributes of the sound. In contrast, subtractive synthesizers use an initial broadband excitation signal transformed using a set of filters. Compared to subtractive synthesizers, hardware additive synthesizers are less common due to the need to integrate individual oscillators for each harmonic in the final sound, quickly increasing the final cost of the machine.

The birth of computer-based synthesizers have lifted these constraints, allowing to build synthesis systems with an arbitrary number of oscillators with no additional cost. Hence, new algorithms for sound synthesis also appeared, such as *frequency modulation* and *granular synthesis*. Processing audio signals using a computer is called Digital Signal Processing (DSP), which operates on discretized audio signals. The discretization process aims at transforming amplitude- and time-continuous signals into discrete amplitude and time arrays. Important parameters involved during the discretization are the *sampling rate* and *resolution* (*bit depth*). The sampling rate defines the number of voltage measures needed to discretize one second of audio, and is measured in Hertz (Hz). The resolution defines how many bits are allocated to approximate the continuous value of each voltage measure. We show in Figure 2.1 an example of a continuous signal being discretized with different discretization parameters.

Choosing the right values for both parameters is crucial to obtain a faithful discrete version of the original signal. According to the Nyquist criterion, the sampling rate must be at least two times higher than the maximum frequency present in the signal. If this requirement is not met, the resulting signal will show signs of aliasing, which produces spurious distortions frequencies, hence, degrading the overall quality of the signal. The resolution has a direct impact on the Signal-To-Noise Ratio (SNR) of the discretization process, defined as

$$\mathrm{SNR}_{\mathrm{dB}} = 20 \log_{10}(2^n), \tag{2.1}$$



(a) 50Hz, 3bits (b) 67Hz, 7bits

Figure 2.1: Example of the discretization of a continuous sinusoid using different sampling rates and resolution values. Increasing the sampling rate and resolution helps reducing the discretization error.

where n is the number of bits used during the discretization. A higher SNR means a larger dynamic range for the discretized signal, which is important for high-fidelity audio. Overall, the discretization process of a continuous audio signal requires both a high sampling-rate and a high resolution in order to obtain a faithful discrete signal. While modern Analog to Digital Converters (ADCs) are perfectly capable of discretizing audio signals without any perceptual difference with their continuous counterpart, this comes at the cost of a large number of individual audio samples per second. As an example, 20 seconds of reasonably high-quality audio sampled at 44.1kHz (i.e. the most common sampling rate for digital audio) produce a vector composed of nearly a million audio samples.

2.1.1 Fourier analysis

As discussed previously, a straightforward representation for audio signals is the raw waveform. However, this representation does not accurately reflect the way sounds are perceived. Indeed, signals with extremely similar perceptual attributes can be represented with largely different waveforms, as shown in Figure 2.2.

A more perceptually-relevant representation for audio signals can be obtained through the use of the *Fourier transform*. This can be used to compute the *spectrum* of an audio signal, yielding information about the frequencies present in the signal, alongside the phase for each frequency. In particular, for discrete signals, we use the Discrete Fourier Transform (DFT) defined as

(a) (b) (c)

Figure 2.2: Three audio signals presenting large differences in terms of raw waveform distance but sounding nearly identical.

$$\mathbf{X}[k] = \sum_{n=1}^{N} \mathbf{x}[n].e^{-2j\pi kn/N}, \tag{2.2}$$

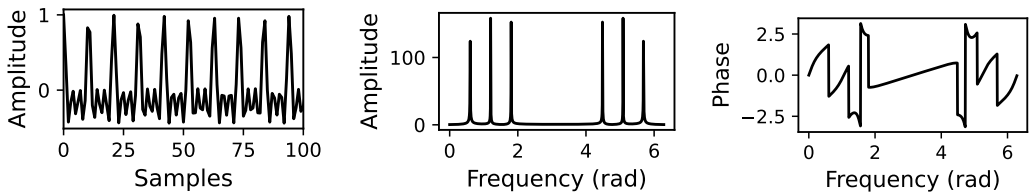
where $\mathbf{X}[k]$ is the Fourier transform of signal $\mathbf{x}$ for the reduced frequency $\nu = \frac{k}{N}$ and N is the total number of audio samples in $\mathbf{x}$. This yields complex numbers, where the second half of the spectrum of a real signal is the complex conjugate of the first half. This implies that any spectrum of a real signal can be cropped up to the first $k + 1$ values without loss of information. The Fourier transform being an invertible operation, we can retrieve the original signal from its spectrum by using the Inverse Discrete Fourier Transform (iDFT), defined as

$$\mathbf{x}[n] = \frac{1}{N} \sum_{k} \mathbf{X}[k].e^{2j\pi kn/N}. \tag{2.3}$$

We show in Figure 2.3 the spectrum of an audio signal obtained by summing three sinusoids with different frequencies. There are six spectral peaks in the amplitude spectrum shown in Figure 2.3(b), corresponding to the frequencies of the three sinusoids, duplicated in the second half of the spectrum. Differences between the phases of each sinusoid can be seen in Figure 2.3(c), where the second half of the phase of the spectrum is indeed a conjugated symmetry of the first half.

2.1.2 Filtering

While the Fourier transform already allows for a finer analysis of spectral information, it can be further used for filtering purposes by applying modifications to the yielded spectrum. A simple way of modifying the spectral content of an audio signal is to multiply the amplitude of the spectrum with



(a) Raw waveform (b) Amplitude of DFT (c) Phase of DFT

Figure 2.3: Spectrum of a real audio signal composed of three sinusoids with different frequencies. The second half of the phase is the conjugated mirror of the first half.

a given mask. This filtering method is called the *window method*, and can be used to define a Finite Impulse Response (FIR) filter. Filtering an input signal $\mathbf{x}$ with a binary mask $\mathbf{H}$ can thus be achieved using both forward and inverse DFT, following

$$\mathbf{y} = \text{iDFT}\left(\text{DFT}(\mathbf{x}) \times \mathbf{H}\right). \tag{2.4}$$

Thanks to the *convolution theorem*, we know that this operation is equivalent to the following convolution in the time domain

$$\mathbf{y} = \mathbf{x} * \mathbf{h}, \quad \mathbf{h} := \text{iDFT}(\mathbf{H}), \tag{2.5}$$

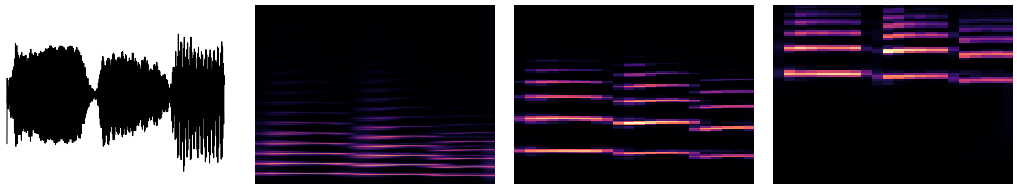
where the convolution $*$ between two discrete arrays $\mathbf{x}$ and $\mathbf{h}$ is defined as

$$(\mathbf{x} * \mathbf{h})[n] = \sum_k \mathbf{x}[n-k]\mathbf{h}[k]. \tag{2.6}$$

2.1.3 Time-frequency representations

While computing the DFT of an audio signal allows to understand its overall spectral content, grasping the temporal behavior of signal becomes impossible, as the time axis is averaged during the transform. A combination of both frequency and temporal information can be obtained by stacking spectrums computed on overlapping windows of signal. The resulting matrix is called a Short-Time Fourier Transform (STFT) or *spectrogram*. Spectrograms are

inherently redundant due to the overlapping signal windows. However, they are often used for analysis as they contain both spectral and temporal information in a single compact representation. An example of the STFT of an audio signal is available in Figure 2.4(b).



(a) Raw waveform (b) STFT (c) Mel-STFT (d) CQT

Figure 2.4: Several representations for the same input audio signal. All spectrograms are plotted in log-scaled amplitude for visualization purposes.

An interesting modification of the STFT is to map its frequencies to a *Mel scale* [5], which more closely matches how humans perceive differences in pitch. The mapping between Mel frequencies m and regular frequencies f is obtained by computing

$$m = 2595 \log_{10} \left(1 + \frac{f}{700} \right). \tag{2.7}$$

An example Mel-scale spectrogram is shown in Figure 2.4(c). Another similar transform is the Constant-Q Transform (CQT). This transform has the particularity to be shift-invariant: a pitch shift of the audio signal results in a vertical translation of the CQT.

2.1.4 Multiband decomposition

Understanding both the spectral and temporal behavior of audio signals can be used for signal compression purposes. Indeed, such representations are useful to identify parts of the spectrum with low energy where fewer bits of information are needed for perceptually similar results. This is important for audio codecs such as *mp3* or *opus*, and for efficient transmission of audio signals. While spectrograms may seem like a good candidate representation for such tasks, the redundancy introduced by the overlapping signal windows imply an increase in the dimensionality of the original signal. A more suited class of transforms are *multiband decompositions*.

The goal of a multiband decomposition is to represent an audio signal sampled at a given sampling rate (e.g. 48kHz) as a combination of several down-

sampled signals (e.g. 3kHz), where each sub-signal covers a particular range of frequencies. Indeed, such decompositions do not introduce additional information, but rather transform a signal vector of length N into a matrix with shape $N/K \times N$, where K is the number of bands. Pseudo Quadrature Mirror Filters (PQMF) [6] allow a near perfect analysis-synthesis transform, based on the modulation of a prototype low-pass filter $\mathbf{h}$ with cutoff frequency

$$f_c = \frac{\text{sampling rate}}{2K}, \tag{2.8}$$

carefully designed to avoid aliasing in the reconstructed signal. Thus, the filter for band k is defined as

$$\mathbf{h}_k[n] = \mathbf{h}[n] \cos\left(\frac{\pi}{4K}(2k+1)(2n-N+1)\right) + \Phi_k, \tag{2.9}$$

$$\Phi_k = (-1)^k \frac{\pi}{4} \tag{2.10}$$

In practice, it is impossible to create a perfect prototype filter (i.e. perfectly rejected bands), but we can instead design a filter that allows the cancellation of aliasing between neighboring sub-bands, using methods such as the optimization of a non-linear objective function as described by [7], or using a simple Kaiser window whose parameters are optimized [8]. Since the filter-bank is an orthogonal basis of the signal due to the cosine modulations implied during its creation, the re-synthesis process can be performed by reapplying a temporally flipped version of the filter-bank to the sub-bands.

2.2 Machine Learning

Machine learning can be defined as the set of techniques involving the optimization of a parameterized function, by minimizing a given cost function based on ground truth examples. The applications of machine learning are numerous, spanning from simple classification and regression tasks to complex prediction and generation problems. Formally, we consider a parametric function f_θ defined as

$$\begin{aligned} f_\theta : \quad & \mathbb{R}^M \times \mathbb{R}^D \to \mathbb{R}^N \\ & \mathbf{x}, \theta \mapsto \hat{\mathbf{y}}, \end{aligned} \tag{2.11}$$

where D is the number of parameters. For simplicity, we define all the variables in $\mathbb{R}$, but extension to other sets are possible (e.g. natural or complex). Learning is achieved through the minimization of a cost function $\mathcal{L}$, also called *loss*, yielding a scalar given $\mathbf{x}, \hat{\mathbf{y}}, \theta$ and f_θ. This cost function should reflect how well the function f_θ is approximating a target ground-truth function f, usually unknown and accessible exclusively through the observation of paired ground truth examples $(\mathbf{x}, \mathbf{y}) \in \mathcal{X} \times \mathcal{Y}$. Thus, by minimizing $\mathcal{L}$ we are implicitly reducing the distance between f_θ and f, eventually reaching a theoretical optimal solution where

$$\begin{aligned} & f_{\hat{\theta}}(\mathbf{x}) \approx f(\mathbf{x}) \ \forall \ \mathbf{x} \in \mathbb{R}^M, \\ & \hat{\theta} = \underset{\theta}{\operatorname{argmin}} \ \mathcal{L}(\mathbf{x}, \mathbf{y}, \theta, f_\theta) \ \forall \ (\mathbf{x}, \mathbf{y}) \in \mathcal{X} \times \mathcal{Y}. \end{aligned} \tag{2.12}$$

A machine learning problem is therefore defined as the combination of a ground truth, a parametrized function and a set of parameters optimized to minimize the cost function. The nature of the function as well as the loss and the ground truth play an important part when addressing a specific task, and is the subject of a lot of research to reach higher accuracies and lower computation complexity. We call the combination of the parametrized function with its parameters a *model*.

2.2.1 Parameter optimization

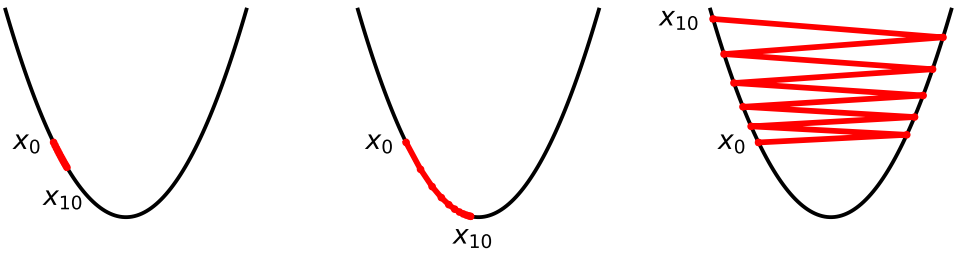
Optimizing a set of parameters to minimize a given cost function can be achieved through different methods, including the selection process of genetic algorithms, the maximization of rewards for reinforcement learning, or

using gradient descent. In the context of this thesis, we focus exclusively on optimization through gradient descent, even though reinforcement learning might show great potential for musical creativity.

Gradient descent is an iterative algorithm designed to find a local minimum of a differentiable function given an initial starting point [9]. It works by taking small steps in the opposite direction of the gradient of the function evaluated at the current point until a local minimum is reached. In our case, we seek to minimize the cost function $\mathcal{L}$, which can be obtained by iteratively updating the parameters θ such that

$$\theta \leftarrow \theta - \alpha \sum_{\mathbf{x},\mathbf{y} \in \mathcal{X}} \frac{1}{|\mathcal{X}|} \nabla_\theta \mathcal{L}(\mathbf{x}, \mathbf{y}, \theta, f_\theta), \tag{2.13}$$

where α is a non-negative hyperparameter called *learning rate*. The choice of the learning rate value is fundamental to make the optimization process converge, as too small values yield slow convergence, while too large values might result in divergence. We show in Figure 2.5 how different values for α can lead to dramatically different outcomes for the optimization procedure.



(a) $\alpha = 0.01$ (b) $\alpha = 0.1$ (c) $\alpha = 1.01$

Figure 2.5: Gradient descent applied to the function $f(x) = x^2$, with different learning rates α. Choosing a too small value for α makes the optimization slow, while a too large value makes the optimization diverge.

In practice, both the large amount of data and the high complexity of models prevent the use of the update described in Equation (2.13). Instead, modern machine learning methods leverage an update rule based on Stochastic Gradient Descent (SGD). SGD optimizes the parameters using batches of examples randomly sampled from the dataset instead of using all samples from it at each iteration, yielding the following update rule

$$\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\mathbf{x}, \mathbf{y}, \theta, f_\theta), \quad (\mathbf{x}, \mathbf{y}) \sim \mathcal{X} \times \mathcal{Y}. \tag{2.14}$$

Using SGD instead of regular gradient descent has empirically proven to be useful to avoid getting stuck in suboptimal minima, at the cost of introducing instabilities during optimization. The popular Adam optimization procedure [10] includes an adaptive momentum to the update rule, most of the time yielding better and faster convergence.

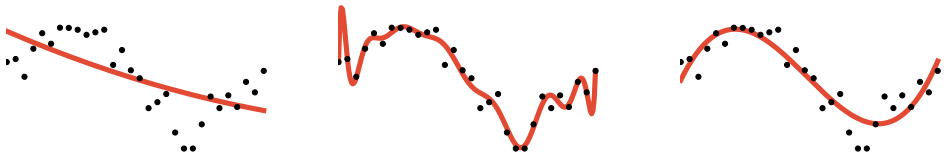
Another important hyperparameter is the *capacity* of the model. This capacity is directly linked to its expressive power, defining an implicit upper bound on the complexity of the ground truth function f that the model can fit. Complex tasks usually require a particularly suited function f_θ or a large capacity. It is important to note that while sub-parametrization of a model is detrimental to its performance, over-parametrizing it might yield generalization issues. To examplify this, we consider a simple polynomial regression task on the following ground truth function

$$\begin{aligned} f : [0, 2\pi] &\to \mathbb{R} \\ x &\mapsto \sin(x) + z, \quad z \sim \mathcal{N}(0, 1/4), \end{aligned} \tag{2.15}$$

which is a full period of a sinusoid perturbed by a Gaussian noise. The training dataset is built by sampling 30 uniformly spaced points between 0 and 2π. We consider a polynomial model consisting in K parameters with the following parametrized function f_θ

$$\begin{aligned} f_\theta : \mathbb{R} \times \mathbb{R}^K &\to \mathbb{R} \\ x, \theta &\mapsto \sum_{k=0}^{K} x^k \theta_k. \end{aligned} \tag{2.16}$$

The model is optimized using the least-square method. As we can see in the predictions in Figure 2.6, it is clear that an over-parametrization of the model helps the convergence on training examples, while missing the underlying structure of the dataset. However, real-life problems often involve complex tasks with high-dimensional data. Therefore, identifying the correct capacity of a model is usually not a trivial endeavor.



(a) Under-fit ($k = 2$) (b) Over-fit ($k = 16$) (c) Correct order ($k = 4$)

Figure 2.6: Effect of different orders for polynomial regression. Data points are represented with the black dots, while the corresponding polynomial approximation is drawn as a red line.

A method to help identify this overfitting regime lies in *cross-validation*, where the input dataset is split into several subsets. The *training* set is used in order to perform gradient descent on the parameters of the model. The *validation* set is used during training to evaluate the performances of the model on unseen data. An optional *test* set can be created and hidden from the researcher or developer to avoid biases in the choice of hyperparameters, implicitly optimizing a specific validation set. In the context of our polynomial example, we create a validation set by redrawing the noise in Equation (2.15) in order to get new data points. We evaluate the prediction error of the model by measuring the average L_1 distance between the prediction and data points, and report the results in Table 2.1.

Capacity	Train set	Validation set
$k = 2$	0.453	0.435
$k = 4$	0.233	**0.200**
$k = 16$	**0.106**	0.278

Table 2.1: Polynomial model prediction errors using different capacities

It is clear that using the over-parametrized model yields the best training error, but missing the global structure of the data makes it unable to properly generalize to unseen data.

2.2.2 Learning types

The learning tasks addressed along this thesis belong to one of four different learning regimes. Those regimes depend on both the nature of the dataset and the type of the target task.

Supervised learning

Supervised learning is the most common learning regime. It is defined as a problem where we have ground truth pairs $(\mathbf{x}, \mathbf{y}) \sim \mathcal{X} \times \mathcal{Y}$, where $\mathbf{x}$ is an input example (e.g. text, image, sound or video) and $\mathbf{y}$ is the output of the unknown target function $f : \mathbf{x} \to \mathbf{y}$ for which we build and optimize an approximation f_θ. The tasks belonging to the supervised regime are usually classification or regression.

Unsupervised learning

In contrast with supervised learning, the goal of *unsupervised learning* is to find patterns or structures dictating the behavior of a set of unlabeled examples $\mathbf{x} \sim \mathcal{X}$. Typical tasks that fall into this category are clustering, representation learning and generative modeling.

Semi-supervised learning

Semi-supervised learning lies at the intersection between unsupervised and supervised learning, where we have access to a dataset $\mathcal{X}$ composed of both labeled *and* unlabeled examples. The task of the model is therefore to build an understanding of the distribution behind the examples $\mathbf{x}$, while using the available pairs $(\mathbf{x}, \mathbf{y})$ to infer pseudo-labels for the unlabeled examples. The usual tasks for semi-supervised learning are classification and regression.

Self-supervised learning

The last learning regime is *self-supervised learning*. This can be seen as the combination between the data availability of unsupervised learning and the task definition of supervised learning. The idea is to exploit structures inside unlabeled examples $\mathbf{x}$ to address classification tasks. *Contrastive learning* is an example of self-supervised learning, where a high-level representation of a dataset is built by proposing a given *pretext task*, which allows classification of positive-negative pairs even without labeled data.

2.2.3 Neural networks

With increasingly complex problems comes the need for larger models, with a number of parameters varying between a few hundreds to several billions. While polynomial models are suited to scalar regression tasks, classification and generation problems need specialized parametrized functions able to fit

a broader range of ground truth functions. One such class of specialized function is the fully-connected layer.

Fully-connected layers

Fully-connected layers are one of the seminal building blocks of deep learning systems. They are defined as an affine transform mapping an input vector $\mathbf{x} \in \mathbb{R}^{D_x}$ to an output vector $\mathbf{y} \in \mathbb{R}^{D_y}$ using a weight matrix $\mathbf{W} \in \mathbb{R}^{D_y \times D_x}$, a bias vector $\mathbf{b} \in \mathbb{R}^{D_y}$ and a non-linear activation function σ, following

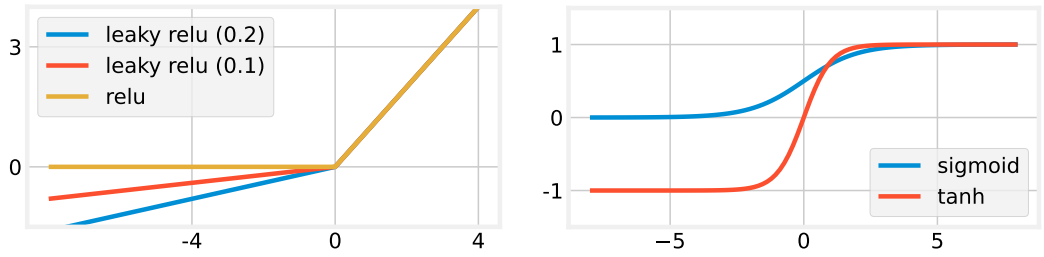
$$\mathbf{y} = \sigma\left(\mathbf{W}\mathbf{x} + \mathbf{b}\right). \tag{2.17}$$

Both the values of the weights and biases are optimized during training, while the activation function usually remains unchanged. Thanks to the non-linear behavior of this activation function, we can increase the computational complexity of the model by stacking linear layers together [11]. Not using any activation function would render this stacking useless, as any linear combination of affine transforms can be achieved using a single affine transform. We define in Table 2.2 and Figure 2.7 several of the most used activation functions that can be found in the literature.

Name	Function
Sigmoid	$f : x \to 1/(1 + e^{-x})$
Tanh	$f : x \to (e^{2x} - 1)/(e^{2x} + 1)$
ReLU	$f : x \to \max(0, x)$
Leaky ReLU	$f : x, \lambda \to \max(\lambda x, x)$
SiLU	$f : x \to x/(1 + e^{-x})$

Table 2.2: Some usual activation functions.

As the complexity of the network increases with the number of stacked layers, naively computing the gradient of the parameters θ with relation to the loss $\mathcal{L}$ becomes tedious. This comes from the fact that any given layer i in a stack of N layers has its gradients with respect to the loss impacted by the following layers. A simpler way of computing those gradients can be achieved through the *chain rule* and back-propagation. We consider a model composed of stacked linear layers $f_l, l \in \{1, ..., L\}$ with corresponding weights $w^l_{j,i} \in \mathbb{R}$ and biases $b^l_i \in \mathbb{R}$, and a cost function $\mathcal{L}$. Each function f_l is applied to the



(a) Rectifier variations

(b) Bounded activations

Figure 2.7: Some usual activation functions.

previous hidden layer h_{l-1} and yields another hidden layer h_l. We can obtain the derivative of the loss with respect to individual parameters using

$$\frac{\partial \mathcal{L}}{\partial w_{j,i}^{l}} = \frac{\partial \mathcal{L}}{\partial h_l} . \frac{\partial h_l}{\partial w_{j,i}^{l}}, \tag{2.18}$$

where the first term in the right-hand side of Equation (2.18) can be recursively decomposed into products of partial derivatives following the same scheme. In practice, derivatives for the parameters of such models are efficiently computed using back-propagation.

Convolutional layers

One of the downsides of using simple linear layers is their inability to process data with variable size, such as temporal sequences or images. Convolutional Neural Network (CNN) [12] address this problem by computing the cross-correlation between a kernel and the input data, as shown in Figure 2.8. This allows the use of variable-size input data, and has proven to be particularly efficient at detecting structured patterns.

Formally, we consider an input tensor $\mathbf{x}$ with shape $[C_{\text{in}} \times D_1 \times \ldots \times D_n]$, where C_{in} is the number of channels, and $n > 1$ is all additional dimensions. Such inputs can be transformed by a n-dimensional convolutional layer parametrized with a weight ω of shape $[C_{\text{out}} \times C_{\text{in}} \times K_1 \times \ldots \times K_n]$, C_{out} being the number of output channels, and K_i the size of the kernel for the input dimension i. The output tensor $\mathbf{y}$ is the result of the cross

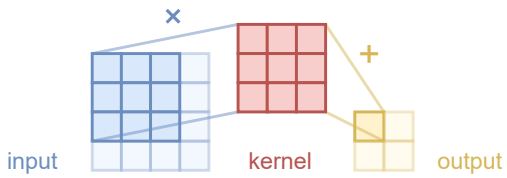
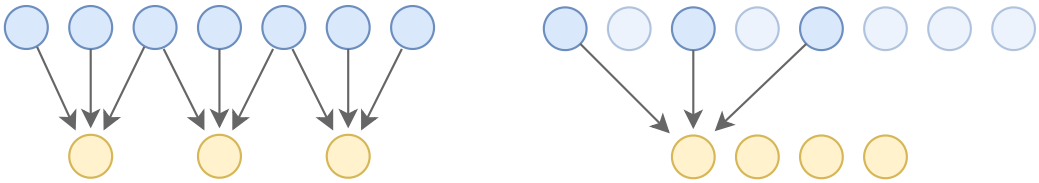


Figure 2.8: A 2-dimensional convolutional operation with a kernel of size 3×3. Parts of the input tensor are multiplied element-wise by the kernel and summed to produce a single element in the output. The operation is repeated for each patch in the input according to a given stride.

correlation between $\mathbf{x}$ and ω, following

$$\mathbf{y}[c_o, i_1, \ldots, i_n] = \sum_{c_i=0}^{C_i-1} \sum_{k_1=0}^{K_1-1} \cdots \sum_{k_n=0}^{K_n-1} \omega[c_o, c_i, k_1, \ldots, k_n] \times \mathbf{x}[c_i, i_1 + k_1, \ldots, i_n + k_n]. \tag{2.19}$$

Usual values of n are 1 for sequences (e.g audio, text), 2 for images (width and height), and 3 for videos (width, height, time). Kernels are usually far smaller than the input, as the convolution operation is computationally demanding. Indeed, a n-dimensional convolution with kernel size $k_i = k$ and input size t has a complexity of $\mathcal{O}(k^n t)$. While the use of the convolution theorem would decrease the computational complexity of the operation using Fourier transforms, most convolution implementations leverage parallel processing abilities of modern accelerators to compute it directly.



(a) Strided convolution with stride 2

(b) Dilated convolution with dilation 2

Figure 2.9: 1-dimensional convolution with stride (a) or dilation (b).

Using a stride $s > 1$ can be used to reduce the size of the input, as shown in Figure 2.9(a). This is often used to gather information around a given point, and yield higher level features about the input [12]. The authors of [3] propose the use of *dilated* kernels (also called *à-trous convolution*), allowing to

increase the effective size of the kernel without introducing new parameters, as shown in Figure 2.9(b).

Recurrent layers

Another kind of layer able to process variable-length inputs are Recurrent Neural Network (RNN). Several variants of those recurrent layers exist, including Long Short-Term Memory (LSTM) [13] or Gated Recurrent Unit (GRU) [14], but all follow the same overall approach, as depicted in Figure 2.10.

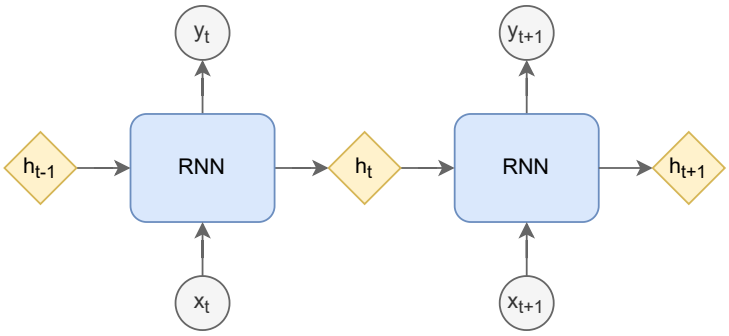


Figure 2.10: Schematic view of a recurrent neural network. Each time step is processed using a recurrent state computed from the previous time steps.

We consider an input sequence $\mathbf{x}$ with shape $T \times D$, where T is the length of the sequence and D is the dimensionality of individual time steps. Recurrent layers are applied sequentially to each time-step $\mathbf{x}_t$, and yields an output $\mathbf{y}_t$ together with a recurrent state $\mathbf{h}_t$ that is fed to the recurrent layer during the processing of the next time step. This recurrent state mechanism allows to progressively build a temporal context that is used to transform a given time step. It is therefore particularly well suited to autoregressive tasks and sequence processing.

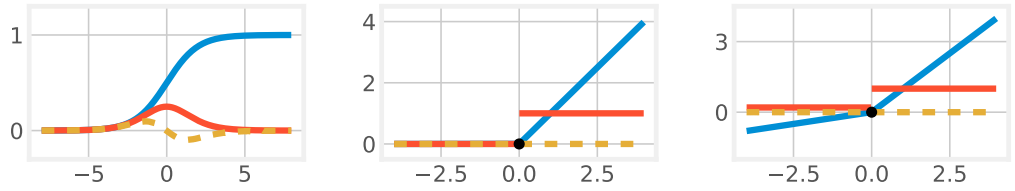
2.2.4 Scaling to deep models

Since their broad adoption by the research community, deep learning models tend to increase in size and complexity in order to maximize performance. However, increasing the number of layers comes at a cost. The gradients coming from the loss computation get increasingly smaller as they get propagated through the successive activation functions. This problem is called *vanishing gradient* [15], and is known to hamper convergence dramatically [16]. Creating deeper models might also result in over-parametrization, as

described in Section 2.2.1, potentially making the model overfit on the training dataset. To address these problems, many regularization techniques and architectural designs have been proposed to build very deep models.

Normalization

While the non-linear behavior of activation functions is desirable, looking at their first and second derivative can give information about the values yielding a null gradient or areas of the function that are acting purely linearly. In Figure 2.11, we plot three activation functions with their respective fist and second derivatives. Outside of the range $[-5, 5]$, the gradients returned by the sigmoid function are close to 0, preventing any learning for the previous layers. Areas with null gradient are called *saturating areas*, and are to be avoided at all cost. Leaky ReLU have been introduced to avoid the saturating area of the original ReLU activation for negative values, by introducing a parametrizable slope. However, if the input values are either strictly positive or negative, this function behaves linearly, which renders stacking useless. Forcing input values to fit in effective activation area is therefore important for proper convergence and is usually addressed using *normalization layers*.



(a) Sigmoid (b) ReLU (c) Leaky ReLU

Figure 2.11: Several activation functions with their derivative. The activation functions (resp. first and second derivatives) are colored in blue (resp. red and yellow). A black dot indicates that the derivative of the function cannot be computed.

There are multiple types of normalization layers, but here we focus on three of the most used ones. *Batch* [16], *Layer* [17] and *Instance* [18] normalization follow the same approach, where the mean and variance of the input tensor are computed and used for normalization purposes, following

$$\mathbf{y} = \frac{\mathbf{x} - \mu(\mathbf{x})}{\sigma(\mathbf{x})} \,.\, \alpha + \beta \tag{2.20}$$

where α, β are optional learnable parameters. The method for computing the mean and variance changes with the type of normalization used. *Batch* computes input statistics averaging over batch and spatial dimensions. In contrast, *Layer* averages over channels and spatial dimensions, while *Instance* averages only over spatial dimensions. Batch normalization additionally learns α, β, whereas others use $\alpha = 1, \beta = 0$.

Using such normalization layers is now widespread, as they yield faster and more stable convergence at the cost of a small computational overhead. However, there are situations where layers themselves need normalization, whether we use Linear, Recurrent or Convolutional layers. One of those model normalization techniques is called Weight Norm (WN) [19], and proposes a reparametrization of the weight $\mathbf{W}$ as

$$\mathbf{W}' = \frac{\alpha}{\|\mathbf{W}\|}\mathbf{W}, \tag{2.21}$$

where α is a learnable scalar. This method allows the decoupling between the magnitude of the weight and its direction, allowing faster training and better convergence. The computation overhead of this technique is relatively small, and can be completely removed during inference by computing the full weight matrix $\mathbf{W}'$ only once. While WN helps with the convergence speed of the model, its usage does not restrict the amplitude of the weight at all. For some applications, it can be needed to apply an amplitude restriction on the weight. Spectral Normalization [20] is a method leveraging the spectral norm of a matrix $\sigma(.)$ to normalize the weight, following

$$\mathbf{W}' = \frac{\mathbf{W}}{\sigma(\mathbf{W})}. \tag{2.22}$$

This ensures that the layer has a Lipschitz constant of 1, which is needed for invertible residual layers [21] or to help stabilize GAN training [20].

Dropout

Dropout [22] is used to prevent models to overfit. It works by randomly dropping values inside outputs of the layers across the network, effectively ensuring that the model does not get overconfident over a particular feature of the dataset, but rather consider an ensemble of co-activating features to make a prediction. Dropout has been successfully used in many architectures, and is notably a crucial part of the Transformer [23] architecture.

2.3 Deep generative models

Early works on machine learning algorithms mainly focused on classification tasks. However, thanks to recent advances in regularization, architecture design and training methods, deep learning approaches have started to be used for generative purposes. This shift allowed to perform text, image, audio or video synthesis using a data-driven approach as an addition to already existing hand-crafted techniques.

Generative models are usually defined using a probabilistic formulation of machine learning. Given individual examples $\mathbf{x}$ taken from a finite dataset $\mathcal{X}$ representing an underlying probability distribution $p(\mathbf{x})$, generative models aim at producing an approximate distribution $p_\theta(\mathbf{x}) \approx p(\mathbf{x})$ with learnable parameters θ. One of the main appeals of building such an approximate distribution is the possibility to sample novel examples from it.

A semi-supervised approach to generative modeling can also be used to define controls over the generation process. We consider a dataset $\mathcal{X}$ composed of pairs of examples $(\mathbf{x}, c)$, where c contains additional information about $\mathbf{x}$ (e.g. labels, descriptors). Learning a conditional model can be achieved by finding an adequate approximate distribution $p_\theta(\mathbf{x}|c) \sim p(\mathbf{x}|c)$. Sometimes, such an additional source of information is not available, and we would like to learn a joint variable $\mathbf{z}$ containing salient information about the example $\mathbf{x}$. This definition of the problem belongs to the unsupervised learning regime. Those variables $\mathbf{z}$ are called *latent variables*, and give rise to a class of models called *latent models*. Variational Auto Encoder (VAE) [24] and GAN [25] are examples of latent models that we describe in detail in Section 2.3.1 and 2.3.2.

2.3.1 Variational Auto-Encoders

VAE are *latent models* as they consider the existence of a latent variable $\mathbf{z}$ containing high-level features about a data example $\mathbf{x}$. However, learning the complete model approximating $p(\mathbf{x}, \mathbf{z})$ is usually intractable given the complexity of real-world data. Therefore, VAEs introduce an *inference model* $q_\theta(\mathbf{z}|\mathbf{x})$, trained to approximate the real posterior distribution $p(\mathbf{z}|\mathbf{x})$ by minimizing the Kullback-Leibler (KL) divergence

$$\begin{aligned}\phi^* &= \underset{\phi}{\operatorname{argmin}} \;\; \mathcal{D}_{\mathrm{KL}}[q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z}|\mathbf{x})] \\ &= \underset{\phi}{\operatorname{argmin}} \;\; -\int_{\mathbf{z}} q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{p(\mathbf{z}|\mathbf{x})}{q_\phi(\mathbf{z}|\mathbf{x})} \mathrm{d}\mathbf{z}. \end{aligned} \tag{2.23}$$

Since we do not have access to the true posterior $p(\mathbf{x}|\mathbf{z})$, an approximate distribution $p_\theta(\mathbf{x}|\mathbf{z})$ is introduced, and the KL divergence in Equation (2.23) is rearranged into

$$\begin{aligned}
0 \leq \mathcal{D}_{\mathrm{KL}}[q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})] &= -\int_{\mathbf{z}} q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{p_\theta(\mathbf{z}|\mathbf{x})}{q_\phi(\mathbf{z}|\mathbf{x})} \mathrm{d}\mathbf{z} \\
&\leq -\int_{\mathbf{z}} q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{p_\theta(\mathbf{x}|\mathbf{z}).p_\theta(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x}).p_\theta(\mathbf{x})} \mathrm{d}\mathbf{z} \\
&\leq -\int_{\mathbf{z}} q_\phi(\mathbf{z}|\mathbf{x}) \log p_\theta(\mathbf{x}|\mathbf{z}) \mathrm{d}\mathbf{z} - \int_{\mathbf{z}} q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{p_\theta(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})} \mathrm{d}\mathbf{z} + \log p_\theta(\mathbf{x}) \\
\log p_\theta(\mathbf{x}) &\geq \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \mathcal{D}_{\mathrm{KL}}[q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})], \qquad (2.24)
\end{aligned}$$

which essentially gives us a tractable lower bound on the log likelihood of $p_\theta(\mathbf{x})$ given an approximate posterior model $q_\phi(\mathbf{z}|\mathbf{x})$, an approximate distribution $p_\theta(\mathbf{x}|\mathbf{z})$ and a prior $p_\theta(\mathbf{z})$. The prior is usually a simple distribution, most of the time set to an isotropic Gaussian $\mathcal{N}(0, 1)$. A visual depiction of the full system is proposed in Figure 2.12.

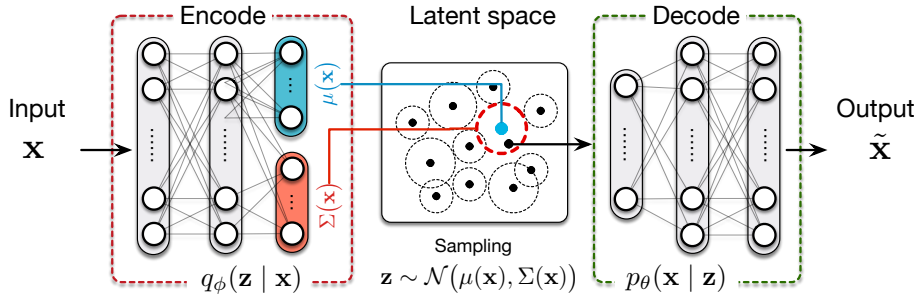


Figure 2.12: Diagram of a Variational Auto Encoder

This lower bound is called the Evidence Lower BOund (ELBO) [24], and is used as a training loss for VAEs, where we can identify a *reconstruction term* and a *regularization term*

$$\mathcal{L}_{\phi,\theta}(\mathbf{x}) = \underbrace{-\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})]}_{\text{reconstruction term}} + \beta \times \underbrace{\mathcal{D}_{\mathrm{KL}}[q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})]}_{\text{regularization term}}. \qquad (2.25)$$

Overall, the ELBO minimizes the reconstruction error of the model through the likelihood of the data given a latent $\log p_\theta(\mathbf{x}|\mathbf{z})$, while regularizing the

posterior distribution $q_\phi(\mathbf{z}|\mathbf{x})$ to match a predefined prior $p(\mathbf{z})$. Both posterior distributions q_ϕ and p_θ are parametrized by neural networks respectively called *encoder* and *decoder*.

The reconstruction and regularization terms in the ELBO are in essence conflicting, as the former is maximizing the mutual information between the input $\mathbf{x}$ and the latent $\mathbf{z}$, while the latter enforces the independence between both variables. Therefore, the authors of [26] propose to weight the regularization term with a parameter β to control the trade-off between accurate reconstruction and strong latent regularization. They show that increasing $\beta > 1$ leads to less entangled latent dimensions, to the detriment of the reconstruction quality.

Nonetheless, VAE are known to produce blurry results due to this conflict between reconstruction and regularization. Removing the regularization term helps with producing finer details at the cost of a loss of information about the resulting aggregated latent distribution. This prevents sampling from the model, and restricts its uses to simple reconstructions.

Figure 2.13: Reconstruction of MNIST digits [27] using a VAE. The reconstructions are similar to the inputs, while being generally blurry.

2.3.2 Generative Adversarial Networks

Generative Adversarial Networks (GANs) [25] are a class of deep generative models implicitly modeling the data distribution $p(\mathbf{x})$ through the use of two separate sub-models G and D, respectively called *generator* and *discriminator*, as shown in Figure 2.14. The core idea behind GANs is the *adversarial* training procedure introduced in [25], where both sub-models are optimized to minimize conflicting losses, defined as

$$\begin{aligned} \mathcal{L}_{\text{dis}} &= -\mathbb{E}_x \left[\log D(\mathbf{x})\right] - \mathbb{E}_x \left[\log(1 - D(G(\mathbf{z})))\right] \\ \mathcal{L}_{\text{gen}} &= \mathbb{E}_x \left[\log(1 - D(G(\mathbf{z})))\right]. \end{aligned} \tag{2.26}$$

The goal of the discriminator is to correctly classify real from generated samples, while the generator is optimized to make its generations classified as real. This framework has proven to outperform VAEs in terms of high-detail output generation, and has been extensively studied to provide more stable training dynamics with higher generation resolution. However, the lack of an encoder model prevents the inference of a latent representation given a data point.

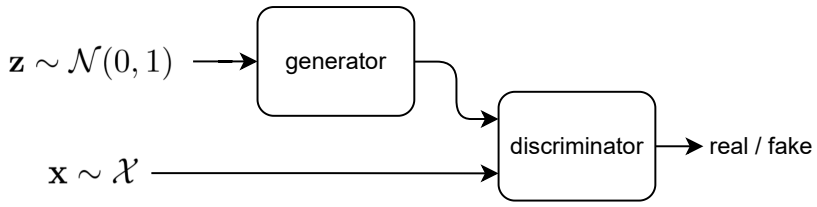


Figure 2.14: A block diagram of a generic GAN. A noise input is used to produce a generation, that is discriminated as real or fake against a true sample from the dataset by a discriminator.

A major issue with GANs is the instability occurring during training. Indeed, if the discriminator outperforms the generator too quickly, it might be impossible for the generator to compete and even start producing adequate generations. This is especially true for saturating GAN loss definitions, such as the one presented in Equation (2.26). Alternative formulations address this problem, including the hinge GAN [28]

$$\begin{aligned} \mathcal{L}_{\text{dis}} &= \mathbb{E}_x[\max(0, 1 - D(\mathbf{x}))] + \mathbb{E}_z[\max(0, 1 + D(G(\mathbf{z})))] \\ \mathcal{L}_{\text{gen}} &= -\mathbb{E}_x[D(G(\mathbf{z}))], \end{aligned} \tag{2.27}$$

or the Wassertein GAN [29] proposing the use of a Lipschitz contractive discriminator combined with the simpler loss

$$\mathcal{L}_{\text{dis}} = \mathbb{E}_z[D(G(z))] - \mathbb{E}_x[D(x)]. \tag{2.28}$$

In practice, the contractive constraint is approximated through the clipping of the weights of the model between $[-c, c]$, with $c = 0.01$. An alternative method to ensure this constraint is the use of Gradient Penalty (GP), as proposed in [30].

Another failure case for GAN is *mode collapse*, happening when the generator focuses on a specific mode of the data distribution instead of covering its entirety. This can be mitigated through the use of regularization techniques [20] or by feeding the discriminator with additional mini-batch statistics as proposed in [31]. Training GANs is notoriously difficult, mainly because the training losses are continuously learned and, thus, can be hard to interpret.

A progressive method for adversarial generation of high-resolution images is introduced in [32]. Instead of defining a fixed model directly generating and discriminating high-resolution images, the authors introduce a *growing* architecture, initially trained with few layers on a low-resolution task, and progressively augmented with additional layers for higher resolutions. This method is the first to generate million pixel images, as shown in Figure 2.15, at the cost of producing alignment artifacts as demonstrated in [33].

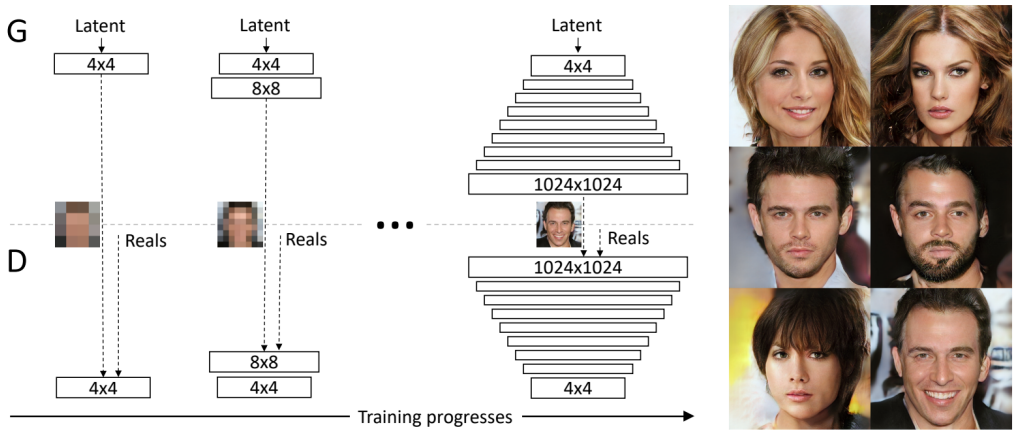


Figure 2.15: Progressive growing training method for high-resolution images adversarial generation. Figure taken from [32]

2.3.3 Normalizing Flows

Compared to VAEs or GANs, Normalizing Flows (NFs) are a class of generative models trained using the exact likelihood of the data under the model. NFs are diffeomorphisms f defining a deterministic mapping between data points $\mathbf{x} \sim \mathcal{X}$ and latent variables $\mathbf{z}$ whose distribution is a usually a simple isotropic Gaussian $\mathcal{N}(0, 1)$. In contrast to VAEs, NFs are *diffeomorphisms*, implying that latent variables $\mathbf{z}$ and input data $\mathbf{x}$ must have the same dimensionality. Therefore, we can derive the likelihood of $\mathbf{x}$ using the following *change of variable* theorem, such that

$$\begin{aligned} \mathbf{z} &= f(\mathbf{x}) \\ p(\mathbf{x}) &= p(\mathbf{z}) \left| \det \frac{df}{d\mathbf{x}} \right|. \end{aligned} \tag{2.29}$$

Since diffeomorphisms are composable, we can build complex transformations by composing multiple instances of simpler transforms without compromising invertibility nor differentiability. Therefore, the function f can be designed as a chain of parametrized diffeomorphisms $f_\theta^k : \mathbf{z}_k \to \mathbf{z}_{k+1}$, with $\mathbf{z}_0 = \mathbf{x}$ and parameters θ optimized to minimize the following log-likelihood

$$\mathcal{L}_\theta(f, \mathbf{x}, \mathbf{y}) := -\log p_{\mathbf{x}}(\mathbf{x}) = -\log p_{\mathbf{z}}(\mathbf{z}_K) - \sum_k \left| \det \frac{df_k}{d\mathbf{z}_k} \right|. \tag{2.30}$$

The advantages of using NFs are numerous. For example, in contrast to VAEs and GANs, we have access to the exact likelihood of the data under the model, which makes the evolution of the training more stable and interpretable. Additionally, the invertible nature of NFs yields an exact inference method to retrieve $\mathbf{z}$ from $\mathbf{x}$, which is not the case for the approximate posterior introduced by VAEs.

However, the computation of the Jacobian determinant in Equation (2.30) is far from trivial, and can be computationally intensive, precluding an efficient training of the model. Therefore, building a computationally efficient Normalizing Flow (NF) requires finding functions f_k whose Jacobian is easy to compute. Proposed approaches include the use of *affine coupling* functions [34, 35], splitting the flow input $\mathbf{x}$ into two separate inputs $\mathbf{x}_a, \mathbf{x}_b$ and finally yielding the following transform

$$\begin{aligned} \mathbf{y}_a &= \mathbf{x}_a \\ \mathbf{y}_b &= \mathbf{x}_b \odot \alpha_\theta(\mathbf{x}_a) + \beta_\theta(\mathbf{x}_a), \end{aligned} \tag{2.31}$$

where α, β are functions parametrized by θ. The output of f is defined as the concatenation of $\mathbf{y}_a, \mathbf{y}_b$. Since the resulting Jacobian is a lower triangular matrix, its log-determinant is the sum of the values in $\alpha_\theta(\mathbf{x}_a)$. The computation of the Jacobian determinant for the coupling layer operation does not require the computation of the Jacobians for α_θ or β_θ, enabling

the use of arbitrarily complex functions modeled by deep neural networks. Furthermore, the inverse operation can be computed using the same method as the forward propagation, making this approach efficient for both sampling and inference simultaneously.

Another efficient parametrization can be found in the form of Residual Flows [21], where each flow f is defined as

$$f : \mathbf{x} \rightarrow g(\mathbf{x}) + x, \tag{2.32}$$

where g is a contractive function. While the resulting function f is not usually analytically invertible, the inverse of an output $\mathbf{y}$ can be iteratively computed, by using

$$\begin{aligned} \mathbf{x}_{i+1} &= \mathbf{y} - g(\mathbf{x}_i) \\ \mathbf{x}_0 &= \mathbf{y}. \end{aligned} \tag{2.33}$$

The contractive constraint on function g is enforced using spectral normalization [20]. The authors propose an unbiased estimator for the log-determinant of the Jacobian, as computing the exact Jacobian is untractable. To compensate for the relatively poor expressive power of contractive functions, the full model is composed of 100 stacked flows.

While choosing simple transforms for flows is crucial for the efficiency of the training, this also harms the expressive power of the model. As a result, NFs have more parameters than GANs for similar results, and are generally slower to train due to the lack of dimensionality reduction.

2.4 Sequence modeling

2.4.1 Transformers

The understanding and modeling of temporal behaviors from a stream of information, whether it is text characters, audio samples or abstract tokens belongs to the *sequence modeling* field. While the RNNs presented in Section 2.2.3 are naturally suited to this task, their use of a fixed-size recurrent state restricts the potential extent of the temporal context needed to make a prediction. For long sequences, this might result in *forgetting* distant information, harming the performances of the model.

A key model addressing this problem is the Transformer [23]. While initially built for Natural Language Processing (NLP) tasks such as machine translation or question answering, Transformers are now used to address various task, from audio modeling [4] to image generation [36]. These models are mostly build around the concept of *attention* mechanisms combined with linear layers and normalization, as depicted in Figure 2.16.

The original Transformer model [23] is able to process pairs of sequences, called *inputs* and *targets*. The input sequence is processed by the encoder and yields a context fed to the decoder, trained to predict the target sequence autoregressively. While many iterations of this model have been proposed, they all share the same core components that we describe here.

Multi-Head Scaled Attention

The main idea behind *attention* mechanisms is to help the model focus on certain elements or combination of elements in a sequence when making predictions, rather than considering the entire sentence as a whole. In the NLP literature, attention is a function of three sequences $\mathbf{Q} \in \mathbb{R}^{T_q \times D}, \mathbf{K} \in \mathbb{R}^{T_{kv} \times D}$ and $\mathbf{V} \in \mathbb{R}^{T_{kv} \times D}$, respectively called *queries*, *keys* and *values*. Intuitively, *queries* are compared with *keys* to identify relevant pairs of elements in both sequences, which are used to define a distribution over the *values*. The scaled attention proposed in [23] implements this mechanism using

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \underbrace{\text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{D}}\right)}_{\text{attention matrix}} . \mathbf{V}, \tag{2.34}$$

where the softmax function is computed over the last dimension of the attention matrix. The query $\mathbf{Q}$ can be thought of as the main input of the layer, as the output shares the same shape (i.e. lives in $\mathbb{R}^{T_q \times D}$). The matrix

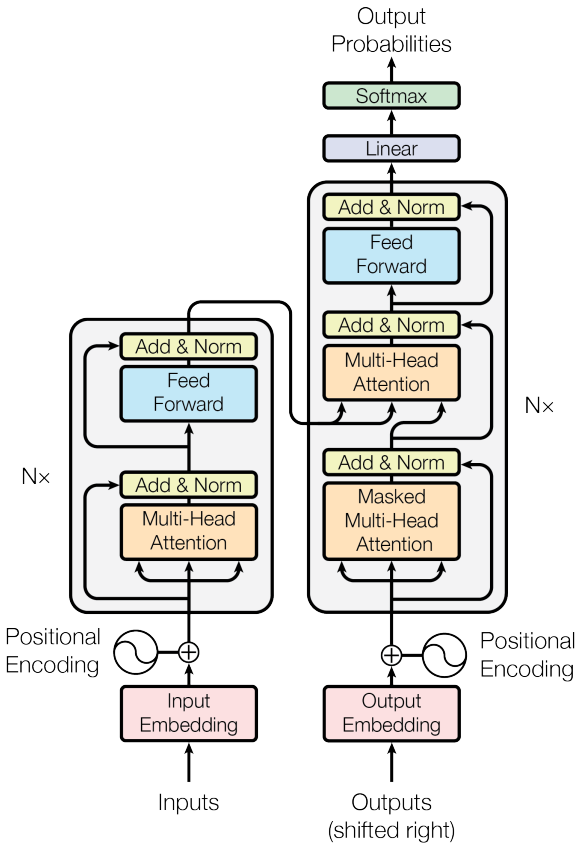


Figure 2.16: The Transformer architecture. Figure taken from [23]

product between $\mathbf{Q}$ and $\mathbf{K}^T$ yields a non-normalized *attention matrix* with shape $T_q \times T_{kv}$. Once normalized by the softmax layer, the attention matrix can be seen as the concatenation of T_q vectors each describing a categorical distribution over the values $\mathbf{V}$, exposing the position of tokens inside $\mathbf{V}$ that are important with relation to the context given by $\mathbf{Q}$ and $\mathbf{K}$. The output of the layer is the matrix product between the attention matrix and $\mathbf{V}$.

Depending on the type of attention considered, further processing of the attention matrix can be needed. *Cross-attention* is obtained by computing $\mathbf{K}, \mathbf{V}$ from a different sequence than that of $\mathbf{Q}$. This is particularly useful in the context of an encoding-decoding model, where we might want to make the queries of the decoder attend to the output of the encoder. In this case, the attention matrix is left unchanged. In addition to cross-attention, autoregressive models rely on *self-attention* in order to attend to their own past predictions. The causality of self-attention is ensured by masking the attention matrix, thus avoiding the introduction of future dependencies. This is achieved by using a triangular matrix as shown in Figure 2.17.

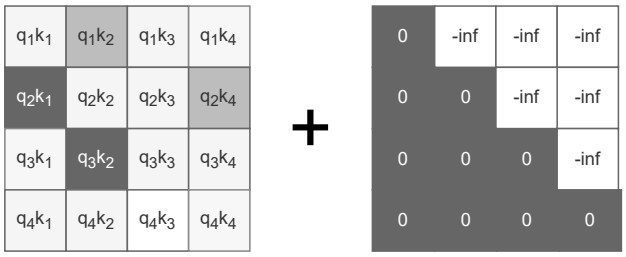


Figure 2.17: Masking attention with a binary mask to ensure the causality of the model.

A *multi-head* attention layer performs the exact same computation, but replicated over multiple subsets of the input tensors. In practice, the $T \times D$ input tensors $\mathbf{Q}, \mathbf{K}$ and $\mathbf{V}$ are split into $H \times T \times D/H$ tensors, where H is the number of heads. The previously introduced attention operation is applied over the resulting tensors, and the heads are gathered using an additional linear layer. The use of multi-head attention allows the model to focus on different temporal scales and feature types, and has proven to be beneficial to the overall accuracy of the resulting model.

While this attention mechanism is responsible for the impressive performances of transformer models, it is computationally expensive as both memory and complexity have a quadratic growth with the length of the sequence n. This implies large memory requirements preventing the vanilla model to be extended to arbitrary lengths. Alternative attention mechanisms have

been proposed to alleviate this issue, such as sparse attention [37], mixed-length attention [38] or linear approximations [39]. However, none of those methods alone is sufficient to properly learn positional information between tokens, as they all focus on finding similarities between tokens, regardless of their position in the sequence. Injecting such positional information is therefore crucial, and is implemented using *positional encoding*.

Positional Encoding

While there are now many ways to implement positional encoding, they all address the same task of biasing input tokens to include positional or temporal information. The first method was introduced alongside the Transformer model [23], and is called Sinusoidal Positional Encoding (SPE). SPE embeds the index n of a token into a D-dimensional vector defined as

$$\begin{aligned} \mathrm{SPE}[n]^{(i)} &= \begin{cases} \sin(\omega_k n) & \text{if } \; i = 2k \\ \cos(\omega_k n) & \text{if } \; i = 2k+1 \end{cases} \\ \omega_k &= \frac{1}{10000^{2k/d}}, \end{aligned} \tag{2.35}$$

which is added to the input of the model. This method allows to encode different time scales for each dimension, from small variations in early dimensions to large changes in late dimensions. We show in Figure 2.18 the output of the encoding defined in Equation (2.35) with $D = 64$ and $n \in [0, 128]$.

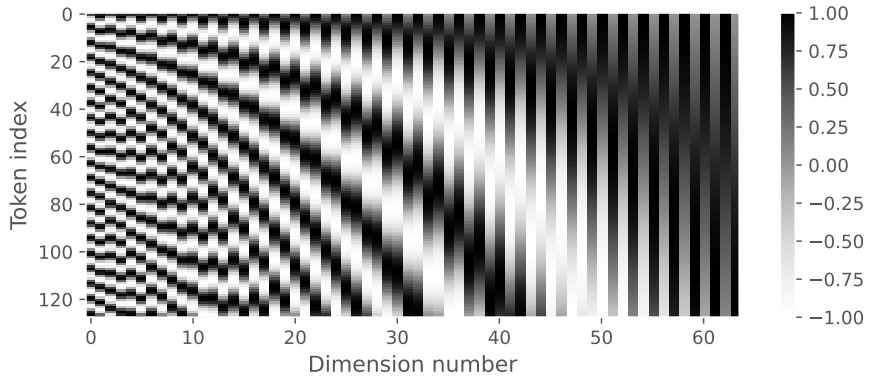


Figure 2.18: Output of SPE for a sequence composed of 128 tokens and a model of dimension 64.

This method encodes the *absolute* position of tokens. This is a problem for generalizing these models to longer sequences where token indices are po-

tentially larger than anything seen during training. Alternative positional encoding techniques use *relative* encoding. Instead of using the absolute position i of a token, relative encoding methods rather consider the distance between tokens $i-j$. Such methods include Rotary Positional Encoder (RoPE) [40] and Attention with Linear Biases (ALiBi) [41], and allow the generalization of the model to unseen length. RoPE proposes a multiplicative approach to positional encoding, in contrast with earlier additive methods. They propose a pairwise rotation matrix, defined as

$$\mathbf{R}^d_{\Theta,n} = \begin{pmatrix} \cos n\theta_1 & -\sin n\theta_1 & \dots & 0 & 0 \\ \sin n\theta_1 & \cos n\theta_1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & \cos n\theta_{d/2} & \cos n\theta_{d/2} \\ 0 & 0 & \dots & \cos n\theta_{d/2} & \cos n\theta_{d/2} \end{pmatrix}, \tag{2.36}$$

that is used to rotate *queries* according to their sequence position, such that

$$\text{RoPE}(\mathbf{q}_n, n) = \mathbf{R}^d_{\Theta,n}\mathbf{q}_n. \tag{2.37}$$

Their method is significantly simpler to implement than previous relative positional encoding approaches such as the one proposed in [42]. As an alternative to RoPE, ALiBi proposes a simple linear bias to encode relative positional information, as shown in Figure 2.19.

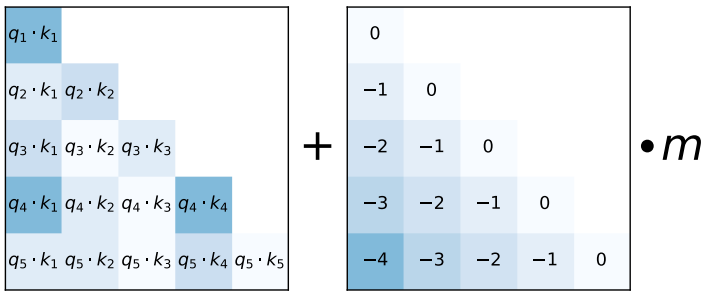


Figure 2.19: Biases applied to the attention matrix by ALiBi to encode relative positional information. The factor m is a positive value called slope, which varies with relation to the index of the head. Figure taken from [41].

The linear biases are weighted according to the values of a *slope*, defined as

$$m_k = 2^{(-8/H)^k}, \tag{2.38}$$

where $H > 8$ is the number of heads, and $k \in [1, H]$ is the head index. They show that their method effectively encodes relative information about tokens, while allowing to extrapolate on larger sequences during inference.

2.4.2 Model pretraining

Tasks involving sequence modeling are often composed of discrete tokens. Whether the dataset is inherently discrete (e.g. binary images or digital audio) or needs to be *tokenized* (e.g. text or pseudo-continuous representations), we consider sequences $\mathbf{x}$ of length T, with individual values $x^t \in [0, V]$, where V is the *vocabulary size*.

Prior to being fed to a model, discrete tokens are often *embedded*. In its simplest form, embedding a token is achieved using of a codebook C with shape $V \times D$, composed of V individual D-dimensional vectors. The codebook is used as a lookup table to transform a T-dimensional discrete sequence into a $T \times D$ continuous tensor. During training, the codebook may be trained to better reflect similarities between tokens. The use of pretrained embeddings such as word2vec [43] allows to embed semantically similar tokens close to each other in the embedding space (e.g. *cat* and *kitten* should be closer to each other in the embedded space than *cat* and *car*). This is beneficial to both training speed and model generalization.

However, there are cases where identical tokens have a deeply different meaning depending on the context (e.g. "he is *running* a marathon" has a different use of the word "running" than "he is *running* a company"). In those cases, the use of a more complex embedding technique can be beneficial. The Bidirectional Encoder Representations from Transformers (BERT) model is addressing this exact task, allowing to produce different embeddings for discrete tokens given their context through the use of masked language modeling.

Bidirectional Encoder Representations from Transformers (BERT)

BERT is trained using Masked Language Modeling (MLM) to provide high level contextual information based on textual sequences. The main idea behind MLM is to randomly mask inputs in a discrete sequence, and train a model to predict those masked tokens given the unmasked context. As shown

in Figure 2.20, this allows the model to build a high-level representation of tokens with relation to their current context. As there are no ground-truth labels, MLM is a *self-supervised* learning task.

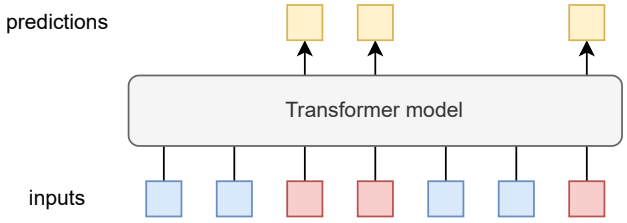


Figure 2.20: The BERT training framework. An input sequence is randomly masked (the masked elements are depicted in red) and fed to a transformer whose task is to predict the masked tokens given the unmasked context.

Once BERT has converged on a specific dataset, it can be used as a pretrained model for a *supervised learning* task, usually targeting classification. This can be done by discarding the n last layers of the model, and use the hidden representation of the sequence as the input of an additional classifier. Thanks to the high level embedding yielded by the pretrained model, the added classifier can be relatively simple, yet the overall system produce state-of-the-art performances in terms of accuracy [44, 45].

The prediction of the masked tokens performed during pretraining is learned through the minimization of the cross-entropy between the categorical prediction yielded by the model and the one-hot vector representation of the masked token. While particularly efficient at producing high-level representations for discrete sequences, the formulation proposed in the original article does not allow an analogous task for continuous values. This is the case of audio signals, for which the w2v-BERT model [46] proposes a way to adapt the BERT approach to self-supervised speech representation learning.

w2v-BERT

The w2v-BERT model leverages the pretraining techniques from BERT and applies it to audio signal contextual information extraction. One of the main differences with the BERT approach is the lack of discrete tokens, preventing the definition of a categorical cross-entropy to optimize the model. Therefore, the w2v-BERT model jointly learns a quantization layer yielding discrete targets given a continuous input. The discrete representation is then used as target for the regular MLM loss. To avoid learning a degenerate quantizer

(i.e. yielding a single discrete value for all signals), a contrastive loss is added to the model, as shown in Figure 2.21.

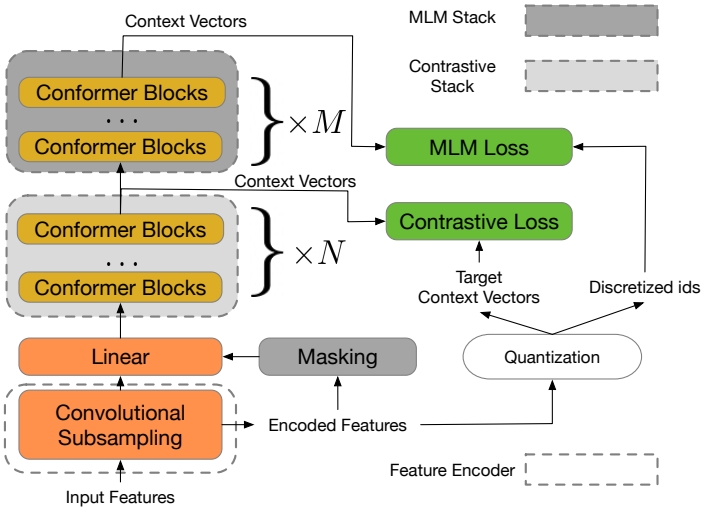


Figure 2.21: The w2vBERT architecture. The model is trained to optimize two losses: a *contrastive* loss building a quantized representation and a *masked language modeling* loss, similar to BERT. Figure taken from [46]

Similarly to the BERT model, the last layers of the trained w2v-BERT model can be either fine-tuned or replaced in a *supervised learning* setup, yielding state-of-the-art results in speech recognition [46].

2.5 Audio modeling

Deep generative models have shown great success at synthesizing images with models capturing both high- and low-level features of a dataset [31, 32, 47]. However, the extension of such models to audio modeling is far from trivial. Approaches addressing this task leverage different representations of audio signals, including the raw waveform (Section 2.5.1), spectral representations (Section 2.5.2) or leveraging classical synthesis techniques (Section 2.5.3).

2.5.1 Waveform models

While directly modeling the raw waveform using deep generative models might look like the most straightforward approach, it is a rather daunting task. As presented in Section 2.1.1, two audio signals with very different waveforms can be perceptually identical, which renders loss computing arduous. Indeed, penalizing a model for not producing a particular waveform can be detrimental to the high-level task of generating a perceptually-similar sound. Several approaches have addressed raw waveform modeling using different formulations and methods that we present here.

Autoregressive approaches

One of the first approaches to raw waveform is autoregressive modeling. In fact, phase alignment issues can be addressed by decomposing the probability of a given signal $\mathbf{x} \in \mathbb{R}^T$ as a product of conditional probabilities following

$$p(\mathbf{x}) = \prod_{t=0}^{T-1} p(x_t|\mathbf{x}_{<t}), \tag{2.39}$$

where x_t is the value of the waveform at time step t, whereas $\mathbf{x}_{<t}$ contains all previous audio samples. Sampling from the resulting distribution p can be achieved autoregressively by producing one audio sample at a time. This is how both WaveNet [3] and SampleRNN [48] are parametrized. In particular, the authors of WaveNet propose an 8-bit quantization of the μ-law transformed waveform defined as

$$f(x) = \text{sng}(x)\frac{\ln(1+\mu|x|)}{\ln(1+\mu)}. \tag{2.40}$$

Using the μ-law transform is known to increase the SNR of an audio signal going through a lossy quantization. This results in an audio waveform $\mathbf{x}_q \in [0, 256]^T$ sampled at 16kHz, which is modeled by WaveNet using a $T \times 256$ categorical distribution. In practice, WaveNet is a strictly causal CNN taking as input $\mathbf{x}_q$ and producing an output l called *logits*. Logits are non-normalized scores that can be normalized using a softmax function. The model is then optimized using the cross entropy between the resulting distribution and the ground-truth samples.

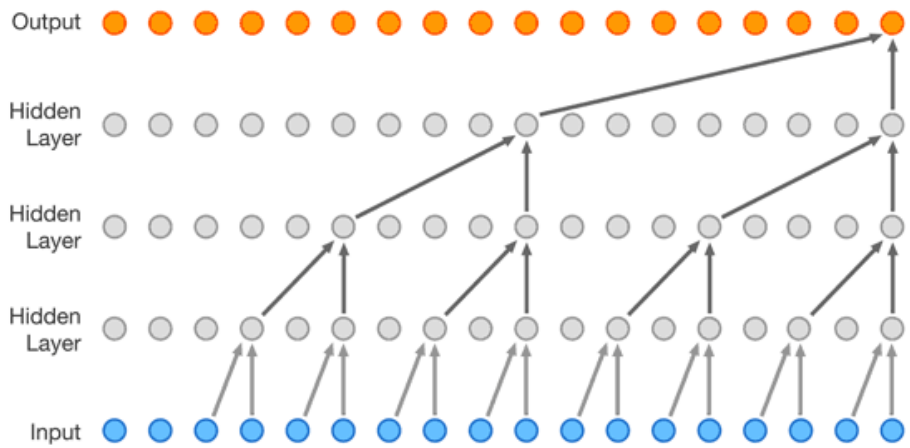


Figure 2.22: The *WaveNet* architecture. During sampling, each audio sample is produced sequentially. Figure taken from [3]

At the time of its release, WaveNet was the state-of-the-art in terms of audio quality and naturalness. The text-conditioned WaveNet yielded a state-of-the-art Text-To-Speech (TTS) system outperforming by a large margin previous approaches. However, the autoregressive sampling procedure enforced by the causal nature of the model results in extensive training and sampling time[1], as well as the need for large datasets.

Teacher-student model

As the long sampling time implied by autoregressive approaches precludes their application to real-world use-cases, several works have studied the use

[1]Open source re-implementation of the model reported an average of 40 minutes of computation per second of generated audio.

of Normalizing Flow (NF) to accelerate the generation process. Parallel WaveNet [49] proposes the use of *probability distribution distillation* to train an Inverse Autoregressive Flow [50]. In practice, their approach is composed of two models based on the WaveNet [3] architecture. The first model, called *teacher*, is an autoregressive WaveNet using a mixture of logistics as a replacement for the categorical distribution, trained following the method described in the original article. The second model, called *student*, follows the same architecture but is trained using distillation. In practice, the *student* model takes noise as input which is processed in parallel to yield a prediction. The *student* model is trained to maximize the likelihood of its generations under the pretrained *teacher* model. This approach is proposed in the context of TTS, where a conditioning signal c is given to both models.

Once trained, the resulting *student* shows a perceptual quality similar to the *teacher*, while being several orders of magnitude faster for sampling. One problem of this approach is the complexity of the training process that is highly unstable and, thus, hard to replicate. The Clarinet [51] model proposes a simpler distribution parametrization for the *teacher* using a single Gaussian instead of the mixture of logistics. This results in a more stable training procedure, but still involves the training of two separate models.

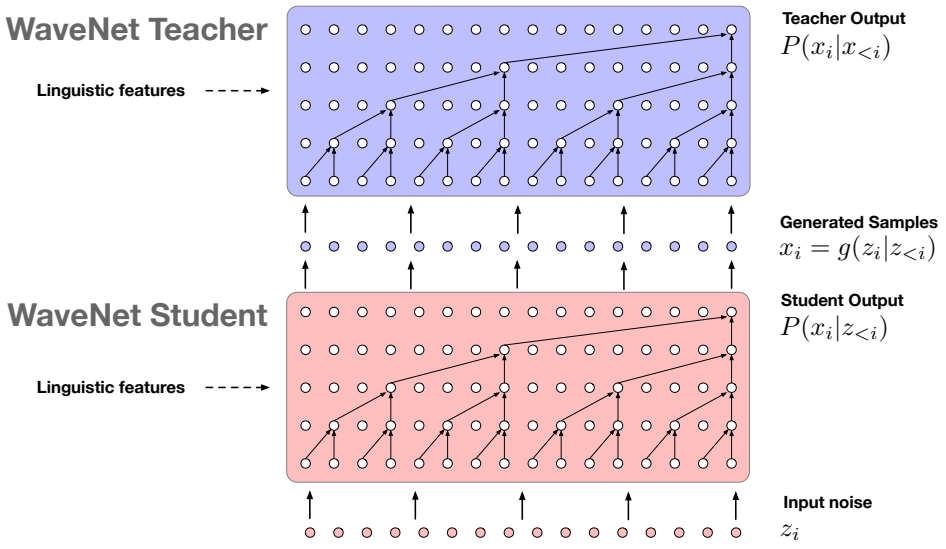


Figure 2.23: The Parallel WaveNet architecture. A pretrained *teacher* is used to distill knowledge about audio into a *student*. Figure taken from [49]

Flow approaches

Following the success of NF models for image modeling, the authors of WaveGlow [52] and WaveFlow [53] propose a method for replacing the teacher student distillation method by a single model trained using maximum log-likelihood. They both use an architecture inspired from WaveNet and implement the invertible mappings using respectively Coupling Flows [35] and Masked Autoregressive Flows [54].

The resulting models show very strong performances in terms of perceptual audio quality and generation speed, but their use of NF implies a large number of parameters to compensate for the limited expressivity of such invertible mappings, as well as a long training time. However, the training stability is greatly improved compared to the previous distillation approach.

Adversarial training

In order to address the computational complexity involved by NF, recent works leverage adversarial training to synthesize high quality audio signals. Both the melGAN [28] and hifiGAN [55] models use a feed-forward architecture upscaling a mel-scale spectrogram into the corresponding raw audio waveform. Both approaches use the Hinge GAN formulation to train the generator and the a set of several discriminators, defined as

$$\begin{aligned} \mathcal{L}_{\text{dis}} &= \max(0, 1 - D(\mathbf{x}_{\text{real}})) + \max(0, 1 + D(\mathbf{y}_{\text{fake}})) \\ \mathcal{L}_{\text{gen}} &= -D(\mathbf{y}_{\text{fake}}). \end{aligned} \tag{2.41}$$

The melGAN model proposes the use of a *feature matching* loss between the feature maps of the discriminator for the target and generated audio signals. This can be seen as a learned perceptual metric ensuring that the generations match the target signals perceptually [28]. In addition to this loss, the hifiGAN model uses a multiscale spectral distance [56] to further regularize the generator.

Both approaches use discriminators processing audio at different scales. The multi-scale discriminator introduced by melGAN operates on progressively down-sampled version of the audio signals, effectively processing different scales of the waveform. The multi-period discriminator added by hifiGAN reshapes the waveform from shape T to shape $k \times T/k$, with $k \in \{2, 3, 5, 7, 11\}$ prior to being classified. This allows the model to focus on windows of signals

with varying period, eventually allowing the resulting model to match the quality of the previously presented autoregressive approaches.

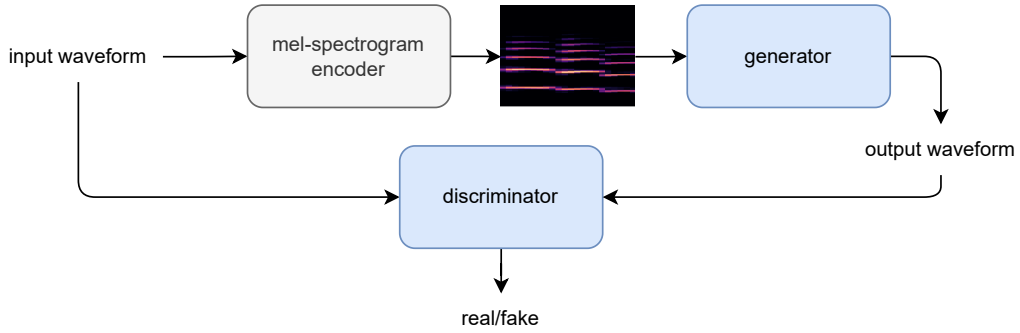


Figure 2.24: The melGAN [28] and hifiGAN [55] architectures. Both models upscale an input mel-scale spectrogram into an audio waveform. Several discriminators are applied to both the ground truth audio and the prediction to increase the perceived audio quality.

2.5.2 Spectral models

One of the main challenges of waveform models is to properly handle the phase of the audio signal. While some waveform approaches rely purely on pretrained models [49, 52] or adversarial training [28], most leverage spectral losses to stabilize the training or help with convergence [51, 55, 57]. In this section, we present how deep audio modeling can be performed using spectral representations as the base training signal.

Amplitude spectrograms

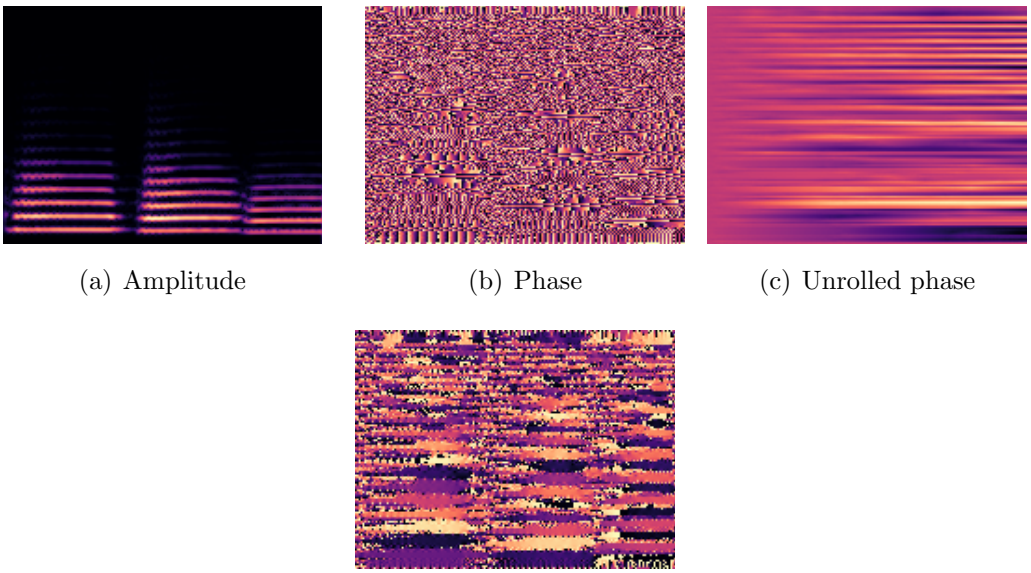
A natural decomposition of a complex spectrogram is obtained through the computation of its amplitude and complex phase. This allows the definition of simple losses on the spectral content of the signal without considering phase alignment problems. The approach proposed in [58] is a VAE trained on amplitude spectrograms. The phase is not taken into consideration during training, thus the raw signal is estimated using the Griffin-Lim (GL) algorithm [59]. Compared to waveform models, their approach allows fast training on a smaller dataset. However, their use of the iterative GL algorithm for phase estimation yields audio examples with a poor audio quality.

The application of large autoregressive models to spectrogram modeling is proposed in the MelNet model [60]. This approach uses several RNN models

to autoregressively predict mel-scale spectrograms using a multi-scale hierarchical approach. Using text-conditioning, they obtain a competitive TTS system in terms of prosody and temporal coherency. However, their approach is still limited by the use of GL, resulting in low-quality audio predictions.

Complex spectrograms

As relying on phase estimation algorithms is a bottleneck for audio quality, models like GANSynth [61] propose an approach to generate both the amplitude and phase of spectrograms. Considering the chaotic nature of the phase component, they do not generate it directly but rather produce a prediction for the *instantaneous frequencies* of the spectrogram, defined as the temporal derivative of the unrolled phase, as shown in Figure 2.25.



(a) Amplitude (b) Phase (c) Unrolled phase

(d) Instantaneous frequency

Figure 2.25: Comparison of several phase spectrogram representations. The original phase has a chaotic behavior, while the derivative of the unrolled phase has some structure.

The architecture of GANSynth is inspired by progressiveGAN [32], more specifically the progressive growing training method allowing a more stable training. Overall, their approach outperforms previous phase estimation algorithms, but their use of the derivative of the unrolled phase discards its initial value. Therefore, audio signals where a precise phase alignment is needed (e.g. percussive sounds) are not correctly reconstructed.

2.5.3 Hybrid models

Whether the audio representation used is raw or spectral, the need to properly model the phase is crucial to synthesize high-quality audio signals. A third class of audio generative models address this problem by modeling the parameters of classical synthesizers instead of using audio signals directly. The FlowSynth model [62] combines a VAE and Normalizing Flows (NFs) to produce a mapping between the perceptual latent distribution learned by the VAE and the parameter space of the synthesizer used, as shown in Figure 2.26. This allows to abstract the complexity of synthesizer parameters through the exploration of a perceptual space. Interestingly, the model can be used to infer synthesizer parameters given an input sound. Hence, further modification of the synthesized signal can be achieved through a manual tweaking of the synthesizer.

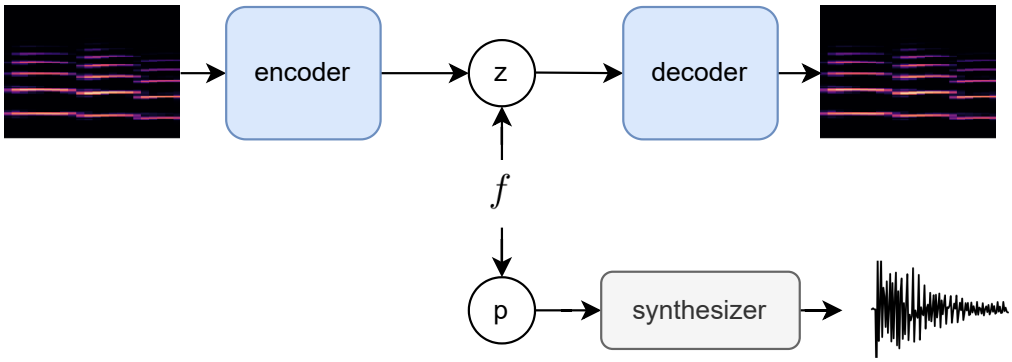


Figure 2.26: FlowSynth architecture. The perceptual distribution learned by the VAE is mapped to the synthesizer parameter space using NFs.

The approach proposed by FlowSynth leverages the synthesizer as a black-box whose outputs are conditioned on a set of parameters. Therefore, the exact relationship between the synthesizer parameters and the output signals are never exploited during training. The authors of Differentiable Digital Signal Processing (DDSP) [56] propose to instead integrate classical synthesizer blocks directly inside the computational graph of the model. By using differentiable operators, they allow the learning of the synthesizer parameters through losses in the audio domain, while abstracting the complexity of audio modeling. In practice, the DDSP model is composed of a harmonic synthesizer, a noise synthesizer, and a reverb module, as shown in Figure 2.27.

The DDSP model is conditioned on the pitch and loudness of the input signal, as well as an optional context vector yielded by an encoder. In order to train

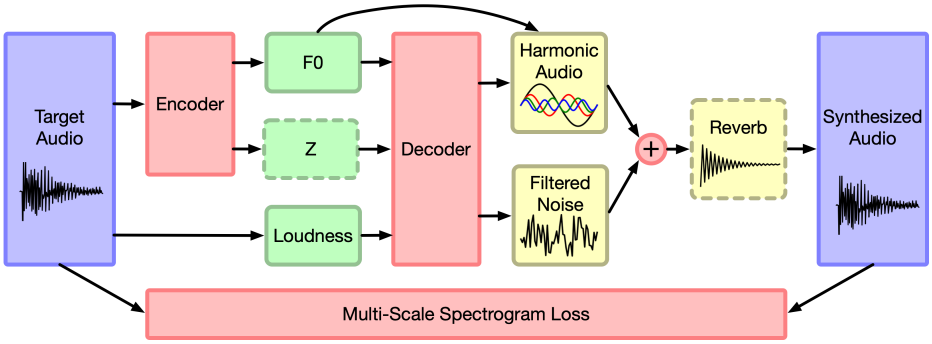


Figure 2.27: The DDSP [56] architecture. Pitch and loudness are extracted from an input audio signal, and are used to drive a harmonic plus noise synthesizer. Figure taken from [56]

the model, they propose a distance called *Multi Scale Spectral Distance*, which is defined as

$$S(\mathbf{x},\mathbf{y}) = \sum_{n\in\mathcal{N}} \frac{\|\mathrm{STFT}_n(\mathbf{x}) - \mathrm{STFT}_n(\mathbf{y})\|_F}{\|\mathrm{STFT}_n(\mathbf{x})\|_F} + \log\left(\|\mathrm{STFT}_n(\mathbf{x}) - \mathrm{STFT}_n(\mathbf{y})\|_1\right), \tag{2.42}$$

where $\|.\|_F$ is the Frobenius norm, and $\mathcal{N} = \{64, 128, 256, 512, 1024, 2048\}$. The resulting model is lightweight and can be trained on a small dataset (e.g. the length of the dataset used in the experiments is approximately 15mn).

Overall, both FlowSynth and DDSP are relatively lightweight and can produce high quality sounds by avoiding modeling the audio directly. However, in contrast with previous waveform or spectral methods, both models are constrained by the synthesizer used during training, and thus cannot generalize to arbitrary types of signal.

We are to admit no more causes of natural things than such as are both true and sufficient to explain their appearances.

Isaac Newton

Chapter 3

Audio representation learning

As described in the previous chapter, modeling high-quality audio signal is a complex task for many reasons, from the complexity of raw waveforms themselves to the numerous audio scales involved in their creation (e.g. timbre features to musical genre or speaker identity). Using large models to process all scales at once is possible, as previously achieved by models such as WaveNet [3] or SampleRNN [48]. However, this straightforward approach has issues. Compromises on the audio quality are made to allow a proper convergence of the model, implying the use of low sampling rates and resolutions. Furthermore, the computational complexity of the resulting system makes the autoregressive sampling method prohibitively slow, resulting in hours of computation for seconds of generation.

The approach proposed in JukeBox [4] involves a hierarchical modeling of the raw waveform using audio auto-encoders with variable temporal compression ratio. To maintain a reasonably high audio quality, they use small temporal compression factors (i.e. 8x, 32x and 128x) producing a latent representation of the raw waveform later modeled by a Transformer model [23]. They rely on three separate transformer models to autoregressively predicts latents from each auto-encoder, conditioned on the previous level. As the small compression ratio of the autoencoders yield large sequences, the authors report an average of 9 hours of computation for a single minute of generation. While the performances of the model are impressive, the computational complexity of the generation process prevents any creative use of the model.

Therefore, a key feature of our proposal is to produce a *compact* learned representation, with a latent sampling rate low enough to ease the downstream temporal modeling task. We hypothesize that a better balance between the

complexity of the audio model and the temporal model is crucial to reach a generative model both high quality and fast enough to allow interaction with the user. A counter-example of this balance is the huge difference in complexity between the audio model ($\sim$ 2M parameters) and temporal model ($\sim$ 5B parameters) of JukeBox. We depict in Figure 3.1 a block-diagram of our overall approach.

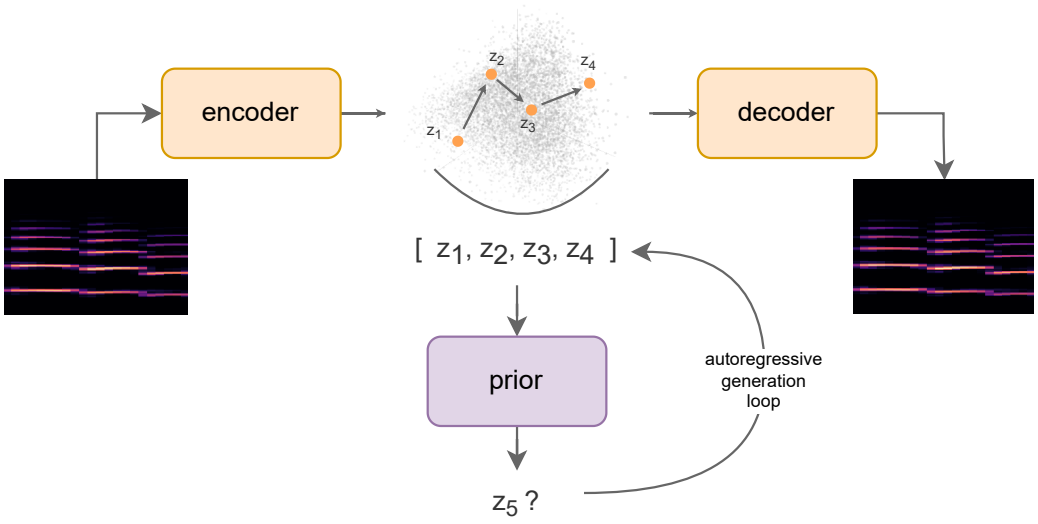


Figure 3.1: Overall hierarchical model proposed. Instead of directly learning the temporal behavior of an audio signal, we propose to use *representation learning* to produce a high level compact representation $\mathbf{z}_t$. The temporal model (or *prior*) then learns the behavior of the audio signal based on the latent trajectories.

We address the problems stated above by introducing the Realtime Audio Variational autoEncoder (RAVE) model, a VAE built specifically for fast and high-quality audio synthesis. We introduce in Section 3.1 a specific two-stage training procedure where the model is first trained as a regular VAE for *representation learning*, then fine-tuned with an *adversarial generation* objective in order to achieve high-quality audio synthesis. We combine a multi-band decomposition of the raw waveform alongside classical synthesis blocks inspired by [56], allowing to achieve high-quality audio synthesis with sampling rates going up to 48kHz without a major increase in computational complexity. Our model is designed to have a high temporal compression ratio, that we render even further efficient by introducing a method to split the latent space between *informative* and *uninformative* parts using a singular value decomposition.

3.1 RAVE

Ideally, the representation learned by a variational autoencoder should contain *high-level* attributes of the dataset. However, two perceptually similar audio signals may contain subtle phase variations that produce dramatically different waveforms. Hence, estimating the reconstruction term in the ELBO (see equation (2.25)) using the raw waveform penalizes the model if those subtle variations are not included in the learned representation. This might both hamper the learning process and include in the latent space those *low-level* variations about audio signal that are not relevant perceptually. To address this problem, we split the training process in two stages, namely *representation learning* and *adversarial fine-tuning*.

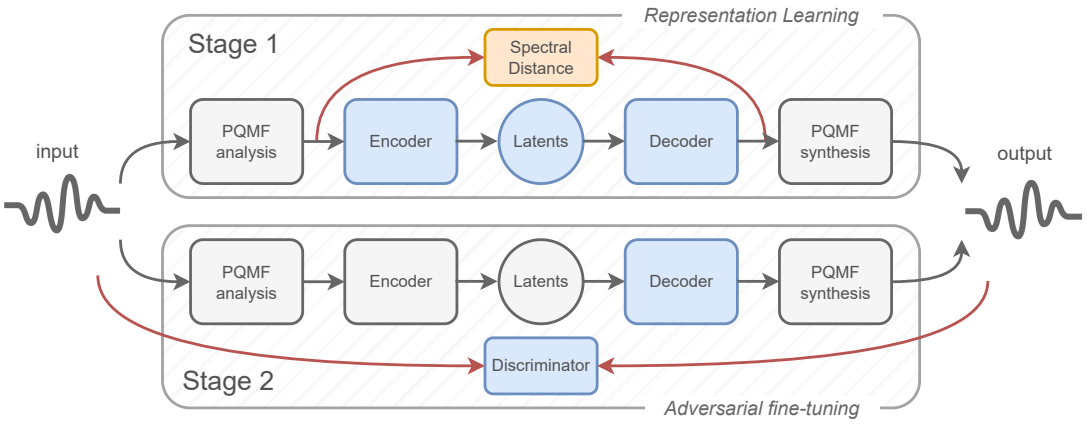


Figure 3.2: Overall architecture of the proposed approach. Blocks in blue are optimized, while blocks in grey are fixed or frozen operations.

3.1.1 High-resolution audio modeling

Learning a high-quality model is challenging as the temporal complexity of an audio signal makes it hard to be both globally coherent and locally detailed. Therefore, we leverage two techniques to achieve high-quality modeling, namely a *two-stage training* scheme and a *multiband decomposition* of the raw waveform.

Two-stage representation learning

The first stage of our procedure aims to perform *representation learning*. We leverage the multiscale spectral distance $S(\cdot, \cdot)$ proposed by [56] in order to compare real and synthesized waveforms, such that

$$S(\mathbf{x}, \mathbf{y}) = \sum_{n \in \mathcal{N}} \frac{\|\mathrm{STFT}_n(\mathbf{x}) - \mathrm{STFT}_n(\mathbf{y})\|_F}{\|\mathrm{STFT}_n(\mathbf{x})\|_F} + \log\left(\|\mathrm{STFT}_n(\mathbf{x}) - \mathrm{STFT}_n(\mathbf{y})\|_1 + 1\right), \tag{3.1}$$

where $\mathcal{N}$ is a set of scales, STFT_n is the amplitude of the Short-Term Fourier Transform with window size n and hop size $n/4$, and $\|\cdot\|_F$, $\|\cdot\|_1$ are respectively the Frobenius norm and L_1 norm. Using an amplitude spectrum-based distance does not penalize the model for inaccurately reconstructed phase, but encompasses important perceptual features about the signal. We train the *encoder* and *decoder* with the following loss derived from the ELBO

$$\mathcal{L}_{\mathrm{vae}}(\mathbf{x}) = \mathbb{E}_{\hat{\mathbf{x}} \sim p(\mathbf{x}|\mathbf{z})}[S(\mathbf{x}, \hat{\mathbf{x}})] + \beta \times \mathcal{D}_{\mathrm{KL}}[q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z})], \tag{3.2}$$

We start by training the model solely with $\mathcal{L}_{\mathrm{vae}}$, and once this loss converges, we switch to the next training phase.

The second training stage aims at improving the synthesized audio quality and naturalness. When the reconstruction loss stabilizes, we freeze the encoder and only train the decoder using an adversarial objective.

GANs are *implicit* generative models allowing to sample from a complex distribution by transforming a simpler one, called the *base distribution*. Here, we use the learned latent space in the first stage as the base distribution, and train the decoder to produce synthesized signals similar to the real ones by relying on a *discriminator* D. We use the hinge loss version of the GAN objective, defined as

$$\begin{aligned} \mathcal{L}_{\mathrm{dis}}(\mathbf{x}, \mathbf{z}) &= \max(0, 1 - D(\mathbf{x})) + \\ &\quad \mathbb{E}_{\hat{\mathbf{x}} \sim p(\mathbf{x}|\mathbf{z})}[\max(0, 1 + D(\hat{\mathbf{x}}))], \\ \mathcal{L}_{\mathrm{gen}}(\mathbf{z}) &= -\mathbb{E}_{\hat{\mathbf{x}} \sim p(\mathbf{x}|\mathbf{z})}[D(\hat{\mathbf{x}})]. \end{aligned} \tag{3.3}$$

In order to ensure that the synthesized signal $\hat{\mathbf{x}}$ does not diverge too much from the ground truth $\mathbf{x}$, we keep minimizing the spectral distance defined in Equation (5.4), but also add the feature matching loss $\mathcal{L}_{\mathrm{FM}}$ proposed by [28]. Altogether, this yields the following objective for the decoder

$$\mathcal{L}_{\mathrm{total}}(\mathbf{x}, \mathbf{z}) = \mathcal{L}_{\mathrm{gen}}(\mathbf{z}) + \mathbb{E}_{\hat{\mathbf{x}} \sim p(\mathbf{x}|\mathbf{z})}[S(\mathbf{x}, \hat{\mathbf{x}}) + \mathcal{L}_{\mathrm{FM}}(\mathbf{x}, \hat{\mathbf{x}})]. \tag{3.4}$$

3.1.2 Representation dimensionality estimation

The loss proposed in Equation (3.2) contains two terms, a *reconstruction* and *regularisation* term. Those two terms are somewhat conflicting, since the reconstruction term maximises the mutual information between the latent representation and the data distribution, while the regularisation term guides the posterior distribution towards independence with the data (potentially causing *posterior collapse*). In practice, the pressure applied by the regularisation term to the encoder during training encourages it to learn a compact representation, where informative latents have the highest KL divergence from the prior, while uninformative latents have a KL divergence close to 0 [26].

Here, we address the task of identifying the most informative parts of the latent space in order to restrict the dimensionality of the learned representation to the strict minimum required to reconstruct a signal. To do so, we adapt the method for range and null space estimation to this problem. Let $\mathbf{Z} \in \mathbb{R}^{b \times d}$ be a matrix composed of b samples $\mathbf{z} \in \mathbb{R}^d$, where $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$. Using a Singular Value Decomposition (SVD) directly on $\mathbf{Z}$ to solve the problem of finding informative parts of the latent space would not be relevant given the high variance present in the collapsed parts of $\mathbf{Z}$. In order to adapt this to our problem, we first remove the variance from $\mathbf{Z}$, by considering the matrix $\mathbf{Z}' \in \mathbb{R}^{b \times d}$ that verifies

$$\mathbf{Z}'_i = \underset{\mathbf{z}}{\operatorname{argmax}} \;\; q_\phi(\mathbf{z}|\mathbf{x}), \tag{3.5}$$

Hence, dimensions of the posterior distribution $q_\phi(\mathbf{z}|\mathbf{x})$ that have collapsed to the prior $p(\mathbf{z})$ will result in a constant value in $\mathbf{Z}'$, that we set to 0 by removing the average of $\mathbf{Z}'$ across the first dimension. The only dimensions of $\mathbf{Z}'$ with non-zero values are therefore correlated with the input, which constitute the informative part of the latent space. Applying an SVD on this centered matrix, we can obtain the matrix $\mathbf{\Sigma}$ containing the singular values of $\mathbf{Z}'$, by computing

$$\mathbf{Z}' = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\mathbf{T}, \tag{3.6}$$

The rank r of $\mathbf{Z}'$ is equal to the number of non-zero singular values in $\mathbf{\Sigma}$. Given the high variation that exists in real-world data, it is unlikely that the vanishing singular values of $\mathbf{Z}'$ are equal to 0. Therefore, instead of tracking the exact rank r of $\mathbf{Z}'$, we define a *fidelity* parameter $f \in [0, 1]$, with the associated rank r_f defined as the smallest integer verifying

$$\frac{\sum_{i \leq r_f} \mathbf{\Sigma}_{ii}}{\sum_i \mathbf{\Sigma}_{ii}} \geq f. \tag{3.7}$$

Given the fidelity value f, and a latent representation $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$, we reduce the dimensionality of $\mathbf{z}$ by projecting it on the basis defined by $\mathbf{V}^T$ and keep only the r_f first dimensions. We obtain a low-rank representation $\mathbf{z}_f$, whose dimensionality depends on both the dataset and f. Before providing $\mathbf{z}_f$ to the decoder, we concatenate it with noise sampled from the prior distribution, and project it back on its original basis using $\mathbf{V}$. We demonstrate in Section 3.1.4 the influence of f on the reconstruction.

3.1.3 Experiments

Encoder

We define our encoder as the combination of a multiband decomposition followed by a simple convolutional neural network, transforming the raw waveform into a 128-dimensional latent representation. In practice, the encoder of RAVE is a convolutional neural network with leaky ReLU activation and batch normalization (see Figure 3.3). For all of our experiments we use $N = 4$, with hidden sizes [64, 128, 256, 512] and strides [4, 4, 4, 2]. The full latent space has 128 dimensions.

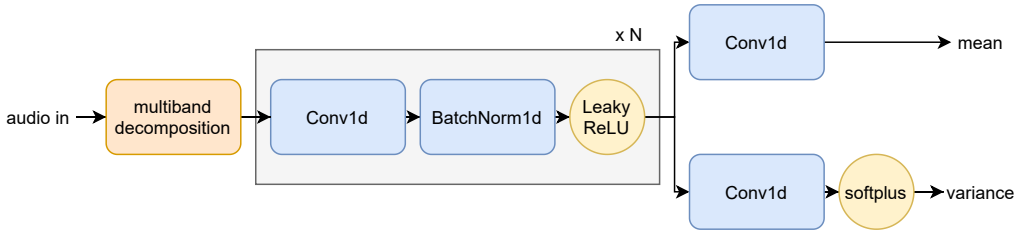


Figure 3.3: Architecture of the encoder used in the RAVE model.

Decoder

Our decoder is a modified version of the generator proposed by [28]. We show in Figure 3.4 and 3.5 details about the architecture of the decoder. We use the same alternation of upsampling layers and residual networks, but instead of directly outputting the raw waveform we feed the last hidden layer to three sub-networks. The first sub-network (*waveform*) synthesizes a multiband audio signal (with *tanh* activation), which is multiplied by the output of the second sub-network (*loudness*), generating a single amplitude

envelope (with *sigmoid* activation) for all bands. The last sub-network is a noise synthesizer as proposed in [56], and produces a multi-band filtered noise signal that is added to the previous signal before being re-composed into a single-band signal using the PQMF bank.

We found that using an explicit amplitude envelope helps reducing artifacts in the silent parts of the signal, while the noise synthesizer slightly increases the reconstruction naturalness of noisy signals (see Section 3.1.4 and 3.1.4).

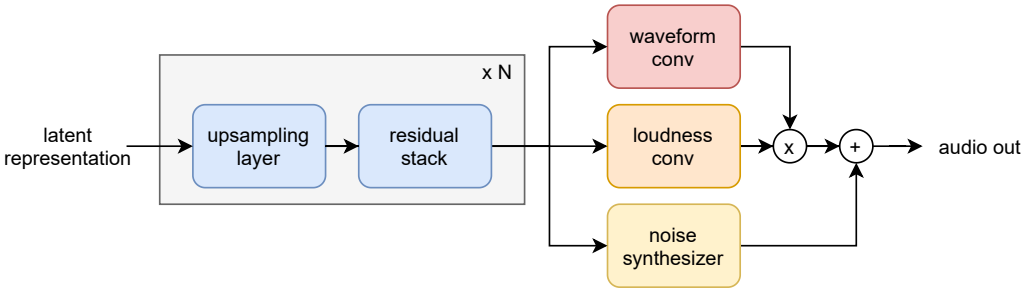
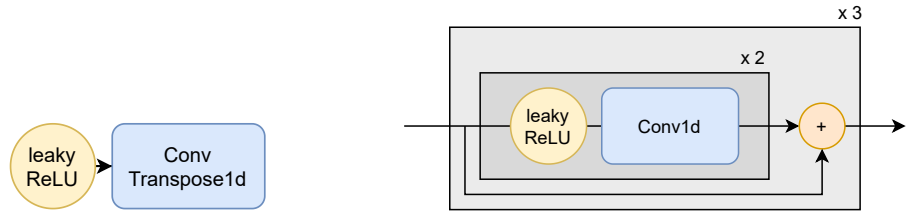
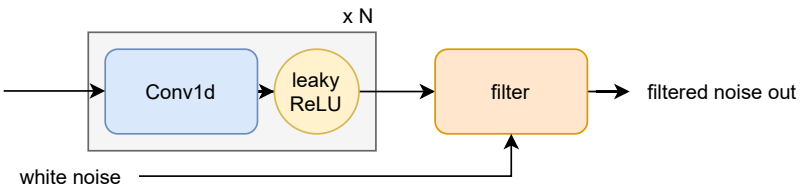


Figure 3.4: Overview of the proposed decoder. The latent representation is upsampled using alternating upsampling layers and residual stack. The result is processed by three sub-networks, respectively producing *waveform*, *loudness* envelope and filtered *noise* signals.



(a) Upsampling layer

(b) Residual stack



(c) Noise synthesizer

Figure 3.5: Detailed architecture of the decoder blocks used in RAVE.

Discriminator

We use the same discriminator as in [28], which is a strided convolutional network applied on different scales of the audio signal to prevent artifacts. We also use the same feature matching loss as in the original paper.

Training

We follow the training procedure proposed in Section 3.1, and train RAVE for 3M steps, specifically 1.5M steps for each stage, summing to a total of approximately 4 days on a single Titan V GPU. Spectral distances are computed both before and after the multi-band decomposition, and we use a cyclic annealing schedule for the regularization term in Equation 3.2, as proposed in [63]. We use the *ADAM* optimizer [10] with a learning rate of 10^{-4}, $\beta = (0.5, 0.9)$, and batch size of 8. Audio samples are 1.5s long sampled at 48kHz, and we use dequantization, random crop and allpass filters with random coefficients as our data augmentation strategy. We experiments with a various number of bands in the multiband decomposition, and eventually settle on $K = 16$.

Baselines

We evaluate RAVE in the context of unsupervised representation learning. We compare it against two state-of-the-art models: an unsupervised NSynth [64] and the autoencoder from SING [65] for both 16kHz and 48kHz modeling. We use the official 16kHz implementations for both baselines, and increase their capacity and receptive field when modeling 48kHz audio signals. We down-sample RAVE generations from 48kHz to 16kHz for a fair comparison with the 16kHz baselines.

Ablations

In order to evaluate the impact on synthesis quality of our 2-stage training strategy as well as modifications applied to the decoder, we train two variants of the RAVE model, one without adversarial fine-tuning, the other without noise synthesizer. We evaluate those variants together with the full RAVE and the baselines both for the perceptual test (see Section 3.1.4) and reconstruction error (see Section 3.1.4). We also demonstrate the importance of freezing the encoder during the second stage of training in Section 3.1.4.

Datasets

We use two internal datasets composed of respectively 30 hours of various *strings* recordings and 10 hours of *darbouka* recordings, both sampled at 48kHz. We employ a 90/10 train/test split. In addition, we use the Voice Conversion ToolKit (*vctk*) [66] dataset, using the same sampling rate and train/test split.

3.1.4 Results

Synthesis quality

We compare the synthesis quality of different models reconstructing audio samples from the *darbouka* dataset. During the qualitative experiment, participants were asked to rate audio samples on a scale of 1 (bad) to 5 (perfect). The test consisted in 12 trials, each presenting an audio sample from the *darbouka* test set alongside its reconstruction by all models. Participants were given either exclusively 16kHz or 48kHz samples during the entire test. A total of 44 participants took the test. We show in Table 3.1 that RAVE has a higher Mean Opinion Score (MOS) than the baselines while maintaining a significantly lower number of parameters. The ablation study of the adversarial fine-tuning shows how important this stage is in achieving high quality audio synthesis, especially for 48kHz signals

Model	16kHz	48kHz	#Params (16/48)
Ground truth	4.31	4.42	-
NSynth	2.36	2.69	64.7M / 103M
SING	2.68	2.92	80.8M / 101M
RAVE w/o adv	2.23	1.94	18.1M
RAVE w/o noise	3.77	3.97	**17.2M**
RAVE	**3.96**	**4.03**	18.1M

Table 3.1: Reconstruction quality (Mean Opinion Score)

As we can see, RAVE outperforms both NSynth and SING in terms of audio quality while having at least 5 times less parameters. The adversarial stage is responsible for a large increase in the assessed audio quality, demonstrating its crucial importance for obtaining high-quality audio generation. On the other hand, the addition of the noise synthesizer has a more subtle effect, but still increases the overall quality of RAVE synthesized signals. There is still a gap in the evaluation between the ground truth and the other models. This might come from the difficulty of modeling the variety of room acoustics

conditions present in the original dataset (as discussed in [56]), sometimes making it obvious that the evaluated samples are synthesized.

Reconstruction error

We evaluate the reconstruction error for all models using two metrics: the perceptually relevant Just Noticeable Difference (JND) score proposed in [67] and a L2 distance using amplitude mel-scale spectrograms. We report all results in Table 3.2.

Model	JND	Spectral
NSynth	2.51	9.54
SING	2.21	7.47
RAVE w/o adv	2.06	**2.92**
RAVE w/o noise	1.90	3.35
RAVE	**1.54**	3.26

Table 3.2: Reconstruction error (lower is better)

NSynth has the highest reconstruction error of all models, which is to be expected given that it has not been trained to minimize any kind of distance between its input and output. SING on the other hand is trained using a spectral distance, but is still outperformed by every variants of RAVE for both metrics. Removing the adversarial fine-tuning yields the RAVE model with the smallest spectral distance, but the highest JND score, which is to be expected since it has been trained using only a spectral distance. Enabling adversarial fine-tuning and adding the noise synthesizer increases the spectral distance, but lowers the JND score, demonstrating how input and reconstructed signals are perceptually closer to each other when using the full RAVE model.

Synthesis speed

Several approaches using NF [52] or GAN [28] can achieve faster than realtime synthesis, but only by relying on GPUs. Other approaches make strong assumptions about the signal in order to simplify the generation process [56, 68], allowing real-time synthesis on CPU by restricting the range of audio signals that can be modeled. In Table 3.3, we evaluate the synthesis speed of all models on both CPU and GPU. Synthesis speed is calculated as the average number of audio samples generated per second for 100 trials.

Model	CPU	GPU
NSynth	18Hz	57Hz
SING	304kHz	9.8MHz
RAVE w/o multiband	38kHz	3.7MHz
RAVE	**985kHz**	**11.7MHz**

Table 3.3: Comparison of the synthesis speed for several models

Being the only model relying on autoregressive synthesis, NSynth is also the slowest, peaking at 57Hz during generation. As expected, the parallel nature of SING and RAVE makes them several orders of magnitude faster than NSynth. The addition of the multiband decomposition speeds up RAVE by a factor of 25, allowing our model to outperform SING on both CPU and GPU. Overall, we obtain audio synthesis at 48kHz with a 20× faster than realtime factor on CPU, and up to 240× on GPU.

Balancing compactness and fidelity

Having a compact learned representation has several benefits, since it is easier to analyse, manipulate and understand. However, it usually comes at the expense of trading off the reconstruction quality. Instead of determining the latent space dimensionality prior to the training, we rely on the fidelity parameter f to estimate it post-training and accordingly crop the learned representation, as explained in Section 3.1.2.

In Figure 3.6, we compute the relationship between f and the estimated number of dimensions r_f for both the *strings* and *vctk* datasets. We depict the influence of f on the reconstruction quality by measuring the spectral distance (see Equation 5.4) between sample and reconstruction

By setting $f = 0.99$, the dimensionality of the learned representation drops from 128 to just 24 on the *strings* dataset and 16 on the *vctk* dataset. The downsampling factor of the encoder is 2048, resulting in a latent representation sampled at $\sim$ 23Hz. Further decreasing f results in a higher spectral distance as shown in Figure 3.6 (right), but significantly reduces the learned representation size. As the fidelity parameter decreases, reconstructed samples get less accurate, loosing parts of their attributes such as phonemes or speaker identity.

Compared to previous approaches combining VAE and GANs, there are no adversarial losses involved when training the encoder. We demonstrate in

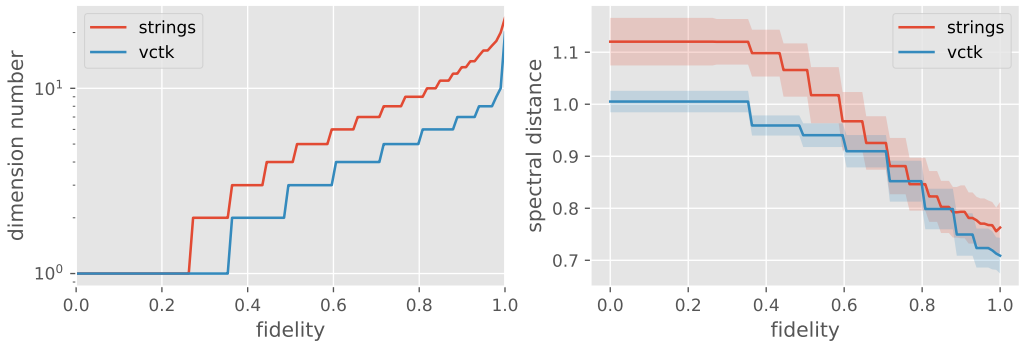
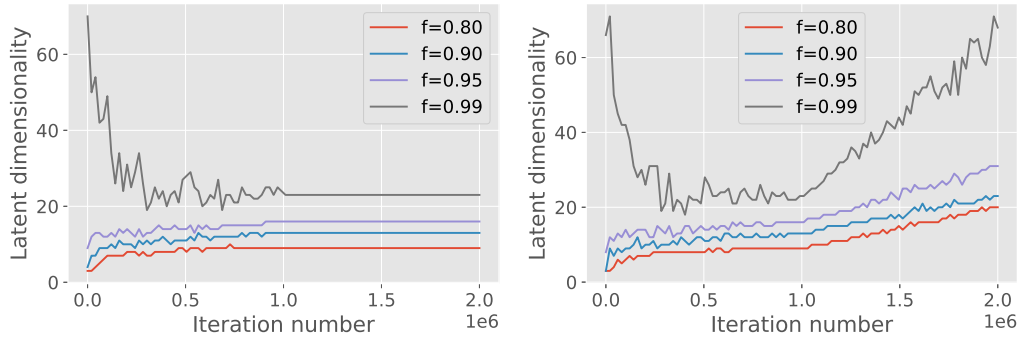


Figure 3.6: Estimated latent space dimensionality according to the fidelity parameter and its corresponding influence on the reconstruction quality.

Figure 3.7 how training the encoder to minimize the feature matching loss as proposed by [69] during the second stage results in a dramatically increased estimated latent space dimensionality, which goes against our objective of building a compact and perceptually relevant representation.



(a) Frozen encoder

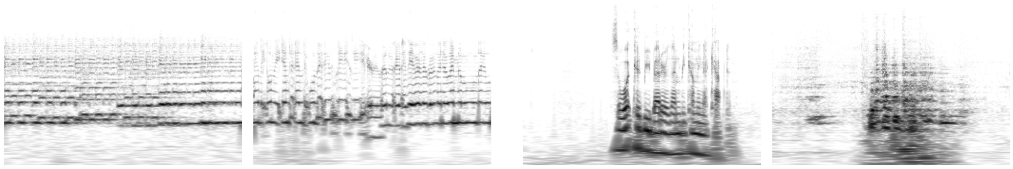
(b) Trained encoder

Figure 3.7: Comparison of the estimated latent space dimensionality for two trainings of RAVE on the Strings dataset with and without freezing the encoder during the *adversarial fine tuning* stage (starting at 1.10^6 iterations).

Timbre transfer

We demonstrate that RAVE can be used to perform domain transfer even if it has not been specifically trained to address this particular task. We perform domain transfer by simply providing to a pretrained RAVE model audio samples coming from outside of its original training distribution (e.g.

violin samples are reconstructed with a model trained on speech).



(a) Transfer from *strings* to *vctk* (b) Transfer from *vctk* to *strings*

Figure 3.8: Example of timbre transfer using RAVE.

High-level attributes such as the overall loudness and the fundamental frequency of the harmonic components are kept after performing domain transfer. Other audio attributes such as formants are absent in the examples coming from the *strings* dataset, and are added by the model in the speech-transfered version.

However, there are no guarantees that the encoder will encode signals from an out-of-domain distribution into a latent representation that matches the prior. We empirically evaluate the KL divergence of two trained RAVE models for in and out-of-domain data distributions, and report the results in Table 3.4.

	RAVE STRINGS	RAVE VCTK
DATA FROM STRINGS	$\mathbf{0.11 \pm 0.09}$	0.16 ± 0.11
DATA FROM VCTK	0.24 ± 0.37	$\mathbf{0.09 \pm 0.02}$

Table 3.4: KL divergence from the prior for in and out-of-domain data

Unsurprisingly, the smallest KL divergences are observed when encoding data sampled from the training distribution, while encoded out-of-domain signals roughly double the KL divergence. Stronger transfer abilities may be achieved by integrating a domain adaptation technique such as the one proposed by [70] or [71].

Latent manipulation

One of the main advantages of using VAEs compared to other generative frameworks is that one can alter the reconstruction of a sample manipulating its latent representation, effectively transforming high-level attributes of the data. We demonstrate in Figure 3.9 how biasing a single dimension from the representation learned with RAVE leads to impactful changes in

the reconstructed sound. In this particular example, there is a strong correlation between the first two dimensions of the latent representation and the amplitude and spectral centroid of the signal.

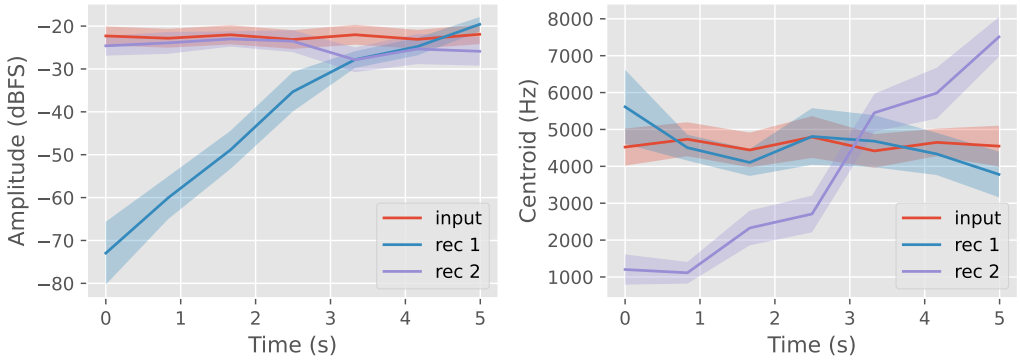


Figure 3.9: Amplitude and spectral centroid of two reconstructions of an input drum sample with altered latent representations. Rec 1 (resp. 2) has its first (resp. second) latent dimension set to a ramp linearly increasing from -2 to 2 over time.

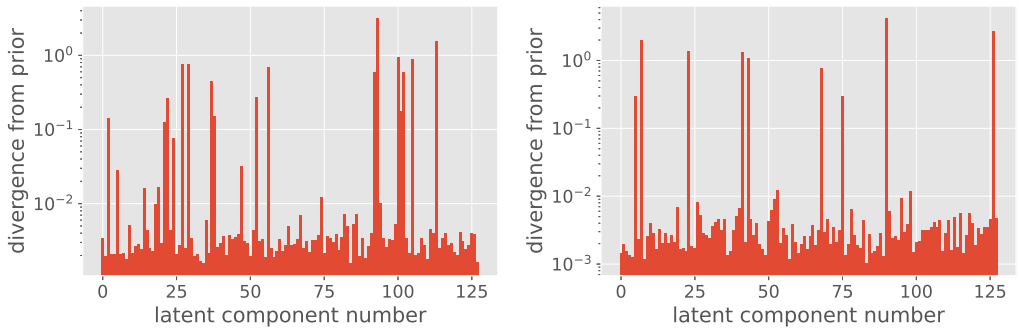
Latent component collapse

The regularization term in Equation 3.2 applies a pressure on the encoder to produce a posterior distribution that is close to the prior and, therefore, not correlated with the input data. Since this objective is contrary to the reconstruction objective, it has been empirically shown by [26] that some latents are more informative than others, as it can be seen in Figure 3.10 in the case of RAVE.

For both datasets, most of the latents have a low KL divergence (lower than 0.01), and only a few (respectively 16 and 9 for the strings and VCTK datasets) have a KL divergence higher than 0.1, which is consistent with the dimensionality estimated by the method proposed in Section 3.1.2 for a fidelity parameter $f = 0.95$.

3.2 Alternative latent regularization

In this section, we study alternatives to the KL divergence in Equation (2.25) as a regularization term, using approaches inspired from Wassertein AEs [72] and Vector Quantized Variational Auto Encoders (VQ-VAEs) [73].



(a) Strings dataset

(b) VCTK dataset

Figure 3.10: Mean Kullback-Leibler divergence for each latent component between the posterior distribution $q_\phi(\mathbf{z}|\mathbf{x})$ estimated over the test set of the strings and VCTK datasets and the prior distribution.

3.2.1 Wassertein regularization

The use of the ELBO puts a lot of pressure on the representation learned by a VAE. This is mainly due to the regularization being applied between two distributions of different natures: the estimated posterior $q_\phi(\mathbf{z}|\mathbf{x})$ and prior $p(\mathbf{z})$. This may eventually be the cause of posterior collapse, which is the extreme case where $q_\phi(\mathbf{z}|\mathbf{x}) = q_\phi(\mathbf{z}) = p(\mathbf{z})$.

An alternative regularization technique introduced in [72] evaluates the divergence between the aggregated posterior $\int_x q_\phi(\mathbf{z}|\mathbf{x})$ and the prior distribution, effectively lifting the independence constraint imposed by the ELBO. They propose the use of the Maximum Mean Discrepancy (MMD) between the posterior and a normal distribution as a regularization strategy. Formally, the MMD between two distributions q_ϕ, p is defined as

$$\text{MMD}^2(q_\phi, p) = \mathbb{E}_p[k(\mathbf{x}, \mathbf{x})]\mathbb{E}_{q_\phi}[k(\mathbf{y}, \mathbf{y})] - \mathbb{E}_{p,q_\phi}[k(\mathbf{x}, \mathbf{y})] \tag{3.8}$$

where k is a positive definite reproducible kernel. We use the Radial Basis Function (RBF) kernel as defined in [72].

We adapt RAVE to include this regularization technique. In practice, we use a deterministic encoder and compute the MMD between an encoded batch of examples and noise sampled from a normal distribution of the same size. We decrease the latent size down to the number of informative dimensions estimated for a RAVE regularized with a KL divergence and trained on the same dataset.

3.2.2 Discrete prior

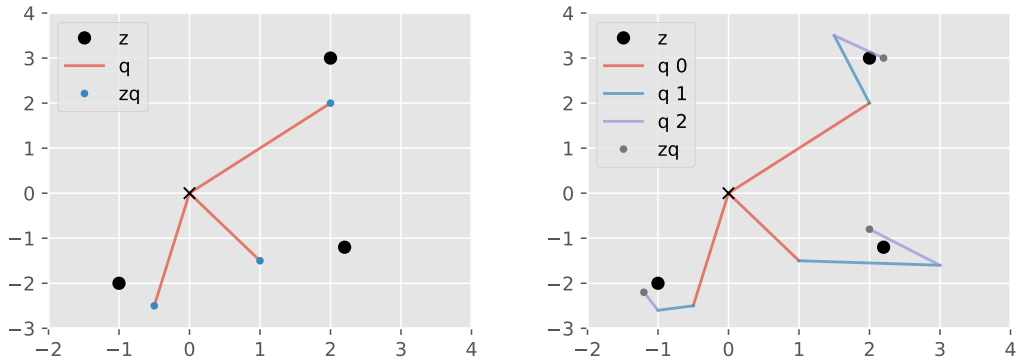
Using a categorical or *discrete* prior over the representation learned by a VAE is introduced in [73]. Their Vector Quantized Variational Auto Encoder (VQ-VAE) shows impressive compression abilities, combining benefits of having an indexed representation with an arbitrary large latent space. They introduce a quantization layer associated with a codebook C composed of V individual D-dimensional vectors. Their encoder yields a continuous D-dimensional vector $\mathbf{z}_c$, quantized to its discrete counterpart $\mathbf{z}_q$ by selecting the closest vector in C. As this operation is not differentiable, the quantization layer is ignored during the backpropagation of the gradients. A popular method to optimize the codebook during training is achieved through Exponential Moving Average (EMA) of each of its elements to the closest continuous vector of an encoded batch. If at some point during training an element of the codebook has not been used for a given number of training iterations, it is re-initialized as a random vector from the encoded batch.

Residual Vector Quantization (RVQ) is introduced in [74] as a way to increase the complexity of the discrete representation without increasing the number of entries in a given codebook. They instead introduce a number Q of quantizers V_q, each one of them quantizing the discretization error of the previous layer. Formally, given a continuous latent vector $\mathbf{z}$, the q vector quantized latent representations $\mathbf{z}_q$ representations are defined as

$$\begin{aligned} \mathbf{z}_0 &= V_0(z) \\ \mathbf{z}_q &= V_q\left(\mathbf{z} - \sum_{k=0}^{q-1} \mathbf{z}_k\right). \end{aligned} \tag{3.9}$$

We show in Figure 3.11 the differences between regular Vector Quantization ($Q = 1$) and RVQ ($Q > 1$).

We implement RVQ and add it to RAVE as an alternative to the previous continuous priors. We propose to use a number of quantizers Q equal to the number of informative latent dimensions estimated for a continuous training on the same dataset. While the size of the discrete latent space is significantly larger ($D = 512$), the indexed representation is a discrete tensor with shape $Q \times T$, which motivates us to take $Q = r_f$ (see Section 3.1.2). As proposed in [74], we use quantizer dropout, randomly sampling the number of quantizers Q during training.



(a) $Q = 1$ (b) $Q = 3$

Figure 3.11: Vector Quantization takes a continuous vector $\mathbf{z}$ and yields the element of its codebook C closest to $\mathbf{z}$. Residual Vector Quantization (RVQ) repeats the operation on the discretization error, refining at each step the quantization of $\mathbf{z}$.

3.2.3 Results

We evaluate the performances of all three regularization techniques (i.e. ELBO, MMD and RVQ) according to metrics reflecting both reconstruction performances and divergence from the prior. We evaluate the reconstruction abilities of all models using the spectral distance defined in Equation (2.42). Regarding the evaluation of the regularization, we estimate the KL divergence between an approximation $q_\nu(\mathbf{z})$ of the aggregated posterior distribution $\int_x q_\phi(\mathbf{z}|\mathbf{x})$ and the prior distribution $p(\mathbf{z})$. Given that the nature of the considered distributions are different (continuous versus categorical), we define different approximations of the aggregated posterior distribution depending on whether $\mathbf{z}$ is continuous or discrete.

Continuous prior. We use a mixture of Gaussian as a model for $\int_{\mathbf{x}} q_\phi(\mathbf{z}|\mathbf{x})$ with 48 independent components. We use the resulting model to draw random samples used for a Monte-Carlo estimation of the KL divergence between $q_\nu(\mathbf{z})$ and $p(\mathbf{z})$, following

$$\mathcal{D}_{\mathrm{KL}}\left[q_\phi(\mathbf{z})\|p(\mathbf{z})\right] \approx \mathcal{D}_{\mathrm{KL}}\left[q_\nu(\mathbf{z})\|p(\mathbf{z})\right] = \mathbb{E}_{\mathbf{z}\sim q_\nu(\mathbf{z})}\left[\log \frac{q_\nu(\mathbf{z})}{p(\mathbf{z})}\right]. \tag{3.10}$$

Discrete prior. An estimation $q_\phi(\mathbf{z})$ of the aggregated posterior distribution can be computed from the codebook usage statistics over the dataset.

We therefore obtain Q independent explicit categorical distributions. Given that we consider a uniform prior distribution $p(\mathbf{z}) = \mathcal{U}(0, N)$, we can compute its KL divergence with the aggregated posterior distribution following

$$\mathcal{D}_{\mathrm{KL}}\left[q_\phi(\mathbf{z})\|p(\mathbf{z})\right] \approx \mathcal{D}_{\mathrm{KL}}\left[q_\nu(\mathbf{z})\|p(\mathbf{z})\right] = \log N - H(q_\nu), \tag{3.11}$$

where $H(q_\nu)$ is the entropy of q_ν.

We train all models up to convergence on the VCTK dataset [75]. Both the reconstruction and regularization evaluations are performed using the validation set, and we report the results in Table 3.5.

Regularization	Reconstruction (↓)	Regularization (↓)
KL	1.22	**0.78**
MMD	**0.71**	2.67
RVQ	0.87	1.89

Table 3.5: Evaluation of reconstruction and regularization metrics for three RAVE models with different regularization methods.

Unsurprisingly, the RAVE trained with the KL regularization yields the highest reconstruction error of all variants, as it is the only training objective with fundamentally conflicting terms. Using the MMD as the regularization strategy yields the lowest reconstruction score, yet the aggregated posterior distribution shows the largest differences with the prior distribution. The discrete RAVE variant has reconstruction scores that are close to the MMD model, while matching its categorical prior more closely.

Choosing the regularization method for a given training depends on the target usage of the model. Applications where reconstruction accuracy is important should rely on either RVQ or MMD, while the KL method is a better fit for generative tasks.

In the strict formulation of the law of causality—if we know the present, we can calculate the future—it is not the conclusion that is wrong but the premise.

Werner Heisenberg

Chapter 4

Temporal learning

Variational Auto Encoders (VAEs) can be used to extract high-level features about a high-dimensional input dataset, resulting in a simpler representation easing both the analysis and manipulation of data [24]. When the model is built in such a way that the latent representation associated with an example is a single vector in $\mathbb{R}^D$, generating new examples using the model can be achieved by sampling from the prior distribution $p(\mathbf{z})$ and decoding the resulting latent vector $\mathbf{z}$. However, fully convolutional models such as RAVE [76] instead encode audio signals into trajectories living in $\mathbb{R}^{T\times D}$, where T (resp. D) is the number (resp. dimensionality) of individual latent time steps. While sampling from the prior distribution is still feasible, there are no guarantees that the resulting samples will preserve any temporal structure between time steps. A way to address this problem is to learn a second model $\hat{q}_\nu$ responsible for estimating the temporal behavior of variables sampled from the following aggregated prior distribution

$$\hat{q}_\nu(\mathbf{z}) \sim q_\phi(\mathbf{z}) := \mathbb{E}_{\mathbf{x}\sim\mathcal{X}}\left[q_\phi(\mathbf{z}|\mathbf{x})\right]. \tag{4.1}$$

The resulting model $\hat{q}_\nu(\mathbf{z})$ is called a *prior model*. Learning prior models give rise to a large number of applications, such as unconditional generation, continuation, outlier detection, conditional exploration, and many others. In this chapter, we study how approaches from the natural language processing literature can be applied to address this task. In Section 4.1 we study the use of different RAVE variants as a base model. We present the challenges of modeling and sampling from a multivariate temporal distribution, and briefly recall the current methods addressing this task in Section 4.2. In Section 4.3,

we propose three methods improving the theoretical computational efficiency of prior models on multivariate sequences. We implement in Section 4.4 three types of neural networks (i.e. recurrent, convolutional and transformer), and test different conditioning methods and computational optimization techniques. We finally present our experimental setup in Section 4.5 and the corresponding results in Section 4.6.

4.1 Continuous vs discrete latents

In this chapter, we rely extensively on the previously presented RAVE model (see Section 3.1) to extract a compact representation from a given audio dataset. While RAVE initially provides a continuous latent representation based on an input audio signal, we have shown in Section 3.2.2 that it can be adapted to yield a discrete representation. This is particularly useful when considering the NLP literature, where a given sequence is defined as a one-dimensional integer-valued tensor. As an alternative to directly using a discrete representation, previous work [3] successfully applied categorical models on continuous variables through the use of an intermediate discretization stage. However, discretizing a continuous variable inevitably comes at the expense of a loss of precision, or *quantization error*. Quantization using a non-linear spacing of quantized bins can be useful to decrease those errors by allocating more quantization bins where the continuous variable is denser. Given a continuous random variable $x \sim p(x) \in \mathbb{R}$, such a non-linear quantization can be obtained using its cumulative distribution function

$$F_x(y) = p(x \leq y). \tag{4.2}$$

Applying F_x to x gives a mapping from $\mathbb{R}$ to $[0, 1]$ where the resulting variable y is uniformly distributed. An optimal discretization of x can therefore be obtained through a linear discretization of the transformed variable y. The authors of [3] do not use this method but instead use a μ-law encoding, since it is a long-standing standard in audio telecommunication designed to reduce quantization errors for low bit-rate digital audio signals.

We study the use of two different discretization schemes for RAVE. The first one uses the standard RAVE regularization method (i.e. two-stage training, isotropic Gaussian prior), and discretizes the representation post-training using the previously presented method. We use the cumulative density function of the prior distribution $p(z) = \mathcal{N}(0, 1)$, defined as

$$F_x(y) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{y} e^{-y^2/2} \, \mathrm{d}y, \tag{4.3}$$

and discretize each dimension of the latent vector individually. The number of discretization bins Q is left as a hyperparameter. The second discretization scheme is the one described in Section 3.2.2, where RAVE is trained with a uniform categorical prior over Q quantizers. We use quantizer dropout as in [74], allowing to learn a representation with variable bit rates. In the end, both discretization schemes result in the encoder yielding discrete latent trajectories $\mathbf{z}$ such as

$$\mathbf{z} \in [0, V)^{T \times Q}, \tag{4.4}$$

where V is the number of potential discretization indices, also called *vocabulary size*, T is the number of time steps, and Q is the number of discrete dimensions. In the case of the continuous RAVE, Q is defined as the reduced latent space size given a fidelity parameter f, which is correlated with the number of *useful* dimensions. For the discrete variant of RAVE, Q is the number of residual quantizers, which is left as a hyperparameter.

One interesting property of both representations is the hierarchy implied between the Q different dimensions. The use of PCA for the continuous representation and variable bit rate for the discrete one organizes the latent representation in such a way that dimensions are sorted from most to least important. This enables us to perform sequence modeling experiments with a variable number of latent dimensions without retraining RAVE.

4.2 Multivariate autoregressive modeling

4.2.1 Discrete sequence modeling

One of the reasons why autoregressive sequence modeling is mostly applied to discrete sequences comes from the possibility to explicitly compute the categorical distribution for a given time step. Continuous random variables are by nature able to take an infinite number of different values. Thus, building a suited parametrized distribution implies a trade-off between expressive power and computational complexity. Complex methods can be used in such cases [77, 47], but are usually coming at the expense of a long training time.

Oppositely, modeling discrete sequences can be achieved through the explicit computation of the probability distribution for each discrete value given some

context. In order to evaluate the likelihood of a sequence composed of T tokens $\mathbf{z} \in [0, V)^T$ given a context extraction function c, a model f should output *logits* defined as

$$\mathbf{l} := f \circ c\ (\mathbf{z}) \in \mathbb{R}^{T \times V}, \tag{4.5}$$

transformed into a probability distribution using a *softmax* function over the last dimension. The nature of the context function c defines how the model predicts the sequence and can range from masked prediction tasks [44] to autoregressive modeling [3]. The model can then be trained using a cross-entropy loss, defined as

$$\mathcal{L}(f, \mathbf{z}) = -\sum_{t,v} \log \operatorname{softmax}(\mathbf{l}_{t,v}) \times \mathbf{z}^o_{t,v}, \tag{4.6}$$

where $\mathbf{z}^o \in \{0, 1\}^{T \times V}$ is a one-hot tensor with a one at the indices given by $\mathbf{z}$, and zeros everywhere else. Optimizing this loss over a sufficiently large dataset results in the model correctly approximating the categorical distribution of $\mathbf{z}$, thus allowing to perform density estimation and sampling. The categorical nature of the yielded distribution allows an easy computation of metrics such as accuracy and perplexity, which are useful to understand the uncertainty or performance of a model given a specific example.

In this chapter, we consider an autoregressive context extraction function c, resulting in the model being trained to approximate the conditional distribution $p(\mathbf{z}_t | \mathbf{z}_{<t})$.

4.2.2 Multivariate extension

The modeling method presented in the previous section has been extensively tested and used for both research and production purposes for tasks like natural language processing or time series forecasting [23]. However, its extension to the modeling of multivariate time series such as the one described in Equation (4.4) remains challenging. This is due to the fact that computing the joint distribution between the Q quantizers results in a prediction living in $\mathbb{R}^{V_1 \times \ldots \times V_Q}$ for each time step. While this is feasible for a multivariate sequence with few parallel dimensions (e.g. $Q \leq 3$), scaling both the vocabulary size V and the sequence depth Q would result in a prediction that could

hardly fit in any current computer memory. For example, predicting the likelihood of a sequence of length $T = 256$ with $V = 1024$ and $Q = 4$ would require one full petabyte (1,000,000 GB) of memory to be stored, without even considering batch of examples nor the cost of backpropagating errors through this prediction.

Previous works have addressed this problem for audio generation using two methods. The JukeBox model [4] uses a representation learning model with a small temporal compression ratio, yielding long univariate discrete sequences allowing to leverage classical univariate temporal approaches, at the cost of an extremely long sampling mechanism. The recent AudioLM model [78] improves the generation speed by using a representation learning model with a high compression ratio, producing a multivariate sequence such as the one described in Equation (4.4). Prior to modeling the resulting multivariate sequence, they flatten it into a $[0, V)^{(TQ)}$ univariate sequence that is Q times longer than the original multivariate one. This increase in temporal dimensionality makes the temporal learning task harder, hence they split the modeling process into two separate stages, the first one modeling the $q < Q$ first dimensions, the second one modeling the others.

While both methods are able to produce convincing new sequences through unconditional generation or general audio signal continuation, the high temporal complexity of the base sequences implies the use of very large models, resulting in a Real-Time Factor (RTF) far larger than 1 even using expensive accelerators. This precludes the use of those models on consumer grade computers, especially for realtime applications.

4.3 Efficient multivariate parametrization

The performances of the JukeBox [4] and AudioLM [78] models show that a sweet spot for efficient sequence modeling lies in the use of a highly compressing representation learning model combined with a temporal model applied directly on multivariate sequences. In this section, we propose three methods to get as close as possible to this sweet spot by leveraging a *temporal model* and several optional *depth-wise* models. As shown in Figure 4.1, both kinds of model (i.e. temporal or depth-wise) are *causal*, the former over time-steps, the latter over dimensions. The outputs of the temporal model are fed to the Q depth-wise models in order to produce a prediction.

We propose three different ways of connecting the temporal and depth-wise models, named *decoupling*, *residual* and *shift* that we evaluate against the

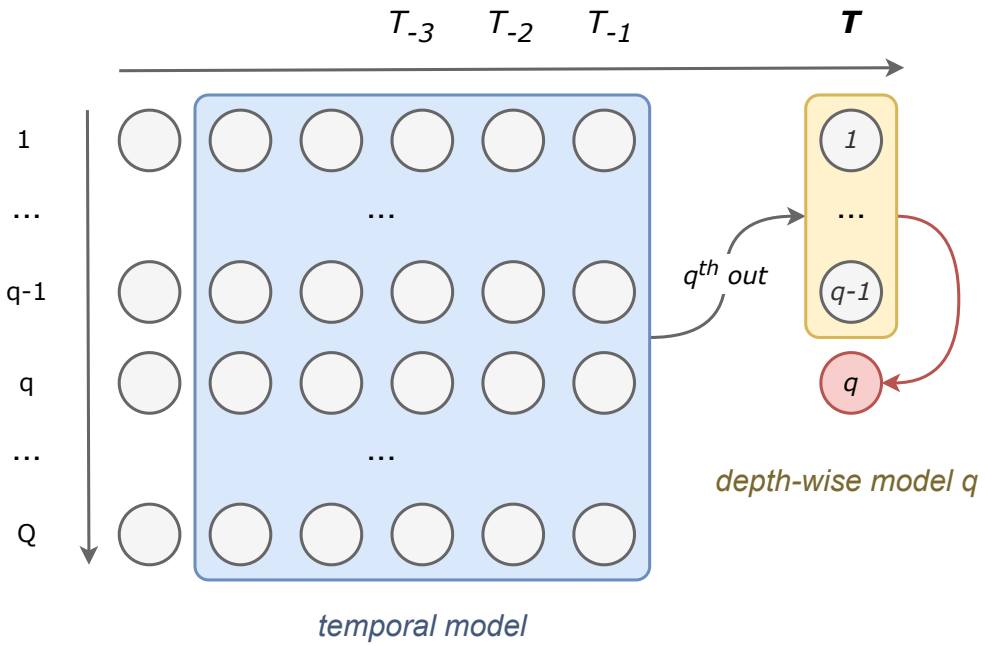


Figure 4.1: Block diagram of our method applied to a sequence with Q quantizers. Dimension q of time-step T is predicted by the q^{th} depth-wise model, taking as input the $q-1$ previous dimensions and the q^{th} output of the temporal model applied on all previous time-steps $t < T$.

flattened baseline proposed in AudioLM [78].

4.3.1 Decoupling method

The first method is called *decoupling*, as the prediction of the model is the combination of three separate distributions: a temporal distribution p_t, a depth-wise distribution p_q and a prior distribution p_p. Formally, we propose to parametrize the autoregressive distribution of the multivariate sequence as

$$p(\mathbf{z}) = \alpha \prod_{q,t} \frac{p_t(\mathbf{z}_t^q | \mathbf{z}_{<t}) p_q(\mathbf{z}_t^q | \mathbf{z}_t^{<q})}{p_p(\mathbf{z}_t^q)}, \tag{4.7}$$

where α is a normalizing factor. Note that we suppose that $\mathbf{z}_t^{<q}$ is conditionally independent from $\mathbf{z}_{<t}$ given $\mathbf{z}_t^q$. Distributions p_t and p_q are parametrized with neural networks, while the prior distribution p_p is statistically estimated over the training dataset. Dividing the product of models by p_p is needed to account for both p_t and p_q implicitly modeling the prior distribution over the codebooks. Thanks to the double auto-regression over time steps and quantizers, this method has the same probabilistic graphical model as AudioLM [78] or JukeBox (with $Q = 1$) [4]. However, we empirically show that the model needed to parametrize the depth-wise distribution p_q can be dramatically simpler than the one used for the temporal distribution p_t. This allows the use of longer sequences during training, while keeping a smaller memory footprint. During generation, the depth-wise model adds a negligible overhead as the sampling time is largely dominated by the temporal model (more details in Section 4.5).

4.3.2 Residual method

A natural modification to the previously proposed decoupling method is to use the depth-wise model as an extension of the temporal model. The base principle is the same, but instead of performing two separate predictions (i.e. p_t and p_q), we use the temporal model to predict a context based only on past samples $\mathbf{z}_{<t}$ that is used by the depth-wise model to autoregressively predict the individual dimensions, resulting in the following parametrization

$$p(\mathbf{z}) = \prod_{q,t} p(z_t^q | \mathbf{z}_t^{<q}, \mathbf{z}_{<t}). \tag{4.8}$$

Given that a single prediction is now yielded by the model, computing the prior distribution p_p is not required anymore. Similarly to the decoupling method, we use a large temporal model and a lightweight depth-wise model. We show in Figure 4.4 and 4.5 the differences between the *decoupling* and *residual* methods. We call this method *residual* as its implementation can be seen as a residual network [79] with Q skip connections providing predictions for all quantizers.

4.3.3 Shift method

Another way of introducing depth-wise dependencies while predicting individual dimensions in parallel can be obtained through the *shift* method. This method only uses a temporal model, and leverages a shifting sequence preprocessing trick to introduce an approximation to the full distribution described in Equations (4.7, 4.8).

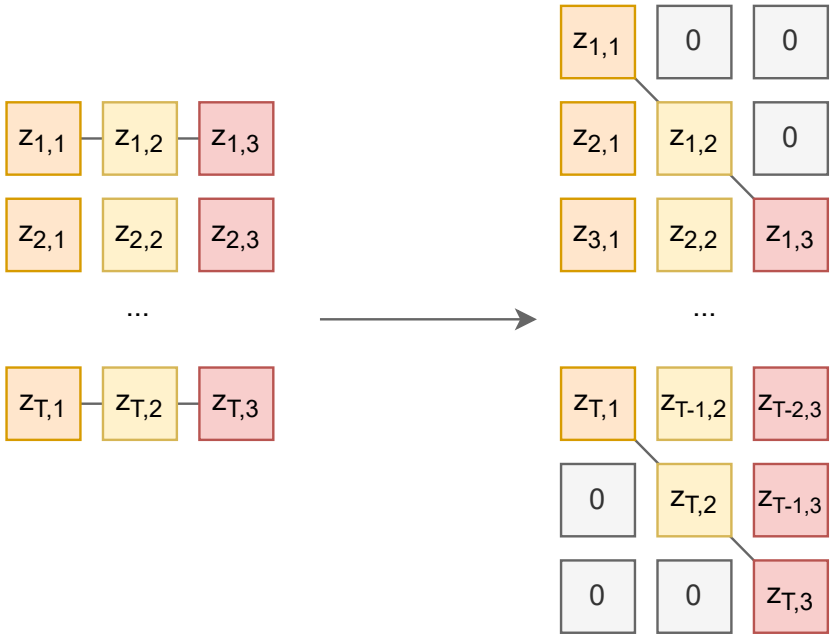


Figure 4.2: Shift preprocessing trick. Given a sequence $\mathbf{z} \in [0, V)^{T \times Q}$ with in this case $Q = 3$, we apply the shift transform by delaying dimension i by i time-steps. We use padding to obtain the final $[0, V)^{(T+Q-1) \times Q}$ sequence.

This shift trick is achieved by introducing a delay to each dimension of the sequence. More specifically, dimension $i \in [0, Q)$ is delayed by i time steps (see Figure 4.2 for a visual depiction of the transform). The model is therefore trained to approximate the following distribution

$$p(\mathbf{z}) \approx \prod_{q,t} p(\mathbf{z}_t^q | \mathbf{z}_{<t+q}^0, \mathbf{z}_{<t+q-1}^1, \dots, \mathbf{z}_{<t+q-Q}^Q). \tag{4.9}$$

It is important to note that contrary to Equations (4.7, 4.8), Equation (4.9) is different from the probabilistic graphical model used by JukeBox [4] or AudioLM [78], which in some cases might not be good enough to produce coherent sequences during sampling. However, we empirically show that the hierarchical nature of the sequence produced by RAVE are compatible with such an approximation (see Section 4.6 for more details.).

4.4 Network definition

In this section, we implement several architectures parametrizing the different distributions presented in the previous section. Various architectures have been proposed in the sequence modeling literature to parametrize autoregressive models, most of those being categorized as either convolutional [3], recurrent [80], or attention-based [78]. We implement one model from each category, namely a WaveNet convolutional model [3], a Gated Recurrent Unit (GRU) [14] and a Transformer [23] model with ALiBi attention [41].

4.4.1 Architecture

All model configurations are composed of a temporal sub-model and optional depth-wise sub-models. In this section, we present implementation details for all sub-models.

Temporal model architecture

We start by adapting the WaveNet architecture [3] to the multivariate sequence prediction task. WaveNet is a fully convolutional architecture leveraging dilated convolutions to expand its receptive field. Every convolutional layer in WaveNet is using causal padding, preventing the model from seeing into the future during training which would make the problem trivial. Our implementation uses 12 stacked residual layers with dimension $D = 512$ and kernel size $K = 3$. When computing the loss, we crop both the target and prediction by an amount of time-steps equal to the receptive field of the model. This prevents training the model on padding positions. We show in Figure 4.3 a block diagram of our implementation of the model.

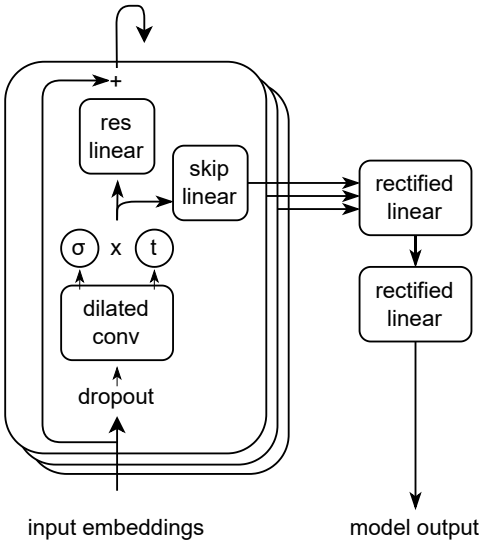


Figure 4.3: Block diagram of the implemented WaveNet-like model. We use sigmoid σ and hyperbolic tangent t activations.

We then study the use of a multi-layer recurrent model for this task. In practice, we implement a GRU with 12 stacked layers and hidden dimension $D = 512$. We use a dropout rate of 0.1.

Finally, we implement a decoder-only Transformer [23] with a pre-norm configuration [81]. We use the multi-head ALiBi [41] relative positional encoding scheme with $H = 8$ heads. The full model is composed of 12 stacked transformer layers with dimension $D = 512$, and use a dropout rate of 0.1.

While having very different architectures, all models share a roughly equivalent number of parameters (≈ 40M).

Depth-wise model architecture

As stated in Section 4.3, we propose the use of a lightweight architecture to model the depth-wise dependencies of the sequence. In practice, all experiments involving a depth-wise model use a Multi-Layer Perceptron (MLP) with 3 layers, no dropout, and SELU activation (see Table 2.2) for each level of quantization. This implies at most Q separate MLPs trained to predict the distribution of each individual dimension.

When using the decoupling method, the prediction for the first quantizer is entirely defined by the temporal model. The remaining quantizer distributions are predicted using all three models (p_t, p_q, p_p), as described in Equa-

tion (4.7). We show in Figure 4.4 how sub-models are arranged together to provide the final prediction.

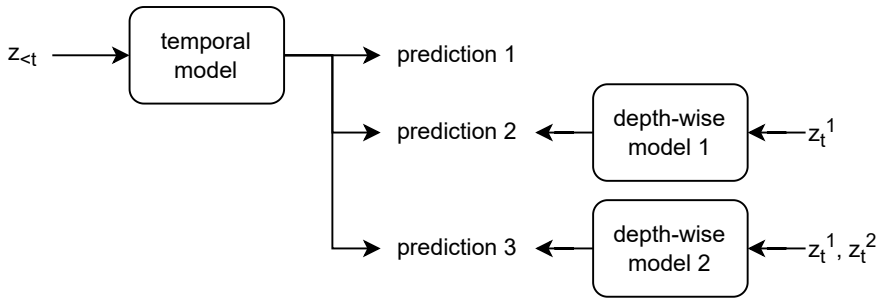


Figure 4.4: Architecture of the decoupling method. The temporal model p_t produces a prediction for each quantizer, while depth-wise models p_q provide a prediction for quantizers $q > 1$ given the previous ground truth targets.

The residual method produces a single prediction for each individual quantizer, using Q depth-wise models taking as input the context c produced by the temporal model, and the previous ground truth targets (during training) or the previous predictions (during inference), as shown in Figure 4.5.

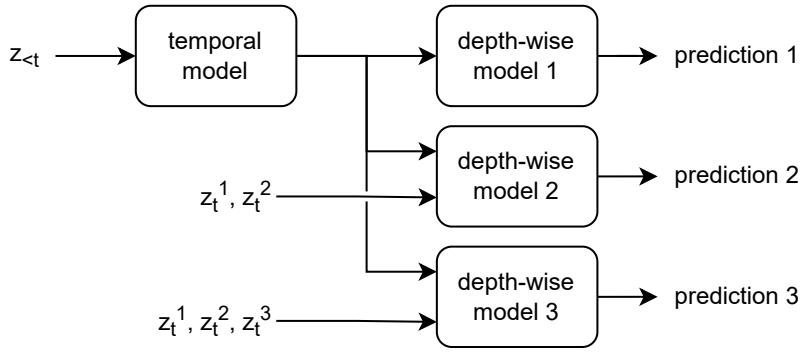


Figure 4.5: Architecture of the residual method. The temporal model p_t produces a context fed to all depth-wise models for which quantizer $q > 1$ are additionally conditioned on the previous ground truth targets.

In contrast with previous methods, the flattened and shifted methods discard the depth-wise models and only use the temporal model for prediction.

4.4.2 Embedding

The first layer of a deep learning model applied to discrete sequences is generally an *embedder*, transforming a discrete sequence $\mathbf{z} \in [0, V)^{T \times Q}$ into a

continuous tensor $\mathbf{x} \in \mathbb{R}^{T \times D}$ with D being the dimensionality of the model, left as a hyperparameter. Most of the time, embedding a discrete sequence is achieved by using a potentially trainable look-up-table called *codebook* of shape $V \times D$. We implement two variants of the embedding layer.

Learnable codebook

The first embedding layer we implement is a randomly-initialized codebook living in $\mathbb{R}^{Q \times V \times D}$, which can be interpreted as Q different codebooks of shape $V \times D$, one for each dimension of the sequence. We embed each individual dimension of the sequence using the corresponding codebook, resulting in a $T \times Q \times D$ tensor. We draw inspiration from Product Quantization [46] and experiment with three aggregation methods to combine the different embedded vectors. The first two methods are simple sums over the Q dimensions, with the second method using a learnable vector to perform a weighted sum. The last method reshapes the embedding into a $T \times QD$ tensor before being projected by a linear mapping $A \in \mathbb{R}^{QD \times D}$ into the final $T \times D$ embedding.

Pretrained codebook

It has been shown that using a good initial codebook dramatically helps the convergence of the downstream model [46], as two different tokens with similar encoded features should be close to each other in the embedded space. Given that both the continuous and discrete RAVE produce Q-dimensional discrete vectors that are computed from a D-dimensional continuous vector, we experiment with using directly the continuous vectors as a form of embedding. More specifically, we retrieve the continuous vectors for each token in the multivariate sequence, yielding a tensor living in $\mathbb{R}^{T \times Q \times Z}$, with Z being the dimension of the pretrained embedding. We combine the gathering modes previously presented with a final linear projection yielding an embedding in $\mathbb{R}^{T \times D}$.

4.4.3 Conditioning

Autoregressive models are by nature able to generate sequences based solely on their own previous generations. However, the resulting generative system is not controllable. While this may be a desirable behavior, most creative applications require a way to control or at least steer the generation using explicit parameters. To achieve such a goal, we experiment with learning conditional models. In practice, we augment our sequence dataset with time-aligned acoustic descriptors that we feed to the temporal models alongside

the discrete sequences. We show in Section 4.6 how this helps with controlling the generation process by modifying the values of the descriptors.

Acoustic descriptors

We choose to conduct experiments based on two different acoustic descriptors, namely the spectral *centroid* of the raw waveform and an estimated *beat track*. The centroid is perceptually related to the *timbre* of a signal, and we use the following equation to compute it

$$\text{centroid}(\mathbf{x})[n] = \frac{\text{sr}}{n_{\text{fft}}} \sum_k |\text{STFT}(\mathbf{x})[k, n]|\,, \tag{4.10}$$

where sr is the sampling rate of the audio signal, n_{fft} is the number of samples used during the computation of the Fourier transform. We also implement *loudness* and *spectral flatness*, but leave their use as conditioning signals for future work. In addition to the centroid, we study the use of a *beat track* as a conditioning signal. Beat tracks represent the positions in an audio signal where human listeners identify a beat, and thus might be tempted to mark it with their feet. We use the implementation of the algorithm proposed in [82] available in the *librosa* python library [83]. The resulting beat track is a vector with zero everywhere except at the positions identified as *beats*, as shown in Figure 4.6.

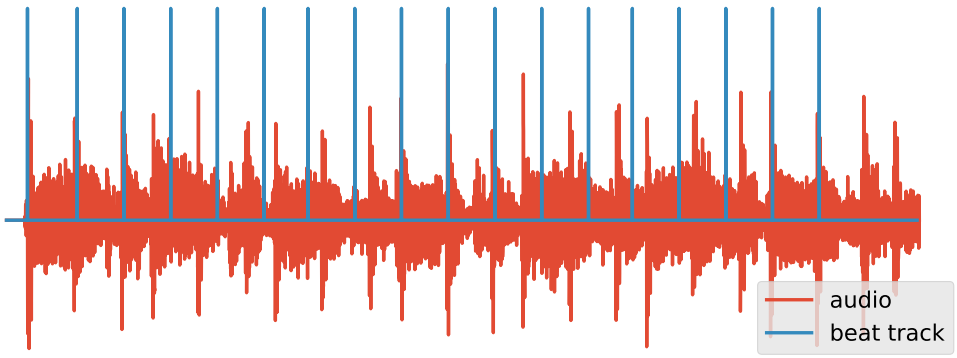


Figure 4.6: Beat track estimated from an audio waveform using *librosa* [83].

Model conditioning

Both the centroid and beat track of an audio signal are 1-dimensional sequences temporally aligned with the audio waveform. However, the centroid

is an inherently continuous descriptor whose values are contained in $[0, \mathrm{sr}/2)$ while the beat track is a binary sequence. Therefore, we propose conditioning methods suited to both descriptors, taking into consideration their continuous or discrete nature. We normalize the centroid so that its maximum value never exceeds 10000, and embed it into a D-dimensional tensor using the Sinusoidal Positional Encoding (SPE) embedding layer presented in equation (2.35). Given the discrete nature of the beat track, we use a regular D-dimensional embedding layer with a vocabulary size $V = 2$. All conditioning embeddings are summed to the embedded RAVE latent sequence prior to being fed to the temporal model.

4.4.4 Optimized inference

As we are targeting a real-time use of our prior models, we implement variant-specific optimizations of the generation loop to prevent unnecessary computation. We leverage the cached convolution mechanism that we propose in Chapter 5 to avoid any redundant computation when using the convolutional model. Similarly, we cache the recurrent state of the GRU model to avoid recomputing the full temporal context at each time-step. Regarding the transformer model, we cache keys and values inside all multi-head attention layers, allowing to compute the attention operation with a linear memory and computation complexity during inference.

Overall, all the variants of our model are faster than realtime. We report computation efficiency benchmarks results in Section 4.6.

4.5 Experiments

Our experimental setup targets the evaluation of all the variants introduced in Sections 4.3 and 4.4.1. In addition, we evaluate the effect of conditioning signals on all configurations using the descriptors proposed in Section 4.4.3. Therefore, we evaluate combinations of 3 architectures (*Convolutional*, *Recurrent* and *Transformer*), 4 parametrizations (*flattened*, *decoupling*, *residual*, *shifted*) and 2 conditioning (*unconditional*, *Beat + Centroid*) setups.

We build a dataset of ≈ 730 hours of royalty-free techno music scraped from internet. The dataset is split into two *train* and *test* subsets composed of respectively 90% and 10% of all the examples. We train two variants of the RAVE model, a continuous RAVE with post-training discretization following the method proposed in Section 4.1, and a discrete RAVE with $Q = 16$ quantizers. We evaluate the reconstruction error of both models with relation

to the number of discrete dimensions Q on the test set using the spectral audio distance defined in Equation (2.42). The results of this experiment are reported in Figure 4.7, and allow the selection of both the RAVE model variant and quantizer truncation level used for the next experiments.

We train all the prior model variants using the Adam optimizer [10] with a learning rate ramping from 10^{-6} to 5×10^{-4} over the first 30k iterations, and then exponentially decayed. All models are optimized until convergence (i.e. when the validation cross-entropy stops decreasing for 10 consecutive epochs). We use sequences composed of 256 tokens, which approximately represent 6 seconds of audio at 44.1kHz, with a batch size of 128.

We are interested in evaluating the impact of our multivariate distribution parametrizations compared to the flattened baseline used in AudioLM [78]. Therefore, we evaluate both the accuracy and perplexity of all model configurations on the test set. We compute the accuracy as the number of correct model predictions divided by the number of total predictions, averaged over time-steps and quantizers. The perplexity of the model is defined as the exponential of the entropy of its normalized prediction, also averaged over time-steps and quantizers. We report the results in Tables 4.1, 4.2 and 4.3.

In order to evaluate the influence of the conditioning on the generation process, we compare the centroid computed from a conditional generation with its target centroid, and show the results for several generations in Figure 4.8.

Finally, we evaluate the computational efficiency of all configurations by computing the Real-Time Factor (RTF) for both feed-forward generation and autoregressive inference.

4.6 Results

We evaluate the reconstruction error of both RAVE variants on the validation set and report the results for a number of discrete dimension Q varying from 1 to 16 in Figure 4.7.

As expected, the reconstruction error decreases with the augmentation of Q for both models. However, the discrete RAVE model has an overall lower reconstruction error than its continuous counterpart, which stabilizes around $Q = 8$. Therefore, we continue the experiments using the discrete RAVE only, with a fixed number of quantizers Q set to 8.

Regarding the discrete sequences embedding strategy (see Section 4.4.2), initial experiments have shown that using a learnable randomly initialized code-

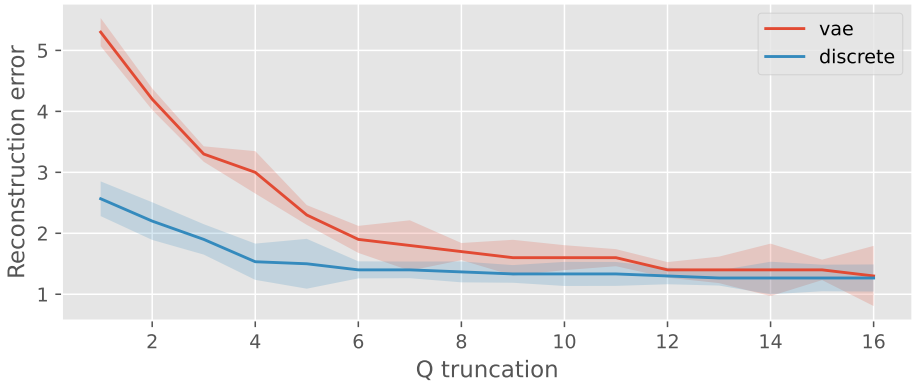


Figure 4.7: Reconstruction error of two variants of RAVE (original VAE and discrete) for various quantizer truncation levels. Shaded areas represent the standard deviation of the error for a given truncation level.

book helps with the convergence of the model. Therefore, all the following experiments are performed using this embedding method.

We train all variants of the temporal model on the dataset discretized by the previously selected RAVE model, and report the accuracies and perplexities of the transformer, convolutional and recurrent variants respectively in Table 4.1, 4.2 and 4.3.

In average, the transformer variants yield the highest (resp. lowest) accuracies (resp. perplexities), closely followed by the recurrent variants. The convolutional variants are far behind in terms of performances. Some configurations (i.e. shifted convolutional, conditioned shifted and flattened recurrent) resulted in systematic divergence of the model. We choose to remove their metrics from the results.

The distribution parametrization method has a large impact on the performances of the system. Both the transformer and recurrent approach yield their highest accuracies with the *residual* method, closely followed by the *decoupling* method. Flattening the sequences as performed in AudioLM [78] yields lower accuracies than both the *residual* and *decoupling* methods, yet outperforms the *shifted* method in most cases.

In almost all cases, the conditioning of the models on the audio descriptors helped with reaching higher performances. However, the gain in accuracy is overall relatively small, suggesting that the conditioning signal might not be strong enough, or should be fed to the network differently (e.g. using FiLM layers [84], or using cross-attention with an encoder-decoder model [23]).

MODE	CONDITIONED	ACCURACY (%)	PERPLEXITY
RESIDUAL	✓	**22.81**	**31.52**
RESIDUAL	✗	22.55	32.23
DECOUPLING	✗	21.20	35.69
DECOUPLING	✓	20.84	36.45
FLATTENED	✓	17.54	44.96
FLATTENED	✗	17.00	46.52
SHIFTED	✓	15.69	53.36
SHIFTED	✗	15.21	54.58

Table 4.1: Accuracy and perplexity values for the *transformer* temporal model, sorted by decreasing accuracy.

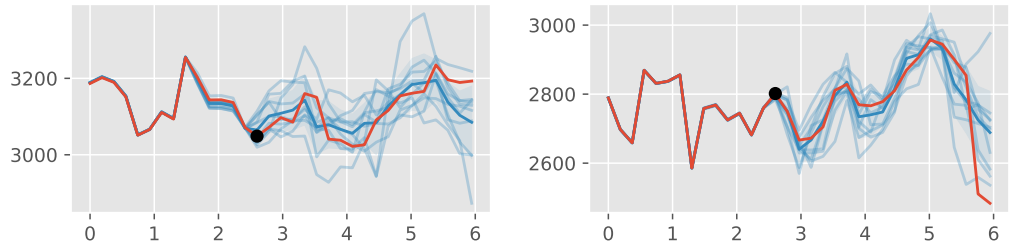
MODE	CONDITIONED	ACCURACY (%)	PERPLEXITY
DECOUPLING	✓	**9.31**	83.14
DECOUPLING	✗	8.97	**78.24**
RESIDUAL	✓	6.88	98.84
RESIDUAL	✗	6.42	102.34
FLATTENED	✓	1.05	418.79
FLATTENED	✗	1.01	371.83
SHIFTED	✓	-	-
SHIFTED	✗	-	-

Table 4.2: Accuracy and perplexity values for the *convolutional* temporal model, sorted by decreasing accuracy.

MODE	CONDITIONED	ACCURACY (%)	PERPLEXITY
RESIDUAL	✓	**19.77**	**36.95**
RESIDUAL	✗	19.56	37.07
DECOUPLING	✗	18.90	42.12
DECOUPLING	✓	18.09	44.78
SHIFTED	✗	13.68	64.23
FLATTENED	✗	12.80	67.86
FLATTENED	✓	-	-
SHIFTED	✓	-	-

Table 4.3: Accuracy and perplexity values for the *recurrent* temporal model, sorted by decreasing accuracy.

Nonetheless, we evaluate the effect of the conditioning signal by computing a centroid estimation on the results of 10 conditional generations using the conditional residual transformer configuration. Then, we compare it to the ground truth conditioning signal. We show in Figure 4.8 the results of this comparison on two different examples from the validation set. Despite its



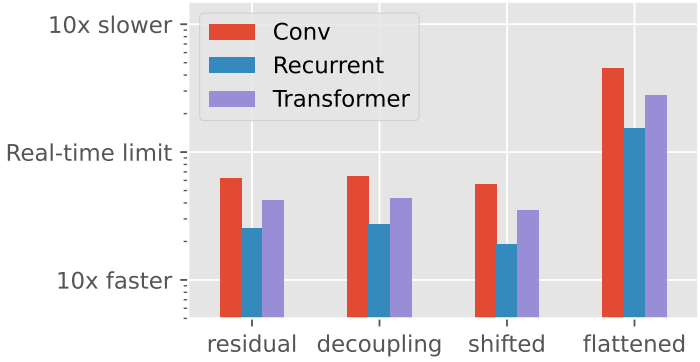
(a) Sample A (b) Sample B

Figure 4.8: Target centroids (red) compared to the centroids computed from the results of 10 conditional generations (blue). The model used is a conditional residual transformer. The black dot indicates a switch from teacher forcing mode to autoregressive mode.

small impact on model accuracy, this comparison shows that our conditioning method effectively makes our temporal model controllable. This is important for many use cases, as the nature of the conditioning signal itself can be defined based on the end application type (e.g. speaker identity or emotion for speech models, tempo, chords or centroid for music models).

Regarding the computational complexity of the resulting models, we compute the RTF of every variant, excepting the conditional ones. This is due to the fact that the conditioning mechanism adds a negligible computational overhead to each model, and would unnecessarily complexify the reading of the results. The resulting RTFs are reported in Table 4.9.

All of our proposed methods are faster than realtime in both autoregressive and teacher forcing mode. However, in autoregressive mode, the baseline flattened method is 1.5× slower than real-time for the most lightweight model possible, and up to 4× slower than real-time when using the convolutional model. In teacher forcing mode, all methods including the baseline are faster than real-time. Overall, given a specific architecture type, our three methods are faster than the flattening baseline. In addition to this, both the *residual* and *decoupling* models are yielding higher accuracies than the baseline, at a fraction of its computational cost during prediction.



(a) Autoregressive

(b) Teacher forcing

Figure 4.9: RTF for variants in autoregressive (a) or teacher forcing (b).

Improvisation is an immediate stream of consciousness, a compositional form where you cannot erase mistakes, but instead should make them work after the fact.

William Basinski

Chapter 5

Real-time interaction

Neural audio signal processing has set a new state of art in many fields, such as audio source separation [85], text-to-speech [51], timbre transfer [56] and unconditional generation [4]. Recent works on neural audio synthesis such as DDSP [56], melGAN [28] or RAVE [76] allows to perform deep audio synthesis faster than real-time. Those methods pave the way towards the integration of neural synthesis inside real-time audio applications.

Amongst these, models based on recurrent layers (DDSP [56] or RNNoise [86]) are built to process time series sequentially. Therefore, they are naturally fit to process live audio streams by caching their recurrent state in-between DSP calls. However, this is not the case for models based on convolutional networks [12] since their reliance on *padding* causes audible phase discontinuities between consecutive audio buffers (e.g clicks), which prevents their use for real-time audio applications. A simple solution to address this problem would be to rely on the *overlap-add* method, where we process large overlapping audio buffers and cross-fade them to smooth out phase discontinuities. While this method is straightforwardly compatible with any generative model, processing overlapping buffers leads to redundant computations and degraded quality during transition phases. In addition, this method requires caching buffers that are large enough to fill the receptive field of the model in order to avoid edge effects. This results in a high latency between the input and output of the model during inference. A more specific solution have been proposed through the idea of *streaming* models [87, 74] that use *causal* convolutional layers. These layers replace padding during inference with a cached internal or external state. Although this mechanism allows the use of convolutional models on live audio streams, it usually degrades the model accuracy due to the aforementioned causal constraint.

In this chapter, we propose a method to make non-causal convolutional neural networks streamable without impacting the audio quality nor introducing computational redundancies. We achieve this by making the model causal *after training*[1], leveraging additional internal delays in order to preserve the original computational graph of the model. Hence, our method can be applied over models that were already trained in a non-causal way. As an application case, we use our method to make our RAVE model (see Chapter 3) streamable in real-time. However, our approach can be applied straightforwardly to any convolution-based model. We compare our method with several *overlap-add* alternatives using both quantitative and qualitative metrics. We demonstrate that our method outperforms all other baselines in inference speed, while behaving exactly like the original model in terms of audio quality. Finally, we develop several applications leveraging our method to provide regular digital audio workstations with real-time neural audio processing abilities. All of our experiments, methods and source code are packaged as an open-source Python library available online[2].

5.1 Streaming models

Processing audio buffers one after the other using a convolutional neural network is not trivial. Indeed, the use of padding in each layer of the model creates discontinuities in the data when processing two consecutive buffers sequentially. In the context of neural audio synthesis, and more specifically raw waveform modelling, this causes audible phase discontinuities that are not acceptable for real-time audio applications. To address this problem, Rybakov et al. [87] proposed to rely on *causal* Convolutional Neural Networks (CNN) altogether with a *cached padding* mechanism.

5.1.1 Cached padding

Cached padding is implemented by retaining the end of one tensor and using it to left-pad the following one, as shown in Figure 5.1. This allows to maintain continuity between the computation of two consecutive audio buffers. It is meant to be used as a replacement for left-padding during inference, retaining the original padding increase in dimensionality without creating discontinuities in-between buffers. Although this method provides a solution for the use of CNN in real-time audio generation, it is constrained by the

[1] We refer to this as *post-training causal reconfiguration* in opposition to training a causal model, i.e. that uses *pre-training causal configuration*.

[2] `https://acids-ircam.github.io/cached_conv`

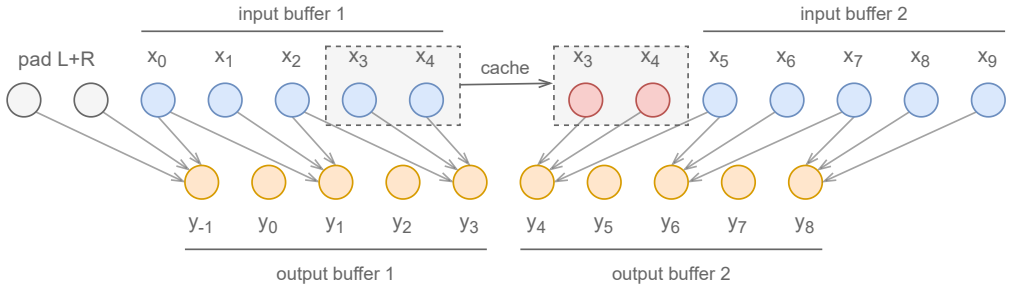


Figure 5.1: Convolution applied on two split buffers using cached padding. Input and output buffers $(\mathbf{x}, \mathbf{y})$ respectively live in $\mathbb{R}^{M \times T}$ and $\mathbb{R}^{N \times T}$. The last P frames from input buffer 1 are cached and concatenated with the input buffer 2 (with P being the original amount of zero-padding) in order to prevent discontinuities between buffers.

strict necessity of using only causal convolutions, which are not widespread. As most CNN in the literature do not satisfy this assumption, it is uncertain if this method can be widely applied to previous architectures or pre-trained models. Finally, it has been shown that a causal constraint makes the learning process more complex [87], which can impact the final audio quality

5.1.2 Non-causal streaming models

The streaming models obtained from the method described in Section 5.1.1 can readily process live audio streams. However, this requires models that use only causal convolutions, which is not the case for most models proposed in the literature. Indeed, training a model causally can lead to a loss of accuracy or audio quality [87].

Here, we introduce our method that allows to make *non-causal* models streamable. Our proposal is constructed around the idea of performing a post-training causal reconfiguration of the model. This allows to consider convolutional networks trained using any type of padding (potentially non-causal) and turn them into streamable models. One idea to do so would be to extend the cached padding mechanism to right-padding. However, this is not possible by nature, as we are processing live audio streams where the next buffer is not known yet.

Therefore, we propose to reconfigure the model as causal *after training*. This can be achieved by transforming right-padding into an additional left-padding. While this reconfiguration allows the use of a cached padding mechanism, making the model causal after training alters its computational graph.

Hence, this might produce unpredictable results if the model includes strided convolutions or has a computational graph with parallel branches (e.g residual connections [79]). In those cases, we propose the introduction of *additional delays* to restore the original behavior of the model. In the following, we detail how we address each of these architectures, in order for our method to be applicable to any type of network.

Aligning strided convolutions

Strided convolutions are often used as a way to reduce the temporal or spatial dimensionality of an input tensor. This is done by skipping some steps in the application of the convoluted kernel, as depicted in Figure 5.2.

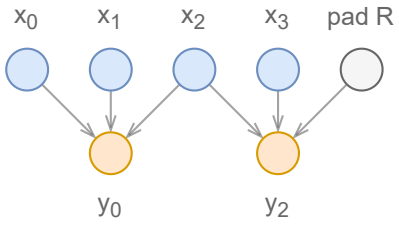


Figure 5.2: A simplified view of a strided convolution using zero-padding during training.

Transforming right-padding to left-padding shifts the input tensor to the right (i.e adds a lag to the input tensor). This has no consequence for convolutions with stride 1 or transposed convolutions as it only delays the output tensor. However, this lag may have an impact on convolutions with a stride greater than one, where a lag of n samples on the input tensor results in a fractional lag of n/s in the output tensor. We show in Figure 5.3 how this fractional lag results changes the behavior of the layer whenever n is not a multiple of s. Therefore, we introduce an additional delay to the input in order to make its overall lag a multiple of the stride, as shown in Figure 5.4.

In the case of a complex convolutional network, it is necessary to keep track of the overall cumulative lag for an input tensor after each convolutional layer. Considering that a convolutional layer with stride S and right-pad length R processes an input tensor with cumulative delay D_c, we need to set the additional delay D_a to

$$D_a = S - (R + D_c \mod S) \mod S, \tag{5.1}$$

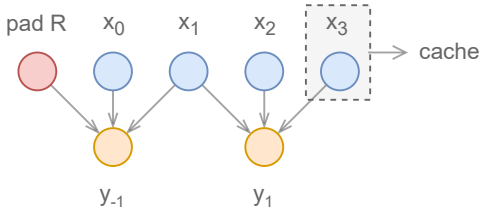


Figure 5.3: A strided convolution with post-training causal re-configuration. Due to the input lag, the output of the layer is not the same as during training (see Figure 5.2 for the regular output).

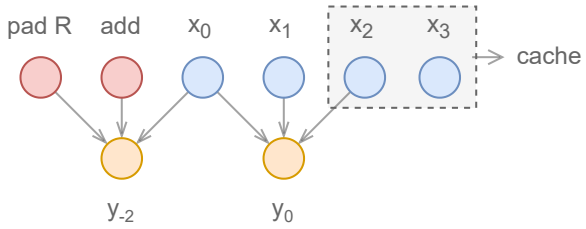


Figure 5.4: An additional delay (*add*) is applied to the input tensor in order to recover the original behavior of the layer.

where the cumulative delay D_c is set to 0 at the beginning of the graph, and is updated at each layer according to

$$D_c \leftarrow \frac{D_c + R + D_a}{S}. \tag{5.2}$$

Aligning parallel branches

When designing neural networks, it has been shown that using residual or skip connections facilitate the training of deeper models [79, 3, 88]. Those connections are responsible for the creation of parallel branches in the computational graph of the model, which are later aggregated (e.g. summed or concatenated) in order to produce an output. However, the delays introduced by a post-training causal re-configuration of the model might result in a relative misalignment between the outputs of each parallel branch. Therefore, models implementing parallel branches must introduce additional delays in order to recover a proper alignment between the different branches, as shown in Figure 5.5. In this case, we set the additional delays A_i to

$$A_i = \max_j D_j - D_i, \tag{5.3}$$

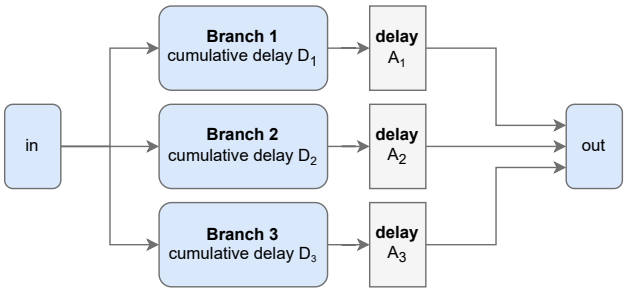


Figure 5.5: Aligning parallel branches using additional delays.

where D_i is the cumulative delay of the i^{th} branch.

Overlap-add baseline

For comparison purposes, we use a simple yet effective baseline method to process live audio streams with non-causal convolutional neural networks. We implement the *overlap-add* method by first collecting an audio buffer large enough to account for the receptive field of the model. Then, we apply the unmodified convolutional neural network on this buffer and window the output signal using the Hann window

$$\mathbf{w}[n] = \sin\left(\frac{\pi n}{N}\right)^2,$$

where N is the buffer size.

Finally, we add the resulting tensor to the previous output with a temporal offset of $N/2$. This implements the overlap-add method with a 50% overlapping factor. We compare this method to another having a 25% overlapping ratio, implemented by scaling w accordingly, as depicted in Figure 5.6. This reduces the computational redundancy of the method and consequently makes it process audio faster. However, using a smaller overlapping window results in harsher transitions between buffers. Hence, we also consider the extreme case of a 0% overlapping factor, where the model is applied on non-overlapping buffers. This last configuration can be seen as an ablation of our method where cached padding and causal constraints are removed.

(a) 50% overlap (b) 25% overlap (c) 0% overlap

Figure 5.6: Windows used by the three overlap-add baseline variants.

5.2 Evaluation

In order to evaluate our method, we apply it on the RAVE model [76] by replacing the convolutions in the regular model with *cached convolutions*[3]. No other changes have been made to the model. As we are evaluating our method on an audio synthesis task, we are interested in its computational efficiency compared to baselines, which we cover in Section 5.2.1. We evaluate the impact of using causal convolutions on the synthesized audio in Section 5.2.2 and 5.2.3. Finally, we verify in Section 5.2.4 that our method does not change the behavior of the model by comparing audio signals synthesized by a non-causal RAVE both in regular and streaming mode.

5.2.1 Performances

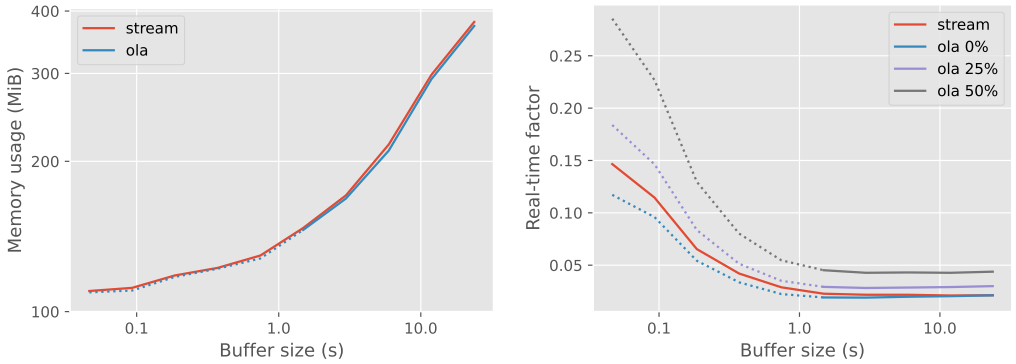
In this section, we evaluate the performances of our proposed streaming method when applied to a non-causal RAVE model compared to different variants of the overlap-add method. Here, we use randomly-initialized models as their parameters values do not affect their computational complexity. We keep the default hyperparameters of the original article.

In order to evaluate the inference speed, we rely on the RTF defined as the ratio between processing time and audio duration when performing an encode-decode pass on an audio signal. A RTF below 1 indicates that the algorithm processes data *faster* than real-time. We also evaluate the amount of memory required during inference on live audio streams, by analyzing the Random Access Memory (RAM) usage[4]. We estimate both memory usage and RTF of the reconstruction process using the various methods applied to 60s long random (white noise) audio signals with varying buffer sizes. We rely on white noise as here the audio output is not relevant to compute the speed of different methods. All results are averaged over 10 trials in order to account for measurement errors.

[3]We release the code for cached convolutions here `https://acids-ircam.github.io/cached_conv/`

[4]We use the `mprof` Python package for memory usage estimation.

We show in Figure 5.7(a) how our proposed *streaming* and different *overlap-add* methods all have a similar memory usage. The only difference comes from a constant 180kiB of additional RAM needed to store the cached padding of the streaming method.



(a) Memory usage

(b) Realtime factor

Figure 5.7: Memory usage and real-time factor for the *streaming* and *overlap-add* methods on a regular RAVE model with varying buffer size. Memory usage is identical for all overlap-add methods. Dotted lines indicate that the model is applied on buffers smaller than its receptive field.

In terms of processing speed, as we can see in Figure 5.7(b), the overlap method with a 0% overlap ratio is the fastest, while also being the less accurate (see Section 5.2.4). Although increasing the overlap ratio to 25% or 50% can reduce the corresponding artifacts, it also makes the overlap method increasingly slower than the streaming method. This is due to the computational redundancies involved in this method.

5.2.2 Processing latency

Introducing delays in the computational graph of a model has an impact on its processing latency. Causal models have the lowest latency, as they only rely on past audio samples when generating an output. Oppositely, non-causal models also leverage future audio samples which, in the context of live audio processing, are not available *yet*. We call the area of the receptive field of the model leveraging future samples its *future receptive field*. While our method allows the use of non-causal models on live audio streams, it also implies a processing latency equivalent to the duration of the future receptive field. In the case of the RAVE model, this latency adds up to

$\sim$ 600ms compared to only $\sim$ 50ms when using RAVE trained with a causal constraint.

5.2.3 Impact of pre-training causal constraint

As discussed in Section 5.1.1, enforcing a causal constraint on the model prior to its training can complexify the modelling task. We evaluate the impact of this constraint on the RAVE model with the following internal datasets:

Darbuka. It has been shown that modelling percussive sounds using a causal model can be difficult [89]. Therefore, we rely on a dataset composed of various solo darbuka performances sampled at 44.1kHz, with a total duration of approximately 3 hours.

Strings. This dataset contains approximately 30 hours of various strings recordings sampled at 44.1kHz that were scraped from different real-life solo violin performances. Compared to the darbuka, it is composed of harmonic signals with smoother attacks.

Speech. The speech dataset is composed of approximately 8 hours of recordings sampled at 44.1kHz. All recordings are produced by a single speaker in a consistent acoustic environment.

All datasets are split into 90%-10% train and validation sets. We use all the augmentation strategies proposed in the original article [76]. We train two variants of the RAVE model for each dataset (pre-training and post-training causal re-configuration). All models are trained for 2M iterations. In order to measure the reconstruction error of audio samples from the validation set as input for a pretrained RAVE model, we use the following spectral distance

$$\mathcal{L}_s(\mathbf{x}, \mathbf{y}) = \| \log(S(\mathbf{x}) + \epsilon) - \log(S(\mathbf{y}) + \epsilon) \|_2, \tag{5.4}$$

where $S(\mathbf{x})$ is an amplitude Short-Term Fourier Transform with a window of size 2048 and an overlap ratio of 75%. We set $\epsilon = 1$ as proposed by Défossez et al. [65]. We report the means and standard deviations of the resulting spectral distances in Table 5.1.

Using the pre-training causal configuration results in a small but consistent loss of accuracy as compared to the regular training of models across all datasets. This confirms that using causal networks in the context of neural audio synthesis complexifies the learning process. We hypothesize that this loss of precision is due to the fact that anticipating a transition or articulation in the sound is made more difficult when the model cannot look ahead.

	PRE-TRAINING	POST-TRAINING
DARBUKA	0.228 ± 0.028	$\mathbf{0.178 \pm 0.038}$
STRINGS	0.055 ± 0.012	$\mathbf{0.054 \pm 0.011}$
SPEECH	0.155 ± 0.005	$\mathbf{0.138 \pm 0.005}$

Table 5.1: Reconstruction errors for pre-training and post-training causal reconfiguration across different datasets.

5.2.4 Fidelity

In contrast to our proposed streaming method, the *overlap-add* approach only yields an approximation of the original model. Hence, we aim to estimate the quality of this approximation by comparing signals coming from the overlap-add method with signals processed offline by a non-causal model. In addition to the spectral distance defined in Section 5.2.3, we use the following metric

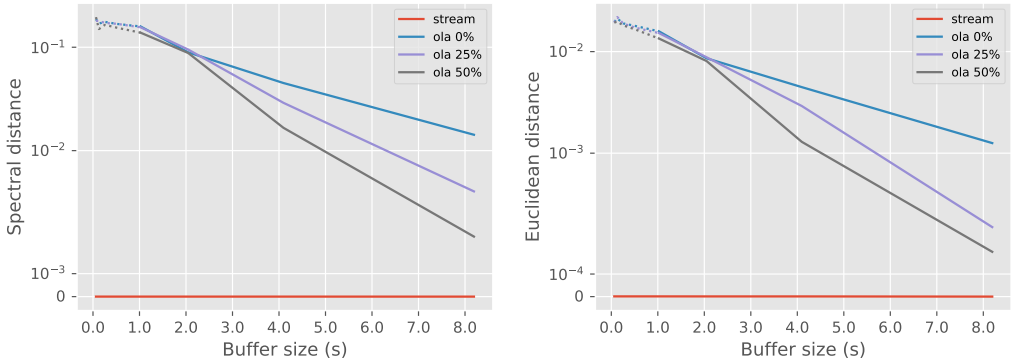
$$\mathcal{L}_w(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2, \tag{5.5}$$

where $\mathcal{L}_w$ is the Euclidean distance between two raw waveforms. While the spectral distance is useful to assess how perceptually similar two audio signals are regardless of their phase, the waveform Euclidean distance is highly phase-dependent, and reflects a sample-wise dissimilarity between the raw waveforms. Combined, those two metrics give us insights about how similar signals are both from a perceptual and sample-wise point of view.

We disable the noise synthesizer and set the encoder variance to 0 in order to make the model behave predictably. This is necessary as any randomness involved in the generation process would bias the fidelity measure.

We compare the *overlap-add* methods with several overlapping ratios (0%, 25% and 50%), and also include our *streaming* method to ensure that it is an exact reproduction of the offline method. We compensate the latency present in the synthesized outputs for all methods prior to their evaluation. We use the three pre-trained non-causal RAVE models described in Section 5.2.3 and compute an average of both the spectral and Euclidean distances over all corresponding validation sets. We report the results for all methods with varying buffer sizes in Figure 5.8.

As we can see, all variants of overlap-add methods have a decreasing spectral and Euclidean distances to the offline method as the buffer size increases. However, those distances never become null even for buffer sizes larger than 8s, stressing out the artifacts introduced by such methods. Oppositely, our streaming method is exactly identical to the offline method, regardless of the buffer sizes. This confirms that the cached padding and post-training causal



(a) Spectral distance

(b) Euclidean distance

Figure 5.8: Spectral and Euclidean distances between different *overlap-add* processing methods (ola) and the offline processing method as a function of the buffer size. Dotted lines indicate that the model is applied on buffers smaller than its receptive field.

reconfiguration of the model allow its use on live audio streams without altering the quality of the output.

5.3 Realtime interfaces

The main deep learning framework used during this thesis is *PyTorch* [90], a Python library for both the design and accelerated training of deep neural networks. While the development of Python scripts for real-time audio processing is possible through the use of dedicated libraries, their integration inside existing DAW is tedious and, thus, not widespread. Therefore, we leverage the *LibTorch* library allowing to import pretrained PyTorch model and use them inside a C++ runtime.

Exporting a PyTorch model into a format compatible with LibTorch can be achieved through either *scripting* or *tracing*. Since the latter involves the export of a static computational graph, we systematically choose the former method as the cached padding operation described in Section 5.1.1 requires the modification of in-memory buffers in-between DSP calls. Once the model is *scripted*, it is saved as a *torchscript* model, which is a self-contained file including the computational graph of the model alongside its pretrained parameters. Models exported for realtime usage must be streamable to avoid inconsistencies or artifacts during inference.

In order to enable the use of *torchscript* models on realtime audio streams, we develop a processing backend and several DAW specific frontends as shown in Figure 5.9.

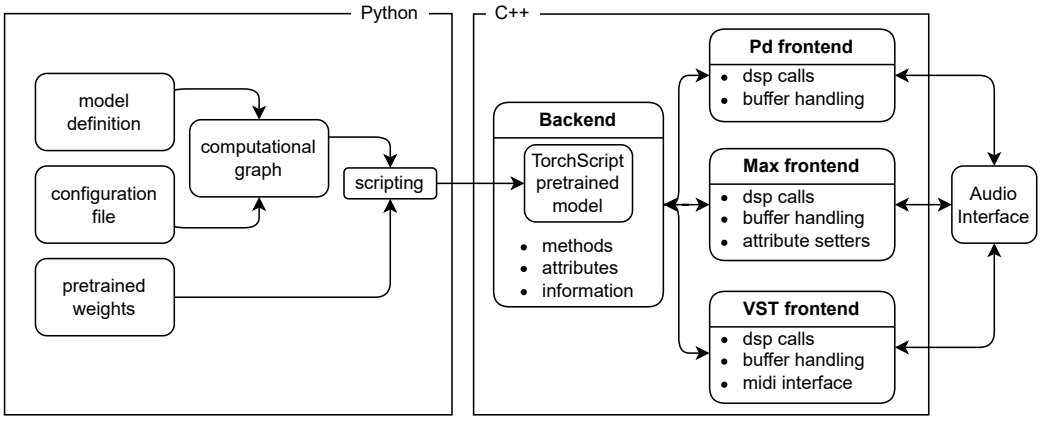


Figure 5.9: Block diagram depiction of our approach to realtime inference with pretrained PyTorch models.

The purpose of the backend is to translate standard DSP calls into model calls. For example, we consider an audio buffer containing N float samples supposed to be processed by the *forward* method of the model. To perform

this operation, the backend converts the buffer into a LibTorch tensor which is casted to a type compatible with the model (usually `float32`), performs check on the dimensionality of the tensor and finally calls the forward method of the model on the tensor. The backend then returns the output of the model converted to a standard C++ `float` table. Depending on the frontend used, the backend can also dynamically configure the model, by modifying some of its attributes. This is done in a separate thread to avoid interfering with the audio processing.

We develop several frontends to allow the use of our backend inside several applications. Our main application is called nn~, standing for *neural network* combined with *tilde* which is a standard for Max/MSP and PureData externals indicating audio signal related objects. The nn~ external is intended to provide a model-agnostic interface to both Max/MSP and PureData, allowing fast interaction prototyping with streaming models. Its features, behavior and use are completely defined by the model being used. Therefore, we present in the following sections a basic presentation of Max/MSP concepts alongside several use cases of nn~ being used in combination with RAVE and temporal models. The nn~ external is available at `github.com/acids-ircam/nn_tilde`.

5.3.1 Reactive programming using Max/MSP

Max/MSP is a visual programming language and graphical user interface used for music and audio processing. It is commonly used by musicians, sound designers, and researchers working with audio, and it is known for its flexibility and ability to create complex and dynamic systems. In Max/MSP, programs are created by connecting together different modules, or *objects*, using patch cords. These objects can represent a wide range of functions, from simple mathematical operations to more complex audio processing units. Instantiating such objects can be achieved by calling the name of a specific function and defining a number of additional arguments (e.g. in Figure 5.10, an oscillator is created by calling the object `cycle` and specifying its frequency as an argument). Max/MSP is often used in combination with other software, such as software synthesizers, to create dynamic and interactive audio environments.

We choose to develop Max/MSP and PureData externals as their arbitrary number of inlets and outlets allows the use of models with various number of inputs and outputs. Furthermore, the compatibility of PureData with embedded systems (e.g. Raspberry Pi) paves the way towards easy deployment of deep learning based DSP units inside hardware synthesizers.

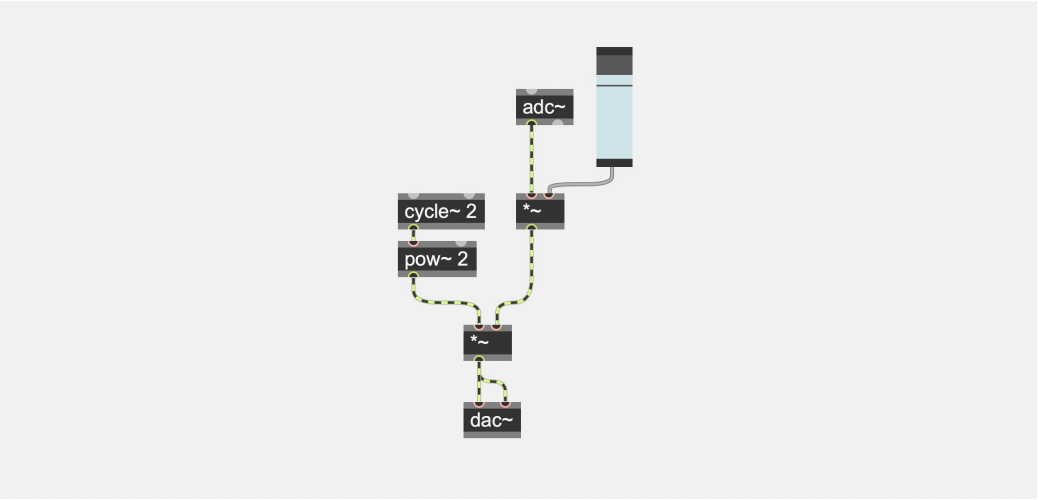


Figure 5.10: Example of a Max/MSP patch applying a linear gain on an input (`adc`), then modulating it with a low frequency oscillator (`cycle`) before being played back (`dac`).

5.3.2 Using the nn~ external

In this Section, we present use cases of nn~ using Max/MSP. However, most features showcased here are compatible with the PureData version. At the time of writing this document, only CPU and CUDA computing are available. Other computing backends might be supported in the future, such as Metal Perfomance Shader (MPS) or Vulkan.

Timbre transfer

We start by presenting the use of RAVE with nn~ in the context of the *timbre transfer* task (see Section 3.1.4). As a reminder, timbre transfer can be achieved by using RAVE to reconstruct out of domain audio signals, i.e. different from what the model has been previously trained on. Using nn~, this can be set up by instantiating an nn~ object with the path of a pretrained RAVE model (in Figure 5.11, the path is '*rave*'). The second argument is the *method* to call, defaulting to *forward*. The availability of different methods depends on the kind of model being imported. In the case of RAVE, the forward method is the combination of encoding an audio signal and reconstructing it.

Using the resulting object on a live audio stream can be achieved by connecting an audio source to the object first inlet and an audio output to its first outlet. The size of the internal buffer can be set using a third positional

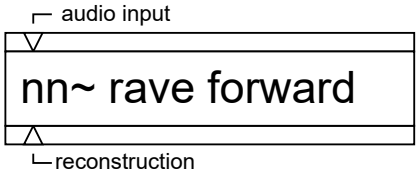


Figure 5.11: An instance of nn~ loading a pretrained RAVE model (here called with the argument *rave*) with its *forward method*. The first inlet is the *audio input* of the model, while the first outlet is the *reconstruction* of the input by the model.

argument, allowing a trade-off between latency and computational requirements.

Latent exploration

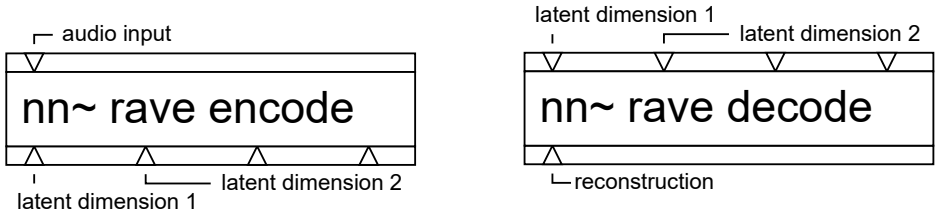
A given pretrained model may have multiple available methods, for example calling different sub-models or addressing a different task. We give the possibility to the user to select a specific method to use when importing a pretrained model to nn~. We show in Figure 5.12 how this can be used to either encode or decode signals using RAVE.

Sometimes, the sampling rate of a particular model is not the audio rate used by Max. This is the case for the latent representation yielded by the RAVE encoder, which is approximately 20Hz. Such low sampling rate signals are therefore upsampled to the audio rate by nn~ using nearest neighbor interpolation. Similarly, the input to the decoder is decimated to the correct rate before being fed to the model. The different sampling rates for each method can be defined during the model export, using a Python library included in the nn~ project.

Unconditional generation and continuation

A final example use case of nn~ is its application to *unconditional generation*, as described in Chapter 4. In this context, we combine both the encoder and decoder of a pretrained RAVE model alongside a prior model trained to approximate the aggregate posterior distribution of the encoder.

The Q inlets of nn~ with the prior model correspond to the Q different discrete dimensions yielded by the encoder model, and can be used to build a temporal context using the ground truth latent representation. The last two outlets of the object outputs accuracy and perplexity metrics of the model



(a) RAVE encoder

(b) RAVE decoder

Figure 5.12: Instances of nn~ loading a pretrained RAVE model with both *encode* and *decode* methods. The latent dimensions are upsampled to the audio rate using nearest neighbor interpolation.

given the ground truth input as shown in Figure 5.13. When the attribute `listen` of the model is set to True, the first Q outputs of nn~ replicate its Q inputs. The model is being used in a *teacher forcing* setup.

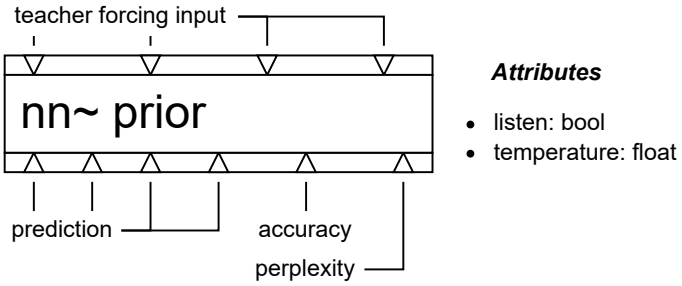


Figure 5.13: An instance of nn~ loading a pretrained temporal model with its default *forward method*. Depending on the inference mode (i.e. listen being set to True of False), the last two outlets output information about both the accuracy (listen mode only) and perplexity of the model.

Setting the attribute `listen` to False modifies the behavior of the model, which switches to *autoregressive prediction*. When using this mode, inputs are ignored, and the model autoregressively predicts the temporal evolution of the latent representation. Switching to this mode while feeding a ground truth latent representation to the model allows to perform *latent continuation*, while using the prior model in headless mode (i.e. without encoder attached) results in *unconditional generation*. The `temperature` attribute can be set to define the sampling temperature used in the generation loop.

There's something to be said about all music being some translation of our languaging, our way of communicating. It's a language. This is a new language.

Suzanne Ciani

Chapter 6

Artistic collaborations

Building tools in order to aid music creation is a complex task that requires both artistic and technical knowledge. During this thesis, we collaborate with professional artists in order to build and refine our research around the idea of creating production-ready models and tools. In this chapter, we present two collaborations conducted during this thesis, and describe artistic and research challenges that we had to overcome. Additionally, we present two creations proposed during this thesis that leverage the entirety of the techniques proposed in Chapters 3 to 5.

6.1 Alexander Schubert: Convergence

Figure 6.1: Still image from the streaming version of *Convergence*.

6.1.1 Technical aspects

Convergence is the result of a one-year-long scientific and artistic collaboration with the composer *Alexander Schubert*. It features an early version of the RAVE model called *WaVAE*. This model is the combination of a VAE trained on mel-scale spectrograms and a MelGAN model [28] inverting the spectrograms into sound. The latent representation yielded by the encoder is regularized using its KL divergence with a normal distribution. We use a small regularization factor $\beta = 0.001$ as a way to address the poor reconstruction abilities of the vanilla VAE model. This comes at the expense of degraded sampling performances.

In addition to the KL divergence, we further regularize the latent representation using the approach proposed in Fader Networks [70]. This approach involves conditioning the decoder of a VAE on labels present in the dataset, allowing the control of the generation process by manipulating those labels. They introduce an adversarial training strategy to minimize the mutual information between the labels and the latent representation using a latent discriminator, therefore allowing their independent control. We adapt their approach to condition WaVAE on continuous descriptors, more specifically using an estimation of the loudness of the signal. We use the method described in Section 4.1 to discretize the loudness, and use the latent discriminator to remove any information about it from the latent space. During generation, the resulting model can effectively be controlled through an explicit loudness parameter. However, this feature is absent from RAVE since this conditioning strategy tends to make the latent representation degenerate

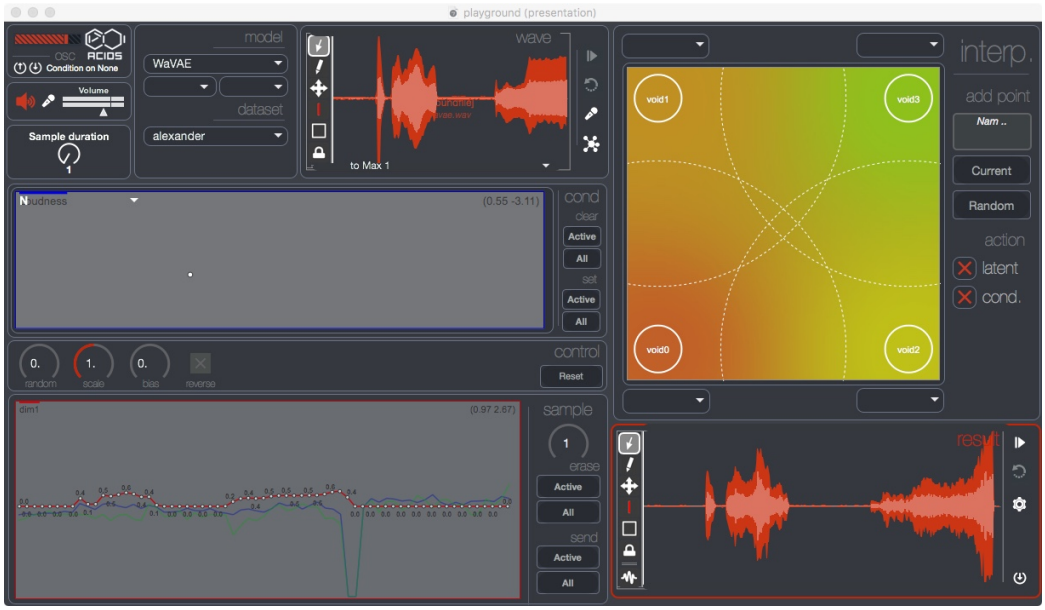


Figure 6.2: Graphical interface built around WaVAE, allowing high level yet offline interactions with pretrained models.

(i.e. the decoder relies *exclusively* on the conditioning signal to reconstruct the input).

6.1.2 Interface

While the WaVAE model served as a test-bed for the streaming method proposed in Chapter 5, its use on realtime streams was not available yet at the time of the collaboration. We therefore developed a graphical interface around the model to provide a more intuitive approach to neural audio synthesis, as shown in Figure 6.2. The implementation of the interactors present in the interface was made in close collaboration with the composer. This was a critical aspect in the development of the model, gradually integrating features (e.g. the loudness conditioning mechanism) during the creation process.

The limitations of both the WaVAE model and the graphical interface eventually led to the development of RAVE and nn~, later used by the composer for his piece ANIMA^{TM}. Nonetheless, *Convergence* was awarded with a Golden Nica in Digital Musics & Sound Art during the 2021 edition of the Ars Electronica Festival.

6.1.3 Program notes

Convergence uses the concept of Artificial Intelligence to learn features of human musicians and then recreate new entities based on these recordings. In the piece the players interact with their generated counter-parts. They see theirselves transform and reshape. The technology used is centered around VAE and GANs. Metaphorically they demonstrate a world that is constructed and parametric. The friction between machine perception and human world perception is the starting point for questions that address the fluidity of the self and the restrictions of perception. Human world and self models are parametric systems that make abstract assumptions and classifications of our surroundings. These processes happen party subconscious and unreflected. They give us the impression of an absolute truth or reality, as the constructed concepts, identities and beliefs are persuasive and internal. That these constructive models are fluid and subject to change is examined through the use of AI in this context. Auto-Encoders allow a formalization of the input data - in this case faces, bodies and voices. The deep learning yields a low-dimensional - abstract or high-level - description of the input. Opposed to our black box human mind the high level parameters in the algorithm are accessible and can be edited and transformed.

In this sense the AI systems allows to warp the representation of the human performers and thus stressing the fluidity of the modeling: A different person, character trait, evaluation or gender is far less substantially disparate as the subject would anticipate. The transformation of the parameters posses the character of (social, societal, clinical or biological) mind-altering states. The AI system is used to enable this altering with the aim to question the robustness and immobility of identity and world models. It tries to expose the internal constructiveness and in this sense works as a mirroring device. It recreates partial aspects of our perception and classification and through its alteration offers the viewer to draw a parallel to our own mental world building processes.

Alexander Schubert

6.2 Maxime Mantovani: Forme improvisée

Figure 6.3: From left to right, Victor Auffray, Maxime Mantovani and Henri-Charles Caget during a rehearsal session around nn~ and RAVE.

6.2.1 Technical aspects

A second artistic and scientific collaboration was conducted with the composer *Maxime Mantovani* during his artistic residency in our research laboratory. The main target for this collaboration was the study of the real-time interaction between artists and pretrained deep learning audio models. This residency started during the development of both RAVE and nn~, allowing the composer to use early versions of both projects as a base material for his work. Building RAVE while having concrete production expectations guided its development toward high-quality audio synthesis (using 48kHz signals instead of 24kHz), dynamic range fidelity (introduction of the loudness envelope generator in Section 3.1.3) and development of a causal variant yielding lower processing latency (see Section 5.1.1). The work of the composer on the creation of the *darbuka* dataset yielded precious insights about the base audio material needed to produce a usable RAVE model.

This project was also the perfect opportunity to test nn~ in a production environment. The feedback of the composer in between rehearsals helped with stabilizing the external and initiated the implementation of important features that a research environment failed to highlight (model bypassing, dynamic internal buffers, optional GPU computation).

An additional research axis included a control surface built by the composer (see Figure 6.4 for an early prototype of the controller), and was explored dur-

ing this collaboration. The objective was the learning of a mapping between latent trajectories yielded by RAVE and signals produced by the controller as a way to drive the generative process through intuitive gestures. The complexity of the task prevented its proper addressing during the time of the collaboration, and the control surface was used in a learning-free context.

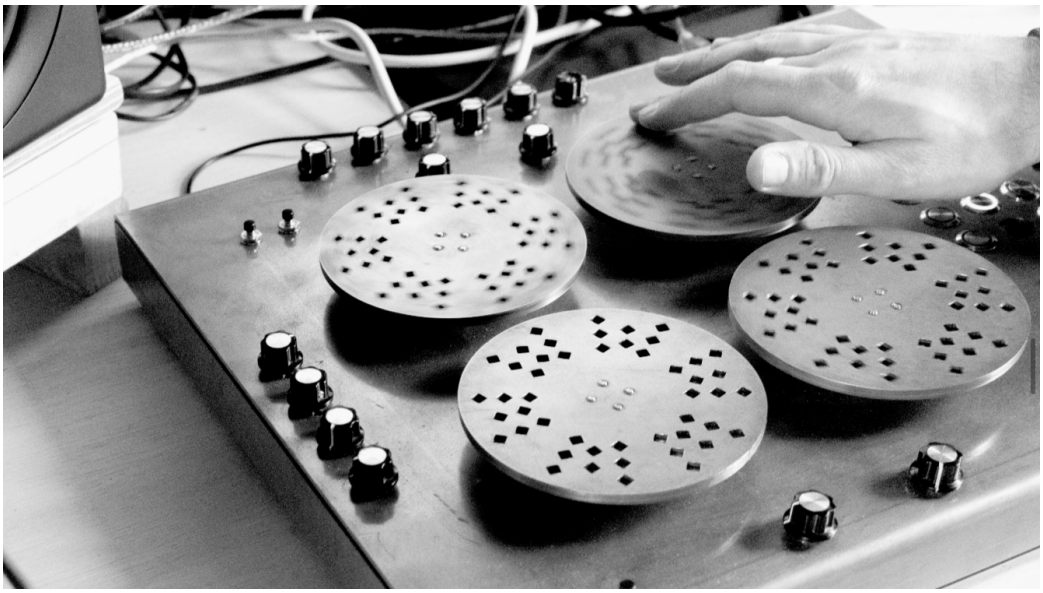


Figure 6.4: Disk-based controller built by Maxime Mantovani, early prototype of the final interface developed during the piece.

6.2.2 Program notes

> As part of my art-research residency at IRCAM, 2021-2022, focusing on Artificial Intelligence systems and electronic instrument building, I invited Victor Auffray (tubist) and Henri-Charles Caget (percussionist) to participate in improvisations in order to confront these technologies with the reality of musical instantaneity, the point of encounter and the genesis of the improvisation proposed as a residency restitution. The presence of these new sound objects proposed by AI, and in particular by the RAVE model, allows for the creation of music that modifies our perceptions of electronic space, transforming it into a creative organism or an autonomous propositional force. Without falling into anthropomorphism that would shift the focus elsewhere, the generative model based on deep neural networks learning is a fantastic tool

that proposes surprising and touching musical solutions in terms of expressiveness.

Our rehearsal work mainly consisted of testing different timbre transfers and interconnections between my interfaces, the various models, and the musicians. We were thus able to identify and control nn~ reaction based on the sounds that were proposed as input. Clément Cannone, a musicologist at IRCAM specializing in the field of improvisation, defined this as a morphological transfer. Thanks to my controllers, I quickly have access to all modes of interconnection between input sources and the latent spaces of RAVE. The morphological transfer is then multiple: from musicians to latent space, from recorded musicians to a granular engine to latent space, from musicians to latent space to recording on the granular engine and then from AI to AI. I can quickly achieve "mises en abyme". The latency is remarkably short and dependent on Max's vector size, but it is still noticeable. To avoid the systematicity that this would impose, all these "mises en abyme" and interconnections of morphological transfer make us forget this state of question/answer, and we find ourselves in a situation of sound creation, construction of the musical space, observation of the playground and constant renewal of musical proposal. AI becomes a force of musical proposal. As interpreting musicians, the RAVE generative model succeeds in surprising us, and it becomes very pleasant to indulge in its imitation game. These systems are surprisingly expressive. I note the crucial importance of creating a very clean and precise dataset. The dataset is fundamental for the quality of the sound results afterwards. This is a problem that has been known since the beginning of recordings. The quality of the recording is essential to maintain the sound quality throughout all the transformations we subject them to. This project aims to connect the expressive vectors of instrumental performance, electronic-acoustic instrument building, and timbre transfer systems to create a more dynamic and impactful musical experience.

One's thoughts automatically turn to the creation of a ghost, an altered digital twin. A parallel sound world opens up. We are no longer at the level of offline work, no longer the stammering of the machine or the chattering that smooths listening over time. We are at the level of interconnected real-time. This allows to link reality and fiction into a surreal sonic space while we play music conceived on the spot. Augmented and interconnected space as

a medium — an Alter-Reality. In learning models, there is currently no notion of musical form; the machine generates, listens, and "simply" repeats. What remains after the fascination of discovery? How can we make it take into account what came before and propose what will come next? A learning model of musical form coupled with the generative model would be an incredible tool in an improvised form with a musician.

Maxime Mantovani

The artificial intelligence, responds instantly to the instrumental gesture and questions the performer in real-time, which is quite innovative in terms of acoustic playing sensations. It takes the expressiveness of each performer into a new dimension. With AI, we are, in all modesty, witnessing the beginnings of tomorrow's writing. It is time to start creating an organology of AI.

Henri-Charles Caget

6.3 Other pieces

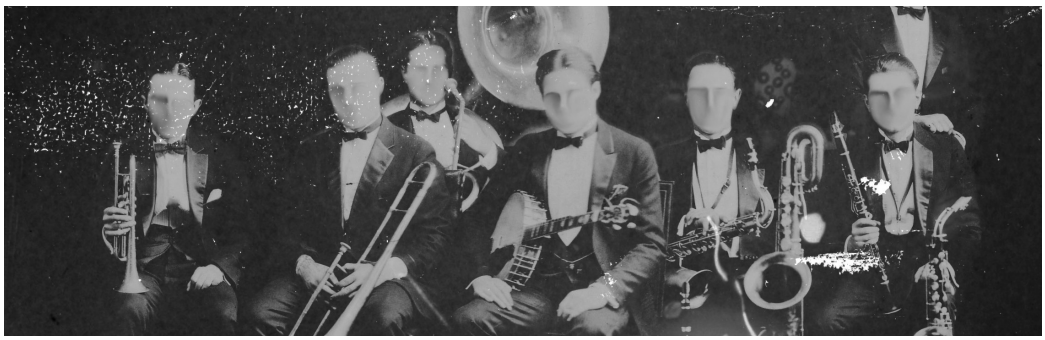


Figure 6.5: Visual of the vintage livestream.

6.3.1 Azimuth Conjunction in Declining State

Azimuth Conjunction in Declining State (ACIDS) is a 24mn long randomly generated piece[1] leveraging RAVE (see Section 3), an unconditional shifted convolutional prior (see Section 4.4.1) and the nn~ external (see Section 5.3). It uses the violin training featured in RAVE [76]. The sampling temperature is oscillating between 2 and 0.4 at a rate of 0.01Hz with random restarts.

The piece has been featured in the 2022 edition of the *MUSAiC festival*.

6.3.2 Vintage experiments

We conduct a set of musical experiments around the idea of using RAVE, a temporal model and nn~ on musical material from the past century. We leverage a large-scale dataset composed of digitized 70RPM disks from the archive.org website that we de-noise using the method described in [91]. The first experiment involved the live monitoring of the training of both a RAVE model and an unconditional shifted convolutional temporal model through a 1-week long livestream. We use a Max patch inspired from the one used for ACIDS, and reload new checkpoints of all models every hour. The livestream started using nearly untrained models, reaching a relative convergence at the end of the stream one week later. We show in Figure 6.5 a visual taken from the livestream.

Another experiment on the subject was led in collaboration with Axel Chemla–Romeu-Santos under the form of a 20mn long live piece called *We get what*

[1]The ACIDS piece is available at this address: `youtu.be/XXqQyeXZpl0`

we leave behind. This piece is a real-time AI triptych on oblivion, hauntology, real and false memories and artificial reveries. It was performed live in the *Cirque Electrique* in Paris, and involved the use of models trained on the previous dataset in addition to music from all around the world. It also involved the use of past technologies such as tape recorders on which the output of the generative models were continuously recorded and played back.

Chapter 7

Conclusion

Musical audio signals are complex to understand, as lots of different information scales are scattered around tens of thousands of individual samples. This makes the analysis and synthesis of such signals with deep learning approaches challenging. Early works introduce the use of large autoregressive models to properly model all scales at once, from fine-grained details to higher-level structure and style features. However, the computational complexity of those models result in expensive training procedures and prohibitively long sampling times. In this thesis, we study the use of hierarchical models to address the problem of music neural audio synthesis. The core idea behind hierarchical approaches is to model each scale of a data example separately, usually using individual sub-models trained on different modalities of this example. This allows to tailor each sub-model to a specific learning task, which eventually results in a smaller and more efficient system. Therefore, we split our approach into two main modules. The first one involves the creation of a representation learning model that extracts a high-level and compact representation of a complex audio dataset. This compact representation is then used as the base signal for our second approach, trained to learn its temporal evolution, either unconditionally or based on external acoustic descriptors. Finally, we introduce a set of techniques and applications to make both previous methods compatible with *real-time* processing, through the use of Digital Audio Workstations (DAWs) such as Max/MSP or PureData. All our methods and applications have been designed in collaboration with professional artists as a way to maintain focus on their usability in the context of creative applications. These collaborations resulted in a series of musical pieces integrating our methods in both an offline and real-time processing context.

Audio representation learning. First, we introduced the Realtime Audio Variational autoEncoder (RAVE) model, a multipurpose audio representation learning model analyzing and synthesizing high-quality raw audio signals 20 to 80 times faster than real-time on CPU. We combine two different classes of generative models, namely Variational Auto Encoders (VAEs) and Generative Adversarial Networks (GANs) through the use of a novel 2-stage training mechanism. Our method learns a *high-level* and *compact* representation extracted from the raw audio signal, while using adversarial training strategies to produce high-quality sounding synthesized sounds. To further compress the learned representation, we propose a post-training analysis of the latent space to identify *informative* and *non-informative* dimensions, and derive from this analysis a *fidelity* parameter. This parameter can be used to control the trade-off between reconstruction fidelity and representation compactness. From an application standpoint, RAVE can be used to perform timbre transfer and high-level feature manipulation. In addition to the usual VAE formulation, we experiment with different latent regularization strategies, including a Wasserstein regularization introduced in [72] and Residual Vector Quantization (RVQ) as proposed in [74]. Overall, these different regularization methods yield RAVE models with different application cases, with both the original and Wasserstein variants being suited to timbre transfer and high-level manipulation of an existing input audio signal, while we use the RVQ variant as the first building block of our higher-level hierarchical model.

Temporal learning. Our next proposal addresses the temporal learning of the representation yielded by RAVE with the RVQ regularization given a specific dataset. We draw inspiration from the recent advances in the Natural Language Processing (NLP) literature, which have provided impressive results in terms of temporal coherency. In contrast with the original VAE formulation which encodes a data point into a single multivariate latent point, the encoder we define in the RAVE model yields a *sequence* of multivariate latent points, that we call *latent trajectory*. Unfortunately, most of the approaches proposed in the NLP literature involve univariate sequences. The current method used to address multivariate sequence modeling tasks involves *flattening* the sequence. In practice, this implies interleaving all the dimensions of the original multivariate sequence into a longer univariate sequence. This method is computationally inefficient as it is introducing an artificial temporal complexity, implying longer training times. Furthermore, in the case of autoregressive generative models, synthesis time is inevitably multiplied by the number of dimensions, which can be a problem for real-

time applications. Therefore, we introduce several extensions to the usual NLP methods to the multivariate sequence distribution estimation task. In practice, we introduce three methods called *decoupling*, *residual* and *shift* which are all able to process multivariate sequences without the aforementioned flattening preprocessing stage. We show that both the *residual* and *decoupling* methods yield higher prediction scores, while being significantly faster to train and to use in a generation setup. In terms of computational efficiency, all our methods are faster than real-time, which is not the case for the flattening baseline. Overall, combining the prediction of our autoregressive temporal model with the decoder from a pretrained RAVE model can be used for autonomous music creation, audio continuation, or descriptor conditioned generation.

Real-time inference and artistic collaborations. While some kinds of deep learning model (e.g. recurrent models) are naturally fit to the *streaming* task (i.e. process live audio streams instead of audio files), we have shown that this is not the case by default for non-causal Convolutional Neural Networks (CNNs). As many recent advances in audio modeling leverage CNNs at the core of their architecture, we propose a set of techniques enabling their use in a *streaming* setup. In contrast with previous methods relying on causal convolutions during training to obtain streamable models, we propose a post-training causal re-configuration of the model to allow any kind of CNN based model to operate on audio streams.

Additionally, we introduce the nn~ external, which provides an interface between the *PyTorch* deep learning library and both Max/MSP and PureData. Combining nn~ with a pretrained model (e.g. RAVE or any model compliant with the nn~ API), we allow artists to perform deep learning based audio processing and synthesis in real-time, without leaving their usual creative framework. Applying those techniques on the RAVE model, we effectively propose the first convolutional-based deep learning generative model integrated in a real-time environment.

We benefit from several collaborations with professional artists to define and improve all our proposed methods and tools. This allowed the introduction of high-quality audio constraints (i.e. high sampling rate, low latency), and helped stabilize the nn~ external, which is particularly important for its compatibility with a production setup. Overall, the combination of the RAVE model with our real-time applications has been used in several artistic pieces around the world.

Future works. Firstly, an interesting development to the hierarchical approach proposed in this thesis can be found in the use of the *semantic* representation introduced in [78] through the use of a self-supervised representation learning model (see Section 2.4.2). Indeed, such a high-level representation used in the context of a generative model yields impressive applications in terms of audio continuations, while reaching an unprecedented temporal coherency during generation. However, the nature of the information encoded in the semantic tokens is difficult to interpret, and even harder to interact with. Therefore, performing an explorative study of this kind of representation to better understand its content might be helpful to design more efficient generative strategies, and avoid redundancies in the different levels of generation.

Secondly, even though the representations extracted by RAVE are significantly more compact than their corresponding audio signals, the yielded trajectories are still living in a latent space whose dimensionality ranges in practice from 4 to 32 dimensions. This high dimensionality makes it hard to build an intuition of the implicit structure learned by the model, and thus might hamper the controllability of the resulting system. Since reducing the number of latent dimensions will inevitably decrease the reconstruction abilities of the model, an alternative solution to this problem might come from the studying of adapted visualization and control methods. More specifically, using the distribution yielded by a temporal model on the latent trajectories of a RAVE model in conjunction with methods inspired from entropy coding might help build a dynamically adapting representation of smaller dimensionality. Indeed, the use of the predictions made by the temporal model would enable the identification of factors of variation that are most likely to occur, drastically simplifying the overall control space of the model.

Lastly, recent advances in Reinforcement Learning offers an alternative framework to the methods presented in this thesis, as they rely on a non-differentiated *reward* function to learn high-level tasks. This could enable the integration of human evaluation in the training loop, increasing the level of customization an artist can have over a model.

Bibliography

[1] Alexandre Défossez, Nicolas Usunier, Léon Bottou, and Francis Bach. Demucs: Deep Extractor for Music Sources with extra unlabeled data remixed. 9 2019.

[2] Josh Gardner, Ian Simon, Ethan Manilow, Curtis Hawthorne, and Jesse Engel. MT3: Multi-Task Multitrack Music Transcription. 11 2021.

[3] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. WaveNet: A Generative Model for Raw Audio. 9 2016.

[4] Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. Jukebox: A Generative Model for Music. *arXiv*, 4 2020.

[5] S. S. Stevens, J. Volkmann, and E. B. Newman. A Scale for the Measurement of the Psychological Magnitude Pitch. *The Journal of the Acoustical Society of America*, 8(3):185, 6 2005.

[6] Truong Q. Nguyen. Near-Perfect-Reconstruction Pseudo-QMF Banks. *IEEE Transactions on Signal Processing*, 42(1):65–76, 1994.

[7] M. Rossi, Jin-Yun Zhang, and W. Steenaart. A new algorithm for designing prototype filters for M-band Pseudo QMF banks, 1996.

[8] Yuan Pei Lin and P. P. Vaidyanathan. A kaiser window approach for the design of prototype filters of cosine modulated filterbanks. *IEEE Signal Processing Letters*, 5(6):132–134, 1998.

[9] Haskell B Curry and frankord Arsenal. The method of steepest descent for non-linear minimization problems. *Quarterly of Applied Mathematics*, 2(3):258–261, 1944.

[10] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, 2015.

[11] HornikK., StinchcombeM., and WhiteH. Multilayer feedforward networks are universal approximators. *Neural Networks*, 7 1989.

[12] Yann Lecun and Yoshua Bengio. Convolutional networks for images, speech, and time-series. In Michael Arbib, editor, *The handbook of brain theory and neural networks*. MIT Press, 1995.

[13] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8), 1997.

[14] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. 9 2014.

[15] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning Long-Term Dependencies with Gradient Descent is Difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[16] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *32nd International Conference on Machine Learning, ICML 2015*, 1:448–456, 2 2015.

[17] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization. 7 2016.

[18] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance Normalization: The Missing Ingredient for Fast Stylization. 7 2016.

[19] Tim Salimans and Diederik P. Kingma. Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks. *Advances in Neural Information Processing Systems*, pages 901–909, 2 2016.

[20] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral Normalization for Generative Adversarial Networks. *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 2 2018.

[21] Ricky T.Q. Chen, Jens Behrmann, David Duvenaud, and Jörn Henrik Jacobsen. Residual Flows for Invertible Generative Modeling. *Advances in Neural Information Processing Systems*, 32, 6 2019.

[22] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 2017-December, pages 5999–6009. Neural information processing systems foundation, 6 2017.

[24] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, 12 2014.

[25] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. *Communications of the ACM*, 63(11):139–144, 6 2014.

[26] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework, 11 2016.

[27] Yann LeCun and Corinna Cortes. The mnist database of handwritten digits. 2005.

[28] Kundan Kumar, Rithesh Kumar, Thibault de Boissiere, Lucas Gestin, Wei Zhen Teoh, Jose Sotelo, Alexandre de Brebisson, Yoshua Bengio, and Aaron Courville. MelGAN: Generative Adversarial Networks for Conditional Waveform Synthesis. *arXiv*, 10 2019.

[29] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. *arXiv*, 1 2017.

[30] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved Training of Wasserstein GANs. *Advances in Neural Information Processing Systems*, 2017-December:5768–5778, 3 2017.

[31] Tero Karras, Samuli Laine, and Timo Aila. A Style-Based Generator Architecture for Generative Adversarial Networks. 12 2018.

[32] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation, 10 2017.

[33] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and Improving the Image Quality of StyleGAN. 12 2019.

[34] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using Real NVP. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, 5 2016.

[35] Diederik P. Kingma and Prafulla Dhariwal. Glow: Generative Flow with Invertible 1x1 Convolutions. *Advances in Neural Information Processing Systems*, 2018-December:10215–10224, 7 2018.

[36] Jiahui Yu, Yuanzhong Xu, Jing Yu Koh, Thang Luong, Gunjan Baid, Zirui Wang, Vijay Vasudevan, Alexander Ku, Yinfei Yang, Burcu Karagol Ayan, Ben Hutchinson, Wei Han, Zarana Parekh, Xin Li, Han Zhang, Jason Baldridge, and Yonghui Wu. Scaling Autoregressive Models for Content-Rich Text-to-Image Generation. 6 2022.

[37] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating Long Sequences with Sparse Transformers. 4 2019.

[38] Curtis Hawthorne, Andrew Jaegle, Cătălina Cangea, Sebastian Borgeaud, Charlie Nash, Mateusz Malinowski, Sander Dieleman, Oriol Vinyals, Matthew Botvinick, Ian Simon, Hannah Sheahan, Neil Zeghidour, Jean-Baptiste Alayrac, João Carreira, and Jesse Engel. General-purpose, long-context autoregressive modeling with Perceiver AR. 2 2022.

[39] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking Attention with Performers. 9 2020.

[40] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced Transformer with Rotary Position Embedding. 4 2021.

[41] Ofir Press, Noah A. Smith, and Mike Lewis. Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation. 8 2021.

[42] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew M. Dai, Matthew D. Hoffman, Monica Dinculescu, and Douglas Eck. Music Transformer. 9 2018.

[43] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. 1 2013.

[44] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. 10 2018.

[45] Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations. *Advances in Neural Information Processing Systems*, 2020-December, 6 2020.

[46] Yu An Chung, Yu Zhang, Wei Han, Chung Cheng Chiu, James Qin, Ruoming Pang, and Yonghui Wu. W2v-BERT: Combining Contrastive Learning and Masked Language Modeling for Self-Supervised Speech Pre-Training. *2021 IEEE Automatic Speech Recognition and Understanding Workshop, ASRU 2021 - Proceedings*, pages 244–250, 8 2021.

[47] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models. *arXiv*, 6 2020.

[48] Soroush Mehri, Kundan Kumar, Ishaan Gulrajani, Rithesh Kumar, Shubham Jain, Jose Sotelo, Aaron Courville, and Yoshua Bengio. Samplernn: An unconditional end-to-end neural audio generation model. In *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, 2017.

[49] Aaron Van Den Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George Van Den Driessche, Edward Lockhart, Luis C Cobo, Florian Stimberg, Norman Casagrande, Dominik Grewe, Seb Noury, Sander Dieleman, Erich Elsen, Nal Kalchbrenner, Heiga Zen, Alex Graves, Helen King, Tom Walters, Dan Belov, and Demis Hassabis. Parallel WaveNet: Fast High-Fidelity Speech Synthesis. Technical report, 7 2018.

[50] Diederik P. Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improving Variational Inference with Inverse Autoregressive Flow. 6 2016.

[51] Wei Ping, Kainan Peng, and Jitong Chen. ClariNet: Parallel Wave Generation in End-to-End Text-to-Speech. Technical report, 9 2018.

[52] Ryan Prenger, Rafael Valle, and Bryan Catanzaro. Waveglow: A Flow-based Generative Network for Speech Synthesis. In *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, volume 2019-May, pages 3617–3621. Institute of Electrical and Electronics Engineers Inc., 5 2019.

[53] Wei Ping, Kainan Peng, Kexin Zhao, and Zhao Song. WaveFlow: A Compact Flow-based Model for Raw Audio. Technical report, 11 2020.

[54] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked Autoregressive Flow for Density Estimation. 5 2017.

[55] Jungil Kong, Jaehyeon Kim, and Jaekyoung Bae. HiFi-GAN: Generative Adversarial Networks for Efficient and High Fidelity Speech Synthesis. *Advances in Neural Information Processing Systems*, 2020-December, 10 2020.

[56] Jesse Engel, Lamtharn Hantrakul, Chenjie Gu, and Adam Roberts. DDSP: Differentiable Digital Signal Processing. Technical report, 9 2019.

[57] Ryuichi Yamamoto, Eunwoo Song, and Jae Min Kim. Parallel Wavegan: A Fast Waveform Generation Model Based on Generative Adversarial Networks with Multi-Resolution Spectrogram. In *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, volume 2020-May, pages 6199–6203. Institute of Electrical and Electronics Engineers Inc., 5 2020.

[58] Philippe Esling, Axel Chemla-Romeu-Santos, and Adrien Bitton. Generative timbre spaces: regularizing variational auto-encoders with perceptual metrics. *DAFx 2018 - Proceedings: 21st International Conference on Digital Audio Effects*, pages 369–376, 5 2018.

[59] Daniel W. Griffin and Jae S. Lim. Signal Estimation from Modified Short-Time Fourier Transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 32(2):236–243, 1984.

[60] Sean Vasquez and Mike Lewis. MelNet: A Generative Model for Audio in the Frequency Domain. *arXiv*, 6 2019.

[61] Jesse Engel, Kumar Krishna Agrawal, Shuo Chen, Ishaan Gulrajani, Chris Donahue, and Adam Roberts. GANSynth: Adversarial Neural Audio Synthesis. *arXiv*, 2 2019.

[62] Philippe Esling, Naotake Masuda, Adrien Bardet, Romeo Despres, and Axel Chemla-Romeu-Santos. Universal audio synthesizer control with normalizing flows. 7 2019.

[63] Hao Fu, Chunyuan Li, Xiaodong Liu, Jianfeng Gao, Asli Celikyilmaz, and Lawrence Carin. Cyclical Annealing Schedule: A Simple Approach to Mitigating KL Vanishing. *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, 1:240–250, 3 2019.

[64] Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Mohammad Norouzi, Douglas Eck, and Karen Simonyan. Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders, 7 2017.

[65] Alexandre Défossez, Neil Zeghidour, Nicolas Usunier, Léon Bottou, and Francis Bach. SING: Symbol-to-Instrument Neural Generator. *Advances in Neural Information Processing Systems*, 2018-December:9041–9051, 10 2018.

[66] Takuhiro Kaneko and Hirokazu Kameoka. Cyclegan-VC: Non-parallel voice conversion using cycle-consistent adversarial networks. In *European Signal Processing Conference*, volume 2018-September, pages 2100–2104. European Signal Processing Conference, EUSIPCO, 11 2018.

[67] Pranay Manocha, Adam Finkelstein, Richard Zhang, Nicholas J. Bryan, Gautham J. Mysore, and Zeyu Jin. A Differentiable Perceptual Audio Metric Learned from Just Noticeable Differences. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, 2020-October:2852–2856, 1 2020.

[68] Xin Wang, Shinji Takaki, and Junichi Yamagishi. Neural source-filter waveform models for statistical parametric speech synthesis. *IEEE/ACM Transactions on Audio Speech and Language Processing*, 28:402–415, 4 2019.

[69] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. *33rd International Conference on Machine Learning, ICML 2016*, 4:2341–2349, 12 2015.

[70] Guillaume Lample, Neil Zeghidour, Nicolas Usunier, Antoine Bordes, Ludovic Denoyer, and Marc'aurelio Ranzato. Fader Networks: Manipulating Images by Sliding Attributes. *Advances in Neural Information Processing Systems*, 2017-December:5968–5977, 6 2017.

[71] Noam Mor, Lior Wolf, Adam Polyak, and Yaniv Taigman. A Universal Music Translation Network. *arXiv*, 5 2018.

[72] Ilya Tolstikhin, Olivier Bousquet, Sylvain Gelly, and Bernhard Schölkopf. Wasserstein Auto-Encoders. *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 11 2017.

[73] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural Discrete Representation Learning. *Advances in Neural Information Processing Systems*, 2017-December:6307–6316, 11 2017.

[74] Neil Zeghidour, Alejandro Luebs, Ahmed Omran, Jan Skoglund, and Marco Tagliasacchi. SoundStream: An End-to-End Neural Audio Codec. *IEEE/ACM Transactions on Audio Speech and Language Processing*, 30:495–507, 7 2021.

[75] Junichi Yamagishi, Christophe Veaux, and Kirsten MacDonald. CSTR VCTK Corpus: English Multi-speaker Corpus for CSTR Voice Cloning Toolkit, 2019.

[76] Antoine Caillon and Philippe Esling. RAVE: A variational autoencoder for fast and high-quality neural audio synthesis. 11 2021.

[77] Danilo Jimenez Rezende and Shakir Mohamed. Variational Inference with Normalizing Flows. *32nd International Conference on Machine Learning, ICML 2015*, 2:1530–1538, 5 2015.

[78] Zalán Borsos, Raphaël Marinier, Damien Vincent, Eugene Kharitonov, Olivier Pietquin, Matt Sharifi, Olivier Teboul, David Grangier, Marco Tagliasacchi, and Neil Zeghidour. AudioLM: a Language Modeling Approach to Audio Generation. 9 2022.

[79] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-December:770–778, 12 2015.

[80] Nal Kalchbrenner, Erich Elsen, Karen Simonyan, Seb Noury, Norman Casagrande, Edward Lockhart, Florian Stimberg, Aaron van den Oord, Sander Dieleman, and Koray Kavukcuoglu. Efficient Neural Audio Synthesis. *35th International Conference on Machine Learning, ICML 2018*, 6:3775–3784, 2 2018.

[81] Anmol Gulati, James Qin, Chung Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, and

Ruoming Pang. Conformer: Convolution-augmented Transformer for Speech Recognition. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, 2020-October:5036–5040, 5 2020.

[82] Daniel P. W. Ellis. Beat Tracking by Dynamic Programming. *Journal of New Music Research*, 36(1):51–60, 3 2007.

[83] Brian Mcfee, Colin Raffel, Dawen Liang, Daniel P W Ellis, Matt Mcvicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and Music Signal Analysis in Python. *PROC. OF THE 14th PYTHON IN SCIENCE CONF*, 2015.

[84] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. FiLM: Visual reasoning with a general conditioning layer. In *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, pages 3942–3951. AAAI press, 9 2018.

[85] Daniel Stoller, Sebastian Ewert, and Simon Dixon. Wave-U-Net: A Multi-Scale Neural Network for End-to-End Audio Source Separation. *Proceedings of the 19th International Society for Music Information Retrieval Conference, ISMIR 2018*, pages 334–340, 6 2018.

[86] Jean-Marc Valin. RNNoise , 2017.

[87] Oleg Rybakov, Natasha Kononenko, Niranjan Subrahmanya, Mirko Visontai, and Stella Laurenzo. Streaming keyword spotting on mobile devices. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, 2020-October:2277–2281, 5 2020.

[88] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9351, pages 234–241. Springer Verlag, 2015.

[89] Sicong Huang, Qiyang Li, Cem Anil, Xuchan Bao, Sageev Oore, and Roger B. Grosse. TimbreTron: A WaveNet(CycleGAN(CQT(Audio))) Pipeline for Musical Timbre Transfer. *7th International Conference on Learning Representations, ICLR 2019*, (2), 11 2018.

[90] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach

DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems*, 32, 12 2019.

[91] Eloi Moliner and Vesa Välimäki. A Two-Stage U-Net for High-Fidelity Denoising of Historical Recordings. 2 2022.