Neural networks for automatic musical projective orchestration

Léopold Crestel

LÉOPOLD CRESTEL

NEURAL NETWORKS FOR AUTOMATIC MUSICAL PROJECTIVE ORCHESTRATION

NEURAL NETWORKS FOR AUTOMATIC MUSICAL PROJECTIVE ORCHESTRATION

LÉOPOLD CRESTEL

Philosophical Doctor (PhD) in Computer Science

Equipe Représentations musicales

Institut de Recherche et Coordination Acoustique/Musique (IRCAM)

Sorbonne Université

October 2018

Citation
— Someone

ABSTRACT

Orchestration is the art of composing a musical discourse over a combinatorial set of instrumental possibilities. For centuries, musical orchestration has only been addressed in an empirical way, as a scientific theory of orchestration appears elusive. Indeed, whereas harmony and counterpoint can rely on solid theoretical grounds, orchestration remains taught from collections of examples drawn from the repertoire.

In this work, we address these questions within the machine learning framework, by proposing the first projective orchestration system. Hence, we start by formalizing this novel task. We focus our effort on projecting a piano piece (seen as an harmonic draft) onto a full symphonic orchestra, in the style of notable classic composers such as Haydn, Mozart or Beethoven. Hence, the first objective is to design a system of live orchestration, which takes as input the sequence of chords played by a pianist and generate in real time its orchestration. Afterwards, we relax the real-time constraints in order to use slower but more powerful models and to generate scores in a non-causal way, which is closer to the writing process of a human composer.

By observing a large dataset of orchestral music written by composers and their reduction for piano, we hope to be able to capture through statistical learning methods the mechanisms involved in the orchestration of a piano piece. Deep neural networks seem to be a promising lead for their ability to model complex behaviour from a large dataset and in an unsupervised way. More specifically, in the challenging context of symbolic music which is characterized by a high-dimensional target space and few examples, we investigate autoregressive models. At the price of a slower generation process, auto-regressive models allow to account for more complex dependencies between the different elements of the score, which we believe to be of the foremost importance in the case of orchestration.

RÉSUMÉ

L'orchestration est l'art de composer un discours musical en combinant les timbres instrumentaux. La complexité de la discipline a longtemps été un frein à l'élaboration d'une théorie de l'orchestration. Ainsi, contrairement à l'harmonie ou au contrepoint qui s'appuient sur de solides constructions théoriques, l'orchestration reste de nos jours encore essentiellement enseignée à travers l'observation d'exemples canoniques.

Notre objectif est de développer un système d'orchestration automatique de pièce pour piano en nous appuyant sur des méthode d'apprentissage statistique. Nous nous focalisons sur le répertoire classique, cette technique d'écriture étant à l'époque courante pour des compositeurs tels que Mozart ou Beethoven qui réalisaient d'abord une ébauche pianistique de leurs pièces orchestrales. Ainsi, notre premier objectif a été de réaliser un système capable d'orchestrer en temps réel l'improvisation d'un pianiste. Dans un second temps, nous abandonnons les contraintes liées au temps-réel afin d'utiliser des modèles de génération non linéaires dans le temps, plus proches du processus compositionel d'un humain, mais dont l'étape de génération est plus lente.

En observant une large base de donnée de pièces pour orchestre et leurs réductions pour piano, nous évaluons l'aptitude des réseaux de neurones à apprendre les mécanismes complexes qui régissent l'orchestration. La vaste capacité d'apprentissage des architectures profondes semble adaptée à la difficulté du problème. Cependant, dans un contexte orchestrale, les représentations musicales symboliques traditionnelles donnent lieu à des vecteurs parcimonieux dans des espaces de grande dimension. Nous essayons donc de contourner ces difficultés en utilisant des méthodes auto-régressives ou en cherchant des représentations mieux adaptées.

PUBLICATIONS

[1] Léopold Crestel and Philippe Esling. "Live Orchestral Piano, a system for real-time orchestral music generation." In: *14th Sound and Music Computing Conference 2017*. Proceedings of the 14th Sound and Music Computing Conference 2017. Tapio Lokki and Jukka Pätynen and Vesa Välimäki. Espoo, Finland, July 2017, p. 434. URL: https://hal.archives-ouvertes.fr/hal-01577463.

[2] Léopold Crestel, Philippe Esling, Lena Heng, and Stephen Mcadams. "A database linking piano and orchestral midi scores with application to automatic projective orchestration." In: The 18th International Society for Music Information Retrieval Conference. July 2017. URL: https://hal.archives-ouvertes.fr/hal-01578292.

[3] Andrew Shin, Léopold Crestel, Hiroharu Kato, Kuniaki Saito, Katsunori Ohnishi, Masataka Yamaguchi, Masahiro Nakawaki, Yoshitaka Ushiku, and Tatsuya Harada. "Melody Generation for Pop Music via Word Representation of Musical Properties." 2017. URL: http://arxiv.org/abs/1710.11549.

ACKNOWLEDGMENTS

Acknowledgments

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

Part I

INTRODUCTION

1 MUSICAL ORCHESTRATION

1.1 ORCHESTRAL MUSIC COMPOSITION

Orchestration is the art of writing music for the orchestra [88]. The almost infinite palette of sounds offered by combining the properties of different instruments represents an astoundingly expressive medium for composers. However, this variety comes at the cost of an overwhelming compositional complexity. An extended knowledge about each instrument and the way they blend together appears as an absolute necessity. This complexity adds up to the usual subtleties of elaborating a musical discourse. Therefore, orchestration is often considered as a pinnacle among different musical disciplines.

1.1.1 *The art of manipulating timbre*

Timbre, or tone colour, defines the perceived sound qualities of a musical event. While there is no consensus over its precise definition, it is commonly referred to as the property that allows listeners to distinguish two sounds produced at the same pitch, loudness, duration and environment (e.g. identical room reverberation) [73]. Hence, timbre often embeds perceptual attributes such as the *brightness* [118] or *roughness* [78] of a sound. Compared with solo works, the range of timbre that can be created by an orchestra is gigantic, and the search for a particular sound becomes a central aspect of the creative process. Hence, orchestration is often referred to as the art of manipulating instrumental timbres [70].

Johan Sebastian Bach's Cantata *Brich dem Hungrigen dein Brot*[1] testifies about the early preoccupation of composers for timbre in orchestral music. In this example, a simplistic melody is spread across the different instruments of the orchestras to create a sensation of instability and movement.

The structuring role of timbre is particularly noticeable in didactic pieces such as Benjamin Britten's *Young person's guide to the orchestra*[2] or Maurice Ravel's *Bolero*[3] which features constant re-orchestrations of the same theme.

The *Klangfarbenmelodie* (melody of timbre) is a concept which appeared and have been extensively used in the early XXth. It consists in splitting a melody between several instruments. Anton Webern's *Concerto for nine instruments, Op.24*[4] is one of the most notable example of *Klangfarbenmelodie*.

1 Johan Sebastian Bach - *Brich dem Hungrigen dein Brot, BWV 39* (1726): https://www.youtube.com/watch?v=pDnwQ-YIAoQ
2 Benjamin Britten - *Young person's guide to the orchestra* (1945): https://www.youtube.com/watch?v=4vbvhU22uAM
3 Maurice Ravel - *Bolero* (1928): https://www.youtube.com/watch?v=pNlXrdJFTAM
4 Anton Webern - *Concerto for nine instruments, Op.24* (1934): https://www.youtube.com/watch?v=pVQambrIKNo

Timbre became an intense source of reflection for contemporary composers. Among the many notables examples we can cite *Quattro pezzi su una nota sola*[5] written by Giacinto Scelsi. Each of the four piece is constructed on a single note which serves as a foundation for exploring orchestral timbre. In the early 1970s, the spectral approach to music composition emerged under the impulsion of Tristan Murail and Gérard Grisey. It is based on the study of the spectral representations of the different instruments of the orchestra in an attempt to merge their timbral properties in order to create unique sounds. Gerard Grisey has been the pioneer of this movement and his piece *Partiels*[6] has notably influenced the next generations of composers.

1.1.2 *Learning orchestration*

Orchestration involves a wide set of intricate mechanisms, most of which have not yet been satisfactorily theorized. Indeed, some renowned composers conjectured that orchestration would remain an empirical discipline, which could only be learned through experience and never axiomatised in books [61]. Even if several notable musicians have written treatises [9, 33, 61, 97], those mostly provide a collection of existing orchestration examples from which one can draw inspiration. This scarce set of knowledge is to be compared with other traditional composition domains, like harmony, which benefits from a long history of theoretical principles and research [94, 116]. Finally, even for a trained composer, finding the instrumental combination that best express a musical idea can remain particularly elusive.

Hence, the combinatorial complexity carried by orchestration, the difficulty to predict the resulting sound of an instrumental mixture and the harder control over the final performance are three of the major difficulties faced by composers. We detail these issues and what they entail in the following three subsections.

1.1.2.1 *Combinatorial complexity*

Charles Koechlin pointed out that "*most likely an orchestration treatise would not be complete given the richness of the sonic material and their combinations.*" [61, p.1]. Indeed, given the number of different instruments in a symphonic orchestra, their respective tessitura, intensity ranges and variety of playing styles, one can foresee the extensive combinatorial complexity embedded in the process of orchestral composition. If we consider an orchestra only composed by 5 instruments, each of them having a pitch range of 2 octaves (24 notes) and 3 possible intensities, we can derive a widely under-estimated lower-bound of the number of possible combinations for a symphonic orchestra equal to $(24 * 3)^5 > 10^9$. One can easily understand the impossibility for a composer (or even a computer) to exhaustively explore all of the existing orchestral combinations.

5 Giacinto Scelsi - *Quattro pezzi su una nota sola* (1959): https://www.youtube.com/watch?v=MfTjz6emd7c

6 Gerard Grisey - *Partiels* (1975): https://www.youtube.com/watch?v=X6S7W8BkKmw

1.1.2.2 *Unpredictability of instrumental mixtures*

Instrumental mixtures are complex phenomena and being able to accurately predict their resulting timbre is extremely difficult, even for a trained musician. Orchestral effects, such as blends in which the timbre of several instruments are transformed into a unique sound from which the original sources are undistinguishable, have been observed [111]. Hence, not only an accurate knowledge about each individual instrument is necessary, but also an extended experience about the resulting sound of their combinations.

1.1.2.3 *Control over the performance*

The notation used in classical western music for orchestral scores allows to define the melodic, rhythmic, harmonic and timbral structure of the piece. However, many aspects are not covered by the traditional notation and depend on the interpretation of the score. For instance, performers often take the freedom to slightly bend the precise rhythmic structure described in the score by adding subtle accelerations and decelerations.

This observation is particularly true in the case of orchestral pieces, where the conductor has to carefully balance the different instrumental sections to create the desired sound. This demanding task is particularly important in order to faithfully recreate the orchestral effects imagined by a composer.

However, the notation does not permit to precisely specify this balance, and important variations in the sonic rendering of the piece may occur between different interpretations of the same piece. Learning the ability to anticipate which orchestral arrangement will be prone to important variations is particularly difficult for composers.

1.2 SCIENTIFIC INVESTIGATIONS OF ORCHESTRATION

In the preceding section, we detailed the considerable difficulties that composers may be facing when writing orchestral music. Walter Piston wrote that "orchestration [is] an art and not a science" [88, p.356], and that it might be the reason why there is not necessarily an explanation for what sounds "good" or not. However, perhaps these difficulties are to some extent due either to our limited understanding of the acoustical laws and the principles underlying human sound perception, or to the inability of the human brain to embrace the wide number of combinations offered by an orchestra. Hence, investigating the art of orchestration through the scope of acoustics, signal processing and computer science could provide useful insights for composers by disentangling the relations between written scores and their perception. Ideally, this could lead to the development of tools for assisting composition.

We can distinguish to axes underpinning a scientific investigation of orchestration. The first axis is an attempt to decipher the mechanisms behind the human perception of orchestral music. It intensely relies on perceptual studies and acoustics, and mostly have an analytical purpose. The second axis attempt to exploit the resources of computers to develop generative tools for assisting composers.

1.2.1 *Acoustics and perception*

1.2.1.1 *Characterising timbre*

As discussed earlier, timbre is often defined as the quality which allows to distinguish two sounds played at the same pitch, loudness, duration and environment [73]. This negative definition of timbre might be intuitively satisfying, but it is clearly not convenient to manipulate. Besides, it provides no information about the relation between a given waveform propagated in a room to a listener and the timbre perceived by that same listener. This remark suggests two axes of research: studying how waveforms propagate and mix in an acoustic space, and studying the link between a given waveform heard by a listener and the corresponding perceived timbre.

The first problem is related to waveform propagation in a medium, which involves highly non-linear and complex mechanisms [37, 41]. This phenomenon is modelled by equations, and methods for solving them have been investigated for centuries. Closed form solutions have been found for the simplest ones, while numerical approximations are usually accessible for the most complex ones [6].

The second question relates to how the human brain processes a physical sound and associates it with a perceived timbre. Researches in signal processing have provided various spectro-temporal descriptors [86]. While most of them have originally been designed for music information retrieval tasks, such as genre classification [85], some of these descriptors correlate well with perceptual attributes such as the *brightness* or *roughness* of a sound [86], thus providing a mathematical definition for some aspects of timbre.

1.2.1.2 *Timbre as a structuring force in music*

The variations in timbre along a composition can segment and structure a musical discourse into logical units. McAdams and al. introduces the concept of auditory streams which is a psychological grouping of musical events which are interpreted by the listener as a coherent "whole" [70, 71]. A taxonomy of the different types of grouping is proposed, from which powerful analysis methods can be derived to extract large-scale structuring elements from a complex orchestral score [44]. The extracted structure provides a clear presentation of the temporal organisation of the piece and layering scheme among the different voices of the orchestra. In particular, this analysis scheme has been used for developing *Orchard* [7], an annotated database of orchestral scores.

1.2.2 *Computer-sciences*

We mentioned in Subsection 1.1.2.1 the combinatorial complexity of orchestral music. Computers are particularly adapted for rapidly exploring large ensemble of solutions and, thus, prove to be helpful for assisting composers in the many aspects of their compositional process.

In particular, "translation" tasks from a medium to another one can be particularly tedious for composers, while computer can excel at doing them. These

7 https://orchard.actor-project.org/search/

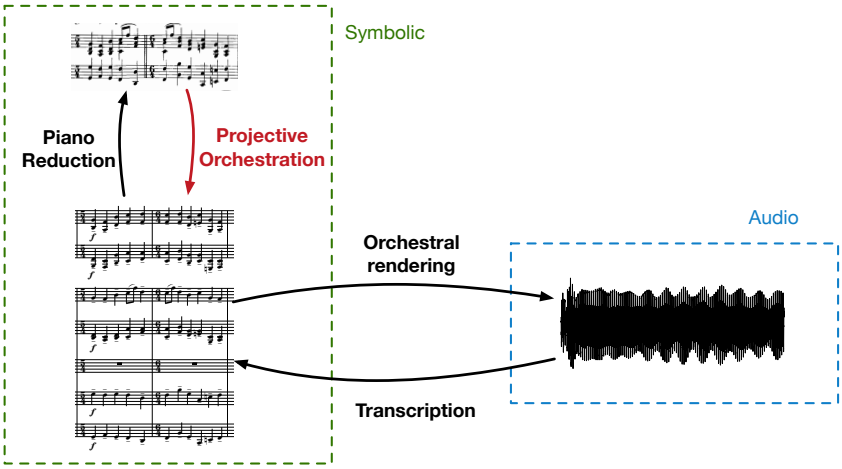

Figure 1.1: We can distinguish four orchestral "translation" problems, represented by the four arrows on the figure. Piano reduction and projective orchestration take place in the symbolic domain (framed in green) and link an orchestral score with its possible arrangements for piano. Orchestral rendering and transcription link the symbolic domain with the audio domains (framed in blue). Computer-assisting tools have already been developed for all these translation tasks, with the exception of the projective orchestration task (red arrow), which we propose to address in this work.

four translation tasks, *transcription*, *rendering*, *reduction* and *orchestration*, are depicted on Figure 1.1 and detailed in the following paragraph.

TRANSCRIPTION We refer to *transcription* as writing the musical score corresponding to a perceived sound. The ensemble of timbre possibly created by an orchestra forms a wide and convoluted space. As discussed earlier, how the exploration of this space is complex for composers, because of the number of possible combinations and the intricacy of instrumental mixtures. To address this question, several systems have been proposed, [35, 92, 102] which aim at assisting composers for navigating in the space of instrumental combinations.

Hence, when a composer wants to produce a certain sound with an orchestra (often in the form of a recorded sound, such as a screaming person or a creaking door), these systems try to produce an orchestration that best mimic this target while focusing on the micro-temporal structure of sounds.

ORCHESTRAL RENDERING Rendering an orchestral score is usually done by an orchestra playing the music written. Rendering software allow composers to better imagine how their piece will sound when played by a real orchestra by simulating a performance. Most of the recent notation software, such as *Sibelius* or *Finale*, and digital audio workstations, such as Logic Pro, embed rendering engines dedicated to the orchestra. Most of these tools are based on concatenating recordings of individual notes played by the different instruments of the orchestra. However, these tools often lack of realism as the articulation between the successive notes or the balance between the different sections is grossly rendered. Hence, more advanced tools such as *OrchSim* have been developed, which allow for a more precise control of these dynamics aspect, and a more realistic rendering.

PIANO REDUCTION The reduction of an orchestral score is the generation of a piano score which preserve the melodic, harmonic and rhythmic structure of the original orchestral score while being playable for a pianist. As the number of notes that can simultaneously be played by a pianist is limited, performing a reduction often involves removing some notes of the original orchestral score. Famous examples of orchestral reductions can be found in the repertoire, such as Franz Liszt reductions of Ludwig Von Beethoven's *Symphonies*.

The exercise undoubtedly has a pedagogical interest, as it requires to be able to synthesize a complex orchestral work and reduce it to its most essential elements. However, being able to automatically perform reductions would be extremely useful for pianists in order to be able to rapidly obtain a playable solo version of their favourite orchestral music. This complex task has been addressed with complex probabilistic models, for instance in [34].

PROJECTIVE ORCHESTRATION Among the different writing techniques for the orchesstra, one of them consists in first laying an harmonic and rhythmic structure in the form of a piano score and then adding the orchestral timbre by spreading the different voices over the various instruments [88]. We refer to this operation of extending a piano draft to an orchestral score as *projective orchestration*. This technique has been widely used by classic composers. One such example is the orchestration by Maurice Ravel of *Pictures at an Exhibition*, a piano work written by Modest Mussorgsky (see Figure 1.2). By observing an example of projective orchestration (see Figure 1.2), we can see that this process involves more than the mere allocation of notes from the piano score across the different instruments.

To our best knowledge, no computer-assisting tool has been developed for addressing this task. The objective of this work is to design the first system for automatic projective orchestration.

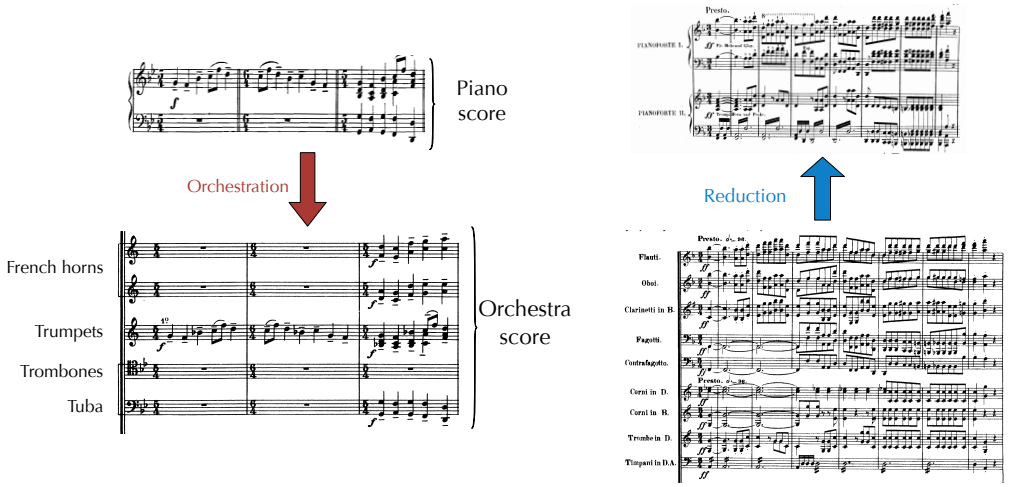

Figure 1.2: (Left) The top-most part of the figure represents an excerpt of the musical score from *Pictures at an Exhibition*, a piece for piano composed in 1874 by Modest Mussorgsky (top), and the musical score of its orchestration by Maurice Ravel (1929) (bottom). (Right) First bars of the fourth movement of Beethoven's 9th symphony (bottom) and its reduction for two pianos by Liszt (top). See how the combination of the bass sections (*Trombe*, *Fagotti* and *Contrafagotto*) and the roll of the *Timpani* section are rendered altogether by repeated 16^{th} notes at the left hand of the *Second Piano*.

2

AUTOMATIC COMPOSITION

Computer-aided composition refers to algorithms and softwares designed to help composers writing music. Hence, the definition of this research field might cover a wide range of scientific domains. Indeed, the development of a diverse range of audio synthesis methods [20, 106] and digital effects (filters, delay, reverberation, spatialization) [99] benefited from intensive research in signal processing, acoustics and auditory perception. Furthermore, theoretical works in mathematics led to the development of a variety of representations for sounds or scores, with applications in both compression tools [83] or music analysis [7, 12]. Music composition has also been a wonderful playground for computer science and artificial intelligence. The overarching challenge of creating an automatic composition system is still one of the most vivid and thrilling issue for the computer music community [101].

The field of automatic music generation has been widely investigated since the 1950s and led to important advances in musical creation. One of the first example of such research is probably the *Illiac Suite*, a string quartet created by an algorithm designed by Lejaren Hiller and Leonard Issacson in 1957. The piece has been automatically generated by sampling various musical quantities, such as the pitch and rhythm, from probability distributions [104]. Since this seminal work, computer-generated music has been a substantial source of inspiration for composers. Stochastic processes have been further investigated by many composers. In particular, Iannis Xenakis explored a wide range of different distributions in some of his pieces (for instance the ST series [1]). Magnus Lindberg developed an algorithm based on constraint programming for his piece *Engine* [2]. Since these pioneering creations, the field has considerably evolved, from rudimentary probabilistic models and rule-based systems to the latest innovations in machine learning. Last year, Daniele Ghisi created the music of *La Fabrique des monstres* [3], for which entire sections of audio have been generated using neural networks.

These artistic productions illustrate the evolution of the automatic composition field, which was itself widely influenced by the successive trends in computer science. In particular, *formal grammars*, *constraint programming* and *machine learning* have been intensely investigated at different periods and shaped the successive generations of automatic composition systems [101].

Across these research, one of the most major distinction between different approaches is the type of information manipulated. Two modalities are commonly distinguished and referred to as *signal* and *symbolic* representations (see Figure 1.1). First, audio recording allows to store the musical information as the variation of atmospheric pressure caused by the propagation of sound waves. As such, audio recording is the closest representation of the physical phenomenon. By extension, signal representations encompass time-frequency

1 Iannis Xenakis - *ST/10=1,080262* (1956-1962): https://www.youtube.com/watch?v=9XZjCy18qrA
2 Magnus Lindberg - *Engine* (1996): https://www.youtube.com/watch?v=yKJgTqRWsKg
3 Daniele Ghisi - *La Fabrique des monstres* (2018): https://vimeo.com/groups/494916/videos/275248479

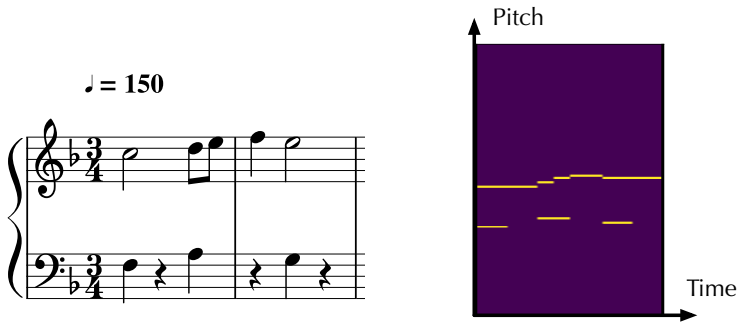

Figure 2.1: First bars of *Notebook for Anna Magdalena Bach* using the standard western notation (left) and the piano-roll representation (right).

representations such as the Fourier Transform of an audio waveform. Generative systems for raw audio have been recently proposed, with promising results [75, 80]. However, these systems have been essentially applied to speech synthesis and their development mainly focused on the quality of the audio signal generated rather than the coherency of the macro-temporal structure. More abstract representations may prove useful to manipulate higher-level musical structures. Symbolic representations designate any compressed encoding of the sound using a set of discrete symbols describing musical events. A typical example of symbolic representation is the western classical musical notation which has long been used as a communication medium between composers and performers. In order to develop tools for automatic composition, symbolic representations seem to be the most adapted, as it seems easier for an artificial intelligence to grasp the temporal organization and important high-level factors. Here, we will focus on symbolic representations as they allow to capture higher-level relations in complex multi-instrument relations.

In Section 2.1, we review the most widespread symbolic representations for music. In Section 2.2, we review algorithms for automatic music generation and outline the lack of research applied to orchestral music. Then, we review orchestral research in Section 2.3 while trying to bridge the gap with more traditional and extensively investigated problems of computer-aided composition. Finally, we present our approach and detail the structure for the remainder of this thesis in Section 2.4.

2.1 SYMBOLIC REPRESENTATIONS OF MUSIC

A crucial step when building a generative system for symbolic music is to find an adapted representation of the musical information. Several symbolic representations have been developed for different musical styles and purposes. As an exhaustive list would be too long, we introduce here only the representations that were used in this thesis, which focuses on western classical music.

2.1.1 *Western classical notation*

The most common symbolic representation for western classical music is the musical scores with western notations, as represented on Figure 2.1 (left). As we

can see, both time and frequencies are discretised. The rhythmic information is represented as fractions of a given symbolic temporal unit (typically a quarter note) which relates to an absolute time duration by defining the *tempo* of the piece. Fundamental frequencies are mapped to a discrete pitch scale, whose degrees are represented equivalently by the vertical position on staves or by a letter between A to G (possibly modulated by a sharp or bemol) in the English system. The relationship between a note on the stave and its frequency is determined by the clef. However, depending on the period and the instruments played, the scale may vary, even though it tends to be standardised nowadays (12-tone equal temperament with A4 being equal to 440Hz).

The number of scanned musical scores available on internet in an image format is gigantic, and the notation is usually accurate. However, this format is particularly difficult to parse with a computer. To address that issue, the field of *Optical Music Recognition* (*OMR*) [96, 117] provides promising results, but as of today, no system is sufficiently robust for orchestral scores.

2.1.2 *Standard MIDI files*

The Musical Instrument Digital Interface (*MIDI*) is a technical standard published in 1983 and intended to unify the communication protocols and interfaces between synthesizers and computers [125]. Originally designed for live performances, the *MIDI* protocol is based on sending a message each time an event occur (such as a key being pressed). By assigning timestamps to *MIDI* messages, sequence of events can be stored in a file in order to be played back later. The *MIDI* format became one of the most widespread digital score format, and is still widely used in audio synthesizers.

The same pitch scale as in musical scores is used, but labelled by a number ranging from 0 to 127 with A0 being mapped to 21. The use of timestamps enforces a form of time discretisation. However, it is easy to choose a minimum duration smaller than the shortest event occurring in the score. Hence, if that condition is respected, the *MIDI* format is lossless regarding the pitch, time and intensity information contained in the standard western score notation.

Finally, *MIDI* has been an extremely popular format with the apparition of synthesizers, and many classical scores have been encoded into *MIDI* files available on the internet. Hence, the *MIDI* format can provide a large collection of data, which are easily parsed by a computer.

2.1.3 *MusicXML*

In the same way that *MIDI* has been invented for unifying the communications between electronic music instruments, the *MusicXML* format has been specifically designed for sharing sheet music files between applications [126]. Based on the Extensible Markup Language (*XML*), *MusicXML* is extremely convenient to automatically parse and manipulate, while a lot of meta-data can easily be embedded. The major drawback of the *MusicXML* format is the relative paucity of data available.

2.1.4 *Piano-roll*

The piano-roll representation of a score is a two-dimensional matrix with its dimensions representing the discretised time and pitch and its values contain the intensity of notes (see Figure 2.1). The intensity is typically normalised between 0 and 1, 0 being a silence and 1 the loudest possible sound (corresponding to 127 in the *MIDI* format).

2.2 AUTOMATIC SYMBOLIC MUSIC COMPOSITION

Automatic symbolic music generation has been a major topic of interest since the premises of computer science, and the field has witnessed many evolutions. Here, we start by introducing the seminal problems addressed in automatic symbolic music composition. In the following subsections, the different approaches used to tackle these tasks are detailed. We divide the literature between rule-based approaches and probabilistic inference methods. In the last subsection, the different contributions to the field of automatic composition are summarized, and two important aspects, the data representation and evaluation measures, are discussed.

2.2.1 *Seminal problems*

Creating a system able to automatically generate interesting and novel music appears as a daunting task. Indeed, depending on the genre, the whole music-writing process is composed of several compositional sub-problems. Each of these sub-problems constitutes a fertile source of questions for the algorithmic music community, allowing to both reduce the initial complexity of the task to a more manageable question, while exploring a wider variety of methods.

First, a distinction can be made between systems generating monophonic music (a single voice melody is played), and polyphonic music (multiple voices are simultaneously conducting independent melodies with concurrent notes). Second, the input representation may vary depending on the musical genre. Classical music is usually written in the western notation system, whereas in Jazz music only the melody and chord chart are often indicated. Chord chart are sequences of symbolic chords, a compact representation of an harmonic information in the form of a bass note and the chord's quality (*e.g.* major, minor, seventh).

Monophonic music generation consists in creating a sequence of non-overlapping notes played successively. Musically, it refers to the generation of a single voice. It can be unconstrained, related to a solo monophonic instrument score [1, 25, 110, 119–121], or constrained, related to the generation of a single voice in a polyphonic score (one voice in a cantata) [38, 81]. We refer to that last case as monophonic in a polyphonic context. An example of application is a jazz solo generation over a chord chart [11, 13, 43].

Polyphonic music generation refers to a system that jointly generates several concurrent voices. This problem is widely more challenging than monophonic generation due to the intricate relations between different voices. Furthermore, the dimension of the generation space increases with the number of voices.

Similarly to the monophonic case, polyphonic generation can be constrained or not. Hence, a common application is to generate a set of voices that could best accompany a given fixed melody. We refer to this task as *harmonization*. For instance, harmonization of Bach's chorales has been an extremely popular topic [2, 27, 30, 49], but this task has also been investigated for other musical genres [119]. Recently, unconstrained polyphonic music generation has been one of the most proficient topic, with a wide variety of approaches proposed [16, 21, 28, 31, 32, 48, 63, 67, 72, 76, 108, 114, 122].

Symbolic chords are also widely used in classical music for analysing pieces or drafting a work. Hence, *chord sequence generation* is another popular task in the polyphonic context. The symbolic notation for chords allows to drastically reduce the dimension of the generation space (compared with the polyphonic case) and to focus on the harmonic aspect of music. Chord sequence generation can be unconstrained [82], but will often be constrained by a melody [15, 40, 124]. As that case is very similar to the previously introduced harmonization task, we refer to it as *symbolic chord harmonization*.

Note that generative symbolic models can be used as prior for other tasks. *Automatic transcription* is a notorious example in which a generative model can be used for discriminating between different detected notes by deciding which one is the most musically relevant given the context [103, pp. 41–44].

2.2.2 *Existing approaches*

The two major families of approaches that have been investigated in the automatic music composition field are *rule-based systems* and *probabilistic models*. These families reflect two essential aspects of a compositional process. First, a musical style can be characterized by a set of rules which entail a particular vertical and horizontal organisation of the notes. However, a composer will sometimes seek to bend these rules in an unexpected way in order to create various types of effects and attract the attention of the listener. Hence, rule-based approaches strive to integrate musical theory in generative systems while probabilistic approaches attempt to model the crucial variability in a creative process. Both approaches complement one another and some systems actually aim at combining them [1, 25, 42, 72, 121]. However, there always is a predominant aspect in a given approach. Here, we present various systems for automatic composition based on this classification.

2.2.2.1 *Rule-based approaches*

The underlying idea behind rule-based systems is to encode music theory knowledge as a set of rules able to drive a generative algorithm. In this category, *formal grammars*, *constraint programming* and *evolutionary algorithms* provide powerful frameworks that have been the most widely used for music generation.

FORMAL GRAMMARS Formal grammars can generate strings of symbols with complex patterns from a simple set of rewriting rules. The repeated application of these rules iteratively transform and extend an initial seed sequence. Originally developed for modelling human languages, musical patterns can also be elegantly represented by formal grammars [52, 98]. However, generating

sequences of symbols at a fine-grain level (each note for instance) will necessitate the tedious elaboration of a large number of rules. Hence, formal grammars have been mostly used for relatively simple tasks such as monophonic melody generation. A particular type of formal grammar which has been profusely used in music composition for their simplicity and ability to generate convoluted patterns are L-systems [72, 114]. The purely deterministic behaviour of formal grammars in their simplest form can be prejudicial in a music creation context. Hence, adding stochasticity in rewriting rules has been extensively used in order to emulate the multiplicity of potential choices in music [25, 72, 114, 121]. An other solution is to automatically infer the rules using statistical learning on a corpus of desired examples [25, 43]. This approach is closely related to the probabilistic methods that we detail in Subsection 2.2.2.2.

CONSTRAINT PROGRAMMING Music theory can also be interpreted as a set of constraints, which implicitly designates which notes can be played depending on the context. For instance, a student trying to learn counterpoint will typically explore the different possibilities for each successive note before selecting one of them. Constraint Satisfaction Problems (*CSP*) is a computer science field, which aims at solving problems stated as a list of constraints over a given domain. A musical example could be to generate a sequence of 10 pitches with only fourth and fifth intervals, and without the pitch C. Here both the interval and pitch removal conditions are constraints over the set of allowed sequences. Constraint programming refers to the family of techniques which attempt to solve *CSP*. Assisting tools based on constraint solver have been designed by composers and led to the creation of renowned musical pieces, such as *Engine* written in 1996 by Magnus Lindberg. [4]). Convincing systems for automatic music generation have been proposed in the constraint programming framework [3, 30, 110]. The two main advantages of this approach is that the constraints are conveniently enunciated in a declarative style, which directly translates sentences such as "intervals must be lower than 5 semitones" into rules for the generative system. Furthermore, it provides the user with a lot of control over the algorithm and allows for proposing an entire ensemble of potential candidates instead of a unique solution. Hence, it is possible to build an interactive system where the user can browse this proposed ensemble of solutions [110].

EVOLUTIONARY ALGORITHMS Traditional algorithms for *CSP* struggle in high-dimensional search spaces, and the time needed for finding solutions might be prohibitive. Evolutionary algorithms are approximate search procedures that seek to alleviate this issue [107]. To do so, an initial population of solutions is randomly generated and the best solutions are selected thanks to a fitness function. Similarly to the mechanisms of genetic evolution, the selected subset of solutions is increased by crossovers and mutations. This process allows for a fast, but non-exhaustive search.
The design of the fitness function is of paramount importance. However, in the musical context, defining a measure of musical quality is a tedious and intricate task [81, 119]. An interesting alternative is to use human feed-back for selecting the best examples in the population [13, 56]. Corpus-based distances have also

4 https://www.youtube.com/watch?v=yKJgTqRWsKg

been developed, which evaluate the proximity of the generated examples with some reference musical pieces [1, 42, 120]. A particularly interesting refinement in the context of creative systems is to define a multi-objective fitness function. Hence, rather than a unique solution, an ensemble of propositions (referred to as the *Pareto front*) is selected as providing various qualities over the selected dimensions [27, 36, 40], which is a desirable feature for a computer-assisted composition tool.

2.2.2.2 *Probabilistic models*

Music can hardly be modelled by a deterministic mapping because it is extremely rare that only a single solution is acceptable given a certain context. Instead, a variety of solutions are all musically relevant and interesting to various degrees. Furthermore, the relative preferences inside this set appear to be mostly subjective. Besides, a desirable property for a creative system is to be able to propose different solutions, even though the same context is repeatedly presented. A first solution consists in returning as many solutions as possible and let the user chose (as in interactive constraint solving systems and evolutionary algorithms). A second option is to incorporate a form of stochasticity in the system to emulate the variability of human decisions (which is done in most of the aforementioned systems, in the choice of transition rules for formal grammar or the choice of mutations for evolutionary algorithms).

This choice for stochastic systems can be pushed further by embedding the compositional problem in a probabilistic framework, where the different musical variables are sampled from probability distributions. Parametric probabilistic models have benefited from intensive research over the last decades [45, pp.12-26] and efficient parameter inference techniques allow to design elaborated models in order to tackle complex musical tasks. In the next paragraphs, we review graphical probabilistic models and neural networks applied to various automatic music composition tasks.

GRAPHICAL PROBABILISTIC MODELS The term *graphical models* refers to a general probabilistic framework in which complex joint probability distributions are modelled in the form of products of conditional probabilities. These factored distribution can be compactly represented as graphs in which nodes are random variables and directed edges represent conditional dependencies. *Markov Models* (*MM*) are a particular class of directed graphical model, which have been specifically designed for modelling causal dependencies. Their successful applications to various time series modelling problems [14, pp.605-607] logically drew the attention of the computer music community. *Hidden Markov Models* (*HMM*) introduce a modelling of unobserved hidden explanatory factors of the visible observations. This idea can be illustrated by considering a singer performing a solo improvisation over a sequence of chords. Only the melody is heard (observed), but the choice of notes is largerly driven by the (hidden) chord progression. Both *MM* [15, 38] and *HMM* [2] have been widely investigated in the context of harmonization of Bach's chorales. A system for polyphonic music generation has been proposed in [76]. In this approach, all the musical parameters (rhythm, chords progression and melodic curves) are successively generated by different modules implemented as a *MM* or *HMM*.

Unfortunately, for modelling polyphonic music, the number of states in a *HMM* grows exponentially with the number of voices, which will result in a prohibitive training time. Some workarounds can be introduced to maintain a reasonable number of state (such as using symbolic chords), but they artificially restrict the possibilities of the model. Hence, more elaborated graphical probabilistic models have been proposed, which model the activation of each pitch separately. In [63], Directed Random Markov Fields are investigated the authors propose to use a lattice of binary units directly derived from the piano-roll representation to generate classical piano music. However, drastic loss of informations, such as the octave and rhythm structure, are made in order to reduce the complexity of their model which may prove to be prejudicial for in the most general musical context. A binary tree graphical model is proposed in [82] for modelling jazz chord sequences of fixed length. A measure of perceptual similarity between chords is developed, which allows for modelling a probability of substitution between chords. This substitution mechanism elegantly allows the model to play chords unseen during the training process. Recently, [49] proposed a maximum entropy model which allows for efficiently modelling polyphonic sequences with a fixed number of voices. The value of each note is conditioned by all neighbouring notes, either temporally (horizontally) and harmonically (vertically). The flexibility of this model allows to address a variety of tasks, from complete generation to harmonization. Besides, it allows to select a fragment of the score and generate alternative version of this precise fragment, which can be a stimulating feature for composers.

NEURAL NETWORKS Neural networks can be cast as a special case of graphical probabilistic model that allow to model complex distributions, at the cost of an approximate parameters inference procedure (see Chapter 3 for a more detailed presentation) [14, pp.225-232]. Both the modelling power and efficiency of the learning process allowed to manipulate larger databases [45, p.18]. Consequently, increasingly complex tasks in music composition have been addressed, such as unconstrained polyphonic generation [28, 67, 122]. Furthermore, new datasets appeared, which considerably widened the spectrum of musical genres addressed by automatic composition systems [10, 77, 93].
In this family of models, *Recurrent Neural Networks* (*RNN*) have been specifically designed for modelling time series [45, pp.367-368]. An early attempt was made for generating chord and melody sequences in the style of blues music [32]. Their approach consisted in encoding a musical score as a time-series (here a piano-roll), and use *RNN* as one-step predictors to generate successive musical frames. The same one-step predictive scheme has been applied to the generation of Irish reels in [31], folk music in [108].
A hierarchical organisation of *RNN* layers has been proposed in [21], in which each layer is in charge of a different temporal granularity. Indeed, there system generates pop music and output three information (a melodic line, a drum track and a chord sequence) at different rates. The layers in charge of the drums and chord sequences generate one output per bar, while the melodic layers output a value for each quarter note. Besides, by conditioning the melody and chord sequence generation under a musical scale, they reduce the number of possible outputs and decrease the chance of musically aberrant generations, but at the price of a system dedicated only to pop music.

In [69], a *biaxial RNN* architecture is used, which instantiates recurrent connections both along the time axis and the pitch axis of a piano-roll representation. The bidirectional structure of their model is very similar to the Markov Random Field architecture developed by Lavrenko et al. in [63]. Both approaches suffer from the same conceptual flaw which is to consider that the pitch axis is directed from the lowest to the highest note, which has no musical justification. However, this architecture allows to efficiently implement a form of pitch translation invariance by sharing weights across all networks of the time axis. Indeed, most qualities of a musical piece are preserved when transposing it a few pitches up or down. Hence, to model this relative insensitivity to pitch translation, the weight sharing mechanism ensures that each note is processed in a manner which is independent of its absolute pitch location, but only relies on the preceding notes along the time and pitch axes. A style-labelling mechanism allows to condition the generation on the style of a particular composer, specified by a one-hot vector.
In [48] Hadjeres et al. proposed a *RNN*-based architecture based on a pseudo-Gibbs sampling process to generate Bach chorales. This generative procedure is very similar to the maximum-entropy model they proposed in [49], and also based on the construction of a probability distribution modelling the individual notes of one voice conditioned by the surrounding notes. This approach differ in the use of *RNN* instead of exponential families. On the same task, an earlier proposal was made by mixing energy-based models (another form of graphical model) with recurrent neural networks [11, 16].
A recent trend in machine learning has been to use a secondary neural network to act as a loss to train the primary model instead of traditional error functions such as the mean squared error. This approach is called Generative Adversarial Networks (*GAN*) [45, pp.265-266]. Several works underlined the fact that traditional measures may not be suited for musical applications and that adversarial approaches provide an adequate way to circumvent that issue [45, pp.265-266]. Using *GAN*s, several systems were proposed for polyphonic music generation [28, 122] with a convolutional architecture. The same model has been applied to lead sheet generation and arrangement in [67]. However, a major drawback of adversarial training is that the optimisation process is particularly unstable, mostly relies on heuristic and requires a lot of training data.
Although neural networks are agnostic approaches that do not naturally provide a declarative encoding of music theory concepts, the reward function of a reinforcement learning system can be designed with such musical rules [57].

2.3 AUTOMATIC SYMBOLIC ORCHESTRATION

Unfortunately, orchestral music did not receive as much attention from the algorithmic music community as the polyphonic automatic composition did. One of the major reasons behind this lack might be that the supposedly simpler question of automatically generating mono-instrumental music is already far from being adequately addressed. However, a wide range of satellite tasks revolve around orchestral composition and offer thrilling challenges.

Unavoidable analogies with mono-instrumental music can lead to attempting to adapt the previously introduced methods. However, the intricate relations between instruments bring a new order of complexity which prevents from tack-

ling orchestration simply as a mere stacked version of the previous problems. Indeed, as mentioned in Chapter 1, timbre is a central question in orchestration. However, very few information about timbre is provided in a musical score. Even though the instrument names are indicated, inferring the resulting sound from this reduced information is a difficult task. Besides, methods such as constraint programming do not scale properly with the combinatorial explosion of orchestral possibilities. Hence, orchestration represents a daunting challenge for the computer music community and we present the few systems revolving around symbolic orchestral music which have been developed so far in Subsection 2.3.1.

However, useful insights can be gained from the past experiences in polyphonic music generation, in particular in the choice of data representation and the design of an evaluation measure. Hence, we review the major contributions in Subsection 2.3.2 for the data representations, and Subsection 2.3.3 for evaluation measures.

2.3.1 *Symbolic orchestral music tasks*

First, the inverse problem of automatically reducing an orchestral score to a playable piano version has been tackled in [54] and [109] with a rule-based approach. In [34], a modular approach in a probabilistic framework is proposed. An interesting representation of chords in the form of a list of notes instead of the usual piano-roll allows to resort to relatively simple probabilistic models such as *HMM*, while obtaining impressive results.

One of the first attempt to produce an automatic orchestration system tried to cast the question as a Constraint Satisfaction Problem (CSP) [17, 113]. Those tools are limited to symbolic constraints, typically enclosed to the set of selected instruments and the notes they are allowed to play (pitch-range or inter-instruments symbolic relationships). However, a major flaw of all these systems is that timbre is not taken into consideration, whereas it is a fundamental dimension of orchestration. This remains a critical limitation as symbolic constraints are not expressive enough to tackle the spectral complexity of orchestration.

As discussed in Section 1.2, some approaches were proposed to orchestrate a given sound target. However, as the introduction of signal information dramatically increases the dimensions of the search space, approximate search procedures have to be used, such as evolutionary algorithms in [36].

Automatic accompaniment has been addressed in [24]. Given a certain style and a melodic line, the system generates an accompaniment by a full symphonic orchestra. However, this approach heavily relies on a tedious implementation of a large set of musical rules and heuristics.

As mentioned in the previous section, the recent progresses in machine learning and particularly the advent of deep neural networks allow to tackle increasingly more complex tasks. In [28, 67], a complex end-to-end system was proposed to generate multi-track music. This system generates pop music composed by five tracks (bass, drums, guitar, piano and strings) by relying on complex and deep generative architectures trained using adversarial networks. In [28], the system is composed by two modules where the first one generates

lead-sheet scores (a melody and chords with no orchestration) and the second one is dedicated to the arrangement of the previously generated lead-sheet.

In summary, very few attempts have been made to tackle automatic orchestral generation. Constraint programming allows to efficiently encode symbolic constraints such as instruments tessitura. However, the lack of expressiveness of symbolic constraint drove to include spectral constraints. Unfortunately, spectral representations lie in high-dimensional spaces, and only approximated search procedure can be used, such as evolutionary algorithms. On the other side, machine learning have recently witnessed important developments and the field seems mature enough to tackle difficult tasks such as orchestration, as recent research showed.

2.3.2 *Data representations*

Machine learning approaches necessitate to find an adequate mathematical representation of a musical score. We detailed in the first section the different formats used for encoding symbolic music.

The piano-roll representation of a musical score and has been used in many systems [16, 69]. However, compact representations can be extracted from this piano-roll. For instance, the pitch-class representation consists in removing the octave information, hence obtaining pitch vectors of dimension 12 [28, 67]. The pitch-class representation is often used for its compactness if the loss of the octave information is not prejudicial to the target task or as a rough approximation of the harmonic content (such as for chord detection). However, pitch-class representation remains a discretisation of the frequency and time axes, which do not radically differ from a piano-roll.

Most systems employ a more refined data representation specifically tailored to the tackled problem. For monophonic music, Cruz-Alcazar et al. [25] proposed to encode a melody as the difference in semitones between two successive notes (C4 to D4 would then be equal to 2). This data representation has the advantage of being invariant to transposition (except for the first note). Indeed, transposition preserves most of the musical qualities of a piece, and such representations allow to enforce a form of pitch translation invariance. A similar idea is developed in [2] for polyphonic music, where the three upper voices of Bach's chorales are encoded as a difference from the root note.

For probabilistic models, the chosen data representation directly impacts the shape of the distribution used for modelling musical events. For instance, a binary piano-roll representation will likely be modelled by a product of independent Bernoulli distributions. In [49], the four voices of chorale music are represented as four different piano-rolls. As each piano-roll is now monophonic, a softmax distribution can be used, which provides more information about the overall relationships inside the data to be modelled. Besides, it becomes easy to condition each voice over the others, which is not easily possible in piano-roll representations. The problem of this representation is that it is not adapted to cases where the number of voices is unknown. This assumption over the number of voices can be relaxed by representing each chord as a list of notes which may have various length at different time frames. However, this data representation, which is developed in [34], is less easy to manipulate, as it each frame is now a list of various lengths.

2.3.3 *Evaluation measures*

In order to compare different models, separate evaluation frameworks have been developed for each task. Unfortunately, as only few attempts have been made to develop a system for automatic orchestration, there is no reference evaluation framework for this task. However, the closely-related task of polyphonic orchestration can provide insights and directions for developing our own evaluation framework that we will introduce in Chapter 5.

For polyphonic music generation, models are frequently evaluated through a one-step predictive task [16, 63] using either the binary cross-entropy or the prediction accuracy measure. These two quantitative evaluation frameworks will be further explored in Section 5.3.

However, these mathematical criteria do not account for human preferences. Hence, more truthful measures of performance could be obtained by performing listening tests on a sufficiently large group of individuals. Recently, most articles include these listening tests in order to assess the perceptual qualities of the automatically-generated examples. In [48], listeners have to guess whether an example is generated by Bach or the proposed algorithm. In [67], different variations of the generative model are compared by a voting system. Similarly, a pair-wise comparison between different models is led in [28].

Qualitative evaluations based on human preferences provide a more truthful evaluation of a model. However, the time needed for obtaining the feedbacks of all tested candidates might be prohibited, and not adapted at an early development stage of a project. Hence, quantitative criteria provide convenient proxies for roughly and rapidly evaluating the performances of a system. Finally, both frameworks complement each others, and a framework for both quantitative and qualitative evaluation of orchestral music generation is introduced in Chapter 5.

2.4 OBJECTIVES AND MOTIVATIONS

2.4.1 *Orchestrating a piano score*

In this work, our objective is to build a system able to automatically perform the projective orchestration of a piano score (see Figure 2.2). The input piano score provides an harmonic, melodic and rhythmic structure, and the role of the system is to output an orchestral score which enhance this structure with a learned timbre expressiveness. Because orchestrating a piano-score often requires harmonic enhancement or doubling, this problem is more complex than the mere allocation of the different voices of the original piano score to different instruments.

Our approach differs from previous proposals (such as *Orchids* [36]) as the orchestration is not driven by a sound target, but rather by an harmonic, melodic and rhythmic template given in the form of a piano score. However, our proposal is very close from the arrangement module proposed in [28]. The main differences in our work is that the piano score is imposed beforehand and not generated, and that we focus on classical orchestral music ranging from the late XVIIIth to the early XXth centuries, approximately corresponding to classical and romantic periods. Hence, the number of possible instruments is

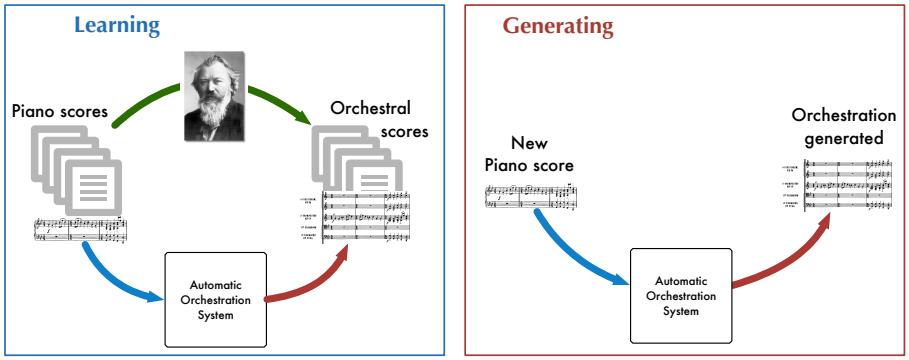

Figure 2.2: We propose to tackle projective orchestration with a statistical learning approach. Statistical inference consists in learning a set of rules by the repeated observation of a reference dataset. Once the rules have been inferred, new orchestrations can be generated for previously unseen piano scores.

much larger in our case (typically 12 against 5 in [28]), while fewer examples are available.

2.4.2 *Using purely symbolic information?*

Almost no attempts have been made to tackle a scientific exploration of orchestration based solely on the study of musical scores. Indeed, as discussed in Chapter 1, timbre is a prevailing aspect in orchestration. Hence, it may seem illogical to rely solely on symbolic information. Yet, symbolic representations implicitly convey high-level information about the spectral knowledge that composers have exploited for timbre manipulation. Here, we target that this implicit knowledge about timbre that appears through the observation of a large set of purely symbolic information.

2.4.3 *An ill-conditioned problem?*

A single piano score could produce a very large variety of orchestrations in a given style, with all of these being "valid" in the sense that they respect the musical rules of the style. Besides, there is no objective criterion for ranking the different propositions except for personal preferences. In that regard, the task addressed in this thesis may seem ill-conditioned. However, a similar problem is faced when attempting to create polyphonic music generative systems, and addressed by reducing this complex problem to a simpler one-step predictive task. This is the approach we adopt in this thesis and that will be developed in Section 5.2

2.4.4 *Proposed approach*

In this work, we investigate parametric probabilistic models, and more particularly neural network architectures. Indeed, the strong correlations that exist between a piano score and its orchestral counterpart appear as a fertile framework for statistical learning methods. Besides, the recent developments of the

neural network framework suggest that the field is mature enough for tackling complex practical tasks such as projective orchestration. A review about neural networks and statistical learning is detailed in Chapter 3 and 4.

Neural networks necessitate to first embed a given problem inside a specific mathematical framework. Hence, we introduce the *projective orchestration task* in Chapter 5.

Statistical learning requires a large collection of reference data. In the case of projective orchestration, as we mentioned in Chapter 1, examples can be found in the repertoire, such as the orchestration by Maurice Ravel of Modest Moussorgsky piano piece *Pictures at an exhibition* (see Figure 1.2). Hence, the objective of our work is to automatically infer musical rules from the repeated observation of orchestration performed by famous composers, in order to be able to orchestrate previously unseen piano scores (see Figure 2.2). To do so, the construction of a first-of-kind database for projective orchestration is detailed in Chapter 6.

In Chapter 7, specific neural networks and energy-based models are implemented and evaluated on the projective orchestration task. Chapter 8 details further experimentations to evaluate the crucial parameters and factors for projective orchestration. Given the challenging context of projective orchestration, we introduce in Chapter 9 a novel and more informed method based on the Neural Auto-regressive Distribution Estimation method.

Finally, Chapter 10 is dedicated to the real-time implementation of the previously presented models, in which we build a live orchestral piano that can orchestrate the input of a piano player instantaneously.

Part II

STATE OF THE ART

3

ARTIFICIAL NEURAL NETWORKS

In this chapter, we review artificial neural networks and the fundamental principles of machine learning. For the sake of brevity, we restrict this discussion to the fundamental aspects used throughout this work. However, more thorough development of machine learning's theory can be found in [14] and in [45] specifically for neural networks.

Originally designed in the forties to model the behaviour of neurons inside a biological brain [74], neural networks have then become independently studied as statistical models. Presumed flaws in neuron's architecture lead to a decline of their popularity. However, a spectacular renewal of interest has been witnessed over the last ten years, due to both theoretical improvements and a dramatic increase of computational power. Indeed, neural networks recently exhibited exceptional efficiency on several seminal machine learning tasks such as classification and regression. Successful applications in image classification and generation, speech recognition and synthesis, or text generation, have drawn the attention of the artificial intelligence community [45, p.22-26]. Nowadays, neural networks are employed in a wide range of applicative domains such as biology [4, 18, 95] or astrophysics [23, 89, 100].

3.1 FEED-FORWARD ARCHITECTURES

A feed-forward neural network is composed by many simple computational units called neurons. Those neurons are connected together by weighted connections and organized by layers. A neuron in a given layer receives as input a weighted sum of the output of neurons from the previous layer. It outputs the result of its activation function applied to this input (see Figure 3.1).

Input units take the value of a data vector x which can represent any kind of information (images, text or music) and will be discrete or real valued depending on the application. The propagation of information through the l^{th} layer of the network is defined by

$$h^l = \sigma(W^l.h^{l-1} + b^l) \tag{3.1}$$

where h^{l-1} is the input to that layer and h^l its output. The layer indexing starts at $l = 1$, while the first layer is defined as the input $h^0 = x$. The output of the last layer is noted $y = h^L$. $W \in \mathbb{R}^{H \times N}$ and $b \in \mathbb{R}^H$ are learnable parameters. σ is the activation function, which is chosen to be non-linear, monotonic and differentiable. The list of candidates is long, but we can cite the *sigmoid function* for its historical importance and naturalness in binary classification problems [14, p.227-228], the *hyperbolic tangent*, widely used in temporal models, and the more recent Rectified Linear Unit (ReLU) which gained considerable popularity over the past decade for its improved performances [64].

In the standard feed-forward neural networks illustrated in Figure 3.1, the information flows from the input units through successive hidden layers and

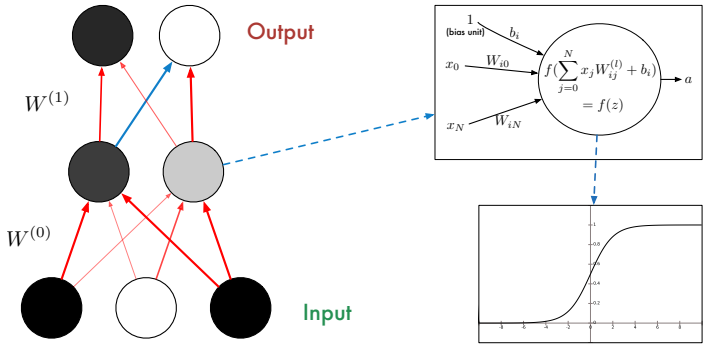

Figure 3.1: A neural network is a connectionist architecture where simple computational units called neurons are organized in layers. Layers are stacked on top of each other, and the input of a neuron is a weighted sum of the output of the all the previous layer's neurons. The weights W^l connecting the layer l and $l+1$ are graphically represented by arrows. The red arrows are positive coefficients while blue arrows are negative ones On the graph, inputs are binary values while hidden layers are real values represented in grey-scale and one can observe how hidden units represent co-activations in the input vector.

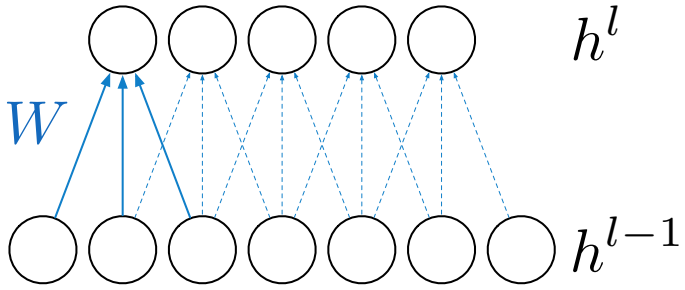

Figure 3.2: Example of a 1-dimensional convolution layer. A kernel W is convolved along the inputs of layer $l-1$. Important benefits of convolutional layers are their reduced number of weights (3 in this figure) and enforcing translation invariance of the detected patterns along the convolved axis.

up to the output layer. The depth of an architecture refers to the number of such layers. Hence, a *deep* neural network is simply an architecture with a large number of computing layers.

3.1.1 *A hierarchical pattern detector*

The weights W and b of a given layer l represent co-activation patterns between the dimensions of its input vector h^{l-1}. Hence, the output h^l can be understood as an encoding of these internal correlations. When stacking layers on top of each other, higher-level correlations are detected. Intuitively, more abstract representations of the input data are successively extracted and more complex structures are encoded as the information is processed through the network. This idea is depicted on Figure 3.1.

Depending on the nature of the data, the computation performed by hidden units may take various forms. For instance, images naturally embed translational invariance along there axes, as any given shape (for instance a cat) can

appear everywhere on a picture. This a priori knowledge about the translation invariance property of images can be used to compute hidden representations more efficiently. Thus, Convolutional Neural Networks (*CNN*) implement filters that are convolved along the image instead of computing an affine transformation of all pixels (see Figure 3.2). The original paper introducing convolutional layers traces back to 1998 [66] and tackles the digit-recognition task on the now famous *MNIST* dataset [65].

3.1.2 *Universal function approximator*

A neural network can also be seen as a function, which computes an output vector y given an input data x. This function can be written as the composition of affine transformations and non-linear mappings

$$y = \sigma\Big(\quad ... \quad \sigma(W^2.\Big[\sigma\Big(W^1.x + b^1\Big)\Big] + b^2)\Big) \tag{3.2}$$

The Cybenko theorem states that feed-forward neural networks such as the one defined in the above Equation 3.2 can approximate any function defined on any compact subset of $\mathbb{R}^n$ [26, 53].

Hence, if a problem can be stated in the form of a mapping between input and output values, then, there exists a neural network able to implement that function. Finding the set of parameters which optimally fits that ideal function is called the *training procedure* and is detailed in Section 3.3.

3.1.3 *Probabilistic formulation*

Real life data are prone to uncertainty, either due to intrinsic variability (a drawn digit admits a large number of valid shapes) or measurement errors. To account for these variability, neural networks can be embedded in a probabilistic framework. Hence, instead of outputting a deterministic value, neural networks rather define a probability distribution $p(y|x, \theta)$ conditioned over the input x and the network parameters θ.

3.2 ENERGY-BASED MODELS

Recently, traditional feed-forward neural networks have been extremely popular [45]. However, alternative architectures such as auto-encoders or energy based models provide interesting properties. Indeed, in traditional feed-forward architectures all units of a given layer are assumed to be independent from each others. In particular, this hypothesis might not be desirable for the output layer. Oppositely, energy-based models can represent the joint probability of a set of variables with no independence assumption.

More precisely, given a vector of random variables x, the probability distribution is defined as [39]

$$p(x) = \frac{1}{Z} e^{-E(x)} \tag{3.3}$$

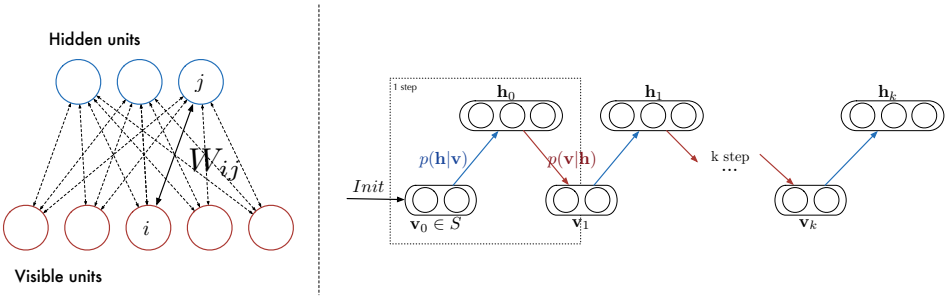

Figure 3.3: (Left) Graphical representation of a Restricted Boltzmann Machine (*RBM*). The weight W_{ij} represent the connection between visible and hidden units. Visible (respectively hidden) units are conditionally independent from each other. (Right) Gibbs sampling can be used to obtained a sample from a distribution close to the true distribution of the *RBM*. It consists in iteratively sampling the value of one unit given the others through their conditional probabilities. The independence of the hidden and visible units allows for a fast implementation known as *block Gibbs sampling*.

where $E(x)$ is called the energy function and $Z = \sum_x e^{-E(x)}$ is the partition function that ensures that probabilities sum to one. The definition of this energy function will shape the form of the modelled probability distribution.

The Restricted Boltzmann Machine (*RBM*) is a particular instance of energy-based models [39]. A *RBM* is defined by a set of m visible units $v = (v_1, ..., v_m)$ and n hidden units $h = (h(1), ..., h(n))$. The parameters of the model are the weights W_{ij} between visible and hidden units, the biases over visible units a_i and the biases over hidden units b_j (see Figure 3.3 (Left)). The energy function of a *RBM* is given by

$$E(v,h) = -\sum_{i=1}^{m} a_i v_i - \sum_{j=1}^{n} b_j h_j - \sum_{i=1}^{m}\sum_{j=1}^{n} v_i W_{ij} h_j \tag{3.4}$$

Even for a reduced number of dimensions, sampling directly from the joint distribution (Equation 3.3) is impossible because the sum over all possible configurations in the partition function is intractable. A solution consists in approximating the joint distribution by performing Gibbs sampling [14, p.542]. This method allows to obtain the joint probability by successively sampling each individual variable given the other variables. In the particular case of the *RBM*, the visible (respectively hidden) units are independent from each others when conditioned on the hidden (respectively visible) units

$$p(v_i = 1|h) = \sigma\left(a_i + \sum_j W_{ij} h_j\right) \tag{3.5}$$

$$p(h_j = 1|v) = \sigma\left(b_j + \sum_i W_{ij} v_i\right) \tag{3.6}$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function. Hence, the particular shape of these conditional probabilities allows for an efficient block sampling strategy where all the visible units are sampled at the same time given the value of the hidden units. Then, all the hidden units can be sampled given the visible units. This process can be repeated until convergence is reached (see Figure 3.3

(Right)). In practice, this may take infinite time and only a limited number of step is performed, ranging from 1 to 100 depending on the application. This efficient sampling method was introduced in [50] and is often referred to as *Gibbs block sampling*.

3.3 TRAINING NEURAL NETWORKS

The main assumption of probabilistic models is that an underlying probability distribution structures the data that we try to represent. Hence, the training procedure aims at finding the set of parameters for which a given neural network best approximates this hypothetical data distribution.

An important distinction has to be made between the feed-forward architectures and energy-based models. Feed-forward architectures model a mapping between input data x and target data y, whereas energy-based models directly represent the input data x. This difference implies distinct modelling of a same problem, and the distribution of interest is $p(y|x)$ in the first approach while it is $p(x)$ in the second case.

Hence, feed-forward neural networks can be trained in a supervised way, by observing examples of input x and corresponding target t collected from observations of the desired mapping gathered in a dataset $\mathcal{D} = \{(x, t)\}$. Energy-based models are trained in an unsupervised fashion, by the observation of a dataset composed by vector's singletons $\mathcal{D} = \{x\}$. In both approaches, the distribution of interest can only be accessed through the observation of points in $\mathcal{D}$, and its inference is conducted in fashion similar to the approximation of a function by interpolation.

This learning process requires designing an error function, which evaluates how tightly the neural network approximates the true distribution. Then, optimization techniques can be used to find the parameters' values which minimize this error function. The large number of parameters and the highly non-convex nature of the error function in the case of neural networks represent a tremendous challenge for optimization.

3.3.1 *Error function*

3.3.1.1 *Supervised models*

The error function $E_{\mathcal{D}}(\theta)$ aims at evaluating how closely the neural network $p(y|x, \theta)$ approximates the true data distribution by evaluating the model performances on a set of known data points x and corresponding targets y drawn from a dataset $\mathcal{D}$. Hence, a natural measure of a probabilistic model's performances is to evaluate how likely the data in $\mathcal{D}$ could have been generated by the model. This measure is the likelihood of a dataset under a model assumption and is defined as the product of all dataset points likelihood

$$\mathcal{L}(\mathcal{D}|\theta) = \prod_{(x,y)\in\mathcal{D}} p(y|x, \theta)$$

To simplify this product to a sum and improve numerical stability, the log of the likelihood is often used. Historically, machine learning algorithms have

been conceived as the minimization of an error function. Hence, the negative log-likelihood is preferred. Therefore, for a given set of observed data $\mathcal{D}$, the error criterion is defined as

$$\begin{aligned} E_{\mathcal{D}}(\theta) &= -\,log(\mathcal{L}(\mathcal{D}|\theta)) \\ &= \sum_{(x,y)\in\mathcal{D}} -log(p(y|x,\theta)) \end{aligned}$$

A frequent assumption is made that the dimensions of the output of a feed-forward neural network are independent. Thus, the negative log-likelihood of a single vector y is given by

$$\begin{aligned} -log(p(y|x,\theta)) &= -\,log(\prod_i p(y_i|x,\theta)) \\ &= \sum_i -log(p(y_i|x,\theta)) \end{aligned} \tag{3.7}$$

Hence, in the special case of binary targets y, the output distribution is a product of independent Bernoulli distributions and the negative log-likelihood is equal to the *binary cross-entropy*

$$-log(p(y|x,\theta)) = \sum_{(x,y)\in\mathcal{D}} \sum_i -y_i log(p_i) - (1-y_i) log(1-p_i) \tag{3.8}$$

where $p_i = p(\hat{y}_i = 1|x,\theta)$ is the probability for the unit i to be equal to one.

3.3.1.2 *Unsupervised models*

Energy-based models directly represent the probability distribution of a given variable of interest through $p(v) = \sum_h p(v,h)$. Their training process is unsupervised and an error criterion is the likelihood of a visible vector v is this time given by $p(v)$ and can be computed by marginalizing out the hidden units

$$p(v) = \sum_h p(v,h) = \sum_h \frac{exp^{-E(v,h)}}{Z} \tag{3.9}$$

At this point, the *free energy* is often introduced to lighten the notations

$$\mathcal{F}(v) = -log\left(\sum_h exp^{-E(v,h)}\right) \tag{3.10}$$

With this notation, for a given set of observations, the error criterion is defined as

$$E(\theta) = \sum_{v\in\mathcal{D}} [\mathcal{F}(v) - \mathcal{F}(\tilde{v})] \tag{3.11}$$

where $\tilde{v}$ is obtained by initializing a Gibbs chain with the sample v and computing one full step of Gibbs sampling.

This criterion is an approximation of the likelihood of a sample under a model distribution. A detailed explanation can be found in Chapter B.

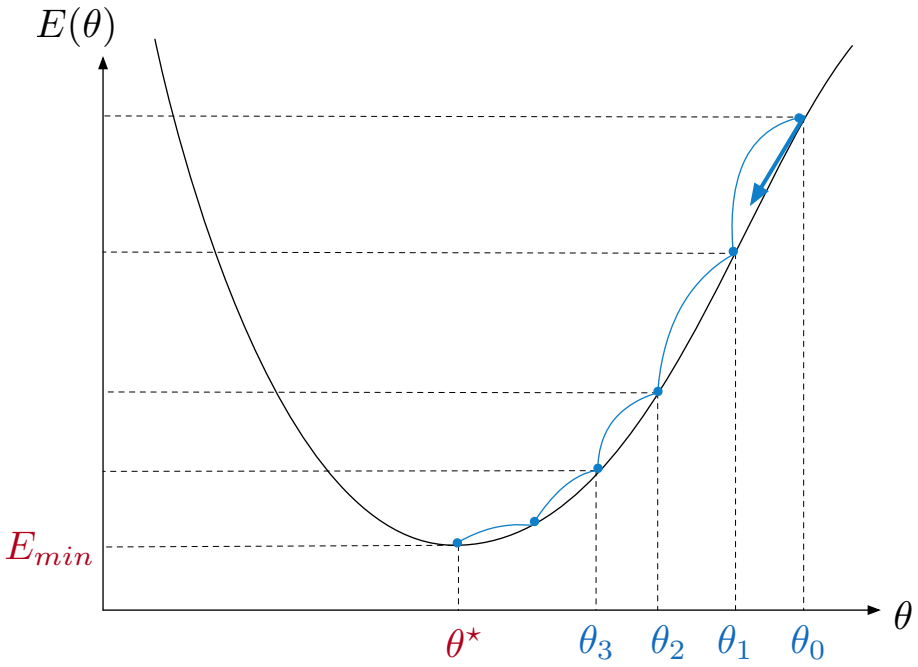

Figure 3.4: The gradient descent algorithm is a numerical iterative method for minimizing a function. Here the evolution of the algorithm is represented for a single parameter θ with error of the model $E(\theta)$. The parameter value is successively modified in the direction of the steepest gradient of the error surface curve (represented by a blue arrow) until it reaches the optimal value in red.

3.3.2 *Setting parameter values: back-propagation*

Training a neural network is defined as an optimization problem over the error function with respect to its parameters. However, the error function has a highly non-linear dependency over the model parameters (as it is a composition of several non-linear functions). Furthermore, it usually contains a large number of local minima, and their occurrences increase with the model complexity. In the end, finding a closed-form solution for minimizing the error function of an even relatively small neural network is usually impossible [14, p.236-237].

A solution to this issue is to rely on the gradient descent method, which is a numerical iterative technique [14, p.239-241]. At each step of the process, the parameters are modified in the direction which provides the largest reduction of the error function

$$\theta_{t+1} = \theta_t - \lambda \nabla(E(\theta_t)) \tag{3.12}$$

where ∇ is the gradient operator and λ is the learning rate which sets the amount of modification made at each iteration (usually set around $1e^{-3}$). A small value of λ can avoid missing (jumping over) local minima but at the cost of a longer convergence time. Visually, gradient descent can be seen as performing small jumps along the steepest slope of the error surface as depicted in Figure 3.4.

The vanilla gradient descent algorithm suffers from several flaws [45, p82-83]. Depending on the function to optimize or the current position in the parameter

space, the optimal value for the learning rate might vary widely. Besides, vanilla gradient descent can still easily be trapped in saddle-points, which happen to be frequent in the high-dimensional error functions of neural networks. For these reasons, the convergence may remain very slow for high-dimensional problems.

Recently, Kingma et al. proposed *ADAM* [59], an optimizer that proved to be particularly efficient for sparse gradients in high-dimensional spaces, which is often the case when training neural networks. *ADAM* also depends on hyper-parameter values such as the learning rate. However, unlike vanilla *SGD* and like most of the recent optimization techniques, they do not require to be carefully tuned.

3.4 TRAINING PROCESS AND HEURISTICS

3.4.1 *Early-stopping*

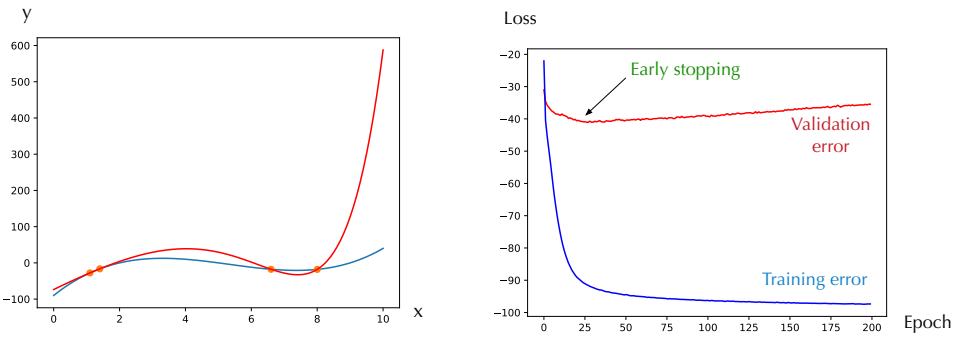

Figure 3.5: (Left) Over-fitting on a polynomial regression problem. The blue curve is the target polynomial of degree 3. The red curve represents a polynomial of degree 8 fitted over 4 points. We can see that the red polynomial perfectly approximates the original function on the 4 training points, but completely fails outside the bounds of these training points. Hence its generalization error would be important. (Right) Early-stopping can be used to prevent over-fitting. The evolution of the train (blue) and validation (red) errors along the training process are compared. After the 25th epoch, the validation error stops decreasing while the training error is still diminishing. This indicates that the model looses in generalisation power and the training process should be stopped after this epoch (green arrow).

A frequent issue when performing statistical learning is that a model over-fits the distribution of the training data and looses its generalisation ability. This phenomenon is depicted on Figure 3.5. This issue can be prevented by monitoring the performances of the model on a distinct validation set. A decrease in the performance over the validation set reveals that the model begins to over-fit the training set. Early stopping consists in monitoring the validation performances and stop the learning process when this quantity increases significantly. However, error curves rarely evolve smoothly, and making the distinction between a temporary and a consistent increase is not straightforward. Hence, early-stopping heavily rely on heuristics.

The *UP-criterion* presented in [91] appears to be a robust early-stopping criterion with a good trade-off between training time and final performance.

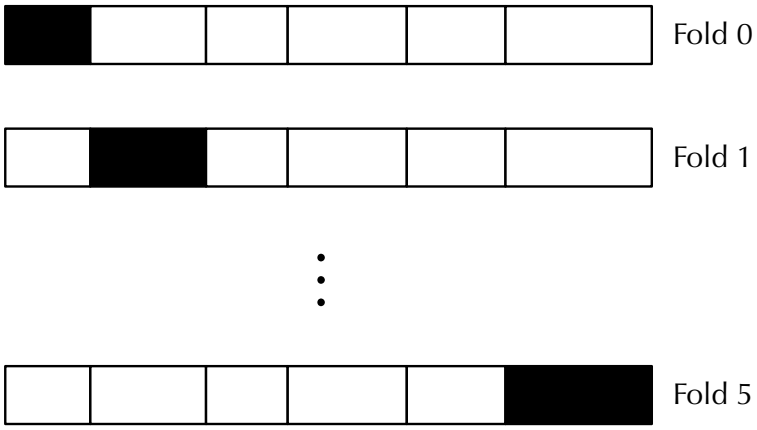

Figure 3.6: K-fold partition of a dataset with K=6. The black part is the test set kept for evaluation while the 5 other blocks are used for training. Among the $K-1$ training blocks, one would typically be used as a validation set for detecting over-fitting

Considering the evolution of the validation score over the successive iterations, the UP-criterion checks that the measure is increasing (or diminishing in the case of a validation error) on s successive strips of a certain size k. A more detailed description of the algorithm can be found in Chapter B.

Note that, in the most general case, the measure used for assessing the validation error can be different than the one used during the training process. An evaluation framework for the projective orchestration task will be detailed in Chapter 5.

3.4.2 *K-fold validation for time-series*

The validation set used to prevent over-fitting usually cannot be used to compare different models. Indeed, early-stopping is part of the training process and the validation set could be over-fitted too. For that reason, a third distinct dataset called *test set* is used to assess the performances of different trained models.

Usually, these three datasets are obtained by splitting one large collection of data into distinct subsets. (80% for the training set, 10% for the validation and 10% for the test set are usual proportions). However, when a low amount of data is available, the score of the same model can vary greatly depending on the dataset division, and, consequently, the ranking between models, which is problematic for selecting the best one. To avoid that issue, a K-fold evaluation is recommended [14, p32-33]. It consists in partitioning a dataset in K approximately equally sized parts (see Figure 3.6). The model is successively evaluated on each of the K parts after being trained on the excluded points. The pseudo-code of this training process can be found in Chapter B.

3.4.3 *Hyper-parameters search*

Hyper-parameters define the set of parameters not optimized during the training process. Hyper-parameters will vary depending on the model considered,

but typical examples for neural networks architectures would be the number of units and layers, the weight decay coefficient or dropout probability.

Common practices in machine learning is to perform either a grid search or a random search on the hyper-parameters space [45, p.422-429]. A requisite of these non-informed methods is that a sufficient number of configurations are evaluated to densely sample the hyper-parameter space. However, the required number of configuration increases exponentially with the number of hyper-parameters.

Hence, Bergstra et al. introduces an hyper-optimization search framework based on Bayesian optimisation [8]. Essentially, the search procedure consists in building a probability distribution $p(score|configuration)$, which relies on a set of priors for each parameter and is iteratively updated as more pairs (score, configuration) are evaluated. To speed-up the process, the next configuration to be evaluated is chosen among the most promising points in the hyper-parameters space (those with a large probability to obtain a high score). This framework has been implemented in a Python package named *Hyperopt*[1].

1 https://github.com/hyperopt/hyperopt

4

DEEP LEARNING FOR AUTOMATIC PROJECTIVE ORCHESTRATION

In this chapter, more elaborate structures with features particularly relevant to the context of automatic projective orchestration are presented. First, time dependencies require a special care when attempting to generate musical sequences. Hence, Section 4.1 introduces deep learning models dedicated to temporal data. Second, in the projective orchestration task, the piano score and its orchestral counterpart are strongly correlated. Conditioning mechanisms, which allow to modify the behaviour of a model depending on an external set of conditioning data (in our case the piano score) are explored in Section 4.2. Finally, Section 4.3 is dedicated to auto-regressive models, developed for modelling dependencies between the units of the predicted vector.

4.1 TIME MODELLING IN NEURAL NETWORKS

Structuring patterns in musical scores can occur at different time scales with related events separated by long temporal intervals. Hence, modelling musical series represents a daunting task for machine learning.

The *receptive field* of a computational unit designates the extent inside the input data that this unit can access to. By extension, we can designate the receptive field of a model as the receptive field of its last layer, which delineates all the information available to the model for making a prediction. This concept is crucial in time series modelling as it delimits the temporal regions from which a model can search for correlations. A naive approach to time series modelling could be to input several successive frames of the series to a feed-forward neural network. However, as we observe on Figure 4.1, with this approach, the receptive fields have a fixed length, and increasing their size comes at the price of a concomitant increase in the number of parameters.

4.1.1 *Recurrent architectures*

In order to efficiently increase the size of the receptive field, state space models have been proposed [14, p.609-610]. At each time step t, the information about

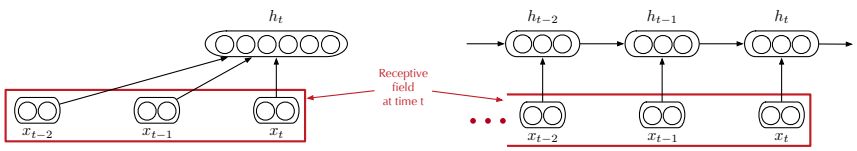

Figure 4.1: (Left) A time series processed with a standard feed-forward neural network. The receptive field of the model has a fixed temporal horizon which can be increased only at the cost of more parameters. (Right) State space models encode information about all the previous frames of the input time series in hidden vectors h.

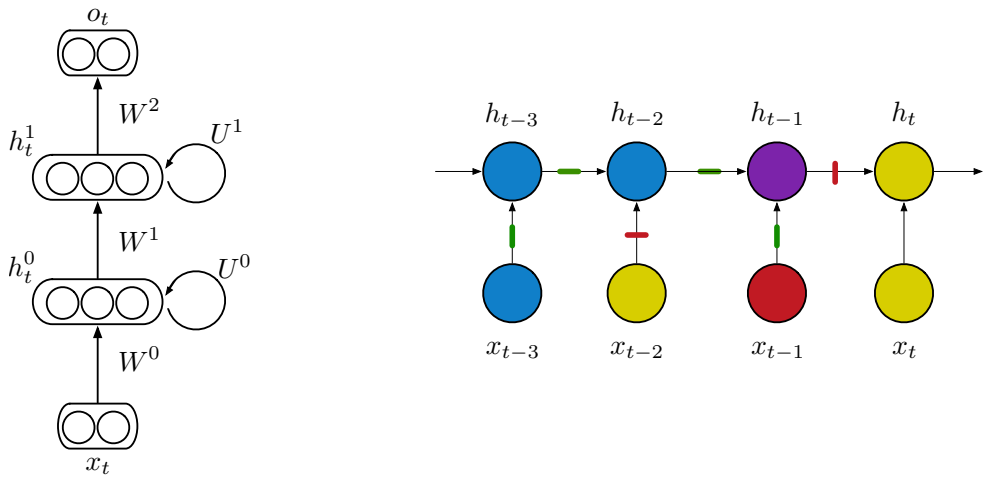

Figure 4.2: As weights are time-invariant, a more compact representation of the graph can be obtained by representing recurrent weights U with a curved arrow. Similarly to the *MLP* architecture, layers can be stacked on top of each others in order to extract successively more abstract representation.
The gating mechanisms allow to control the information flowing through the model along time (output nodes are omitted for clarity). The input at x_{t-2} is blocked and the hidden state is preserved. Between $t-1$ and t, the hidden state is reset. The colours illustrate how the different informations are selected and combined.

all past events of the time series is encoded in a single vector h_{t-1}. Hence, this vector evolves as a function of its previous value h_{t-1} and the current frame of the input time series x_t

$$h_t = f(h_{t-1}, x_t) \tag{4.1}$$

where f is a function (which does not depend on time). Interestingly, state space models can theoretically access all of the previous samples in the input time series (see Figure 4.1), while maintaining a reasonable number of trainable parameters.

Recurrent neural networks are a type of state space model. They extend the feed-forward architecture by implementing the function f in Equation 4.1 as

$$h_t = \sigma(W.x + U.h_{t-1} + b) \tag{4.2}$$

were parameters are similar to Equation 3.1 but the temporal information is added through vector h_{t-1} transformed with parameter matrix U. Similarly to simple feed-forward architectures presented in Section 3.1, recurrent layers can be stacked on top of each others in order to extract higher-level representations (see Figure 4.2).

4.1.2 *Modelling longer time dependencies*

Although state space models allow for theoretically infinite receptive fields, long-term correlations are actually diluted along the recurrent connections by the input time series. To alleviate this problem, an architecture called *Long Short-Term Memory* (*LSTM*) units was proposed in 1997 by Hochreiter and

Schmidhuber [51]. Each *LSTM* unit is composed by a cell c which stores a value akin to an internal memory. Three gates allow to control how the information is flowing through the unit by implementing forget f, input i and output o mechanisms. These gates scale the different values which enter and leave the cell. f controls the cell's value at previous time steps, i controls the input vector and o the value going to the output [47]:

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f c_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i c_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o c_{t-1} + b_o) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \sigma_c(W_c x_t + U_c c_{t-1} + b_c) \\
h_t &= o_t \odot \sigma_h(c_t)
\end{aligned}
$$

where matrices W_x, U_x and vectors b_x are trainable weights, $\odot$ is the element-wise (or *Hadamard*) product, and σ_g, σ_c and σ_h are non-linear activation functions. A usual choice is to use a sigmoid for σ_g and an hyperbolic tangent for σ_c and σ_h. These gating mechanisms provide different useful operations for modelling long-term dependencies. The internal memory of the cell can be reset if f_t is null, the input information x_t can be bypassed if i_t is null, and the output disabled if o_t is null (see Figure 4.2)

Many variations around the *LSTM* units have been developed with the *Gated Reccurent Units* (*GRU*) being one of the most popular [22]

$$
\begin{aligned}
z_t &= \sigma_g(W_z x_t + U_z h_{t-1} + b_z) \\
r_t &= \sigma_g(W_r x_t + U_r c_{t-1} + b_r) \\
h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot \sigma_h(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h)
\end{aligned}
$$

Here, the internal state is removed and the hidden units h are directly propagated along time. Besides, the input and forget gates are merged as a single vector z_t which acts as a balance between the previous state value and the new candidate.

Unfortunately, contradictory results have been found in different empirical studies. Chung et al. suggests that *GRU*s lead to better performances [22] than the *LSTM*, while a more recent study claims the opposite [47]. Overall, results seem sensibly identical and we did not observe radical differences in terms of performances between the two architecture (see Section C.1). However, *GRU* provides a lower amount of parameters and we will use them in the rest of this thesis unless specified.

4.1.3 *Back-propagation through time*

Gradient-based methods for updating the weights of a network can easily be adapted to recurrent models. A solution consists in *Back-Propagating Through Time* (*BPTT*) the error calculated at each time frame [84] (see Figure 4.3, where the model has been unrolled along time axis). Mathematically, if we consider a one-layer recurrent neural network, and a parameter θ which is either a recurrent connection or an input weight (either U or W in Equation 4.2), the derivative of the error function E is given by

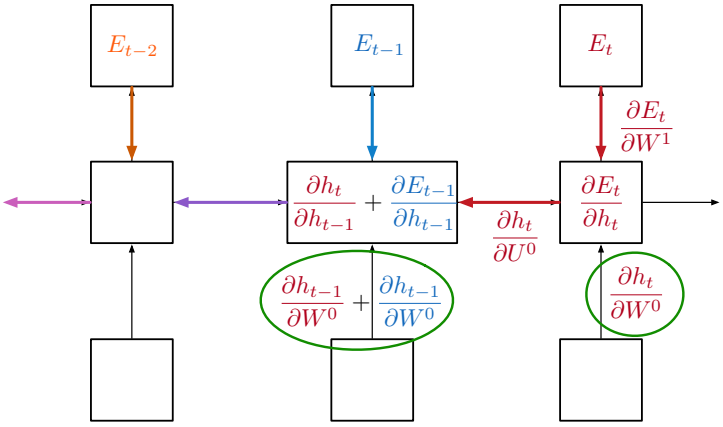

Figure 4.3: The *Back-Propagation Through Time* (*BPTT*) algorithm allows to train recurrent architecture. By unrolling along the time axis the recurrent network, we can observe the multiple influence of each weight matrix over the error function at each time-step. Hence, back-propagating each error term E_t for t in $[0, ..., T]$ along recurrent connections allows to modify the weights considering this influence over time. For instance, all the terms framed with green circles will contribute to the update of W^0.

$$\begin{cases} \frac{\partial E}{\partial \theta} = \sum_{1 \leq t \leq T} \frac{\partial E_t}{\partial \theta} \\ \frac{\partial E_t}{\partial \theta} = \sum_{1 \leq k \leq t} (\frac{\partial E_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial \theta}) \\ \frac{\partial h_t}{\partial h_k} = \prod_{t \geq i > k} \frac{\partial h_i}{\partial h_{i-1}} \end{cases}$$

The chain terms $\frac{\partial h_t}{\partial h_k}$ account for the propagation of the gradient backward in time (see the horizontal chain in Figure 4.3). As they define a geometric series, the values of the chain term can either shrink to zero or explode if their successive values are smaller or greater than 1. These phenomena are called respectively *vanishing* and *exploding* gradients. Exploding gradients can suddenly push the parameters very far from their current position, possibly in a zone where the error is much larger. Reciprocally, vanishing gradients can prevent long-term correlations to be learnt.

The gating mechanisms implemented in *LSTM* and *GRU* help fighting these diverging gradients. Indeed, in the case of *GRU*, if the z gate is locked to 1 between two distant time indices t_a and t_b, then the derivative between these two states is equal to one ($\frac{\partial h_{t_b}}{\partial h_{t_a}} = 1$) and the gradient is preserved during the propagation.

However, gating mechanisms are not a sufficient guarantee against gradient's divergence. The risk increases with the sequence length, as the size of the chain

and, thus, the number of terms grows. A solution called *gradient clipping* [84] consists in normalizing the gradient by its norm when it exceeds a threshold

$$\hat{g} = \frac{K}{||g||} g \quad if \quad ||g|| > K \tag{4.3}$$

where $g = \nabla E(\theta)$ is the gradient of the error function with respect to the model's parameters, $||.||$ is a matrix norm (typically the L2-norm), and K is a threshold value (typically set to 1).

4.2 CONDITIONING MECHANISMS

A number of canonical problems in artificial intelligence require the ability to condition a generative model under a set of observed inputs, such as text-to-speech generation, image captioning or question answering [29]. Implementing a conditioning mechanism usually boils down to finding a way to modulate the behaviour of a model depending on the value of a conditioning vector z. Depending on the type of conditioning information, z can take different forms, such as one-hot encodings for representing mutually exclusive class-belonging, or three-dimensional real-valued vectors accounting for coordinates in space. Note also that, in the case of time series conditioning, the condition itself depends on the time frame and is denoted z_t.

A major challenge in conditioned generative models is to factor the information and structures which are shared across different conditioning. For instance, a model generating music in the style of Mozart and another model specialized in Beethoven probably both learnt structures which could be informative for the other. Various sources of information can be encoded by distinct sub-networks (or sub-modules) and then combined through concatenation. For inputs x and conditioning z, intermediate representations $\tilde{x}$ and $\tilde{z}$ are computed by sub-networks f_x and f_z, These representation are concatenated and processed by the network g which predicts the output $\hat{y}$

$$\begin{cases} \tilde{x} = f_x(x) \\ \tilde{z} = f_z(z) \\ \hat{y} = g(\tilde{x}, \tilde{z}) \end{cases}$$

4.2.1 *Feature-wise conditioning*

Bringing this idea further, the Feature-wise Linear Modulation *FiLM* is a general and powerful framework for feature-wise conditioning [87]. The main idea is to modify the values of intermediate hidden representations (also called features) by shifting and scaling them by variables computed from the conditioning vector. If h is a hidden representation extracted at any point of the model, its value can be modulated as

$$\hat{h} = \gamma(z) \odot h + \beta(z) \tag{4.4}$$

where $\gamma(.)$ and $\beta(.)$ can be any transform of the conditioning vector, typically defined as neural networks with trainable parameters. (see Figure 4.4). This

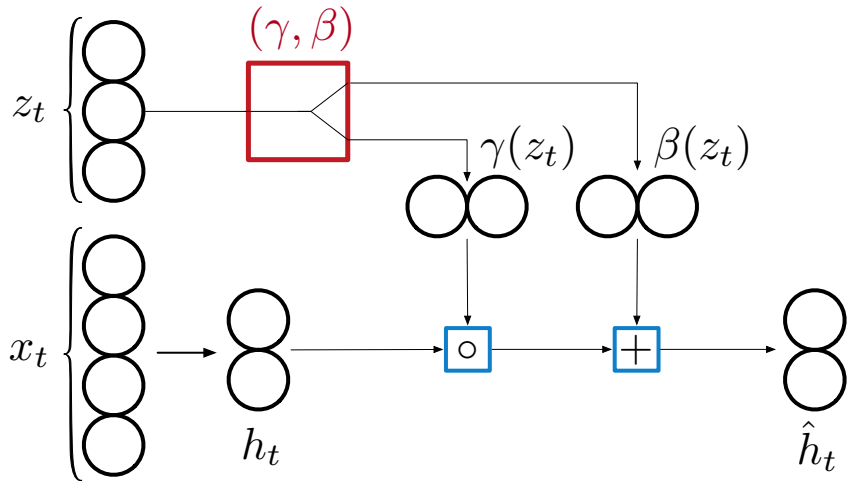

Figure 4.4: The *FiLM* conditioning mechanism applies a linear transformation to each unit of a feature vector h. Thus, any intermediate hidden layer in the network can be modulated by different *FiLM* layers.

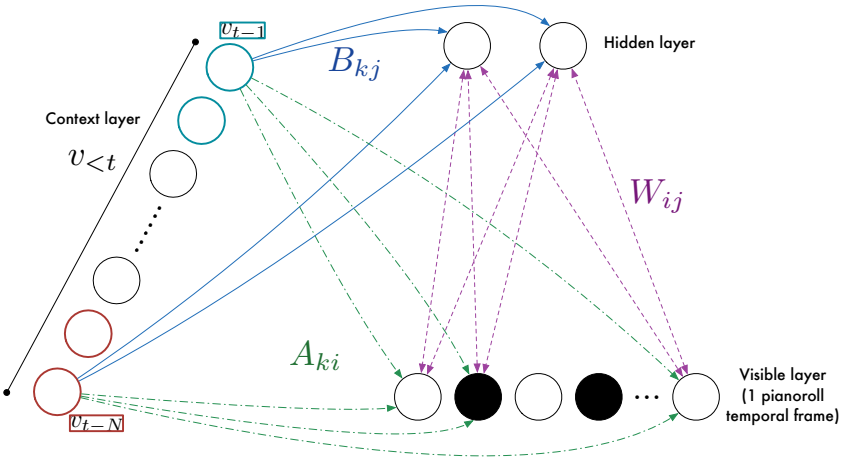

Figure 4.5: *CRBM*. The weights A_{ki} and B_{kj} model the influence of the past visible states on the biases of the current visible and hidden units.

equation defines a *FiLM* layer [87] and can be added after any hidden representation, but different functions γ and β might be used if the conditioning occurs at several places in the network.

4.2.2 *Conditioning for RBMs*

The strict theoretic framework of *RBM*s does not allow for feature-wise conditioning. However, a solution lies in modifying directly the weights of the network instead of the computed features. In a *RBM* the shape of the density function $p(x)$ depends on the weight values. Hence, modulating these weights modifies the shape of the energy function and, consequently, the distribution of the visible units.

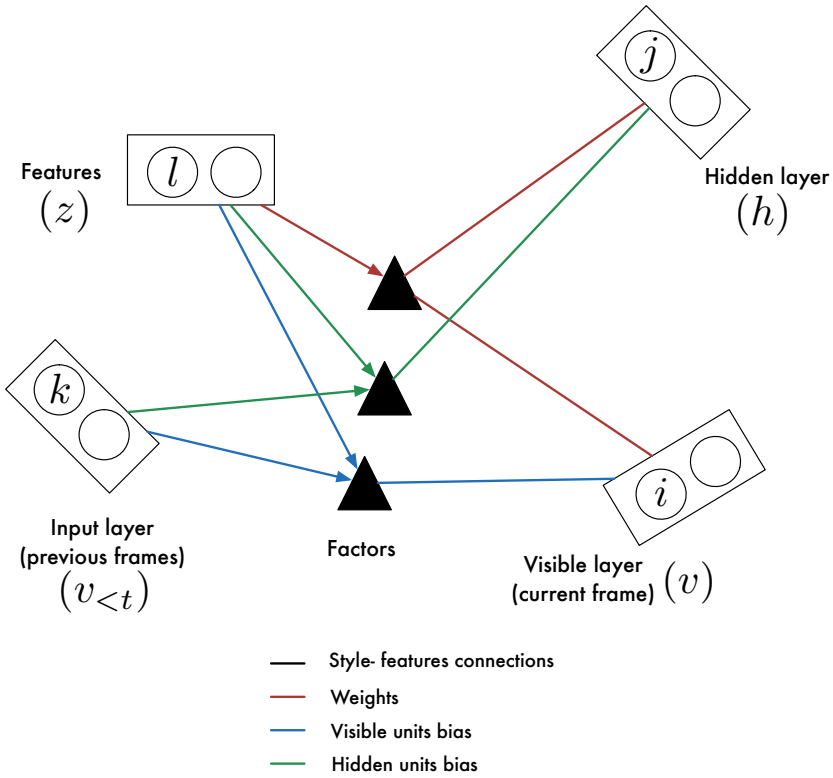

Figure 4.6: A style-gated FGcRBM. The three sub-models are represented by three different colors. The style features are gated on the three interactions: weights between visible and hidden units (in red), bias on hidden units (green), bias on visible units (blue)

4.2.2.1 *cRBM: modulating biases*

The conditional Restricted Boltzmann machine (*cRBM*) is a conditioned generative model proposed in [112] for modelling an animated character walking with different motion styles. A set of conditioning labels can modulate the behaviour of a standard *RBM* by dynamically modifying its biases. Thus, the energy function of a *cRBM* is given by

$$E(v_t, h_t | v_{<t}) = -\sum_i \hat{a}_{i,t} v_{i,t} - \sum_{ij} W_{ij} v_{i,t} h_{j,t} - \sum_j \hat{b}_{j,t} h_{j,t} \tag{4.5}$$

where the biases are defined by $\hat{a}_i = a_i + \sum_k A_{ki} v_{k,<t}$ and $\hat{b}_j = b_j + \sum_k B_{kj} v_{k,<t}$. Here, a and b are static biases as in the *RBM* model, and A_{ki} and B_{kj} model the influence of the context on the visible ad hidden states respectively (see Figure 4.5). The same inference and generative methods as in a standard *RBM* (see Section 3.2) can be performed by simply replacing the static biases a and b by their dynamic counterparts.

4.2.2.2 *FGcRBM: a multiplicative influence over bidirectional weights*

Instead of modifying the biases, another conditioning mechanism consists in modulating the bi-directional weights W [112]. Hence, the energy function becomes

$$E(v,h|x) = -\sum_{ijk} W_{ijk} v_i x_k - \sum_{ij} c_{ij} v_i h_j - \sum_i a_i v_i - \sum_j b_j hj \tag{4.6}$$

where W_{ijk} are the components of a 3-dimensional tensor and c_{ij} are gated biases.

The computational cost implied by the manipulation of 3-dimensional tensors can be reduced by factoring W into a product of pairwise interactions such that $W_{ijk} = \sum_f W^v_{if} W^h_{jf} W^x_{kf}$ (where the superscript indicates which unit the weight refers to and f indexes the factors). If $I \approx J \approx K \approx N$, this assumption over the shape of W leads to a decrease in the number of parameters from N^3 to $3N^2$. Besides, gated biases c can also be abandoned further reducing the model complexity.

When modelling time series, an other proposed improvement [112] is to separate the influence of the recent past events from the conditioning units. Hence, the recent past is modelled using dynamic biases, and the labels z_t are gated to both the bi-directional weights W and the dynamic matrices A and B. Finally, the energy function of this *Factored-Gated cRBM* (*FGcRBM*) is equal to

$$E(v_t,h_t|v_{<t},z_t) = -\sum_f \sum_{ijl} W^v_{if} W^h_{jf} W^z_{lf} v_{i,t} h_{j,t} z_{l,t} - \sum_i \hat{a}_{i,t} v_{i,t} - \sum_j \hat{b}_{j,t} h_{j,t} \tag{4.7}$$

where the dynamic biases of the visible and hidden units are defined by

$$\begin{aligned}\hat{a}_{i,t} &= a_i + \sum_m \sum_{kl} A^v_{im} A^{v<t}_{km} A^z_{lm} v_{k,<t} z_{l,t} \\ \hat{b}_{j,t} &= b_j + \sum_n \sum_{kl} B^h_{jn} B^{v<t}_{kn} B^z_{ln} v_{k,<t} z_{l,t}\end{aligned}$$

where m indexes the factor for the gated interactions between features, input and visible units and n the factor for the interactions between features, input and hidden units. The training process is similar to the *RBM*'s procedure. However, the conditional probabilities are now expressed by

$$p(v_i = 1|h,c,z) = \sigma\left(\hat{a}_i + \sum_f W^h_{jf} \sum_{ik} W^v_{if} v_i W^z_{lf} z_l\right) \tag{4.8}$$

$$p(h_j = 1|v,c,z) = \sigma\left(\hat{b}_j + \sum_f W^h_{jf} \sum_{ik} W^v_{if} v_i W^z_{lf} z_l\right) \tag{4.9}$$

A graphical representation of this model can be observed on Figure 4.6.

4.3 NEURAL AUTO-REGRESSIVE DISTRIBUTION ESTIMATION

In Section 3.1, we introduced the feed-forward neural networks architecture whose output variables can model independent Bernoulli distributions when a sigmoid activation is used in the last layer (see Equation 3.7). In some cases,

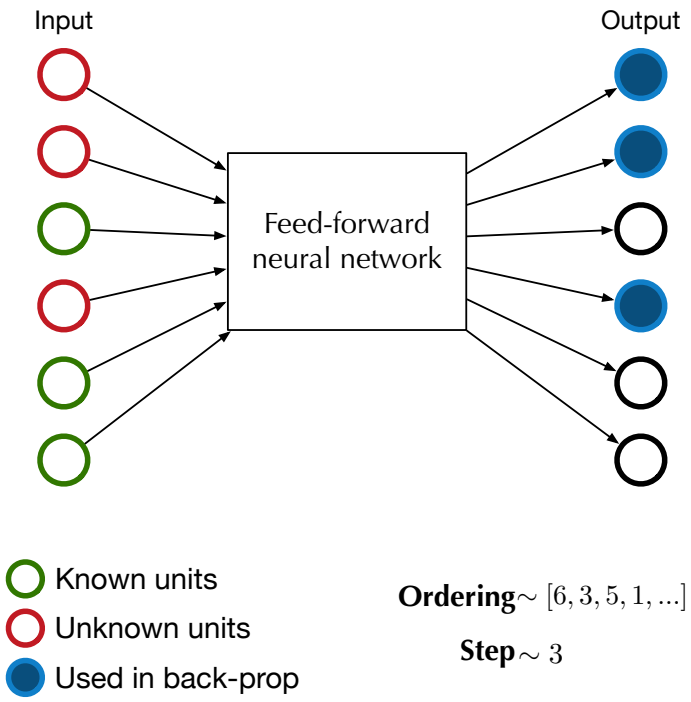

Figure 4.7: The training procedure of the *Orderless Deep Neural Auto-regressive Distribution Estimation ODNADE* consists in randomly sampling an ordering and an order (or step). These determine the known units in the input (green circles) and those unknown which are masked out (red circles). The opposite mask is applied to the error function so that only the unknown units are used for back-propagation (blue circles).

this independence assumption over the dimensions of the output may not be justified. For instance, when considering the case of image generation, the intensity of pixels in an image may strongly depend on the value of the surrounding pixels.

Neural Auto-regressive Distribution Estimation (*NADE*) [62] has been proposed for modelling any joint probability of a set of variables by using neural networks. *NADE* is based on a solid theoretical framework. For the sake of brevity, only the algorithm is presented here without its theoretical justification. We refer interested readers to [115] for a thorough explanation.

NADE on the chain rule, which states that any joint distribution over a set of variables $\{y_i, i \in [1, D]\}$ can be decomposed in a product of conditional distributions

$$p(y_1, ..., y_D) = \prod_{d=1}^{D} p(y_{o_d} | y_{o_{<d}}) \tag{4.10}$$

where o is a permutation in $[1, D]$, and $o_{<d}$ designate its $d-1$ first elements. In theory, any ordering can be chosen.

In principle, each conditional probability $p(y_{o_d} | y_{o_{<d}})$ could be modelled by a different feed-forward neural network. Instead, the main idea of *NADE* is to share the same unique network between all conditionals. This *weight sharing* assumption amounts to consider that any two variables are always related in the same manner, independently of the ordering.

In practice, the conditioning vector $y_{o_{<d}}$, whose length $|o_{<d}|$ depends on the step d, is fed to this unique network through a fixed-size input vector x. The vector $x_{o_{<d}}$ can be set to the known values $y_{o<d}$, while the other values $x_{o_{\geq d}}$ are set to zero. This idea is implemented using a binary mask $m_{o_{<d}} \triangleq [1_{1 \in o_{<d}}, ..., 1_{D \in o_{<d}}]$, which indicates the units already that are already predicted at step d, and allows to mask in or out the known or unknown information in the input vector. Finally, for a data vector x, an ordering o and a network composed by L layers denoted each by h^l, a feed-forward pass can be derived as

$$\begin{cases} h^0 = & x \odot m_{o_{<d}} \\ a^l = & W^l h^{l-1} + b^l \\ h^l = & \sigma(a^l) \\ h^L = & sigmoid(a^L) \end{cases} \tag{4.11}$$

where σ is any activation function (typically a *ReLU*) (see Figure 4.7). Finally, the probability distribution of the unit o_d is given by $p(y_{o_d} | x) = h^L_{o_d}$.

4.3.1 *Orderless NADE*

Training a neural network with the process described above is slow, as only a scalar value $h^L_{o_d}$ is predicted for a complete forward pass and, thus, can be used for back-propagation. An efficient training procedure called Orderless Deep *NADE* (*ODNADE*) has been proposed in [115]. It relies on two essential ideas. First, it uses all the unknown units $h^L_{o_{\geq d}}$ for computing the error function instead of a single unit. Second, the ordering o and order d are considered as stochastic

variables over a uniform distribution, and the error function is computed as an expectation over the ordering and order.

As discussed earlier, any ordering can be used in the chain rule decomposition of a joint probability (see Equation 4.10). Hence, at a given step d, all the indices that are not in $o_{<d}$ are potential candidates for o_d, and all units $h^L_{o_{\geq d}}$ can be used for computing and back-propagating the error function.

When considering the ordering and order as random variables, the error function in a *NADE* architecture is the expectation of the negative log-likelihood under the ordering and order distributions. This quantity is intractable in most cases, and an approximation is obtained by sampling both o and d.

Finally, in the case of binary output units, an ordering and order are sampled at each iteration, and the negative log-likelihood is computed as a masked binary cross-entropy

$$\begin{cases} o \sim & \mathcal{U}(\mathfrak{S}_D) \\ d \sim & \mathcal{U}(1, D) \\ E(\theta) = & \frac{D}{D-d+1}(1 - m^T_{o<d}) \left[-x \odot log(h^L) - ((1-x) \odot log(1-h^L)) \right] \end{cases}$$

where $m_{o<d}$ is a binary mask, $\mathcal{U}$ the uniform distribution, and $\mathfrak{S}_D$ all the permutations over D elements $\frac{D}{D-d+1}$ normalises the error function by the number of units used for computing the error, which depends on the order d.

The masking mechanism does not allow the model to distinguish between zero-valued inputs or missing values. To alleviate this issue, it is recommended to concatenate the mask to the input, thereby replacing h^0 in Equation 4.11 by

$$h^0 = (x \odot m_{o_{<d}}, m_{o_{<d}})$$

If, theoretically, any order can be chosen for decomposing the joint probability of the output variables, in practice, conditionals are only approximated during the training process, and different orderings will not assign the same probability to a given output value. To alleviate this issue, it is recommended to use an ensemble of randomly chosen orderings $\{o^k\}_{k=1}^K$ and use the average probability $\frac{1}{K}\sum_k (h^L)^k$ as an estimator [115].

Part III

SYMBOLIC PROJECTIVE ORCHESTRATION

5 THE PROJECTIVE ORCHESTRATION TASK

In this section, we introduce and formalise the projective orchestration task. In Section 5.1, we derive a mathematical framework, in which the projective orchestration task is defined as a mapping between two multidimensional time-series. In Section 5.2, we introduce a probabilistic formulation of the task adapted to statistical learning approaches. Finally, in order to evaluate and compare different models, we propose a one-step predictive task and discuss its relevance in Section 5.3.

5.1 MATHEMATICAL FORMULATION

Here, we consider the piano-roll representation (see Figure 5.1) of a musical score, which can be interpreted as a multidimensional time-series. We denote $p = (p_1, ..., p_{L_p})$ and $o = (o_1, ..., o_{L_o})$ the piano-rolls of a piano and orchestral scores. The subscript indices represent time and L_p and L_o are the respective length of the two time-series. Each piano frame p_t is a real-valued vector defined on $[0, 1]^{N_p}$, where N_p represents the tessitura of the piano (typically 88 notes). Hence, $p_t[i]$ is a real value between 0 and 1 representing the intensity of the pitch i at time t, 0 representing a note off and 1 the maximum intensity. Each orchestral frame o_t is defined on $[0, 1]^{N_o}$ where N_o is equal to the sum of the tessitura of the different instrument of the orchestra. In this framework, the projective orchestration of a piano score if a mapping $f : p \longmapsto o$ between an input piano time-series p and an output orchestral time-series o (see Figure 5.1).

5.1.1 *Preserving the overall structure*

In the most general setting, the length of the orchestral sequence L_o is not necessarily equal to the length of the input piano score L_p. For instance, an orchestral score can be longer than the original piano score if repetitions are inserted in the orchestral version. However, this thesis focuses on mapping an existing melodic, rhythmic and harmonic structure to a given orchestra without modifying it. Therefore, we purposely restrict ourselves to an orchestration which does not alter the temporal organisation of the original piece. Consequently, the two time-series p and o will share the same length $L \triangleq L_p = L_o$.

Besides, for a given time index t, the piano frame p_t and the orchestral frame o_t share the same harmonic and melodic function. In other words, there is a frame-to-frame correspondence between the two time-series p and o.

5.1.2 *Binary representation*

We consider a simpler orchestration problem, which consists solely in deciding which notes of the orchestral score are played but not their intensities. Hence, the orchestral time-series becomes a binary multidimensional time-series, and

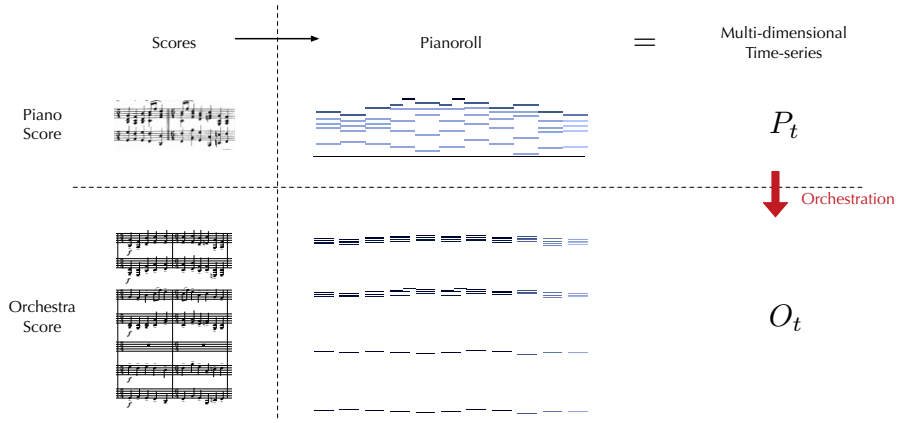

Figure 5.1: Projective orchestration is defined as the mapping between a multidimensional time-series representing a piano score and a multidimensional time-series representing an orchestral score.

each frame o_t is now a binary vector defined on $\{0,1\}^{N_o}$. We argue that the operations of deciding which notes will be played and their intensities can be performed in two successive steps, and we focus only on the first one.

However, the situation is different regarding the piano score, since its dynamics structure can dramatically impact the choice of instruments used in its orchestration. For example, a *fortissimo* passage may combine the powerful timbres of the trumpets, trombones and a bass trombone doubling at an octave lower [61, p.293], whereas a different choice of instruments will probably be used for orchestrating a *pianissimo*. Hence, the dynamics information of the piano score is theoretically important, and will be studied in detail in Chapter 8.

5.2 SPECIFYING THE TASK FOR STATISTICAL LEARNING APPROACHES

In the following of this thesis, the upper case will denote random variable, while the lower case designates values. To keep notations uncluttered, we use the unambiguous notation $p(x)$ for denoting $p(X = x)$, the probability for a random variable X to take the value x.

As discussed in Chapter 1, a single piano score can be orchestrated in many different ways. Sampling from a probability distribution provides an elegant and efficient mechanism for emulating this variability. Hence, for an input piano score p, we consider a model f which outputs a probability distribution over all possible orchestrations $f(p) = p(O)$, where O has the same length as p.

Modelling the joint probability distribution of the successive frames of an orchestral time-series given a piano score $p(O) = p(O_1, ..., O_L|P)$ is most likely intractable. Hence, the problem is frequently simplified by modelling independently each orchestral frame O_t. We present two different ways achieve this: a *causal* and a *random walk* decomposition (see Figure 5.2). Both approaches are presented in Subsection 5.2.1.

At training time, both piano and its orchestration by a famous composer are accessible. Thus, the rich information contained in the known orchestral score can be advantageously used to further simplify the task. This is not the

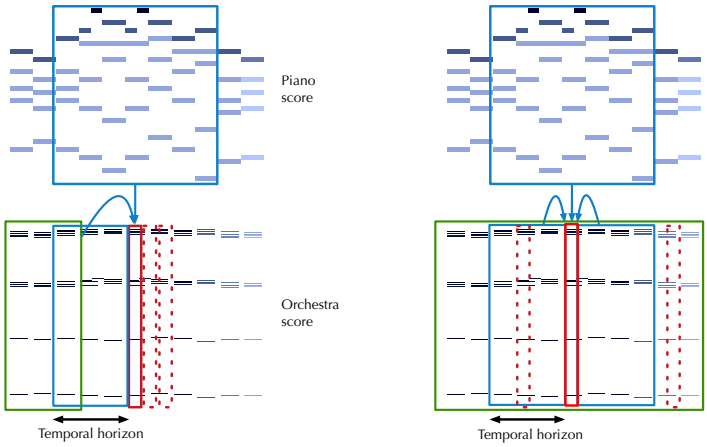

Figure 5.2: *Causal* (left) and *random walk* (right) generation processes. The temporal horizon for the contextual information of both the piano and orchestral scores is delimited by the black arrow. The contextual information (blue frames) is used to generated the current orchestral frame (red frame). The next orchestral frames to be generated are represented with dotted frames. Hence, one can observe that in the causal case, the orchestral frames are successively generated, while a random-walk over the temporal indices is performed in the second case. The random-walk generation process implies two major differences: the future of the orchestral score can be used in the contextual information (blue frame), but all the orchestral score needs to be initialised (green frames).

case during the generation step, as a reference orchestral score no longer exists. Hence, we review the differences between the two steps in Subsection 5.2.2.

5.2.1 *Generative schemes*

5.2.1.1 *Causal decomposition*

The chain rule states that a joint probability can be decomposed into a product of conditional probabilities over any ordering of the variables. For time series, causal ordering appears as a natural decomposition

$$p(O_0, ..., O_L|P) = \prod_{t=0}^{L} p(O_t|P, O_{<t})$$

Ideally, the whole piano score P and orchestral past $O_{<t}$ would be used for inferring the orchestral frames O_t. However in practice, when L is large, only fragments of these time-series are used to limit the computational cost. We denote as $X_{t_a:t_b}$ the sequence extracted from the time series X between time indices t_a and t_b and we define the piano context as $P^c_t = P_{t-T:t+T}$ and the orchestra context as $O^c_t = O_{t-T:t-1}$ where T defines the temporal horizon of the model (the value of this parameter will be discussed in Section 8.1). Using a reduced piano context centred around the same time index t is only possible because we hypothesized in Subsection 5.1.1 that the piano and orchestral scores share the same structure.

$$p(O_0, ..., O_L|P) \approx \prod_{t=0}^{L} p(O_t|P_t^c, O_t^c)$$

Besides, when orchestrating a piano score from scratch, the contextual information on the orchestral score is obtained from the orchestral frames previously generated by the model or initialised, that we denote $\widetilde{o}_t^c$. Finally, for a model f whose output represents the individual distribution $f(P_t^c, O_t^c) = p(O_t|P_t^c, O_t^c)$, the causal generation process consists in sampling $\widetilde{o}_t$ for the successive values of time index, and is described by the following algorithm

for $t \in [1, L]$ **do**
  $\widetilde{o}_t^c = \widetilde{o}_{t-T:t-1}$
  $\widetilde{p}_t \sim f(p_t^c, \widetilde{o}_t^c)$
end

Different initialisation strategies can be adopted for time indices smaller than the temporal horizon T and will be discussed in Section 5.2.2.2.

5.2.1.2 *Random walk generation*

The causal decomposition does not accurately reflect human approach to orchestrating a piano score, which is rarely a linear process in time. Indeed, a composer often writes a first draft of the piece and then goes back an forth to perform corrections while incrementally refining the score. That writing strategy can be approximately rendered by re-sampling orchestral frames at time indices randomly chosen, in a process akin to a random-walk along the time axis. Then, for each time index chosen, all contextual information is used, and the orchestral context is now equal to $O_t^c \triangleq (O_{t-T:t-1}, O_{t+1:t+T})$

Hence, the random walk generative process consists in repeatedly sampling the time index from a uniform distribution $t \sim \mathcal{U}(1, L)$ and update the orchestral frame at time t with the sampled value $\widetilde{o}_t$.

for $i \in [1, N_{sample}]$ **do**
  $t \sim \mathcal{U}(1, L)$
  $\widetilde{o}_t^c \triangleq (\widetilde{o}_{t-T:t-1}, \widetilde{o}_{t+1:t+T})$
  $\widetilde{o}_t \sim f(P, \widetilde{o}_t^c)$
end

The whole orchestral score needs to be initialised before the generation process, and the initial values are progressively replaced by the sampled values. The crucial initialisation step is discussed in Section 5.2.2.2.

Finally, both the *causal* and *random-walk* generation boil down to building a probabilistic model f of individual orchestral frames

$$f(p_t^c, o_t^c) = p(o_t|p_t^c, o_t^c) \tag{5.1}$$

The only differences are the definition of the orchestral context and the generative process. Hence, similar models can be implemented in place of f for the two approaches, and we detail our proposition in Chapter 7.

5.2.2 *Differences between training and generating*

5.2.2.1 *Training with teacher forcing*

We explained in Chapter 3 that the parameters of a parametric probabilistic model can be inferred by minimising an error function which is equal to the negative log-likelihood of a training dataset $\mathcal{D}_{train}$. In the case of projective orchestration, the dataset contains piano scores and their corresponding *ground-truth orchestration* written by famous composers $\mathcal{D}_{train} = \{(p, o)\}$.

Both the causal and random walk decompositions detailed in the previous subsection lead to the probabilistic independence of the orchestral frames, and a negative log-likelihood over the training dataset which is equal to

$$E(\theta, \mathcal{D}) = -\sum_{(p,o)\in\mathcal{D}_{train}} \sum_{t=0}^{L} log(p(o_t|p_t, o_t^c)) \tag{5.2}$$

where o_t is taken form the ground-truth orchestration.

A major difference between training and generative steps is that the complete orchestral score is known at any moment during the training process. Hence, the ground-truth orchestral context o_t^c can be used instead of the generated or initialised value $\widetilde{o}_t^c$ in Equation 5.2. This is necessary as reusing the orchestral frames generated by the model may quickly lead to a strong divergence from the ground-truth orchestration and reach a point where the two contexts o_t^c and $\widetilde{o}_t^c$ are so widely different that predicting the ground-truth o_t given the context $\widetilde{o}_t^c$ does not make sense any more. This strategy is frequently used when training temporal models and known as *teacher forcing* [55].

5.2.2.2 *Initialisation strategies during generation*

During generation, teacher forcing can not be used as the orchestral score is unknown at the beginning of the generative process. In particular, as most models require a temporal horizon for predicting an orchestral frame, it become necessary to provide the algorithm with an initial seed for the orchestration.

In the causal case, only the first T frames of the orchestral score have to be initialised. However, in the random-walk generative process, the whole orchestral score has to be initialised.

Regarding the value of the initialisation, we explored two options, either by sampling from a Bernoulli distribution or setting all units to a constant value. Both approaches are detailed and explored in Section 8.3.

5.3 EVALUATION FRAMEWORK

The advent of artificial intelligence promulgated a quantitative approach of algorithmic composition [16, 63]. In particular, assessing the quality of models in an objective manner would require to define a rigorous evaluation framework.

However, defining a quantitative metric for music appears to be a dubious endeavour as it mostly relies on subjective evaluations. Nevertheless, some tendencies and preferences about musical excerpts might emerge from a given group of individuals. Perceptual tests aim at assessing if such preferences exist in a statistically significant way [58].

As they directly evaluate the preferences of human listeners, perceptual listening tests seem to be the most reliable method for comparing musical excerpts. However, gathering a group of subjects sufficiently large for extracting statistics and collecting the corresponding amount of results is a tedious process, which cannot be used in the development stage of a system. Hence, it is necessary to develop a computable quantitative measure of the model performances.

5.3.1 *Corpus similarity measures*

In order to design an evaluation measure for music, a logical choice might be to rely on a rule-based system which would encode the music theory principles. However, it might be inherently biased and, most of all, limited to a specific set of musical genres. Instead, corpus-based measures are often preferred for probabilistic models. These aim at evaluating how distant the examples generated by a model are from a set of reference examples. In the remainder, $\mathcal{D}_{test} = \{(p, o)\}$ designates a ground-truth set of piano and corresponding orchestration drawn from the existing repertoire.

The iterative generative process described in Section 5.2 suggests a frame-by-frame comparison between the model output and a test dataset. Hence, we can define the overall error of a given model as

$$E(f, \mathcal{D}_{test}) = \sum_{(p,o)\in\mathcal{D}_{test}} \sum_{t} d(f(p_t^c, o_t^c), o_t) \tag{5.3}$$

where f is a model for projective orchestration, and d is a frame-to-frame metric.

As discussed in Section 3.3.1, the negative log-likelihood appears as the most natural choice for evaluating probabilistic models. However, the accuracy measure has been extensively used in the music generation field [16, 63]. We introduce and compare both of these measures in the following paragraphs.

5.3.1.1 *Negative log-likelihood*

The negative log-likelihood is the usual training measure for many probabilistic models, and has already been introduced in Equation 5.2. We remind its definition for a test dataset D_{test}

$$\mathcal{L}(\theta|\mathcal{D}_{test}) = \frac{1}{|\mathcal{D}|} \sum_{(p,o)\in\mathcal{D}_{test}} \sum_{t=0}^{L} -log(p(o_t|p_t, o_c^t)) \tag{5.4}$$

In the special case where the model outputs independent Bernoulli distributions, we explained in Section 3.3.1 that the negative log-likelihood is equal to the binary cross-entropy

$$\begin{aligned} Xent(o_t) &= -log(p(o_t|p_t^c, o_t^c)) \\ &= -o_t \cdot log(q) - (1 - o_t) \cdot log(1 - q) \end{aligned}$$

where $q = p(O_t = 1|p_t^c, o_t^c)$ and $\cdot$ is the scalar product.

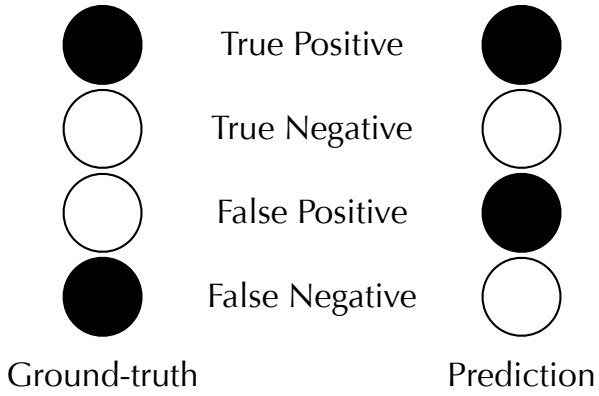

Figure 5.3: Precision, recall and accuracy measures are based on counting the number of correct prediction (true positive and true negative) and mistakes (false positive and false negative).

5.3.1.2 *Accuracy measure*

In the field of decision theory, *precision*, *recall* and *accuracy* measures are commonly used to evaluate how closely a model is able to predict a given target [90]. These measures are based on counting the number of matching elements between the two vectors. In the context of music prediction, the target vector is the set of orchestral notes o_t extracted from the test dataset. The prediction is sampled from the model $\widetilde{o}_t$, and depends on the piano and orchestral contexts p_t^c and o_t^c extracted from the test dataset.

We denote as TP (true positives) the number of notes correctly predicted (notes played in both $\widetilde{o}_t$ and o_t). TN (true negative) is the number of silence correctly predicted (notes off in both $\widetilde{o}_t$ and o_t). FP (false positive) is the number of notes predicted that are not in the original sequence (notes played in $\widetilde{o}_t$ but not in o_t). FN (false negative) is the number of unreported notes (notes absent from $\widetilde{o}_t$, but played in o_t). Figure 5.3 represents the four possible cases.

In the case where the model output is a probability distribution over binary values (here $\widetilde{o}_t \sim p(O_t)$ with $p(O_t)$ being a Bernoulli distribution)[1], the activation probabilities $q(i) = p(O_t(i) = 1 | p_t^c, o_t^c)$ can be used directly, in order to avoid adding noise by sampling from the distribution

$$\begin{aligned} TP &= q \cdot o_t \\ TN &= (1-q) \cdot (1-o_t) \\ FP &= q \cdot (1-o_t) \\ FN &= (1-q) \cdot o_t \end{aligned} \tag{5.5}$$

The traditional *accuracy*, *precision* and *recall* measures are ratio of the aforementioned quantities. However, in the case of music generation, targets vectors o_t are highly sparse. Therefore, the impact of the number of true negatives can rapidly overweight the other quantities, and bias the accuracy to favour models which predict only zeros. Hence, a modified accuracy measure has been used

1 If the dimension of o_t is N, $p(O_t)$ is actually N independent Bernoulli distributions.

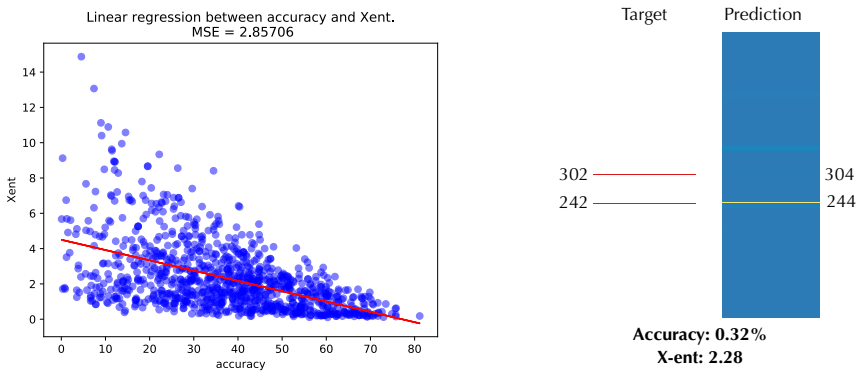

Figure 5.4: The cross-entropy is denoted *X-ent* on the figure, and a lower value indicate a better predictive power. (Left) Each point represents the cross-entropy and accuracy obtained for one prediction of a model. We can see that the two measure do not rank predictions similarly. (Right) An example of a target vector and a prediction which obtains an extremely low accuracy score and an average cross-entropy. This is due to the sparsity of the target vector and the fact that the modified accuracy does not treat true positives and true negatives equally, whereas the cross-entropy does.

as a reference for evaluating models in the automatic music generation task [16, 63, 123]

$$MA(q_t, o_t) = 100 * \frac{TP}{TP + FP + FN}$$

Finally, the modified accuracy of a model over a test dataset is given by

$$MA(\theta|D_{test}) = \frac{1}{|D|} \cdot \sum_{(p,o)\in D_{test}} \sum_{t\in L} MA(q_t, o_t) \tag{5.6}$$

5.3.1.3 *Comparison*

The Figure 5.4 compares the two measures. Each point represent a prediction, its abscissa being the accuracy score and its ordinate the cross-entropy. We can clearly see that the two measures do not rank models similarly. Here, the points which obtain a good cross-entropy but a poor accuracy are sparse targets with no true positive prediction. Indeed, positive and negative targets roles are symmetric in the case of the cross-entropy, while it is not the case for the accuracy. However, we intuitively do not want these empty predictions to obtain a good score. This underlines the fact that we give more importance to true positives than true negatives in sparse targets.

A qualitative evaluation of the best examples selected with the two measures led to the conclusion that the binary cross-entropy is not adapted in the context of projective orchestration task and using a piano-roll representation. Indeed, the cross-entropy validation error quickly reaches its maximum value. Hence, over-fitting occurs very early in the training process (around the epoch 1 or 2). On the other side, the accuracy validation curve is much smoother and the over-fitting detection occurs after the 20^{th} epoch. When listening at the

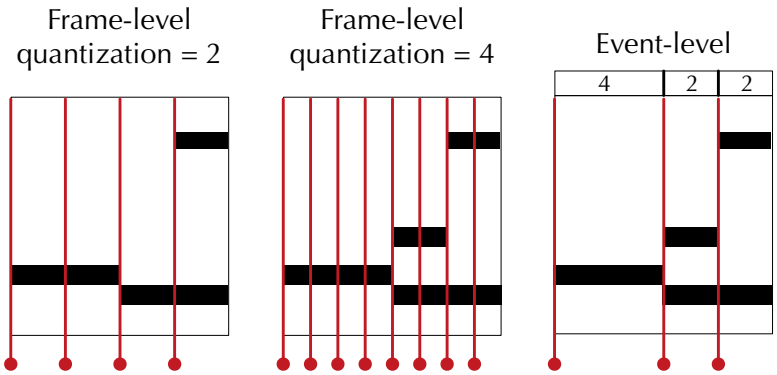

Figure 5.5: Comparison of frame-level and event-level time quantisation. We can see that the number of repeated frames increase with the quantisation. This becomes problematic for statistical learning since the model might mostly focus on learning repetitions. Event-level representation alleviate this issue by discarding the duration information. To preserve rhythmic information, the duration can be added to each vector of the event-level representation.

generated orchestrations using one or the other criterion for selection[2], the results are clearly in favour of the accuracy measure, even though it has some imperfections.

This observation raises the question of which training criterion and selection measure should be used in the context of projective orchestration. Hence, we investigate alternative training and evaluation measures in Section 7.2.1.

5.3.2 *Event-level quantisation*

The piano-roll representation extracted from a piano score depends on the choice of the rhythmic quantisation (see Figure 5.5). This parameter defines the symbolic duration of the smallest event. A coarse rhythmic quantisation may lead to missed events. On the other side, the number of repeated successive frames increases as the quantisation gets finer. As a consequence, even for relatively large quantization of one frame per quarter note, a model which simply repeats the previous frame O_{t-1} at time t obtains an high accuracy score. As the quantization gets finer, the results of the repeat model dramatically increase (see Table 5.1). This effect can also be observed in the field of polyphonic music generation, where the state of the art system obtains an accuracy score of 33.12% on a dataset of Bach's chorales [16], whereas the repeat model obtain a score of 39.60%.

Hence, we propose to rely on a more robust *event-level* framework, which measures the similarity between a true and a predicted frame each time a new *event* occurs, instead of every new time frame (see Figure 5.5). In that case, models will also be trained using the event-level representation.

Note that the orchestration task can be learned with an event-level representation, but easily transformed back to a frame-level quantisation output for generation. Indeed, it can be done by simply applying the rhythmical structure of the original piano score to the generated event-level orchestral score.

2 https://qsdfo.github.io/LOP/results.html#xent_vs_acc

The event-level representation amounts to discard the rhythmic information. However, note durations might condition the decisions regarding orchestral choices. To restore that information, a solution is to append the duration information to the piano-roll representation. This is investigated in Section 8.1.3.

The results comparing the score obtained by a recurrent neural network (implementations are detailed in Section 7.1.2) with a repeat model can be observed Table 5.1. The score of the repeat model diminishes with the event-level quantisation, but it still outperforms the *RNN* model. Besides, the ratio between the repeat and *RNN* scores is similar for all representations.

Even though there is quantitatively no obvious advantage of one representation over the other one, we preferred the event-level for several reasons. First, because the resulting representation is more simplistic and compact. Second, because it does not depend on the definition of a rhythmic quantisation parameters. Finally, because it is more adapted to the real-time implementation we will present in Chapter 10.

Model	**Frame-level accuracy (Q = 4)**	**Frame-level accuracy (Q = 8)**	**Event-level accuracy**
Random	0.73	0.73	0.72
Repeat	58.36	76.66	47.91
RNN	56.60	63.21	42.47

Table 5.1: Results of a different models on the projective orchestration task based on frame-level accuracies with a quantization of 4 and 8 and event-level accuracies.

6

A DATABASE LINKING PIANO AND ORCHESTRAL *MIDI* SCORES

In the previous chapter, we introduced the projective orchestration task along with a framework for building probabilistic models using statistical learning. However, this framework heavily relies on the construction of a large database of piano midi files and corresponding orchestration. As only very few attempts have been made to work on symbolic orchestral data, no reference dataset exists for this task. Hence, we detail in this chapter the creation of two first-of-kind datasets.

- the *Symbolic Orchestral Database* (*SOD*) is a collection of orchestral *MIDI* scores
- the *Projective Orchestral Database* (*POD*) is a collection of piano scores and corresponding orchestrations

Because the quality of collected examples will strongly impact the final performances of the system, we selected examples written by famous composers or orchestration teachers, and manually checked the files. The databases are freely available on a website [1] along with Python scripts for importing the data.

As most of the files were collected on the internet, the piano score and its orchestral version were most of the time not aligned. This proves to be problematic for the approach we proposed in Section 5.2 which relies on a frame-by-frame comparison of the two scores. Hence, we introduce a method for automatically aligning a piano score and its orchestration. Finally, we review the specificities of the *POD* database and the challenges when tackling the projective orchestration task with statistical learning methods.

6.1 ORCHESTRAL SYMBOLIC DATABASES

ORGANIZATION OF THE SYMBOLIC ORCHESTRAL DATABASE (*SOD*) The *SOD* contains 5876 *MIDI* and *MusicXML* orchestral files. The files have been collected from several free-access databases. Files are organized in folders reflecting their origins and a sub-folder indexed by a number.

ORGANIZATION OF THE PROJECTIVE ORCHESTRAL DATABASE (*POD*) The *POD* contains 392 *MIDI* files. Those files are grouped in pairs containing a piano score and its orchestral version. Each pair is stored in a folder indexed by a number. The files have been collected from several free-access databases or created by professional orchestration teachers. The list of composers and their respective representativeness can be found in Table B.1.

INSTRUMENTATION As the files gathered in the database have various origins, different instrument names were found under a variety of aliases and

1 https://qsdfo.github.io/LOP/database

abbreviations. Hence, we provide a comma-separated value (*CSV*) file associated with each *MIDI* file in order to normalize the corresponding instrumentations. In these files, the track names of the *MIDI* files are linked to a normalized instrument name.

METADATA At the root of the database, *CSV* files provide the name of the composer and the piece for the orchestral and piano works.

INTEGRITY Both the meta-data and instrumentation *CSV* files have been automatically generated but manually checked. We followed a conservative approach by automatically rejecting any score with the slightest ambiguity between a track name and a possible instrument (for instance *bass* can refer to *double-bass* or *voice bass*).

FORMATS Links to Python parsers for both *MIDI* and *MusicXML* formats can be found on the companion website [2]. For a given rhythmic quantization, these functions output the piano-roll representations of the scores. Tools for piano-rolls manipulations such as event-level representation extraction (see Section 5.3) are also provided.

In the *MIDI* files of our corpus, the intensities are encoded as an integer between 0 and 127 which we normalize to the real range $[0, 1]$ by dividing by 127. Intensities are encoded with literal musical dynamics in the *MusicXML* format . The mapping between the tags and the real values can be found in Table B.2.

To facilitate further research work, we also provide pre-computed piano-roll representations (see Section 2.1.4). These matrices can be found in Python (.npy) and raw (.csv) data formats.

SCORE ALIGNMENT Two versions of the database are provided. The first version contains unmodified *MIDI* files. The second version contains *MIDI* files automatically aligned using the *Needleman-Wunsch* [79] algorithm as detailed in the next Section 6.2.

6.2 AUTOMATIC ALIGNMENT

Given the diverse origins of the *MIDI* files, a piano score and its corresponding orchestration are almost never aligned temporally (see Figure 6.1). These misalignments are very problematic for learning or mining tasks, and in general for any processing which intends to take advantage of the joint information provided by the piano and orchestral scores. Hence, we propose a method to automatically align a single-instrument score to a corresponding multi-track score. More precisely, we consider the piano-roll representations that interpret scores as a sequence of vectors. By defining a specific distance between two piano-roll frames, the problem of aligning two scores can be cast as a univariate sequence-alignment problem. We provide the Python implementation on the companion website [3].

2 https://qsdfo.github.io/LOP/code
3 https://qsdfo.github.io/LOP/code

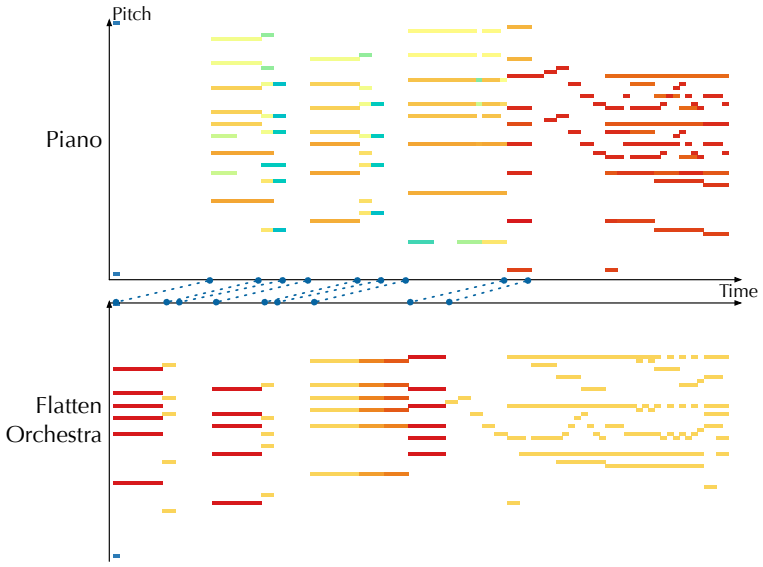

Figure 6.1: Piano-roll representation of a piano score (Top) and its corresponding orchestration (Bottom). The different instruments of the orchestral piano-roll have been summed along the pitch axis instead of concatenated for reducing the size of the representation, which we refer to as *flatten orchestra*. The Needleman-Wunsch algorithm allows to find the optimal alignment of two time-series which proves to be crucial in order to study frame-to-frame correlations.

6.2.1 *Needleman-Wunsch*

The *Needleman-Wunsch* (*NW*) algorithm [79] is a dynamic programming technique which searches for the optimal alignment between two symbolic sequences by allowing the introduction of gaps (empty spaces) in the sequences.

An application of the *NW* algorithm to the automatic alignment of musical performances was proposed in [46]. As pointed out in that article, *NW* is the most adapted technique for aligning two sequences with important structural differences, such as skipped parts for instance [4].

Depending on the domain of application, a possible variation of the original *NW* algorithm consists in deleting symbols instead of inserting gaps. Indeed, replacing gaps with silences in music seems misleading as it would relate them with non-silent events. On the other side, deletions perfectly handle the aforementioned structural differences by completely obliterating the unmatched section. Hence, deletions have been preferred over gaps.

We consider two time series $x = (x_0, ..., x_N)$ and $y = (y_0, ..., y_M)$ and a frame-to-frame distance $d(i, j)$. A deletion in one of the two series is associated with a negative score γ. Hence, a numerical score can be assigned to any alignment between two time-series and the goal of the *NW* algorithm is to find the alignment with the highest score.

Let $S(i, j)$ be the highest possible score for aligning the sub-sequences $x_{0:i}$ and $x_{0:j}$. Thus, $S(N, M)$ is equal to the maximum score possible when aligning

4 A modified algorithm is actually proposed in [46] in order to allow both dilations and deletions operations when aligning the scores. This is of the foremost importance when trying to align audio data, since both deformations can occur. However, in the case of event-level symbolic series, dilation are not expected to happen, and the traditional version of *NW* is preferable.

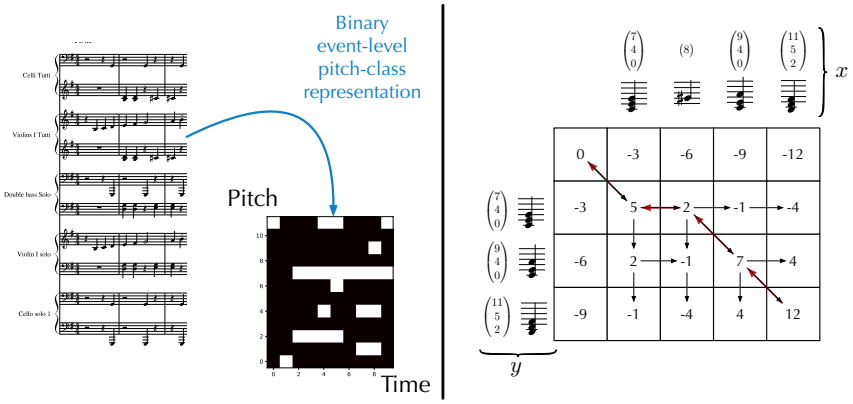

Figure 6.2: *Needleman-Wunsch* (*NW*) algorithm for aligning scores x and y based on their event-level pitch-class representation (Left). An equivalent but more compact representation of a pitch-class vector can be given as a list of activated class, (Right), In the construction of the score matrix S, black arrows illustrate the forward iterative process while the back-tracking process is represented by the red arrows. The gap insertion parameter γ used in this example is equal to 3.

the complete sequences x and y. The matrix S can be iteratively constructed by observing that the value $S(i,j)$ is computed from the surrounding values $S(i-1,j)$, $S(i,j-1)$ and $S(i-1,j-1)$ which covers all the possible cases (symbols are matched, or a deletion is inserted in one of the two series)

$$\text{S(i,j)} = max \begin{cases} S(i-1,j-1) + d(i,j) & \text{diagonal} \\ S(i-1,j) + \gamma & \text{horizontal} \\ S(i,j-1) + \gamma & \text{vertical} \end{cases} \tag{6.1}$$

The direction indicates whether a deletion or a match has been chosen, and the optimal alignment can be found by back-tracking the path taken to reach $S(N,M)$. An illustration of the computation of matrix S is given in Figure 6.2.

6.2.2 *Similarity Function*

The design of a frame-to-frame measure $d(i,j)$ strongly determines the efficiency of the *NW* algorithm. While it can be a particularly complex and delicate task in many contexts, a relatively coarse measure proved to be sufficient in the projective orchestration framework. Indeed, aligning a piano score with a corresponding orchestration is actually not an extremely challenging task as both time-series will exhibit similar harmonic, rhythmic and melodic structures at a fine temporal precision. Differences between the two time-series will mostly occur in the instrumentation and doubling of notes. By finding a common representation that discards these differences, the two time-series can be easily compared. Hence, to find the best alignment path between a piano score and its orchestral counterpart, we propose to extract the following representation as illustrated on Figure 6.2

- compute the event-level piano-roll of both scores

- flatten the orchestral score by summing all instruments along the pitch dimension.
- discard intensities by representing notes being played as one and zero otherwise.
- compute the pitch-class representations of the two matrices, which flattens all notes to a 12 dimensional vector by discarding the octave information. In our case, we set the pitch-class to one if at least one note of the class is played. For instance, we set the pitch-class of C to one if there is any note with pitch C played in the piano-roll vector. This provides an extremely rough approximation of the harmony, which proved to be sufficient for aligning two scores.
- if one of the vectors is only filled with zeros (silence) the similarity is automatically set to zero (note that the score function can take negative values).

Then, we define the score function between two frames $x_i = (x_i^0, ..., x_i^{11})$ and $y_j = (y_j^0, ..., y_j^{11})$ as

$$d(i,j) \ = C \ \times \ \frac{\sum_{p=0}^{11} \delta(x_i^p + y_j^p)}{max(||x_i + y_j||_1, 1)} \tag{6.2}$$

where δ is defined as:

$$\delta(x) = \begin{cases} 0 & \text{if x = 0} \\ -1 & \text{if x = 1} \\ 1 & \text{if x = 2} \end{cases}$$

C is a tunable parameter and $||x||_1 = \sum_i |x_i|$ is the $\mathcal{L}1$ norm.

Based on the values recommended in [79] and our own experimentations, we set C to 10. The gap-open parameter, which defines the cost of introducing a gap in one of the two sequences, is set to -3 and the gap-extend parameter, which defines the cost of extending a gap in one of the two sequences, is set to -1.

6.3 SPECIFICITIES OF THE *pod* DATABASE

In Chapter 5 we showed that the projective orchestration task is akin to a multi-label classification problem for time-series. However, the *POD* database has inherent differences with traditional datasets for multi-label classification.

6.3.1 *Imbalance*

Figure 6.3 highlights the activation ratio of each pitch ($\frac{\#\{\text{pitch on}\}}{\#\{\text{pitch on}\}+\#\{\text{pitch off}\}}$, where # is the cardinal of an ensemble) across the orchestral scores of the dataset. Note that this activation ratio does not take the duration of notes into consideration, but only their number of occurrences. The pitch range of each instrument is depicted beneath the horizontal axis.

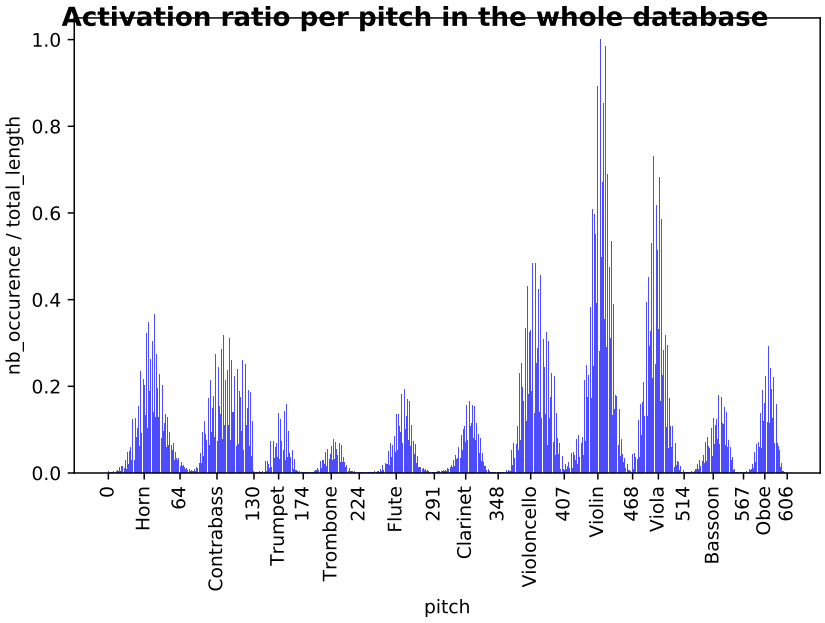

Figure 6.3: Activation ratio per pitch in the whole orchestral score database. For one bin on the horizontal axis, the height of the bar represents the number of notes played by this instrument divided by the total number of frames in the database. This value is computed for the event-level representation. The tessitura of each instrument is indicated underneath the pitch axis, and one can observe the peak in the activation ration curve around the mean tessitura of each instrument.

Two different kinds of imbalance can be observed in Figure 6.3. First, the activation ratio of each individual note is much lower than one (< 0.06) This implies that any given pitch is only rarely played. This is problematic because a model learning on these sparse activation data might be pushed towards a constantly null prediction.

Second, some pitches are played way more often than others. This imbalance might push the model to always choose the same notes and never take the risk to play the notes at the extremes of the instrument ranges.

Class imbalance is known as being problematic for machine learning systems These two observations highlight how challenging the projective orchestration task is. More statistics about the whole database can be found in Chapter B.

6.3.2 *Stylistic consistency*

The wide stylistic range covered by our dataset can be problematic in the learning context we investigate. Indeed, a kind of stylistic consistency is *a priori* necessary in order to extract coherent sets of rules. On the other side, given the paucity of data, reducing the dataset to an even smaller and more consistent dataset might prove even more problematic. In chapter ch:8, we will explore these aspects and show how to alleviate this issues by scheduling the learning process by successively focusing on smaller subsets.

Part IV

MODELS FOR SYMBOLIC PROJECTIVE ORCHESTRATION

7 IMPLEMENTED MODELS

The projective orchestration task has been formalised as a one-step predictive task in Section 5.2. To address it, a fundamental stage is to model the probability distribution of a single orchestral frame O_t given the piano and orchestra context, P_t^c and O_t^c. Depending on the generative scheme, causal or random-walk, context vectors accept different definitions (see Section 5.2). To tackle this task, we investigate two classes of parametric probabilistic models which have been introduced in Chapter 3: energy-based models and feed-forward neural networks. Here, we detail how to apply them to the projective orchestration task.

To evaluate which approach is best suited for the projective orchestration task, we assess the performances of several models belonging either to energy-based or feed-forward neural network models. In Section 5.2, we mentioned that different generative schemes and simplifications can lead to slightly different variations of our task. In order to be able to fairly compare the different models, a common evaluation framework is presented in Section 7.2, along with the results obtained by the different models proposed.

7.1 PROBABILISTIC MODELS FOR PROJECTIVE ORCHESTRATION

In the projective orchestration task, the main challenge is to efficiently encode and combine the various informations of the known piano score P_t^c and already generated orchestral parts O_t^c.

First, the piano score contains the crucial information of the harmonic progression around the orchestrated frame. Hence, P_t strongly conditions the notes that can be played in the subsequent orchestral frame O_t. Second, a common rule in classical music is to ensure continuous melodic lines for each instrumental section. This principle is sometimes referred to as *voice leading*, and is based on minimising the distance between the successive pitches in a given voice. Hence, the prediction of the orchestral frame O_t needs to be informed by the orchestral context O_t^c, composed by the orchestral past, and future in the case of random-walk generation).

7.1.1 *Energy-based models*

In *RBM*-based models, the visible units represent the input that we are trying to model. Thus, in the projective orchestration case, the visible units model an orchestral frame $v = O_t$. The piano and orchestral contexts can be concatenated in a single *context vector* $c = (P_t^c, O_t^c)$ (see Figure 7.2).

The simplest way to condition the generation of the visible units v by the context vector c in a *RBM* is to concatenate the visible and context vector together $\hat{v} = (v, c)$. At generation time, the value of the context vector c is clamped to its known value, and the Gibbs sampling procedure described in

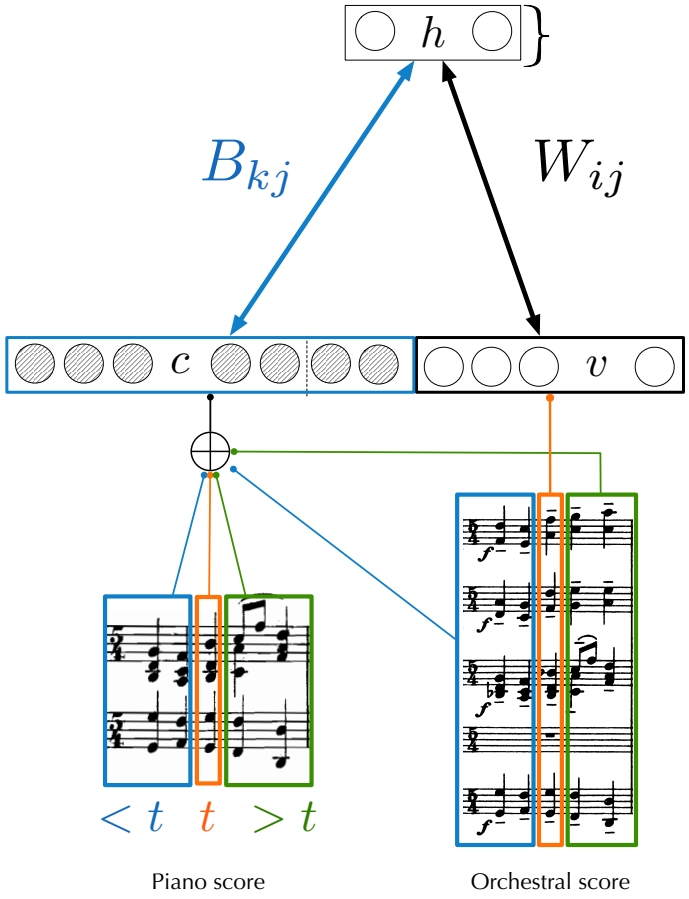

Figure 7.1: Conditioning a *RBM* with a set of context units c can be achieved by using in-painting and concatenating the context vector c to the visible units v. The context units are clamped to their known values when Gibbs sampling is performed, which is represented with hatched circles. The weights for clamped context and non-clamped visible units are separated into a matrix W and a matrix B, but can be manipulated as a single matrix $W_{stacked} = (W, B)$. For projective orchestration, the context vector is defined as the concatenation of the piano and orchestra contexts $c = (P_t^c, O_t^c)$.

Section 3.2 can be applied as in a normal *RBM*. This technique is known as in-painting [39], as it consists in filling in missing values of the visible vector $\hat{v}$.

More sophisticated approaches consist in defining a set of conditioning units that modulate the network weights. The *Conditional RBM* (*cRBM*) architecture consists in simply adding a term to the visible and hidden biases, while in the *Factored Gated Conditional* (*FGcRBM*) model, both the biases and weight matrices are modified, as detailed in Section 4.2.2.

In the *cRBM* model, a set of conditioning units c alter the value of the static biases a and b by adding a term

$$\hat{a} = a + A \cdot c$$
$$\hat{b} = b + B \cdot c$$

where $\cdot$ is the scalar product, A and B weight matrices, and $c = (P_t^c, O_t^c)$ as for the *RBM* in-painting case. $\hat{a}$ and $\hat{b}$ are referred to as dynamic biases, as their value depends on the context vector c and will differ depending on the context.

The weight matrix of the visible units of the in-painting model can be separated in two parts. One part is associated with the context vector and denoted B, and the other part is associated with the orchestral frame and denoted W. Then, the probability distribution of the hidden units given the visible units becomes

$$p(h = 1|v, c) = \sigma(b + B \cdot c + W \cdot v)$$

$$= \sigma(\hat{b} + W \cdot v)$$

Hence, we observe that the probability distribution of the hidden units given the visible and context units is strictly similar to the in-painting *RBM* case. However, a major difference between the two models is that the context units also directly influence the distribution of the visible units through the dynamic biases $\hat{a}$ in the case of the *cRBM*.

The *FGcRBM* allows to dissociate the influence of the orchestra and piano contexts by modelling them with two separate sets of units. Hence, the conditioning units now represent the orchestra context $c = O_t^c$ while the label units represent the piano context $l = P_t^c$. This implementation is depicted on Figure 7.2. Instead of directly encouraging the activations in the hidden or visible units, as this is the case for the in-painting *RBM* and *CRBM*, here, the multiplicative influence of the style vector acts directly on the weight matrices of the model. Hence, the style units can inhibit or favour the correlations extracted by the sub-graph formed by the visible, hidden and context units.

We mentioned in Section 3.3.1 that a single step of Gibbs sampling was performed during training. If this proves to be sufficient for estimating the steepest direction of the gradient, this is not sufficient for obtaining a decent approximation of the joint probability of the hidden and visible units. In practice, we observed that 10 Gibbs sampling steps was sufficient.

7.1.2 *Feed-forward architectures*

Feed-forward neural networks offer an extremely flexible framework for representing and manipulating various sources of information. Their interpretation as a composition of functions (see Section 3.1.2) allows to structure the whole network as sub-graphs, each of them processing a different information. Hence, the output of each sub-graph can be seen as another representation of its input, and the term *embedding* is sometimes used for these intermediate representations. This modular view proves to be particularly convenient in the case of projective orchestration as it allows to easily manipulate and combine the different sources of information.

Hence, the sequences $O_{<t}$, $O_{>t}$, $P_{<t}$ and $P_{>t}$ are processed by Recurrent Neural Networks (*RNN*) (see Section 4.1). We used stacked *RNN*s composed by *Gated Recurrent Units* (*GRU*) and used the last output frame of the last layer

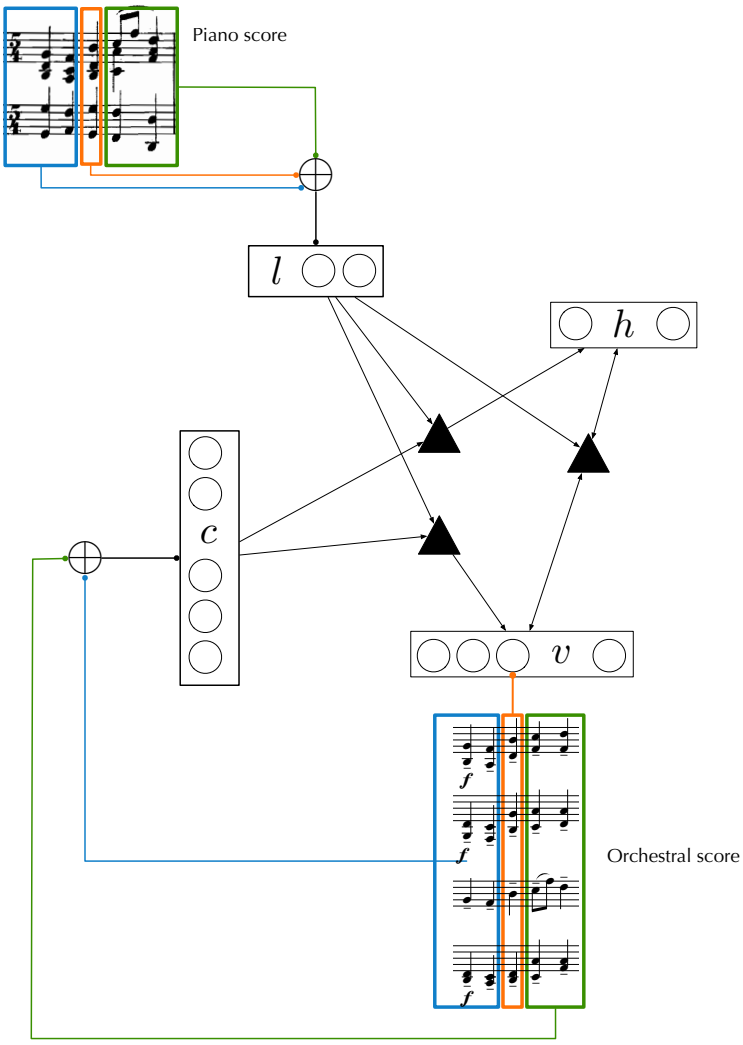

Figure 7.2: The *FGcRBM* model applied to projective orchestration. Context units c represent the orchestral context O_t^c, while the label units l represent the piano context P_t^c. The triangle represent the factor units.

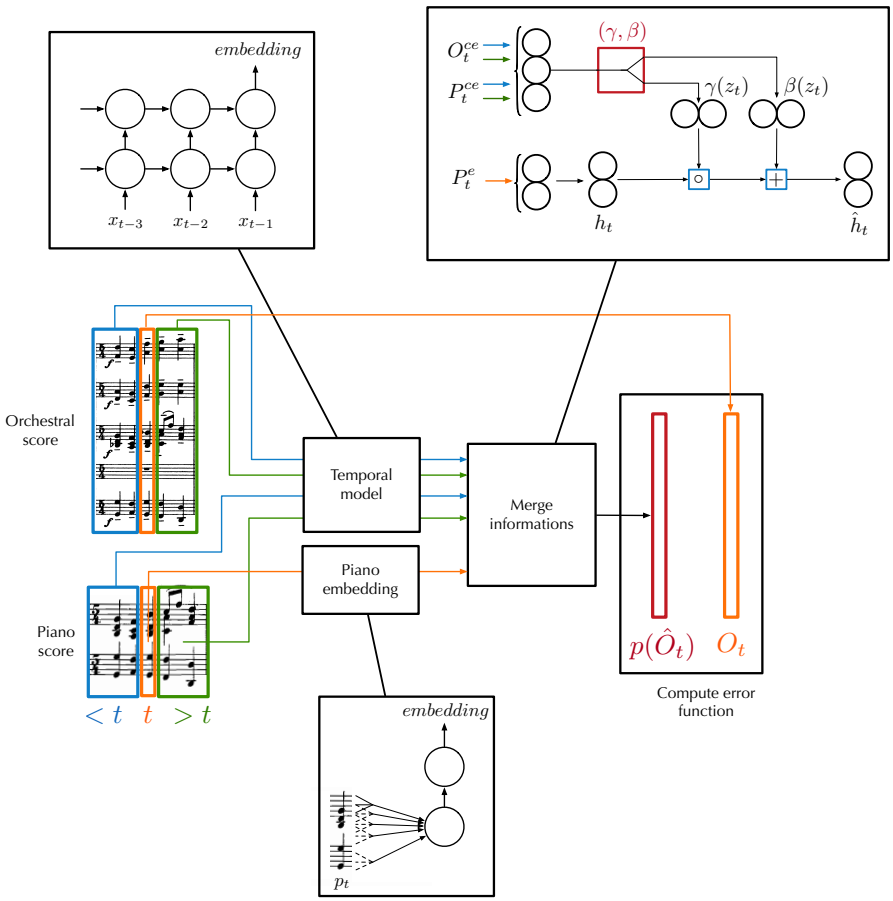

Figure 7.3: A modular approach developed with feed-forward neural networks. The different informations from the piano and orchestral scores are processed by different modules. Past and future events of the piano and orchestral scores are encoded by temporal models. The piano frame P_t is processed separately by a stack of dense layers and possibly a first 1-dimensional convolutional layer. To combine the information extracted by the different modules, affine conditioning is illustrated here.

as the embedding of the time-series (see Figure 7.3). Different units have been tested and the results can be found in Section C.1.

The current piano frame P_t could have been included in the past and future information of the piano score and processed in the recurrent modules. However, we mentioned in the introduction that this information is of the foremost importance for predicting the orchestral frame, as it strongly conditions the possible notes in the orchestration. Hence, we purposely dissociate the current piano frame from the piano time-series and process it separately through a stack of densely connected layers (see Figure 7.3).

In all feed-forward architectures, *Rectified Linear Units* (*ReLU*), $relu(x) = \max(0, x)$, are used in every layer, except in the recurrent units, where sigmoid function, $sigm(x) = \frac{1}{1+\exp^{-x}}$, are used for the activations and hyperbolic tangent, $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, for the recurrent activations, and in the last layer, where sigmoids are used to output a value in the range $[0, 1]$.

We saw in Subsection 2.2.2.2 that some generative models for polyphonic music attempt to model a form of invariance along the pitch axis. This idea comes from the observations that, first, the transposition of a musical piece would preserve most of its musical qualities, and second, that similar patterns can be observed at different pitch heights. This statement is obliterating all considerations about timbre, as the same musical phrase will produce widely different effects if played at different pitch heights. However, this assumption is acceptable for transpositions of a reasonable amplitude, while allowing for introducing strong structuring elements in a model. In particular, we mentioned in Section 2.2.2.2 that convolutional layer has been used to implement a form of transposition invariance in the context of music generation. Hence, we also evaluated the performances when using a 1-dimensional convolution as the first layer of the module processing the piano frame P_t.

Each sub-network is in charge of processing the most adapted representation in order to predict the orchestral frame O_t. To combine the output of the different modules, we investigated two approaches: simply concatenating the extracted representations, or combining them through an affine transformation as proposed in the FiLM framework (see Section 4.2). The affine conditioning requires to make a distinction between the conditioned and conditioning variables. Several options are possible here, but conditioning the current piano frame P_t by all the others embeddings seemed a logical choice to us, as P_t represents the critical information of the harmonic content. Affine conditioning is illustrated on Figure 7.3.

Finally, the network outputs a vector h^L of the same dimension as the orchestral frame O_t. The values of this output vector h^L are scaled in the range $[0, 1]$ by implementing a sigmoid function as its non-linearity $sigm(x) = \frac{1}{1+e^{-x}}$. These values correspond to the probability of activation of each note in the predicted orchestral frame $p(\hat{O}_t = 1) = h^L$. Hence, a binary prediction can be obtained by sampling from a Bernoulli distribution with parameters equal to the network's output values, $\hat{o}_t \sim Bernoulli(h^L)$.

7.2 RESULTS

In the previous section, we presented two families of model for addressing the projective orchestration task: energy-based models and feed-forward neural networks. The goal of this section is to compare in a very general way the respective advantages of both approaches, in order to select the architecture best suited for projective orchestration and further investigate it in Chapter 8.

The goal here is to obtain a first estimation of the capabilities of different models. Hence, we evaluate various architectures in a relatively simple framework, which allows for reasonable training and evaluation time. To do so, we rely on a causal one-step predictive task, as defined in Subsection 5.2.1. We restrict the piano score information to the single frame P_t and discard its intensities. Consequently, both piano and orchestral piano-rolls have binary values. Besides, the event-level representation described in Subsection 5.3.2 is used.

7.2.1 *Alternate training criterion*

In Section 5.3, we observed that the binary cross-entropy and the accuracy do not rank models similarly. Hence, using the first as a training criterion and the second as a test measurement may seem dubious. The binary cross-entropy seems to be the most logical choice in a probabilistic framework. However, we observed that using the binary cross-entropy for selecting models leads to qualitatively unsatisfying results 5.3.

Hence, our objective in this subsection is to propose an alternate training criterion. In particular, we believe that emphasizing the impact of the true positives in the prediction is necessary in the context of highly sparse orchestral vectors. We investigate four training criteria: directly using the modified-accuracy, a weighted modified accuracy and two weighted binary cross-entropy.

Their accuracy scores are given in Table 7.1, and the measures are analysed in the following paragraphs. To gain better insight about the properties of the different training criteria, their error surfaces in a simplified two-dimensional targets case can be observed in Figure C.6.

MODIFIED ACCURACY The modified accuracy is equal to

$$Accuracy\_mod(O_t, \hat{O}_t) = 100 * \frac{TP}{TP + FP + FN}$$

This is the measure defined in Chapter 5 as an evaluation criterion. Note that when used as a training error, its opposite value is used. However, the modified accuracy is a poor training criterion. Indeed, because true negative are completely removed from the equation, plateaus are created along the zero targets axes, and the generated orchestration are not sparse enough.

WEIGHTED ACCURACY To alleviate this issue, the true negative term can be re-introduced in the accuracy measure. To maintain a greater influence of the true positive terms, we introduce a weighted accuracy measure

$$Accuracy\_mod(O_t, \hat{O}_t) = 100 * \frac{TP + \boldsymbol{\lambda TN}}{TP + FP + FN + \boldsymbol{\lambda TN}}$$

where different values of the parameter λ will weight differently the influence of the true negatives.

For large values of the true negative weight ($\lambda > 0.02$), the model simply learns silences since it is the optimal strategy (most of the notes are equal to zero). For small values ($\lambda < 0.02$), the model learns an optimal combination of notes on and off, and play that combination disregarding the piano input.

Hence, using the accuracy measure or a modified version with weighted true negatives leads to a very poor model that predicts a constant vector. Indeed, the non-strictly convex shape of these function may poorly condition the use of optimisation algorithms.

WEIGHTED CROSS-ENTROPY To lower the influence of the true negatives in the learning process, a solution consists in weighting the negative part of the binary cross-entropy function. This lead to the definition of the following function

$$W - Xent = -x.log(p) - \boldsymbol{\lambda}\ (1-x).log(1-p) \tag{7.1}$$

A grid-search over the value of the parameter led to set $\lambda = 0.3$. We could not link this value with a sensible statistic. Indeed, a reasonable value could have been the ratio between positive and negative values, However, this term is equal to

$$\frac{positive}{negative} \approx 0.016$$

PER NOTE WEIGHTED CROSS-ENTROPY A refinement of the weighted cross-entropy is to weight individually each output unit depending on its mean activation across the database (see Figure 6.3). Pitches which are rarely played should have more influence on the positive activation to counter. Hence, we proposed the following error function

$$W - Xent = -\boldsymbol{log(\frac{1}{q})}\ x.log(p) - (1-x).log(1-p) \tag{7.2}$$

where q is the mean activation over the dataset defined in Equation 8.1. $\frac{1}{q}$ is takes values on $[1, \infty)$, and the logarithm limits the strong derivative of the inverse function.

X-ent	M-acc	W-acc	WX-ent	NWX-ent
42.74	3.99	4.49	43.87	44.26

Table 7.1: Best results obtained by a LSTM model mean over a 10-fold validation procedure. Rows are validation measure and column the training measure. The colour indicate the how we qualitatively evaluate the results, green being good, orange medium and red poor. We retained the measures in green for perceptual test on a group of listeners.

An unfortunate consequence of lowering the negative part of the binary cross-entropy is that the impact of the false positives is also reduced. As a consequence, the orchestration become more crowded as too much notes are

predicted at each time step. The weighted binary cross-entropy suffers from this bias for both the per-note and standard versions. Figure D.2 illustrates this effect for the per-note weighted binary cross-entropy on the beginning of the second movement of the 7^{th} symphony of Ludwig van Beethoven (see the original on Figure D.1). When diminishing the weight λ, this effect becomes even more tangible (Figure D.3).

Using the weighted binary cross-entropy as a selection criterion did not produce convincing results, as it can be observed on figure D.4. Using this criterion, over-fitting is detected after only 1 epoch. When listening at the generated examples, an important number of notes are completely out of the harmony defined by the piano score. All these examples can also be observed and listened to on the companion website [1].

Finally, the use of an alternative training criterion quantitatively improved the results of the models, but caused undesired effects in the generated examples. Hence, the problem of finding an adequate evaluation measure for orchestral vectors remains unsolved. In the remainder of this thesis, the pair binary cross-entropy and accuracy is used for training and evaluating.

7.2.2 *Database: building a coherent test set*

We used the *POD* database which has been detailed in Chapter 6. A first observation is that an important number of instruments (120) are represented in the entire database. However, most of them scarcely appear in the different scores. Hence, to avoid dramatically sparse target vectors, which are prejudicial in a learning context, instruments have been grouped together to finally obtain 12 different instrumental sections (Table B.3 in appendix details the sections). For each instrument, only the pitches observed in the database are taken into consideration, reducing the dimension of the orchestral score from $12 \times 127 = 1524$ to 604, where 127 stands for the usual number of possible pitches in the *MIDI* format. The effect of this reduction on the activation ratio of the output targets can be observed in annex on Figure B.1.
A second observation is that this database covers a relatively wide range of orchestration styles from different periods (see Table B.1 in appendix for details). Hence, we believe that the dataset needs to be structured in such a way that a model can focus on a particular style of orchestration. Fortunately, the *POD* database contains all the Symphonies of Ludwig van Beethoven and their piano reduction by Franz Liszt, which constitutes an important and stylistically coherent subset. Besides, the scores of this subset have been encoded with particular care, and the scores are clean from notation errors.

This observation is confirmed when running a leave-one-out training and evaluation on the *POD* database. It consists in training a model with all the files contained in the database except one, which is used for measuring the test error. It allows to detect outliers scores, which often exhibit issues in the notations. The leave-one-out evaluation of a *RNN* model can be observed on Figure 7.4. Several types of outliers were detected.

1 https://qsdfo.github.io/LOP/results.html#alternate_measures

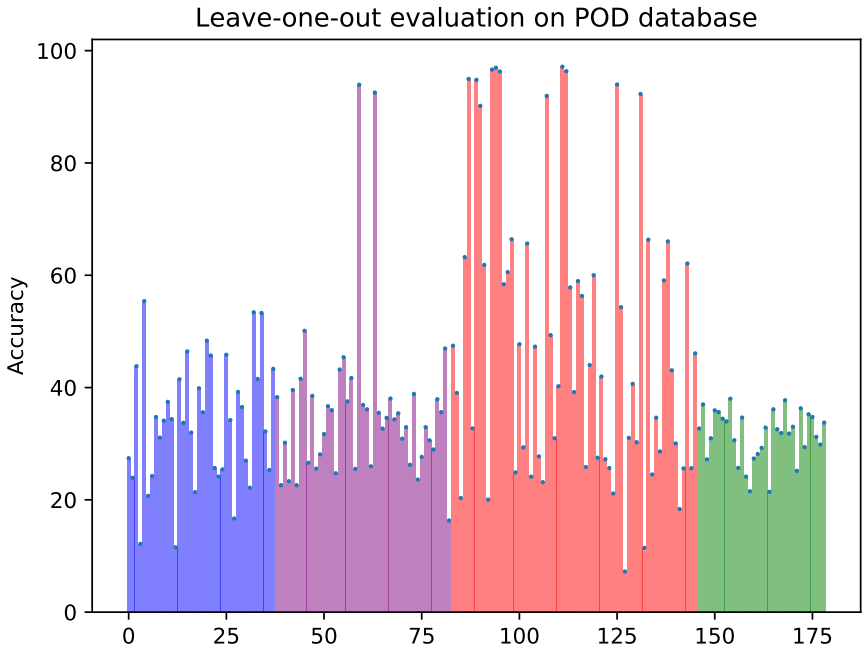

Figure 7.4: The results of a leave-one-out evaluation ran on the *POD* database is reported here. Each bar represents the score obtained by the same model on the unique test file. The colour corresponds to the four different subgroups in the *POD* database. The Liszt-Beethoven subset is represented in green, and show a good consistency. The outlier files were inspected carefully, and most of them showed pathological encodings, such as a violin vibrato written as a repeated note

- files which obtained extremely high scores because of overly repeated notes. For instance, the most precise way to encode a tremolo in the string sections in the *MIDI* format is to write fast repeated notes (see Figure 7.4).
- scores with long percussive sections Indeed, while *MIDI* allows for encoding drums, the encoding used in the different files was not consistent across the database. This needs to be addressed in a later version of the database.
- the leave-one-out evaluation surprisingly detected the files that were stylistically too different from the other files. Hence, the model obtained very poor results on the three pieces written by Schoenberg or Stravinsky, which use a widely different musical language than the other composers.

The Liszt-Beethoven subset shows a great consistency in the accuracy score obtained by the model. Hence, we decided to rely solely on this subset for testing the performances of the different models. However, all files can be used for training purposes.

7.2.3 *Benchmark*

In this subsection, we evaluate the performances of different models belonging either to the feed-forward or the energy-based family. The results are presented in Table 7.2, and implementation details are provided below. The codes for running these experiments is available on Github [2].

A baseline is provided by a *Random* model which consists in N_o independent Bernoulli distributions of parameter 0.5, where N_o is the dimension of an orchestral frame (typically 604). *Repeat* refers to a model which simply repeats the previous orchestral frame o_{t-1} as a prediction for O_t.
Four feed-forward architectures are evaluated. In the *MLP* model, both the piano and orchestral contexts are embedded using stacked densely connected layers. *RNN* designates models for which the orchestral context is embedded with a recurrent neural network. *Conv* is appended when the piano context is embedded using a 1-dimensional convolutional layer. *FiLM* is appended when the piano and orchestral embedding are combined using the *FiLM* framework instead of a simple concatenation.
Three energy based model corresponding to the three architectures we introduced in Subsection 7.1.1 are evaluated: an in-painting *RBM*, *cRBM* and *FGcRBM*.

To evaluate each model, we rely on a 10-fold evaluation (see Section 3.4) of the Liszt-Beethoven subset from the *POD* database. Hence, the training files are composed by the files not in the fold evaluated, plus all the files not in the Liszt-Beethoven subset. It is important to perform the split of the database into folds on the tracks themselves and not fragment of tracks. Indeed, classical musical pieces are often redundant, especially for long works such as symphonies, in which themes are often re-exposed and developed. Hence, if the segmentation in folds is directly made on time frames, there is an important risk that the same frame occurs both in training and test sets. Besides, due to the important variability in the database, the 10-fold split has been computed once, and is

2 https://forge-2.ircam.fr/acids/team/leopold/lop

always the same for all models and configurations. By doing so, we ensure that the mean score over the folds can be compared between models.

The hyper-parameter space of each model has been searched relying on the Bayesian framework described in Subsection 3.4.3. 40 points in the hyper-parameter space were evaluated for each model.

	Accuracy (%)
Random	0.72
Repeat	47.91
MLP	42.50
RNN	42.74
RNN-Conv	40.46
RNN-FiLM	12.30
RBM	1.39
CRBM	27.67
FGCRBM	25.80

Table 7.2: Quantitative evaluation for different models. The score displayed is obtained on the mean score over a 10-fold splitof the database, and the best result obtained over a 40 configurations of hyper-parameters. The first group are simple baseline model, composed by a random prediction and a model which repeat the previous frame. The second group are feed-forward neural networks, while the third group are energy-based models.

Overall, the feed-forward neural networks obtained better accuracy scores than the energy-based models. These results are confirmed by the observation of generated scores[3]. We mentioned in Subsection 5.2.1 that the orchestral score needs to be initialised for time indices smaller than the temporal horizon of the model. Here, the known orchestration is used to seed the first four events.

It can be observed on Table 7.2 that a model which simply repeats the previous events obtains the best results. An explanation is that a relatively frequent case in orchestral scores is that a large number of notes are sustained while few instruments develop the melody (see Figure 7.5).

The fact that the repeat model obtained the best performances is problematic, and denote a problem either in the data representation or evaluation measure. The alternate training criteria we proposed in Subsection 7.2.1 did not solve that issue, and we did not manage to solve that issue in this thesis.

Among the different feed-forward architectures, a simple *RNN* obtained the best results. The convolutional layer do not improve the results. The *FiLM* layer considerably reduces the performances. Indeed, we believe that the piano and orchestral contexts both convey equally important informations, and that the concatenation is the most efficient way to combine the information. Besides, for obtaining embeddings of the same size, *FiLM* requires more weights, and thus a more complex training procedure.

3 Energy-based model VS feed-forward neural networks: https://qsdfo.github.io/LOP/results.html#energy_vs_ff

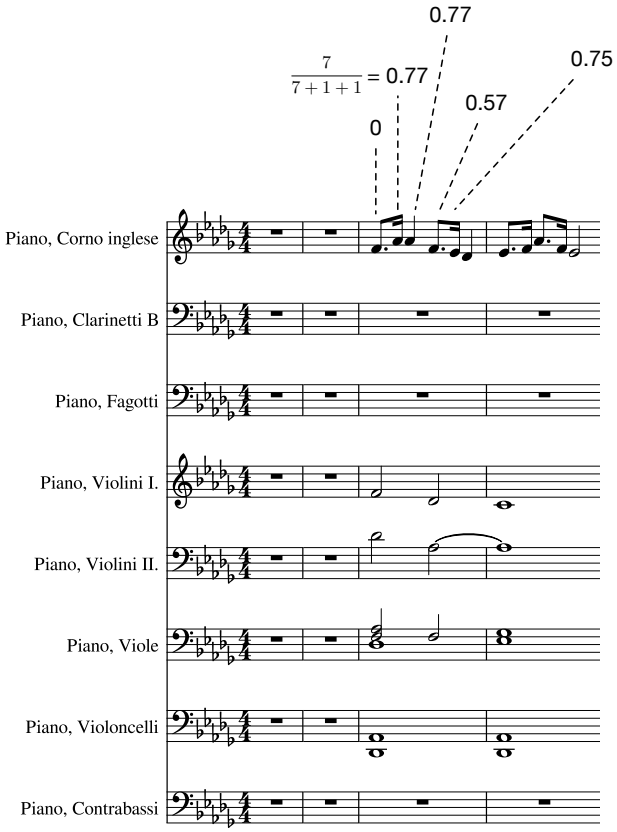

Figure 7.5: The beginning of the second movement of Antonín Dvořák 9th is an example of overly repeated notes due to the accompaniment section. On top of each frame is indicated the accuracy score obtained by the repeat model. On the second frame, there is 7 true positive, and 1 false positive and 1 false negative due to the melody.

The influence of the hyper-parameters over the performances of the *RNN* model, which obtained the best scores, can be observed in annex (Chapter C). From this hyper-parameter analysis it is important to underline that relatively small architectures composed by two stacked GRU layers with a number of units close to 2000 obtained the best results. Increasing the number of layers and units reduces the performances of the model. Indeed, the database is actually rather small (each fold comprises approximately 1500 batches of 128 training points each, for a total of 192000 points).

8

EXPERIMENTS

In Chapter 7, we evaluated the performances of different architectures on the projective orchestration task which has been defined in Section 5.2. We determined that a relatively simple model based *Recurrent Neural Networks* (*RNN*) was rank the best (see Table 7.2).

In this chapter, we attempt to improve the quality of the orchestration generated by this basic architecture. Hence, all experiments presented in the following sections are led using same model. Unless specified, it is composed by a 2 layer recurrent neural network using *GRU* units, which embeds the orchestral past $O_{<t}$ and one layer of densely connected units, which embeds the piano frame P_t. These two embeddings are concatenated and passed through a densely connected layer. *ReLU* activation function are used.

In the remainder of this chapter, a model designates the types of units, choice of layers and their organisation, but is not associated with a value for the hyper-parameters (such as the number of layer or number of units). To refer to a model and an associated set of values for its hyper-parameters, we use the term architecture. Hence, when evaluating one given model in different contexts (e.g. use of training set or generative scheme), the same model name may actually refer to different architectures, since the optimal value of hyper-parameter will be different.

In Section 8.1 we evaluate the impact of passing different informations as input to the network. In particular, the impact of the temporal order, and the dynamics and durations of the piano scores on the predictive power of a model are discussed. In Chapter 6, we mentioned two pitfalls of the Projective Orchestral Database (*POD*): class-imbalance, which refers to the fact that notes activation are sparse and not equally distributed across the different instruments, and the paucity of data.

In Section 8.2 we attempt to alleviate the first issue using pre-computed biases, and the second by using purely orchestral scores to increase the number of examples observed by a model. Finally, we mentioned in Subsection 5.2.1 that two generative schemes could be adopted for orchestrating a piano score: a causal and a random-walk approach. Both are investigated in Section 8.3.

8.1 INPUT INFORMATIONS

An important aspect when designing a neural networks is to determine which informations are useful in order to best accomplish the desired task. In the case of projective orchestration, the temporal order and the dynamics and durations of notes in the piano score are theoretically important in order to decide the notes played by the orchestra. We investigate their impact in the following subsections.

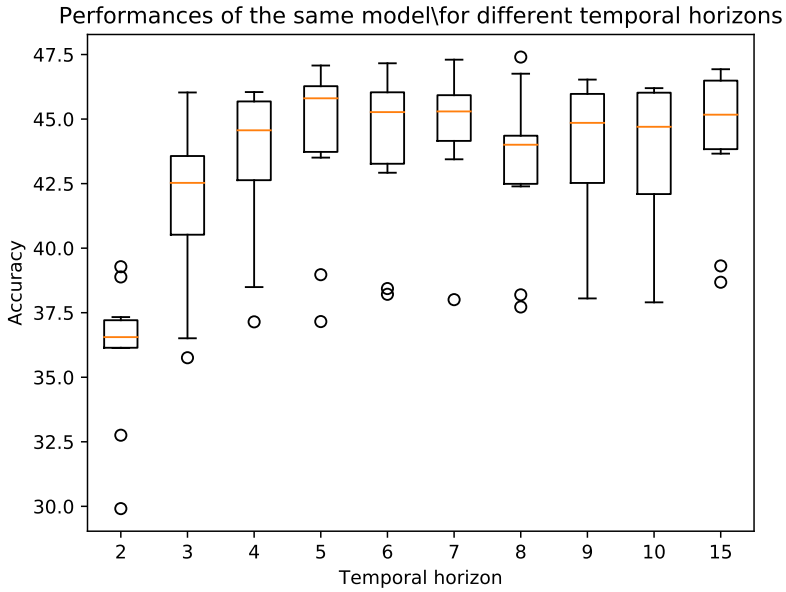

Figure 8.1: Accuracy score obtained by a *RNN* model on a 10-fold evaluation scheme for different temporal horizon. The lowest temporal order which obtains an optimal score is equal to 5.

8.1.1 *Temporal order*

As defined in Section 5.2.1, the temporal order T of a model defines the length of the piano and orchestral context fed to a network. The influence of the temporal order parameter on the accuracy score can be observed on Figure 8.1. The performances of the model improve with the temporal order until a value of $T = 5$. For longer temporal horizon, the performances do not increase, but remain comparable.

When considering this results, It is important to remember that it takes place in the event-level framework (see Section 5.3.2). Hence, the actual context observed by the network will strongly depend on the rhythm of the score being observed, ranging from 1 beat to several bars. However, in rapid sections such as a 16^{th} notes melodic line, a value of 5 would still ensure a scope of approximately 1 beat, depending on the measure. This value seems small, but it is actually sufficient for grasping most informations necessary for basic voice leading. However, the result indicates that the model fails at using longer temporal relations to inform its predictions.

Note that the same number of hidden layer is used for all architectures, which may seems to disadvantage longer temporal horizon which would need more statistical power. However, increasing the number of layers for models with a longer temporal order did not yield better results.

8.1.2 *Dynamics*

Intuitively, the intensity in the piano score seems to be of the foremost importance. Indeed, we mentioned in Section 5.2 how the dynamics of a passage influence the choice in the instruments used for orchestrating it.

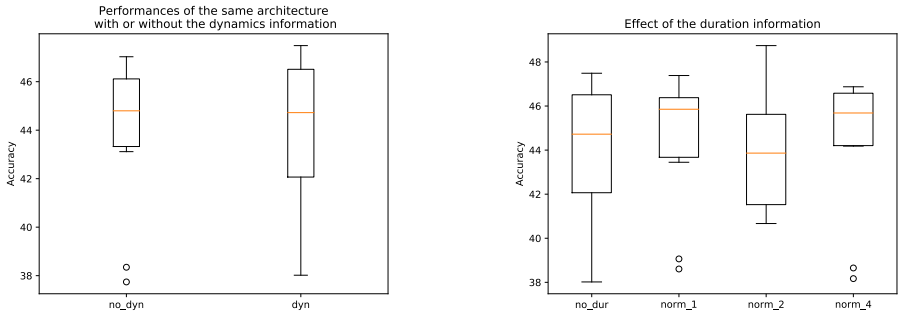

Figure 8.2: (Left) Accuracy score obtained by a *RNN* model with or without the dynamic information. Without the dynamics information, the model obtains an average score of 42.70%, slightly better than the 42.46% it obtains when informed with the dynamics. (Right) Influence of the dynamic information.

Dynamics in the piano score are a indicate by a real value between 0 and 1. No pre-processing or normalisation proved to be necessary. The accuracy obtained by the same model with or without the dynamic information can be observed on Figure 8.2 (left).

It appears that our architecture fails at using the dynamic information to improve its predictive ability. This can be explained by the fact that the model is trained on a one-step predictive task. Hence, the objective of ensuring a continuous voice leading in the orchestration probably surpasses the dynamic information. In other words, the voice leading constraint dominates the dynamic information. This seems logic, as the orchestration of a single frame given all the surrounding context does not involve the choice in instruments, but rather to continue the choices made in the initialisation.

8.1.3 *Durations*

Event-level rhythmic quantization (see Section 5.3.2) has the disadvantage that the notes duration is no longer represented. However, this information might be important when orchestrating a score. Indeed, instruments with a sharp and precise attack might be preferred for rapid sections, whereas slower passages may admit a larger number of instrument being used with no risk of sounding clumsy.

To include this information in an event-level piano-roll, a dimension which encodes the duration of each event in number of quarter notes is added. The longest duration observed was equal to 9.3 quarter notes. Hence, we normalise the symbolic durations as

$$d = \frac{q}{K}$$

where q is the duration in quarter notes and K a normalization constant. Different values of K are tested and the results are reported on Figure 8.2. The influence of the duration information over the predictive performances of the model is not significant. Similarly to the case of the dynamics information, we believe that this is due to the overly constrained framework of the one-step predictive task.

8.2 MANIPULATING THE *pod* DATASET

8.2.1 *Pre-computed bias*

We mentioned in Chapter 6 that two kinds of imbalance were observed in the *POD* database. Between-class imbalance refers to the fact that some instruments are more rarely played than others, whereas class imbalance refers to the fact that a given note is rarely played in the whole set of scores, resulting in highly sparse target vectors.

The alternative training criteria we proposed in Subsection 7.2.1 where a first attempt at addressing sparsity issues by adding a term in the error function which favours sparse predictions. Here, we investigate another solution, which consists in providing to the network the information of the ratio of activation of each notes before starting the training process. The idea was that by informing the network about which notes are supposed to be played rarely or not, the training process should speed-up, and possibly a better score could be reached.

Passing this information can be done via the initialisation of the biases in the last layer of the network. Indeed, a network designed to perform projective orchestration uses a sigmoid function in the last-layer in order to output a vector whose values are comprised in range $[0, 1]$. These values correspond to the probability of activation $p(O_{t,i} = 1)$ for each note indexed by i (see Section 7.1.2).

Hence, if h^{L-1} denotes the output of the penultimate layer, W the weights from the previous layer and b the static biases, the probability activation is given by :

$$p(O_t = 1) = sigm(W.h^{L-1} + b)$$

where $sigm(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function.

In the absence of any input information ($x = 0$), a reasonable prediction for the network would be to output the probability of activation of the different notes observed over the training dataset

$$sigm(b) = q$$

with

$$q = \frac{1}{|D|} \sum_{(p,o)\in\mathcal{D}_{train}} \frac{1}{L} \sum_{t=1}^{L} o_t \tag{8.1}$$

Then, a reasonable value for the static biases of the predictive layer is

$$\hat{b} = sigm^{-1}[min(q, \epsilon)]$$

where $sigm^{-1}(x) = \ln(\frac{x}{1-x})$ is the inverse sigmoid and ϵ is a small value preventing passing zeros to the logarithm.

Hence, to reduce the computational load over the network and initialise it in a state closer to its optimal set of parameters, an idea is to initialise the biases to the mean activation value of each note. Two options are possible:

- maintaining a fixed value for the biases $b = \hat{b}$ in the last layer. This parameter is not modified during the training process, and all the variability expressed in the last layer is in charge of the weights W.
- modelling the biases as $b = \hat{b} + b_{learnable}$ where $b_{trainable}$ is a trainable parameter. The purpose of this decoupling is to reduce the modelling effort of the trainable bias to some smalls variations around the pre-computed mean activation.

The final performances of the models with pre-computed biases are very similar to the result obtained by models with biases initialised to zero. The results are 42.77%, 42.66% and 42.47% for for respectively the fixed bias, variable bias and standard architecture. Besides, no significant speed-up in the training process was observed when using pre-computed biases, with 19.8 and 18.9 epochs in average before over-fitting against 19.6 with a zero initialisation.

We concluded that the initialisation of the biases with pre-computed value was useless with the architectures we investigated.

8.2.2 *Transposition invariance*

The *POD* dataset contains relatively few data points (approximately 192000 training points depending on the fold), which proves to be problematic in statistical learning context. A method frequently used to compensate for the paucity of reference examples is to perform *data augmentations*. This technique consists in artificially increasing the number of training points by creating new valid examples from the one observed in the original database. When learning on musical sequences, a standard data augmentation consists in performing pitch transpositions of the observed musical excerpts. The assumption here is that the transposition of a musical piece constitutes a valid example as well.

In the case of orchestration, pitch translation is a very delicate operation. Indeed, transposing a whole orchestral score might conduct in instruments being played out of their tessitura. Besides, instruments' timbre changes depending on the register, which can affects the whole timbre of the orchestra. Consequently, orchestral effects might suffer from transposition, even in a reasonable range. For instance, the sound of a given chord might change substantially depending on whether or not open chords are used for the string sections, and this occurs for a transposition of only a semi-tone. Finally, pitch transposition can considerably increase the technical difficulty of a passage for some instruments. After discussing with composers, we decided that a pitch transposition of three semitones up and down would be the most extreme transposition still acceptable as a rough approximation.

An other way to introduce a form of transposition invariance is to use convolutional layers. By replacing the first layer of the piano and orchestra embedding networks with a one-dimensional convolution, the model should have the ability to identify the same patterns transposed.

Figure 8.3 compares the accuracy scores obtained when using several ranges of data augmentation and when replacing the first layer of the same model with a convolution. 50 hyper-parameters configurations have been ran for each model.

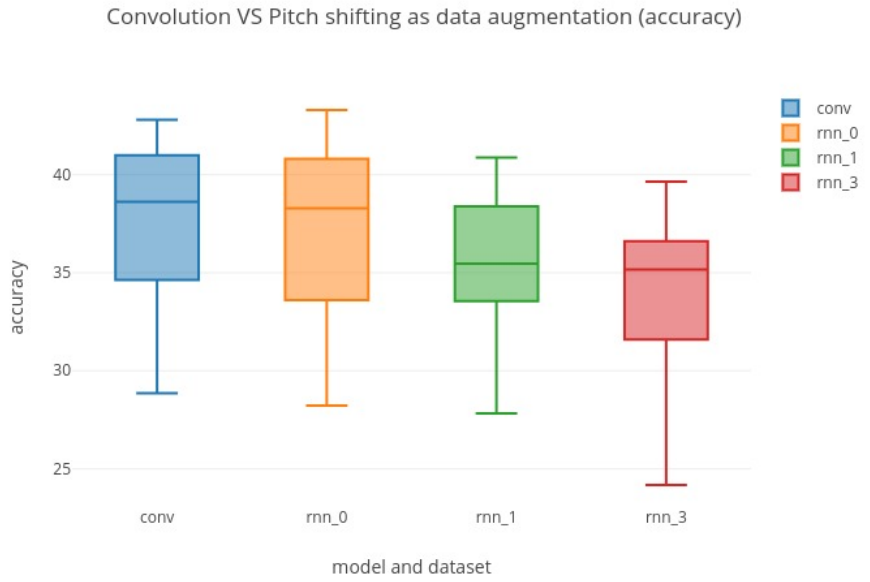

Figure 8.3: Comparison of the accuracy score obtained when performing data augmentations of amplitude 0, 1 and 3 and a model implementing a 1-dimensional convolution as its first layer.

The *RNN* model obtained slightly better results than the convolution model when no data augmentations are used. However, it can be observed that he accuracy score decreases when the range of the data augmentation increases. This can be explained by the fact that when performing data augmentation, the dimension of the target orchestral vector increases. Indeed, the tessitura of each instrument is computed on the dataset, so that the orchestral vector has the lowest possible dimension. For example, considering the simple case of 4 instruments of range 20 pitches each, the dimension of the orchestral vector is 80, but if a data augmentation of 3 semi-tones up and down for each instrument is allowed, the orchestral vector now has a dimension of $80 + 4 * 3 * 2 = 104$. Hence, the additional information provided by the data augmentation does not counterbalance that increase in the dimensionality.

8.2.3 *Scheduled learning*

We mentioned in Chapter 6 the stylistic inconsistency of the dataset. Many styles of orchestration have been developed at different periods. To be able to extract an orchestration style, we believe that it is necessary to have a coherent dataset.

We mentioned in Subsection 7.2.2 that the models are evaluated solely on the subset of the *POD* database composed of Beethoven's Symphonies and their piano reduction performed by Liszt. However, the other files not in the Liszt-Beethoven subset were used as training data, despite the fact that the orchestration style can be different, and the encoded information of poorer quality. Hence, how does the presence of these file in the training set impact the performances of the learnt model ?

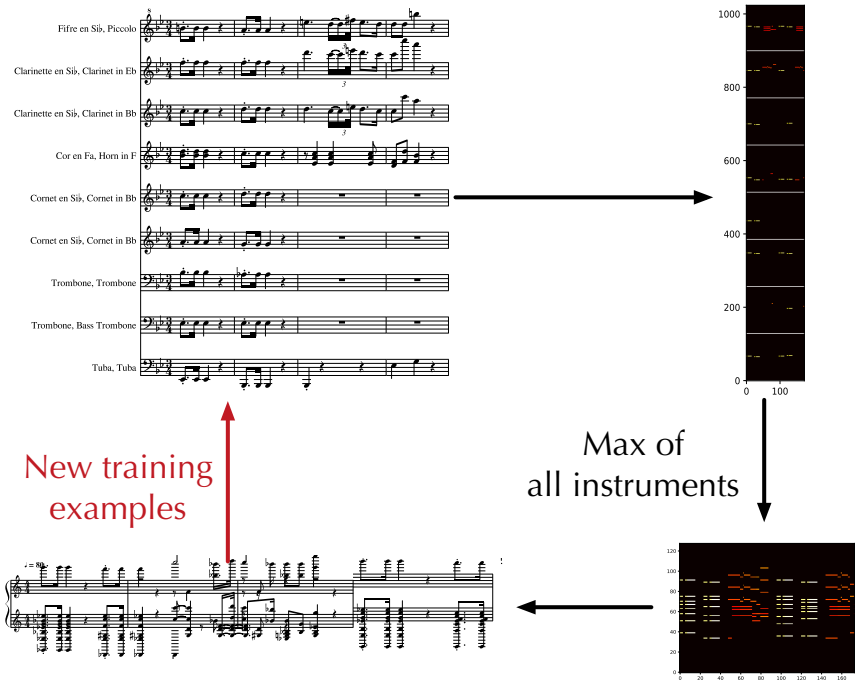

Figure 8.4: New examples can be artificially created from purely orchestral scores by reducing them to a piano version by overlaying all the instrumental sections in a single piano score. The piano score is a very rough approximation of a piano reduction. It can be observed that it contains too many notes for being playable by one pianist. However, we hypothesised that this approximation could be useful pre-training material.

On the other side, deep learning models usually require a large amount of data to efficiently infer parameter values. However, the *POD* dataset we presented in Chapter 6.1 is rather small. To increase the size of the dataset, we try to take advantage of the large quantity of orchestral files contained in the *SOD* dataset which is composed solely by orchestral files, but not associated with a piano reduction (see Section 6.1). Hence, we attempt to artificially create piano scores by reducing all the different instrumental sections in a single piano score (see Figure 8.4), and hypothesize that this rough approximation of a piano reduction is sufficient to be used as a warm-up training set and initialise the weights of a model in a good setting before fine-tuning it on a dataset of better quality.

Here, we investigate how to schedule the learning process by progressively reducing the training dataset to smaller consistent ensembles. We define the three following subsets:

SUBSET A consists in Beethoven symphonies and their reductions for piano by Liszt. It contains 34 pairs of files, each file being a whole movement of symphony.

SUBSET B is composed by 162 pairs of files containing clean and manually checked examples of orchestrations. However, they cover a wide periods, from baroque pieces to a few scores from the XXth century.

SUBSET C contains 5867 orchestral scores of famous classical pieces. However, they cover a wide range of styles.

To compare the effect of the different subsets on the performances of a model, we propose to train a same architecture using three different training strategies, while evaluating them only on the smallest subset A which corresponds to the Liszt-Beethoven dataset. The strategies are:

- training is performed only on A.
- the model is pre-trained on B and fine-tuned on A.
- the model is pre-trained on C then on B and finally on A.

During the pre-training steps, a validation subset is extracted from the pre-training dataset to detect over-fitting. A 10-fold split is performed on the subset A. However, the same pre-trained model is used for all folds of the subset A. This is done to avoid running several times the pre-training step, which can be particularly long (more than 10 hours) in the case of the subset C.

Figure 8.5 summarizes the results obtained for the different training strategies. The best strategy is to train on both B and A. Hence, it seems that the information contained in the subset B helps the model to improve its predictive capacities over the subset A. However, pre-training on the dataset C does not improve the performances. We believe that this is due to the brutal method we used for generating the piano reductions. Hence, the information grasped by the model on these examples is overly simplistic and does not help the model to be accurate on more subtle examples. A confirmation of that assumption is the particularly high validation score obtained by the model at the end of the pre-training step on the subset C, which is equal to 73.73% to be compared with the maximum score of 42.74% obtained on the subset A.

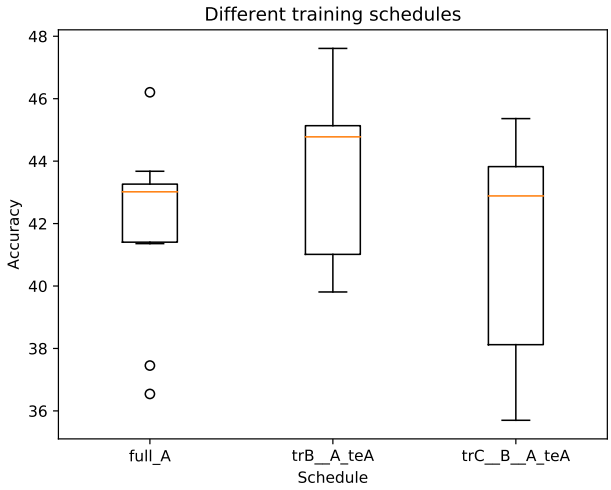

Figure 8.5: The training process can be scheduled by presenting successively different subsets of the database. The idea is to benefit from a large collection of files, while preserving a stylistic consistency by progressively focusing on a more coherent subset.

8.3 GENERATION SCHEMES

By observing the proposed orchestrations, it appears that there is a lack of continuity inside the voices, and the model partly fails to ensure a satisfying voice leading. For instance, in Figure D.2, the Oboe, Bassoon and Trumpet sections exhibit particularly fragmented melodic lines.

A first hypothesis was that the discontinuity might be due to a form of imbalance between the influences of the orchestral context and piano embeddings. In the architecture we proposed, the concatenated embeddings and prediction are directly connected and not separated by any intermediate layer. This allows for directly observing the influence of each input over the prediction, through the observation of the weights connecting these two last layers Hence, we thought that applying a different weight decay penalty for each part of the weight matrix could be used to enforce a stronger influence of one of the two embeddings. However, by observing this weight matrix (see Figure C.7), there does not seem to be any imbalance in its structure, suggesting that both information are equally weighted.

Another approach we investigated in this section consists in trying to improve the continuity in the proposed orchestrations by adopting alternative generation schemes. Several variants are proposed, but the main idea is to "smooth" the orchestration by performing several generative passes over the score, instead of a single forward pass over the successive frames of the piano score. Indeed, all the model presented in the previous section relied on the causal generative scheme introduced in Subsection 5.2.1. However, in that same section, we also mentioned the possibility to orchestrate the frames of a piano score in random order, by performing a random-walk over the time indices. We insisted on the fact that the exact framework can be used for training such model, as it also relies on a one-step predictive task. The main difference with the causal generation being that the future orchestra $O_{>t}$ is now used for predicting

the orchestral frame O_t. More precisely, we propose four generative schemes, combining in different ways the causal and random-walk generative processes. We describe them in the following subsection.

8.3.1 *Algorithms*

FORWARD A purely causal generative scheme in which the orchestration is generated in a single forward pass. Orchestral frames are generated following an incremental order of the time indices (see Algorithm 1 in Section 5.2.1).

FORWARD-BACKWARD By simply reversing along the time axis all training sequences, the exact same architecture as the one developed for building a causal model can be used for building an anti-causal model. Such model generates an orchestration by browsing time indices in decremental order, starting from the last index. By combining a causal and anti-causal model, forward and backward generative passes over the piano scores can be performed, in order to successively refining the orchestration. After initialising the orchestral score (discussed below), the generation can be described by the following algorithm

for $pass \in range(1, K_{fb})$ **do**
  for $t \in range(T, L)$ **do**
    $\widetilde{o}^c_t = \widetilde{o}_{t-T:t-1}$
    $\widetilde{p}_t \sim f(p^c_t, \widetilde{o}^c_t)$
  end
  for $t \in range(L-T, 1, -1)$ **do**
    $\widetilde{o}^c_t = \widetilde{o}_{t+1:t+T}$
    $\widetilde{p}_t \sim f(p^c_t, \widetilde{o}^c_t)$
  end
end

where T is the temporal order of the model, $range(L-T, 1, -1)$ decrements indices between $L-T$ and 1, and K_{fb} denotes the number of forward-backward pass. Different values of this parameter have been tested and discussed in Subsection 8.3.3.

RANDOM-WALK This strategy is the direct application of the random-walk generative process described in Subsection 5.2.1. The time indices of the orchestral frames generated follows a random-walk (see Algorithm 2). The total number of sampling steps performed is a parameter we denoted N_sample. However, a more relevant parameter is the average number of pass over a single frame, which we denote K_{rw}. Both parameters are linked by the following equation $N_sample = K_{rw} * L$, where L is the dimension of an orchestral frame (604 in most of the aforementioned training context). Different values of the parameter K_{rw} are discussed in Subsection 8.3.3.

FORWARD AND RANDOM-WALK We mentioned in Subsection 5.2.1 that the entire orchestral score needs to be initialised in the case of the random-walk generative process. We speed-up the convergence of the random-walk generation, we investigate a strategy which consists in first initialising the

orchestral score by performing a forward pass.

for $t \in range(T, L)$ **do**
  $\widetilde{o}_t^c = \widetilde{o}_{t-T:t-1}$
  $\widetilde{p}_t \sim f(p_t^c, \widetilde{o}_t^c)$
end
for $pass \in range(1, K_{rw} * L)$ **do**
  $t \sim \mathcal{U}(1, L)$
  $\widetilde{o}_t^c \triangleq (\widetilde{o}_{t-T:t-1}, \widetilde{o}_{t+1:t+T})$
  $\widetilde{o}_t \sim f(P, \widetilde{o}_t^c)$
end

where $\mathcal{U}(1, L)$ denotes the uniform distribution over the interval $[1, L]$.

8.3.2 *Initialisation*

All four approaches require to initialise parts or the entire orchestral score, as depicted on Figure 5.2. So far, we used the known orchestral score as a seed for the generation, which greatly simplified the task. However, when orchestrating an new piano score for which no existing orchestration has already been proposed, such initialisation seed is not available.

We explained in Subsection 5.2.2.2 that both the causal and random-walk generative processes require the initialisation of a fragment of the orchestral score. The causal generation only needs the first T frames of the orchestral score, where T designates the temporal order of the model used. However, the random walk process requires the initialisation of the entire score. In the last generative scheme we proposed in the previous subsection, the forward pass performed before the random-walk generation acts as an initialisation.

Three initialisations are compared

- a zero-initialisation, where all unknown orchestration simply takes the value 0.
- constant initialisation with a value equal to 0.1 or $q \approx 0.01$, with q the mean activation per note defined in Equation 8.1.
- random initialisation from a Bernoulli of parameter 0.1 or 0.01

8.3.3 *Results*

A quantitative comparison of the different generative schemes is not really possible here as the tasks are not defined in the same manner. In particular, the accuracy scores of the random-walk models evaluated on a single-frame predictive task are much higher than the scores obtained by causal models. Indeed, the orchestra contextual information in the case of the random-walk model is defined as $o_t^c = (o_{[t-T:t-1]}, o_{[t+1:t+N]})$. By consequence, for a same *RNN* architecture, a causal model obtains an accuracy score of 53.45% whereas the same model obtained a score of 42.74% using a causal generative scheme.

Orchestrations of Liszt reduction of Beethoven 7^{th} Symphony using different generative schemes can be downloaded from the website [1].

1 https://qsdfo.github.io/LOP/results.html#generation_strategies

Random and constant initialisation with an high parameter value of 0.1 generated too crowded orchestrations. This was to be expected, since orchestral vectors are supposed to be sparse. Zero and constant initialisation with a low parameter value of 0.01 produced similar results. Indeed, a very low constant value of random sampling parameter is not significantly different from a zero initialisation, and eventually the three initialisations lead to similar results. Random initialisation does not seem adapted as it can severely mislead the model by indicating completely wrong orchestration. The conclusion is that a simple zero initialisation is adapted for all generative schemes.

More surprisingly, the four generative schemes led to qualitatively similar results. Increasing the number of pass in the forward backward generation, as well as in the correction model did not improve the quality of the proposed orchestration. It even tended to generate too sparse orchestration. Indeed, the sparsity observed in the training orchestral vectors pushes the model to output sparse vectors. Hence, notes are switched off after each successive pass.

We believe that the lack of improvement observed with the more sophisticated generative schemes may be due to an important difference between the task on which the model is trained and the actual generative context. A possible solution is to schedule the learning process by progressively reducing the amount of information passed when performing teacher forcing. By defining a parameter in $[0,1]$ which defines how much information is used. We are currently investigating this solution, but no satisfactorily solution has been obtained for now.

8.4 CONCLUSION

In this chapter, we further investigated the performances of the feed-forward neural networks on the projective orchestration task. First, we studied the impact on the predictive power of a network when passing it more information. The model we developed are not able to take advantage of the additional information passed to the network. Indeed, the one-step predictive task we proposed is already extremely constrained, while these information do not further specify the set of possible notes. Hence, we believe that these additional informations might not be useful in the restrictive framework of one-step prediction. Actually, we believe that these informations would be of the foremost importance at precise locations in the score, such as at section changes. However, the end-to-end approach we adopted is unlikely to be able to infer the structure of the piece. We evoke a possible solution in future works (see Chapter 12.2).

The *POD* is a challenging dataset, which is constituted by sparse target vectors. We proposed to implement pre-computed biases in order to inform the model of the overall probability for each note to be played, but it did not improved the results. Besides, the *POD* database covers a wide range of epochs. We propose to evaluate the performances of a model on a restraint consistent subset composed by Beethoven Symphonies and their piano reduction by Liszt. While evaluating different training schedules which consists in presenting successively different subsets, and tried to include a larger purely orchestral dataset (*SOD*) (see Chapter 6). The examples contained in the *SOD* database did not helped the model to obtain better results, probably because of the crude method we used for automatically generating the piano reductions, which

consists in mapping the notes of all instruments on a piano score. However, reducing the training material only to the Liszt-Beethoven subset did not yield better results. Hence, the other scores present in the *POD* help the model increasing its predictive performances over Liszt-Beethoven subset.

Finally, we observed a lack of continuity in the orchestration generated by our system. In an attempt to alleviate this issue we investigated different generative schemes. Notably, we proposed procedures which perform several passes over the generated orchestration in order to successively refine it. However, none of the more sophisticated generative schemes proved to be better than the most simple causal generative scheme, using a zero initialisation.

From these experiments, we had the feeling of a glass ceiling over the performances. Our belief is that the one-step predictive task we defined is rather constrained. However, few data are available. Hence, we believe that models should be kept as simple as possible, but that they should be more informed. In the next chapter we present an attempt to propose such architecture.

9 AUTO-REGRESSIVE GENERATION

Among the examples generated by the RNN model investigated in Chapter 8, a number of notes would be considered as errors in the stylistic context of the classical period. Figure D.2 shows an example of a problematic semi-tone appearing in one instrumental section. This observation illustrates the fact that the model predicts each note without taking the other generated notes into consideration. Indeed, by using a sigmoid activation in its last layer, a feed-forward neural network model h^L independent Bernoulli distributions, where h^L is the last layer dimension (see Subsection 3.3.1). However, even though all output units are conditioned by the same penultimate layer h^{L-1}, they are not conditionally dependent. This assumption may not be adapted in a musical context, as the notes of a same chord are strongly dependent from each others. In Section 4.3, we introduced the *NADE* framework which has been developed for modelling any joint probabilities of a set of variables. Hence, in Section 9.1, we explain how to apply the *NADE* framework in the case of projective orchestration. Then, we detail how the inference mechanism of *NADE* can be used for injecting musical knowledge in Section 9.2.

9.1 APPLICATION TO PROJECTIVE ORCHESTRATION

NADE has originally been designed for purely generative model. However, the flexibility of the framework allows for easily combine various sources of information. Hence, conditioning in *NADE* can be simply achieved by concatenating the condition information to the input vector.

In the case of projective orchestration, the conditioning information is given by the piano and orchestral contexts, that we denoted p_t^c and O_t^c in Section 5.2. These informations can be embedded by using a neural network before being concatenated to the present orchestral vector $\hat{o}_t$ which is fed to the *NADE* network. Hence, the exact same *NADE* procedure as the one described in Equation 4.11 can be applied by simply modifying the input

$$h^0 = (x \odot m_{o_{<d}} \quad , \quad m_{o_{<d}} \quad , \quad emb) \tag{9.1}$$

This process is illustrated on Figure 9.1.

An efficient training procedure, called *Orderless Deep NADE* (*ODNADE*), consists in randomly sampling an ordering and an order (see Section 4.3). However, during the generation process, each unit of the predicted vector has to be sampled successively. Hence, the generation, ran at each validation step, requires to perform a number of forward passes equal to the output dimension. It rapidly becomes the time bottleneck of the training process for high-dimensional spaces, such as orchestral vectors.

Two networks have been evaluated for embedding the piano and orchestral context. A dense multi-layer architecture, denoted *NADE-MLP*, and a Recurrent Neural Network, denoted *NADE-RNN*. Note that this embedding is shared

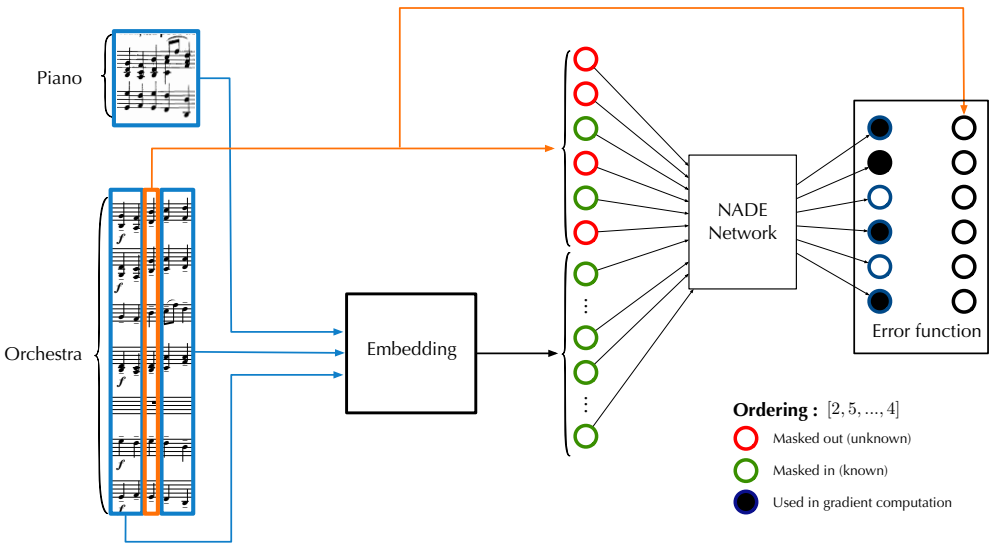

Figure 9.1: The *NADE* framework can be applied to perform projective orchestration by concatenating the embedding of the piano and orchestral contexts to the orchestral vector O_t.

across all orders, and needs to be computed only once at generation time. Hence, the complexity of the embedding network does not impact the generation time.

9.2 INFORMED ORDERING

The piano-roll representation implies that a prediction has to be made for all possible notes of the orchestra, whereas most of them are actually extremely unlikely to be played, given the harmonic context of the piano score and other notes in the orchestral frame. Determining a relation between the notes contained in the piano frame p_t and the notes allowed in the orchestral scores can be defined by hand, relying on musical theory. However, this method requires to be adapted for each different style of orchestration. A more automatized way is to count the number of co-occurrences between all piano and orchestral notes. The co-occurrence matrix C^{po} indicates for each pair of piano and orchestral note if they appeared simultaneously in the training database. Hence, for

$$\forall (i,j) \in [1, N_p] \times [1, N_o] \, , \begin{cases} C^{po}_{i,j} = 1 & if \sum_{(p,o)\in\mathcal{D}_{train}} \sum_t (p_t)_i . (o_t)_j > 1 \\ C^{po}_{i,j} = 0 & else \end{cases} \tag{9.2}$$

The masking mechanism proposed in *NADE* allows to easily enforce orchestral units to be set to zeros during the generation process. Indeed, the notes in the orchestral vector which have never been observed in the training set given the current piano score can automatically be set to zero, and considered as known. Hence, for a piano frame p_t, the mask can be initialised to the value $m = 1 - (p_t)^T . C^{po}$, since the scalar product between the transposed current piano frame and the co-occurrence matrix indicates the possible notes in the orchestral vector. The co-occurrence matrix can be observed in Figure 9.2 (). Then, the remaining indices to sample correspond to the values equal to zero in the mask, and are browsed in a random order.

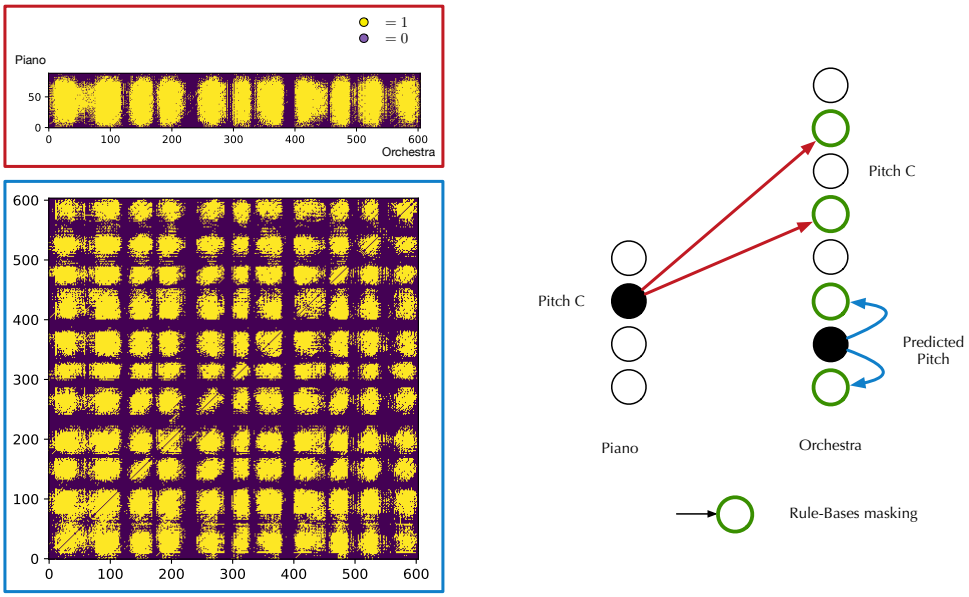

Figure 9.2: The piano and orchestral notes co-occurrence matrix can be used to automatically mask out orchestral units before starting the generation step (red frame and arrows). During the generation, inter-orchestral co-occurrences (blue frame) can be used to mask other orchestral units each time an orchestral unit has been predicted (blue arrows).

Note that this procedure does not speed up the generation. Indeed, since the ordering of the indices to be sampled is now dependent on the piano frame p_t, the generation cannot be performed over a batch of inputs any more.

The same principle can be applied with the co-occurrences of notes in the same orchestral frame. Each time an note is predicted in the orchestral vector on, the other notes which never occurred simultaneously in the training data can be automatically masked out. The inter-orchestral co-occurrence matrix C^{oo} is defined by

$$\forall (i,j) \in ([1,N_o])^2 \, , \begin{cases} C^{oo}_{i,j} = 1 & if \sum_{(p,o)\in\mathcal{D}_{train}} \sum_t (o_t)_i . (o_t)_j > 1 \\ C^{oo}_{i,j} = 0 & else \end{cases} \tag{9.3}$$

and can be observed in Figure 9.2 (blue frame).

Whereas the piano to orchestral co-occurrences are used to initialise the mask before the beginning of the generation process, the inter-orchestral co-occurrences are used along the *NADE* inference procedure, each time a note is predicted to be played.

9.3 RESULTS

The results obtained by different *NADE* architectures are compared in Table 9.1. In Section 4.3, we detailed the ensemble of *NADE* architecture, which consists in using the mean of several orderings as a prediction at generation time. The results presented here are obtained by combining 5 orderings randomly sampled.

	Accuracy (%)
RNN	42.74
NADE-MLP	40.78
NADE-RNN	43.36
NADE-informed	40.53

Table 9.1: Accuracy scores obtained by different architecture derived from the *NADE* framework. All architectures are similar, the *RNN* part being composed of two layers of 2000. The results are the mean score obtained over a 10-fold evaluation.

The *NADE-RNN* slightly improves the performances of the model. However, qualitatively, the results are almost the same. The orchestrations generated by the different *NADE* models can be downloaded on the companion website of the thesis [1]. However, we believe that we did not manage to completely take advantage of the *NADE* framework. Indeed, by observing the weights connecting the two last layers, it appears that the network does not seem to use the mask information (see Figure C.8). However, that information is crucial to distinguish between the notes which have been set to zero or the notes which are not yet predicted. Hence, we believe the performances obtained here do not reflect fully the potential of the *NADE* architecture and plan to further investigate it.

Besides, the informed ordering did not improve neither the scores obtained by the model. The co-occurrence matrix was a first attempt at embedding theoretic knowledge in the *NADE* framework, but it does not seem to be adapted. Indeed, the co-occurrence of a piano and orchestral note do not prove that the piano note conditioned the presence of the orchestral note, which could rather be associated to another note of that same piano vector. Using the ratio of co-occurrence and applying a threshold over that value could alleviate this issue, but with the risk of completely forbidding notes which are rare, but extremely relevant in particular contexts. Finally, we believe that a very careful rule-based masking could eventually be a better solution, that we are currently investigating.

Enforcing theoretic musical knowledge in a neural network by directly altering its sampling process may seem dubious for two reasons. Indeed, because these methods are often praised for allowing agnostic approaches of a problem. However, the piano-roll representation may not be suited for tackling this task, as the model has to make a prediction for each notes in the orchestral vector, whereas the piano score actually strongly constrains the choice of possible notes. Hence, when trying to model a particular style of orchestration, refusing to use the musical rules associated with this style deprives the model from a primordial information. In particular, the paucity of data available in the database needs to be somehow counterbalanced with additional knowledge and assumptions about the data structure.

1 https://qsdfo.github.io/LOP/results.html#NADE

Part V

LIVE ORCHESTRAL PIANO: A SYSTEM FOR REAL-TIME PROJECTIVE ORCHESTRATION

10 THE LIVE ORCHESTRAL PIANO ARCHITECTURE

In this chapter we describe the *Live Orchestral Piano* (*LOP*), a system for performing the live projective orchestration of a piano input played by a human performer. The Max/MSP patch we developed to that purpose is detailed in Section 10.1, while the constraints introduced by the real-time framework over the neural network performing the orchestration are detailed in Section 10.2.

Beside the recreational interest of developing this kind of real-time system for orchestration, it allows to rapidly evaluate the different architectures investigated. Indeed, by simply plugging a trained model in this system, we can directly test its capacity to orchestrate a piano input. Hence, the *LOP* provides an interactive way to test a model, while avoiding the tedious task of writing a musical score using an edition software for generating the orchestration of original scores.

10.1 MAX/MSP IMPLEMENTATION

In this section, we detail the implementation of a *Max/MSP* patch for performing the orchestration of a *MIDI* piano input in near real-time.

MIDI is primarily a communication protocol for electronic instruments (see Section 2.1.2), and the output of a *MIDI* piano can easily replace the *MIDI* scores we manipulated throughout this work. *Max/MSP* is a software for implementing real-time transformations over flows of audio or symbolic data, which provides a convenient way to retrieve and process that information. In our proposed implementation for the live orchestration of a piano input, a *Max/MSP* client collects the notes played on a *MIDI* piano keyboard and request its orchestration to a Python server which performs a forward pass in a neural network. The output of the neural network is a vector representing an orchestral frame, which is sent back to the *Max/MSP* patch to be rendered as an audio waveform. To do so, a Virtual Studio Technology (*VST*) plug-in for the rendering of orchestral sounds is embedded in the Max/MSP patch. We used *HALion*[1]. The communication between the *Max/MSP* client and the Python server is based on the *Open Sound Control* (*OSC*) protocol [2]. *OSC* is a communication protocol for computer and electronic instruments, and is a more modern equivalent of *MIDI*. Figure 10.1 depicts the workflow of the system.

The sequence of notes played by a human performer needs to be converted into a sequence of vectors. A vector is created by buffering all the *MIDI* events occurring during a short period of time, usually a 32^{th} note, using a tempo parameter defined by the user. The buffering mechanism is here to prevent small delays occurring when the finger of the performer does not hit the keyboard exactly at the same time while playing a chord. A vector is constructed only when *MIDI* events are received by the *Max/MSP* client. Hence, the communication

1 https://www.steinberg.net/en/products/vst/halion_und_halion_sonic/halion.html
2 http://opensoundcontrol.org/introduction-osc

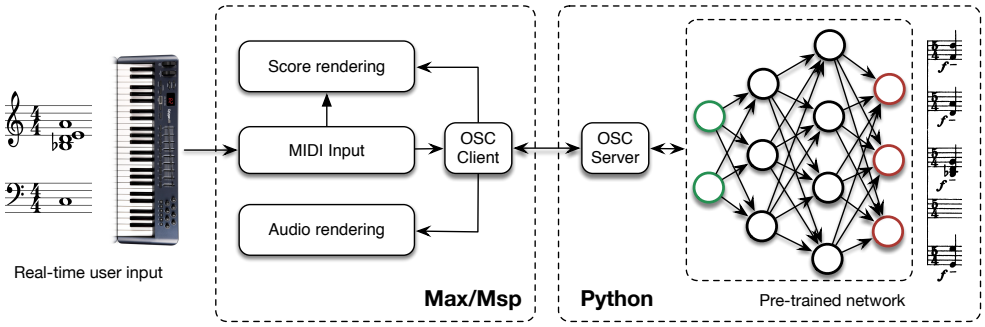

Figure 10.1: The *Live Orchestral Piano* (*LOP*) is a system for performing real-time orchestration of a piano performance. The system consists in a *Max/MSP* patch which receives the output of a *MIDI* piano keyboard, sequences it, and for each frame, requests its orchestration to a Python server which implements an already trained neural network. The network performs a forward pass and sends its output back to the *Max/MSP* patch, who uses it for displaying a musical score and to be rendered as an audio signal.

between the server and client is done asynchronously, and the client requests an orchestration each time a buffer is ready.

From the user perspective, the Live Orchestral Piano takes the form a *Max/MSP* patch whose interface can be observed in Figure 10.2. A musical score displays both the piano input and the generated orchestration. The neural network used by the Python server to perform the orchestration of the input piano frames can easily be modified by selecting an already trained architecture from a drop-down menu accessible in the *Max/MSP* patch. To convert the information of the probability activation into binary values, either sampling from a corresponding Bernoulli distribution, or selecting all the notes above a certain threshold can be used. In that later case, the threshold value can be modified on-line, resulting in more or less crowded orchestrations. A tempo parameter is needed for quantifying the durations written on the displayed musical scores. Its value can be modified by the user, and a metronome activated or not.

10.2 REAL-TIME GENERATION

In this section we detail the neural network architecture implemented on the Python server side.

10.2.1 *Forward generation*

In this section we present the type of model which can be implemented on the Python server. The framework introduced in Chapter 5 can easily be adapted to real-time constraints. Indeed, by relying solely on the past information for defining the piano and orchestral contexts, a causal generative model is adapted to real-time constraints (see Figure 10.3). The orchestral score is initialised with zeros for time indices smaller than the temporal order, as we observed in Section 8.3 that it obtained convincing results.

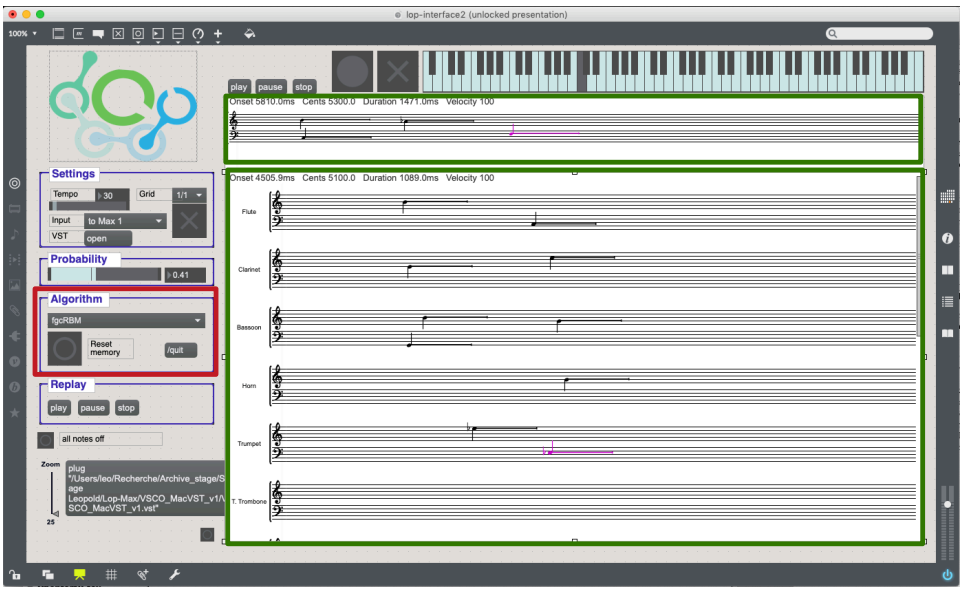

Figure 10.2: The Max/MSP interface of the Live Orchestral Piano allows to see both the piano input and the generated orchestration on a musical score generated as the user plays (green frames). Different models trained beforehand can be loaded in the *Max/MSP* (red frame).

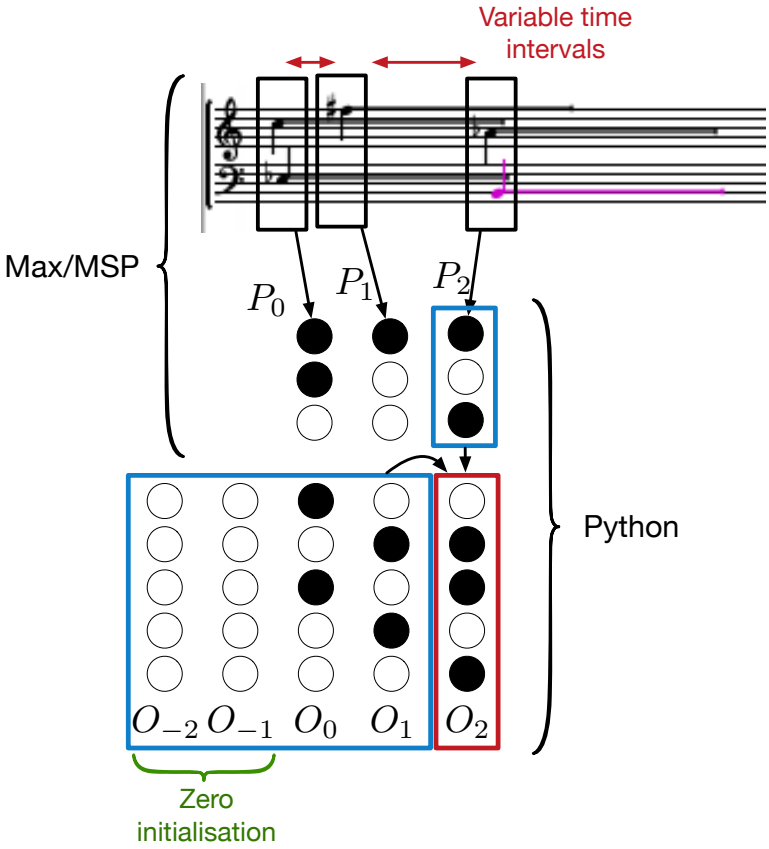

Figure 10.3: A piano vector is built each time a *MIDI* event is received by the *Max/MSP* patch. For the Python server, these vectors are processed in the event-level framework, thus dropping the duration information. The real-time framework requires a causal generative scheme to be employed, and zero-initialisation is used for unknown orchestral frames.

10.2.2 *Event-level representation*

The event-level representation is particularly adapted to the asynchronous communication scheme we established between the server and client sides. Indeed, the buffer vectors collected from the *MIDI* keyboard are sent to the Python server every time an event is detected, but not necessarily at regular time intervals (see Figure 10.3). Hence, in the event-level framework, the Python server side simply needs to keep track of the successive buffer vectors sent by the *Max/MSP* client and process them as they arrive, with no information about their absolute time relations.

10.2.3 *Time constraints*

The forward pass in the neural network needs to be fast enough so that the performer does not notice a delay between his playing and the production of the sound. A forward pass in a *RNN* model composed by 2 layers of 2000 units each implemented with the TensorFlow library takes less than 250 milliseconds on the 2,6 GHz Intel Core i5 processor.

For musical applications, up to 20 milliseconds of latency can be considered insignificant. Hence, our system is above that limit and, thus, is not fast enough for being called real-time. However, we experienced that it is already sufficiently reactive for providing the user with an interesting experience.

Part VI

CONCLUSIONS

11

CONCLUSIONS

In this work, we proposed to address for the first time the automatic projective orchestration of a piano score in the symbolic domain. We investigated statistical learning approaches, and more particularly neural networks models.

A first step was to gather a collection of piano scores and their orchestrations by famous composers in the *MIDI* and *MusicXML* formats. As the files were collected from various sources, important efforts were dedicated to establishing a uniform nomenclature for the instrumentation. Besides, a method for automatically aligning a piano score and its orchestration was proposed, to correct the structural differences that sometimes appeared between the respective files. This resulted in the creation of the Projective Orchestral Database (*POD*).

Then, a formalisation of the projective orchestration in the context of statistical learning was proposed, along with an evaluation framework. Both are based on a one step predictive task, which consists in predicting a single orchestral frame knowing the surrounding piano and orchestral contexts. The generation of a complete orchestration can then be performed in a frame-by-frame manner. Then, to assess the performances of different models on the projective orchestration task, we developed an evaluation framework similar to those derived for polyphonic music generation. We compare two evaluation metrics traditionally used for evaluating the predictive performances of a model, the binary cross-entropy and the accuracy, and conclude that the accuracy is more adapted in the case of orchestration.

In this framework, we implemented and evaluated several algorithms belonging to two families of models: energy-based models and feed-forward neural networks. In our proposed evaluation framework, feed-forward neural networks appeared to be the most suited architecture for addressing the projective orchestration task.

Hence, we decided to focus on neural networks models and investigated different solutions for improving their performances. A first attempt consisted in passing more information to the network, by providing the dynamics and duration of the notes, which are important parameters for a human orchestrator. The results were not improved and showed the inability of the architecture that we developed to take advantage of these informations. Our belief is that this extra knowledge is not useful for a one-frame predictive task, because it is already strongly conditioned by the contextual information of the piano and orchestral score.

Orchestral vectors are highly sparse, and the ratio of activations strongly varies depending on the instrument and pitch. To alleviate this issue, we proposed to use pre-computed biases in the last layer and alternative training criteria. These techniques quantitatively improved the performances of the models, but undesirable effects occurred on the generated orchestration. In particular, the reduced influence of the false positives led to more crowded orchestration.

The *POD* database covers a long period of time and consequently various styles of orchestration. We decided to focus on trying to model a stylistically coherent subset of the whole database, composed of Beethoven's Symphonies and their reductions by Franz Liszt. We investigated different training strategies consisting in presenting as training material successively more reduced subsets. We concluded that there is a trade-off between observing files from various origins, which may also bring information about the style we attempt to imitate, and maintaining a stylistic consistency in the dataset.

In the examples generated by our proposed models, we observed a lack of continuity in each instrumental voices. In order to alleviate that issue, we investigated different generative schemes which perform several passes over the generated score. Causal and random-walk processes were combined in various manner. However, the most simple forward generation with a zero initialisation led to the most satisfying results. We believe that this is due to the important differences between the training process and more sophisticated generative schemes. Indeed, teacher forcing is used during the training step. The extremely rich information provided to the model greatly reduce the complexity of the prediction. However, it is not accessible at generation time.

The observation of aberrant co-occurrences of notes in a same chord in the generated orchestration led us to try to introduce conditional dependencies between the notes of a same orchestral vector. The *Neural Auto-regressive Distribution Estimation* (*NADE*) framework provides an elegant way to model joint distributions while relying on an efficient implementation based on feed-forward neural networks. We proposed an informed *NADE* process in which notes in the orchestral score are automatically masked out given the piano score, and the orchestral notes already predicted at a given moment. In an attempt to maintain an agnostic approach of the problem, we tried to automatically infer the masking rules from the data. However, the crude approach we used did not lead to convincing results, and discussed about the idea of enforcing theoretic music knowledge specific to the style we are trying to model, which might yield better results.

Finally, we proposed a system for the real-time orchestration of a piano performance. The proposed system is based on a Max/MSP server which receives the output of *MIDI* piano keyboard and request its orchestration to a Python client which implements a neural network previously trained on the projective orchestration task.

The ensemble of this work constitutes a first attempt at performing the automatic projective orchestration of a piano score. The quality of the orchestrations generated by our system are still far from being convincing, but they rather define a baseline against which other models and approaches can be compared in the future, and draw directions for the future works which we present in the next chapter.

12 FUTURE WORKS

12.1 SEQUENCE TO SEQUENCE

We observed a form of glass-ceiling over the performances of the different approaches we investigated. We believe that this is partly due to the piano-roll representation we used for the piano and orchestra scores. Indeed, it creates high-dimensional sparse vectors which proved to be challenging in a learning context. Hence, we believe that a more structured representation can be found, with stronger assumptions over the shape of the prediction. A possibility would be to represent each orchestral frame as a list of triplets containing the pitch-class, octave and instrument name for each note. Using this representation, a sequence-to-sequence model, similar to the one used for neural translation tasks [5], can be used to predict a single frame (see Figure 12.1). The softmax cross-entropy function that would be used in this framework contains much more information about the structure of the data than the binary cross-entropy function we used in this work. Similar approaches have been successfully applied to music generation [48] and arrangement for piano [34].

To model temporal relations using this data representation, a bi-directional *RNN* can be used and recurrent connections deployed over the final state of the piano embedding. This idea is depicted on Figure 12.1.

12.2 DETECTING SECTIONS CHANGES

Orchestrating a piano score necessitates a thorough understanding of its structure. In particular, a temporal segmentation of the musical discourse in structuring elements seems important in order to determine whether the orchestration should ensure a smooth and continuous voice leading inside each instrument section (in the middle of a musical phrase), or rather search for discontinuity in the instrumentation (for example at the beginning of a new section, or if a sentence is repeated twice). The approaches we proposed in this work lack this crucial information, and we do not believe the end-to-end approaches we investigated would be capable of learning to detect such structures. Musical analysis is a complex discipline and leading a thorough analysis of a musical piece demands a wide knowledge. However, we believe that unsupervised methods could be used for teaching a model to extract useful informations about the structure of a piece. Indeed, a neural network trained for polyphonic music generation can be used to build a curve of "entropy" of the piece. This entropy curve would be defined by measuring the prediction error of the model on each frame (see Figure 12.2). Hence, high error frames could indicate brutal changes in the musical discourse, whereas extremely low errors may indicate repetitions or a steady structure. Also, variations in the dynamics, or in the tempi could be used to detect important structures.

Such entropy curve could be used in several manner. A first possible solution would be to pass the entropy value associated with the piano frame as an input

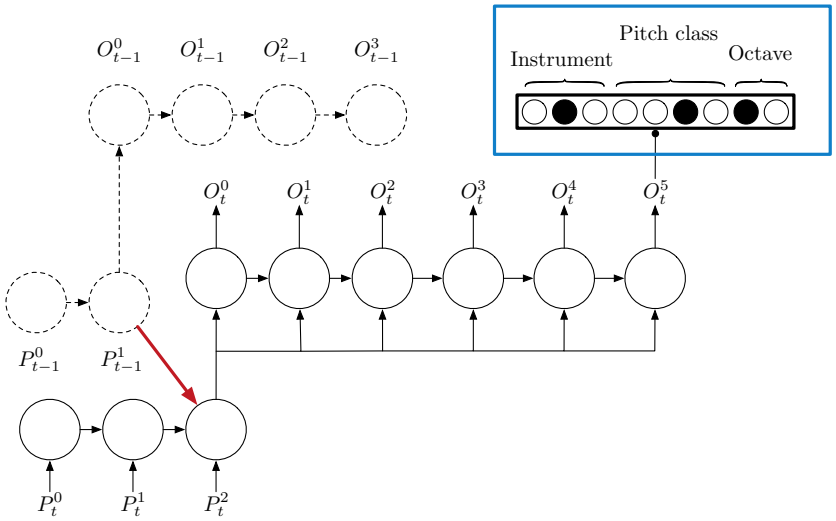

Figure 12.1: A symbolic frame can be represented as a list of notes. On the figure, the piano frame P_t contains 3 notes, and is orchestrated with an orchestral frame O_t containing 6 notes. A given note is represented by the concatenation of three vectors representing the pitch-class, octave and instrument. The interest of this representation is that each vector is now a one-hot vector. Hence, the last layer can implement a softmax function which conveys more information about the structure of the output than independent Bernoulli. Besides, each one-hot vector has a limited dimension (12 for the pitch-classes, 8 for the octaves and 12 for the instruments, using the instrumental simplifications we used in *LOP*). Using this representation (blue frame), a sequence-to-sequence model can be used, each frame being consider as a sequence. Time dependency can be ensured by conditioning the piano embedding at time t under the value of the piano embedding at time $t-1$. Note that piano and orchestral frames at previous time step (dotted circles) do not necessarily contain the same number of notes, which is made possible by the sequence-to-sequence framework.

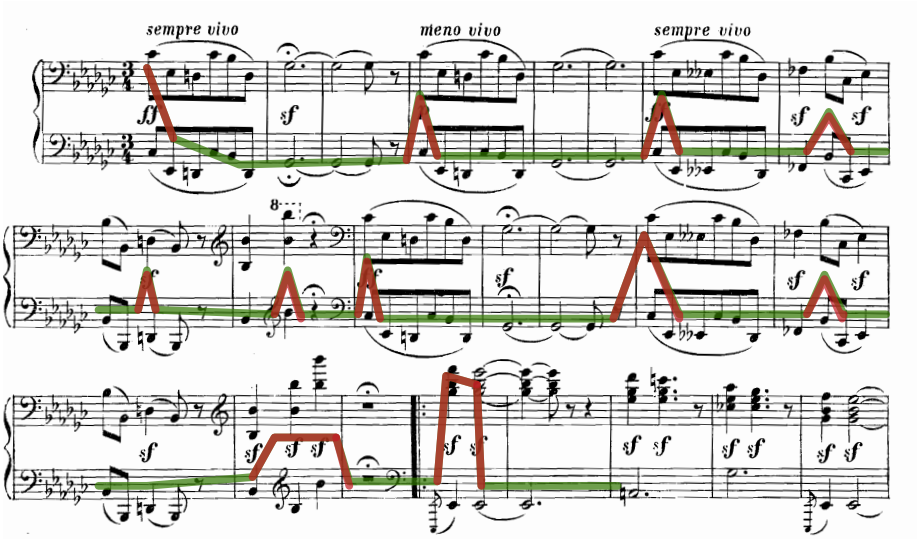

Figure 12.2: An entropy function can be computed over the piano score. Depending on the value of the entropy, a different model could be used. The red part, associated with an important change in the score would be orchestrated by a model which does not attempt to ensure a continuous voice leading for each instrumental section.

to the network performing orchestration. If the network manage to efficiently use this information, it should adapt its behaviour depending on the degree of uncertainty of the frame being orchestrated. A second solution would be to use different models depending on the entropy value.

In particular, a model specifically tailored for orchestrating high-entropy frames, could rely on a different paradigm than the one-step predictive framework we proposed in this work. Indeed, these frames would ideally be associated with important structural changes. Thus, it can be the ideal location for inserting interactivity by asking the user for a particular tone colour, or to select the instruments to be used. Hence, more interactivity with the system could be obtained by requesting the user to provide such inputs every time the system hits a high-entropy frame.

12.3 MULTI-MODAL ARCHITECTURES

12.3.1 *Metric learning embedding*

A major pitfall when attempting to model symbolic sequences is that the usual distances (such as the L2 norm) used to compare two frames of a piano-roll representation do not account for the perceptive proximity between the two corresponding sounds. This is particularly true in the case of orchestral music, where various combinations of instruments, and thus widely different piano-roll representations, can create very similar sounds. Hence, we believe that the symbolic distances used to train and evaluate models are not relevant. Ideally, two musical symbols should first rendered as audio waveform, and compared in this modality. However, this is both time consuming and inadequate in a statistical learning framework as the rendering operation is not differentiable. A possible solution consists in creating an embedding space for the symbolic representation in which geometric and perceptual distances are linked.

Building such space can be achieved by enforcing symbolic vectors which produce perceptually similar sounds to be mapped to points which are close from each others. A methods for automatically finding an embedding space which accounts for pairwise similarity relying on siamese networks has been proposed in [19]. A slightly more sophisticated approach relying on triplet comparisons has been proposed in [105]. This idea could be applied to orchestral vectors by assigning a binary label to pairs of symbolic vectors, depending on their proximity in term of sound (see Figure 12.3).

The mapping toward this embedding space is a neural network, and, thus, is differentiable. Hence, this space can be used for measuring the error function of a model during its training step, as the gradient of the error can be back-propagated through the embedding operation. Note that in the context of projective orchestration, some constraints, such as respecting the harmonic structure of the piano score, might be more clearly defined in a symbolic space rather than in an embedding space as the one we described before. Hence, an efficient training criterion would probably combine a symbolic measure with a distance in the perceptually informed embedding space.

To decide the value of the label assigned to pairs of symbolic vectors, the joint information of symbolic vectors and corresponding audio rendering is necessary. It can be obtained either by collecting musical score and the record-

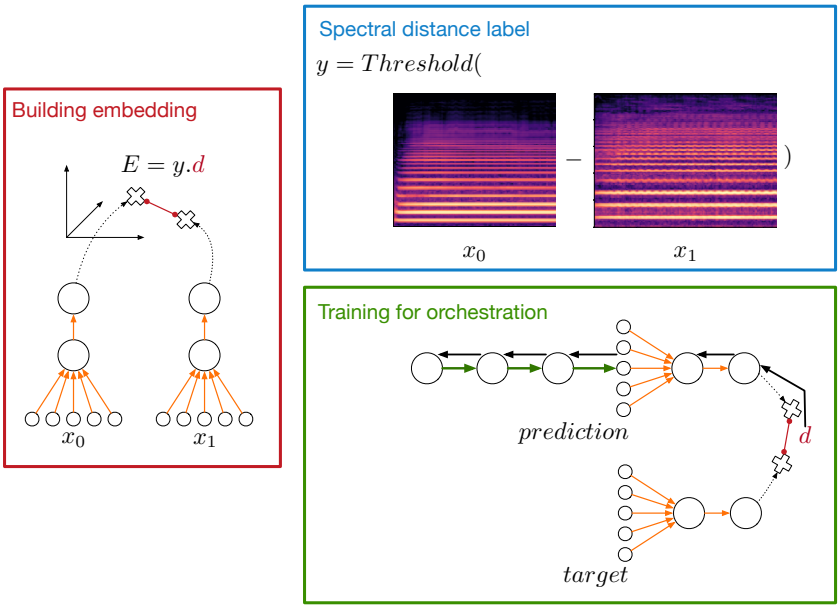

Figure 12.3: An embedding space which accounts for spectral distances can be built using siamese networks (red frame). The two networks with orange connections are one same network duplicated (called siamese networks). That duplicate network is trained to minimise the distance in the embedding space of symbolic vectors which are associated to sounds which are spectrally close. A possible way to do so is to associate pairs of symbolic vectors with a binary label which indicate whether or not their spectral counterparts are close (blue frame), and use that label to modify the sign of the distance used in embedding space for training the siamese architecture. Once such architecture has been trained, it can be used to measure the distance between the prediction of an orchestral symbolic frame and a target (green frame). Because the embedding is computed thanks to a differentiable network, the error measured in that space can be back-propagated through the embedding. Note that in that case, only the green weights are modified, and the orange weights corresponding to the embedding network are frozen.

ing of corresponding performances, or by using a rendering software. Both approach have advantages and drawbacks. Using real performances leads to more realistic audio examples, but the alignment and segmentation of the audio file might be tedious (however solutions exist, such as in [46]). Besides, finding an adequate measure to compare audio segments of different lengths extracted from various sources might be difficult. On the other side, using a rendering software provides a more controlled environment, but at the cost of less realistic examples, in particular regarding complex orchestral effects such as instrumental blending.

12.3.2 *A latent space for transcription*

We mentioned in the Section 12.2 that an interesting model for high-entropy frames could include a user input asking fo a particular tone-colour. To achieve this, a model able to generate symbolic vectors corresponding to particular spectral descriptors has to be designed. This is actually very close to the task

tackled by existing transcription systems such as *Orchids* [36] or *SPORCH* [92]. Hence, a solution would be to include them as a module of a more complex system.

Another solution could be to train an encoder-decoder model on a transcription task. The encoder would receive as input an information describing the audio signal, either in the form of a waveform or a time-frequency representation, and map it as a point in a *latent* space. That point is passed to the decoder whose objective is to find the corresponding symbolic representation (typically a frame of piano-roll). Variational inference can be used to obtain latent representations drawn from a desired probability distribution [60]. The advantage is that the latent space can then easily be sampled as the location of *relevant* latent points is known.

This is particularly interesting in our case, as it provides a generation mechanism for symbolic vectors. A priori, the latent space dimensions do not correspond to spectral descriptors or any intuitive quantity, but most likely embed informations about both the audio and symbolic domains. Exploring the properties of a latent space automatically inferred as being the most adapted for connecting the two modality would surely be a thrilling experience.

Part VII

APPENDIX

A

MACHINE LEARNING

A.1 ERROR FUNCTION IN A *restricted boltzmann machine* (*rbm*)

In a *RBM*, the parameters inference consists in minimizing the negative log-likelihood of a training dataset. However, the likelihood of a set of visible units is intractable in a *RBM*. We describe here the calculus that lead to a tractable estimation of this quantity.

The likelihood of a visible vector v is given by

$$p(v) = \sum_h p(v,h) = \sum_h \frac{e^{-E(v,h)}}{Z}$$

The free-energy is frequently defined as

$$F(v) \triangleq -log\left(\sum_h e^{-E(v,h)}\right)$$

Both the energy and free energy depend on the model's parameters, but we omit the term for keeping uncluttered notations.

Using the free-energy, the summation over the hidden vector possible values can be hidden, which leads to a more compact expression of the likelihood

$$p(v) = \frac{e^{-F(v,h)}}{Z}$$

$$Z = \sum_x e^{-F(x)}$$

Hence, for a parameter of the model $\theta \in \Theta$

$$\begin{aligned} -\frac{\partial}{\partial\theta}\log[p(v)] &= -\frac{\partial}{\partial\theta}log[\frac{e^{-F(v)}}{Z}] \\ &= \frac{\partial}{\partial\theta}F(v) + \frac{\partial}{\partial\theta}log(Z) \end{aligned}$$

The second term is equal to

$$\begin{aligned}\frac{\partial}{\partial\theta}log(Z) &= \frac{1}{Z}\frac{\partial Z}{\partial\theta}\\ &= \frac{1}{\sum_{\tilde{v}} e^{-F(\tilde{v})}}\cdot\sum_{\tilde{v}}\frac{\partial}{\partial\theta}e^{-F(\tilde{v})}\\ &= -\sum_{\tilde{v}}\frac{e^{-F(\tilde{v})}}{\sum_{\tilde{v}} e^{-F(\tilde{v})}}\frac{\partial F(\tilde{v})}{\partial\theta}\\ &= -\sum_{\tilde{v}} p(\tilde{v})\frac{\partial F(\tilde{v})}{\partial\theta}\end{aligned}$$

Finally,

$$-\frac{\partial log(p(v))}{\partial\theta} = \frac{\partial\mathcal{F}(v)}{\partial\theta} - \sum_{\tilde{v}} p(\tilde{v})\frac{\partial\mathcal{F}(\tilde{v})}{\partial\theta}$$

The second term is the expectation of the free-energy value over the model distribution. It can be approximated by averaging the value of the free-energy computed over a fixed number of sample drawn from the model distribution. A approximation which proved to be sufficient for the training step consists in using only one sample for computing the mean, if this sample is drawn after initializing the Gibbs chain with the known sample v. This process is known as contrastive divergence. Finally, the derivative of the negative log-likelihood can be approximated by

$$-\frac{\partial log(p(v))}{\partial\theta} \approx \frac{\partial}{\partial\theta}\left[F(v) - F(\tilde{v})\right]$$

Hence, the difference between the positive and negative free energy can be used as a differentiable criterion for training *RBM*

$$E(\theta) = F(v) - F(\tilde{v})$$

A.2 UP CRITERION

Over-fitting can be detected by observing the error function of the model over a set of validation data which is distinct from the training set. However, validation errors curves are rarely monotonic, and detecting over-fitting is a complex task. The *UP-criterion* is a heuristic for detecting over-fitting during the training

process of a parametric probabilistic model, which checks if the validation error increased repeatedly over successive intervals.

Data: V: list containing the validation score at each iteration epoch: the index of the current epoch
S: number of strips
k: validation order
Result: Overfitting: boolean, true if the training process has to be stopped

```
s = 0;
Overfitting = True;
while s < S do
    t = epoch - s;
    tmk = t - k;
    if V[t] > (V[tmk] - ϵ) then
        Overfitting = False
    end
    s = s + k
end
```

Algorithm 1: The UP-s algorithm checks that on S successive strips of size k the validation score has increased at least once. If this is not the case, the training process is stopped. The validation score is assumed to be increasing with the quality of the model and the algorithm has to be modified if a decreasing validation error is used instead.

A.3 K-FOLD VALIDATION

When few data are available, the score obtained by a model on a test subset may vary importantly depending on the selected subset. To alleviate this issue, a technique consists in partitioning the entire dataset in K folds and evaluate the model on each folds, while training it on the remaining data.

Data: M: model
$\mathcal{D}$: dataset
K: split order
Result: Res: list of length K containing the model's evaluation on each part

```
Partition 𝒟 in equally sized parts 𝒟_0, ..., 𝒟_{K-1}
for fold in range(K) do
    initialize M
    train M on 𝒟 − 𝒟_i
    score := evaluate on 𝒟_i
    Res[fold] := score
end
```

Algorithm 2: K-fold evaluation algorithm. The output is a list containing the performance of a given architecture on each K partition of a dataset $\mathcal{D}$. Note that between each fold, the model is re-initialized and trained

B

LOP DATABASE

Original	Simplified
Piccolo	Flute
Flute	Flute
Alto-Flute	Flute
Soprano-Flute	Flute
Tenor-Flute	Flute
Bass-Flute	Flute
Contrabass-Flute	Flute
Pan Flute	Remove
Recorder	Flute
Ocarina	Remove
Oboe	Oboe
Oboe-dAmore	Oboe
Oboe de Caccia	Oboe
English-Horn	Oboe
Heckelphone	Remove
Piccolo-Clarinet-Ab	Clarinet
Clarinet	Clarinet
Clarinet-Eb	Clarinet
Clarinet-Bb	Clarinet
Piccolo-Clarinet-D	Clarinet
Clarinet-C	Clarinet
Clarinet-A	Clarinet
Basset-Horn-F	Clarinet
Alto-Clarinet-Eb	Clarinet
Bass-Clarinet-Bb	Clarinet
Bass-Clarinet-A	Clarinet
Contra-Alto-Clarinet-Eb	Clarinet
Contrabass-Clarinet-Bb	Clarinet
Bassoon	Bassoon
Contrabassoon	Bassoon
Soprano-Sax	Remove
Alto-Sax	Remove

Tenor-Sax	Remove
Baritone-Sax	Remove
Bass-Sax	Remove
Contrabass-Sax	Remove
Horn	Horn
Harmonica	Remove
Piccolo-Trumpet-Bb	Trumpet
Piccolo-Trumpet-A	Trumpet
High-Trumpet-F	Trumpet
High-Trumpet-Eb	Trumpet
High-Trumpet-D	Trumpet
Cornet	Trumpet
Trumpet	Trumpet
Trumpet-C	Trumpet
Trumpet-Bb	Trumpet
Cornet-Bb	Trumpet
Alto-Trumpet-F	Trumpet
Bass-Trumpet-Eb	Trumpet
Bass-Trumpet-C	Trumpet
Bass-Trumpet-Bb	Trumpet
Clarion	Remove
Trombone	Trombone
Alto-Trombone	Trombone
Soprano-Trombone	Trombone
Tenor-Trombone	Trombone
Bass-Trombone	Trombone
Contrabass-Trombone	Trombone
Euphonium	Remove
Tuba	Contrabass
Bass-Tuba	Contrabass
Contrabass-Tuba	Contrabass
Flugelhorn	Remove
Piano	Piano
Celesta	Remove
Organ	Violin and Viola and Violoncello and Contrabass
Harpsichord	Remove
Accordion	Remove
Bandoneone	Remove

Harp	Remove
Guitar	Remove
Bandurria	Remove
Mandolin	Remove
Lute	Remove
Lyre	Remove
Strings	Violin and Viola and Violoncello and Contrabass
Violin	Violin
Violins	Violin
Viola	Viola
Violas	Viola
Viola de gamba	Viola
Viola de braccio	Remove
Viola dAmore	Remove
Violoncello	Violoncello
Violoncellos	Violoncello
Contrabass	Contrabass
Basso continuo	Remove
Bass drum	Remove
Glockenspiel	Remove
Xylophone	Remove
Vibraphone	Remove
Marimba	Remove
Maracas	Remove
Bass-Marimba	Remove
Tubular-Bells	Remove
Clave	Remove
Bombo	Remove
Hi-hat	Remove
Triangle	Remove
Ratchet	Remove
Drum	Remove
Snare drum	Remove
Steel drum	Remove
Tambourine	Remove
Tam tam	Remove
Timpani	Remove
Cymbal	Remove

Castanets	Remove
Percussion	Remove
Voice	Violin and Viola and Violoncello and Contrabass
Voice soprano	Violin
Voice mezzo	Viola
Voice alto	Viola
Voice contratenor	Viola
Voice tenor	Violoncello
Voice baritone	Violoncello
Voice bass	Contrabass
Ondes martenot	Remove
Unknown	Remove

Table B.3: To reduce the dimensionality of the target space, several instruments are grouped under the same section name. We tried to link together instruments belonging to the same family, and thus sharing similar timbres. For instance, all the possible flutes are grouped under the same section named "flute".
The timbre of a bass flute is widely different from the one of a Piccolo. However, depending on the pitch associated to the flute section, it can easily be decided the type of flute best suited for playing this section.
In term of structure for the training algorithms, this grouping is rather challenging, as the tessitura of each section is large, and they widely overlap between them.
On the other side, the notations used in some scores are in some rare cases fuzzy, as in "Strings". In that case, we can associate this notation with several instruments, for instance "Violin and Viola and Violoncello and Contrabass". In that case, the notes played in that track will be associated to the corresponding instruments if the pitch range corresponds. For instance, if in the "Strings" track some notes are played in the range of the double-bass and violoncello, these notes will be associated to these two sections.
Some extremely rare instruments, such as the Pan Flute are simply removed from the score when encountered. Percussions have also been removed as there were too many different notations across the database.

Composer	Period	Number of piano files	Percentage piano frames	Number of orchestra files	Percentage orchestra frames
Arcadelt. Jacob	1507-1569			1	0.07
Arresti. Floriano	1667-1717	3	0.57		
Bach. Anna Magdalena	1701-1760	3	0.43		
Bach. Johann Sebastian	1685-1750	9	4.57	4	0.81
Banchieri. Adriano	1568-1634			1	0.32
Beethoven. Ludwig Van	1770-1827	1	0.60	38	42.28
Berlioz. Hector	1803-1869			1	0.14
Brahms. Johannes	1833-1897			3	0.28
Buxtehude. Dietrich	1637-1707	1	0.21		
Byrd. William	1538-1623	1	0.13		
Charpentier. Marc-Antoine	1643-1704			2	0.38
Chopin. Frederic	1810-1849			2	0.44
Clarke. Jeremiah	1674-1707			1	0.23
Debussy. Claude	1862-1918	1	0.59	6	0.90
Dvorak. Anton	1841-1904			6	2.42
Erlebach. Philipp Heinrich	1657-1714			1	0.10
Faure. Gabriel	1845-1924			1	0.60
Fischer. Johann Caspar Ferdinand	1656-1746	1	0.10		
Gluck. Christoph Willibald	1714-1787			1	1.61
Grieg. Edvard	1843-1907			1	2.10
Guerrero. Francisco	1528-1599	1	0.12		
Handel. George Frideric	1685-1759	4	1.00	1	0.75
Haydn. Joseph	1732-1809			6	1.01
Kempff. Wilhelm	1895-1991	1	1.58		
Leontovych. Mykola	1877-1921			2	0.22
Liszt. Franz	1811-1886	34	39.98		
Mahler. Gustav	1860-1911			1	0.85
Mendelssohn. Felix	1809-1847			2	1.41
Moussorgsky. Modest	1839-1881			1	0.04
Mozart. Wolfgang Amadeus	1756-1791	1	0.71	8	1.45
Okashiro. Chitose	?-alive	3	1.09		
Pachelbel. Johann	1653-1706	1	0.15		
Praetorius. Michael	1571-1621			2	0.14
Purcell. Henry	1659-1695			1	0.08
Ravel. Maurice	1875-1937	6	6.49	8	6.69
Rondeau. Michel	1948-	2	0.25	1	0.14
Schonberg. Arnold	1874-1951			1	0.21
Schumann. Robert	1810-1856			1	0.05
Shorter. Steve	1958-?	1	0.26		
Smetana. Bedrich	1824-1884			1	0.61
Soler. Antonio	1729-1783	1	0.54		
Strauss. Johann	1825-1899			1	0.04
Strauss. Richard	1864-1949			1	0.22
Stravinsky. Igor	1882-1971			4	0.94
Tchaikovsky. Piotr Ilyich	1840-1893			36	20.08
Telemann. Georg Philipp	1681-1767			2	1.04
Unknown.		107	40.18	28	7.47
Vivaldi. Antonio	1678-1741			4	2.94
Walther. Johann Gottfried	1684-1748	1	0.14		
Wiberg. Steve	1974-?			1	0.75
Zachow. Friedrich Wilhelm	1663-1712	1	0.32	2	0.23

Table B.1: This table describes the relative importance of the different composers present in the database. For each composer, the number of piano and orchestral files are given in number of files, and in number of frames. The number of frames depends also on the duration of each file, and is more relevant in a learning context as it determines the number of training points from one composer a model will observe. The period of each composer is indicated.

Intensity tag	Value
ppp	16
pp	32
p	48
mp	64
mf	80
f	96
ff	112
fff	127

Table B.2: Corresponding *MIDI* intensity used for the dynamic tags when parsing *MusicXML* files. Values found in Logic Pro 9 User Manual [68, p.474]

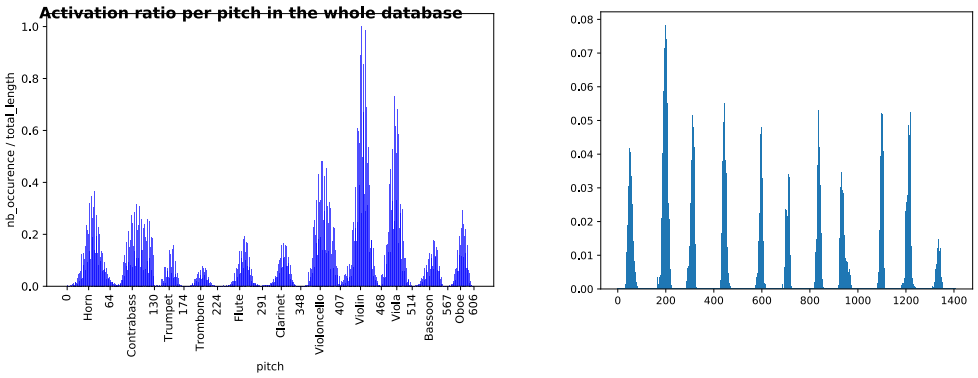

Figure B.1: This figure represents the pitch ratio activations over the complete database when limiting the instrumental range to the tessitura observed in the database (Left) and not (Right). Limiting the tessitura allows to dramatically reduce the dimension of the orchestral vector, from 1409 to 606. Furthermore, it leads to a better conditioning of the orchestral vectors in order to be used for statistical learning, as it avoids units always set to zeros.

C

HYPER-PARAMETERS

C.1 UNITS TYPE

C.2 ERROR SURFACES

NEGATIVE MODIFIED ACCURACY

$$Accuracy\_mod(O_t, \hat{O}_t) = -100 * \frac{TP}{TP + FP + FN}$$

NEGATIVE WEIGHTED ACCURACY

$$Accuracy\_mod(O_t, \hat{O}_t) = -100 * \frac{TP + \boldsymbol{\lambda} \boldsymbol{TN}}{TP + FP + FN + \boldsymbol{\lambda} \boldsymbol{TN}}$$

WEIGHTED CROSS-ENTROPY

$$W - Xent = -x.log(p) - \boldsymbol{\lambda}\ (1 - x).log(1 - p) \tag{C.1}$$

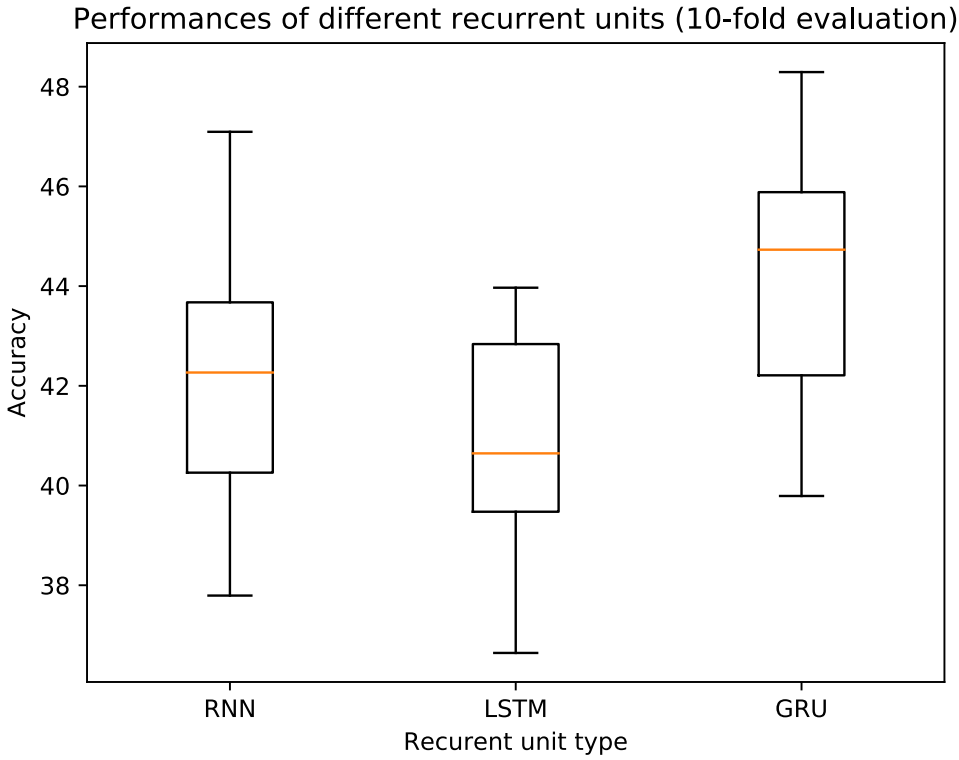

Figure C.1: Performances obtained with three different types of recurrent units on the predictive orchestration task. The boxes illustrate the results over a 10-fold evaluation, for a recurrent network made of 2 layers of 2000 units each and using weight decay as regularization. *RNN* refers to the vanilla implementation of recurrent units. The *GRU* surpasses the *LSTM* and Simple *RNN* units on this particular taskx

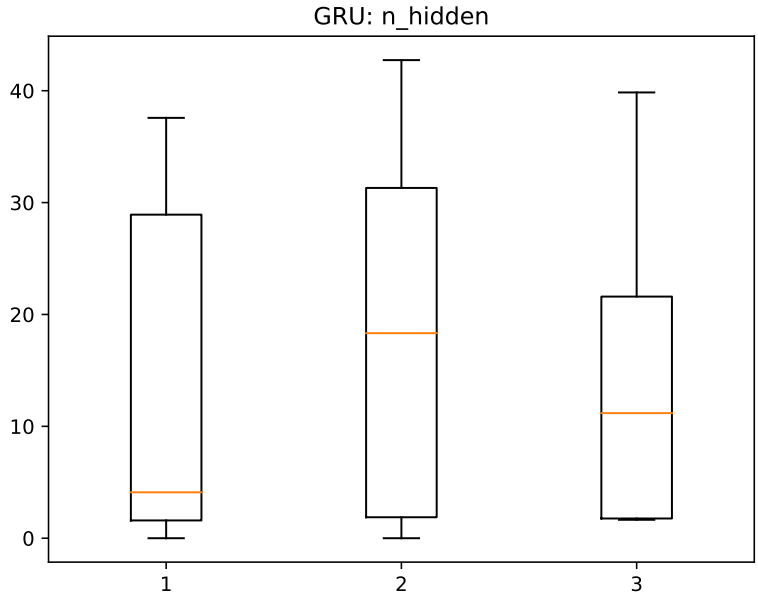

Figure C.2: Influence of the number of recurrent layers on the accuracy performances of the GRU model

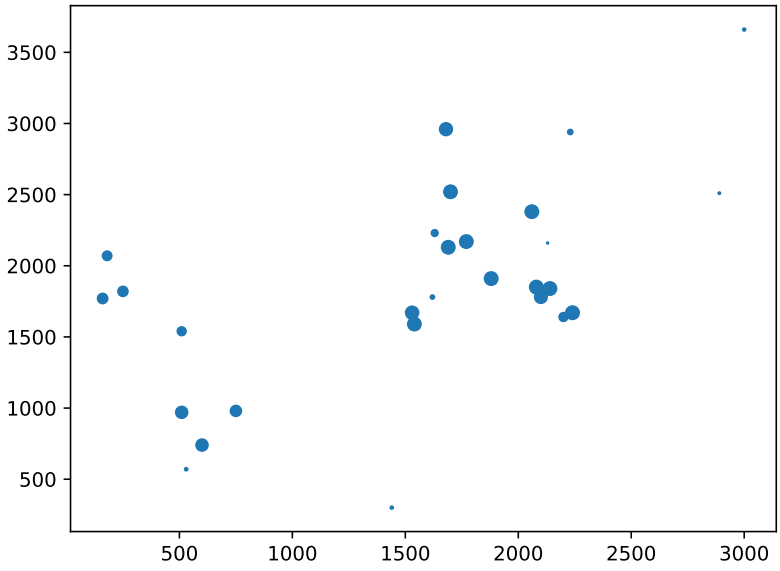

Figure C.3: Influence of the number of units in the two layers recurrent networks of the orchestral embedding on the accuracy performances of the GRU model. Each dot represent a model, and its diameter is the mean accuracy obtained over a 10-fold validation.

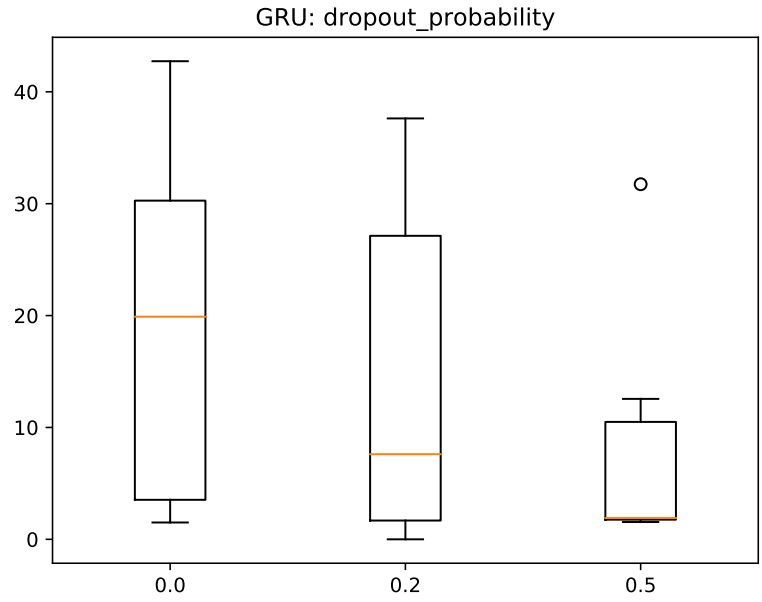

Figure C.4: Influence of the dropout value on the accuracy performances of the GRU model The poor results obtained when a larger value of dropout is used is probably related to the fact that we used *ReLU* in intermediate layers. Indeed, *ReLU* already have a tendency to output sparse intermediate representations. Hence, applying dropout with an high probability on top of *ReLU* will likely shut down most of the intermediate layers units.

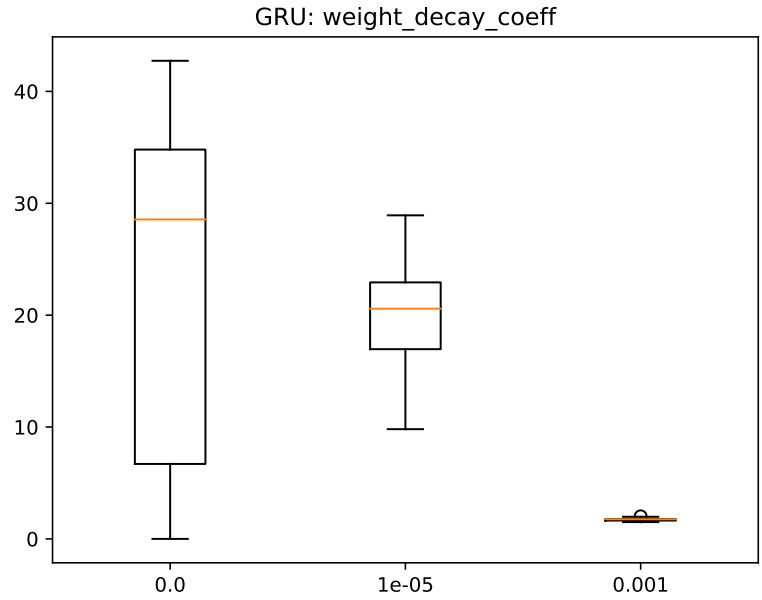

Figure C.5: Influence of the weight decay parameter on the accuracy performances of the GRU model

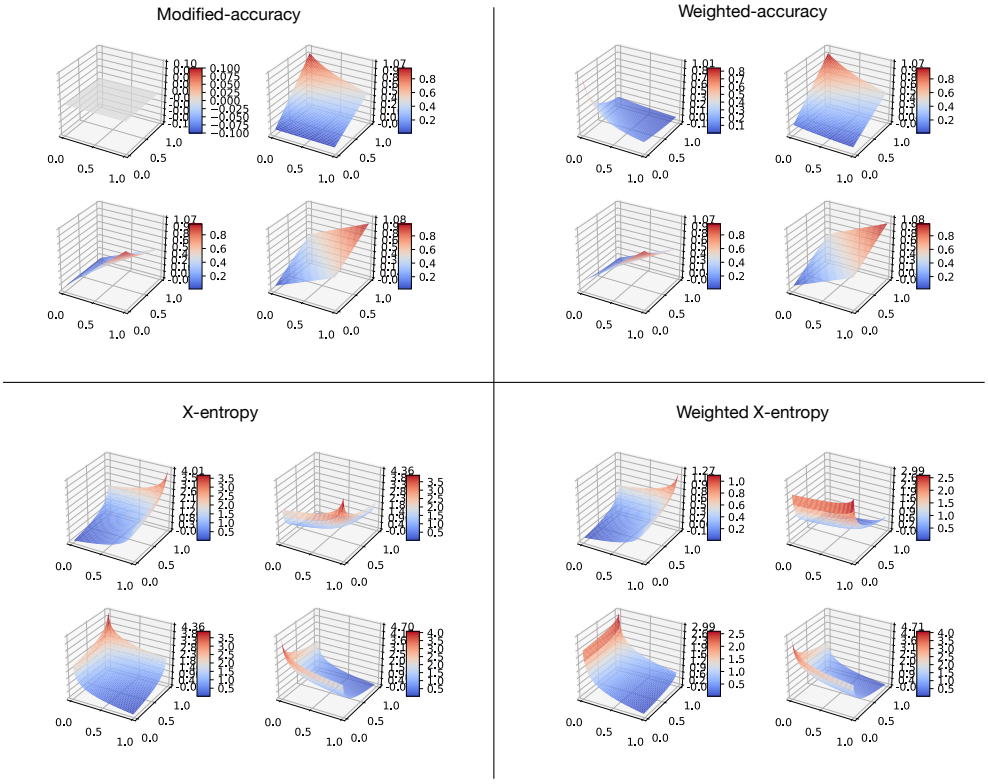

Figure C.6: This figure represents the error surfaces of the negative modified-accuracy, negative weighted accuracy, binary cross-entropy and a weighted binary cross-entropy in the case of a two dimensional target. The two axis represent the probability of activation, which has value between 0 and 1. For each measure, the top-left surface correspond to a target value $(0, 0)$, top-right $(0, 1)$, bottom-left $(1, 0)$ and bottom-right $(1, 1)$.

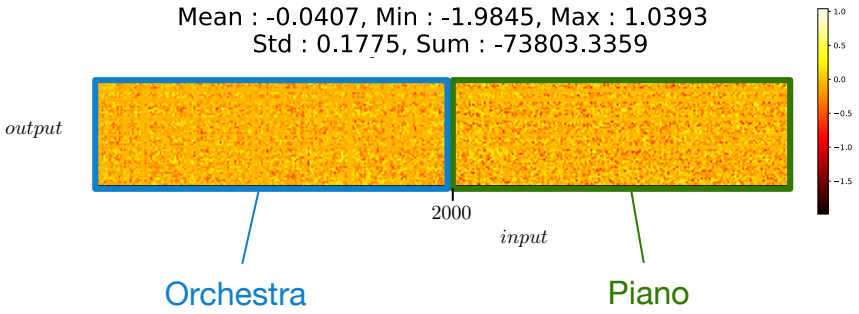

Figure C.7: The piano and orchestra embeddings are concatenated before being used for prediction. Hence, the weights connected the embeddings to the prediction directly represent the respective influence of each information. However, we detected no significant differences in the structure of the weight matrix. Note that, for the purpose of this experiment, no weight decay has been applied to this layer to avoid enforcing a Gaussian distribution of the weights. The results were not deteriorated by doing this.

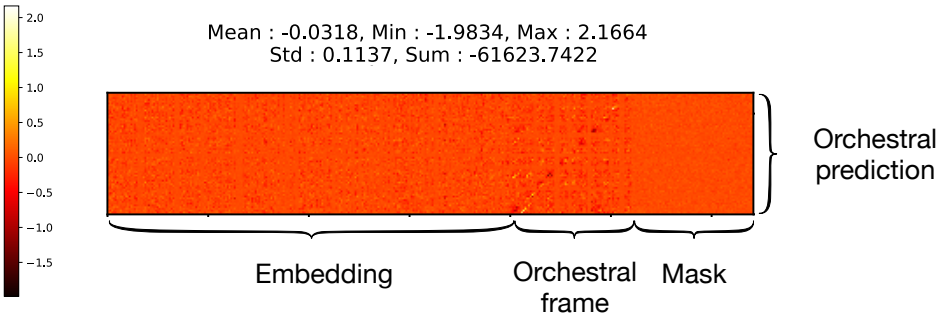

Figure C.8: In our proposed architecture relying on the *NADE* framework, the last layer weights connect the concatenation of the embedding, orchestral frame and mask to the orchestral prediction. However, the weights corresponding to the binary mask are set to zeros, indicating that the model discard that crucial information.

D

GENERATED SCORES

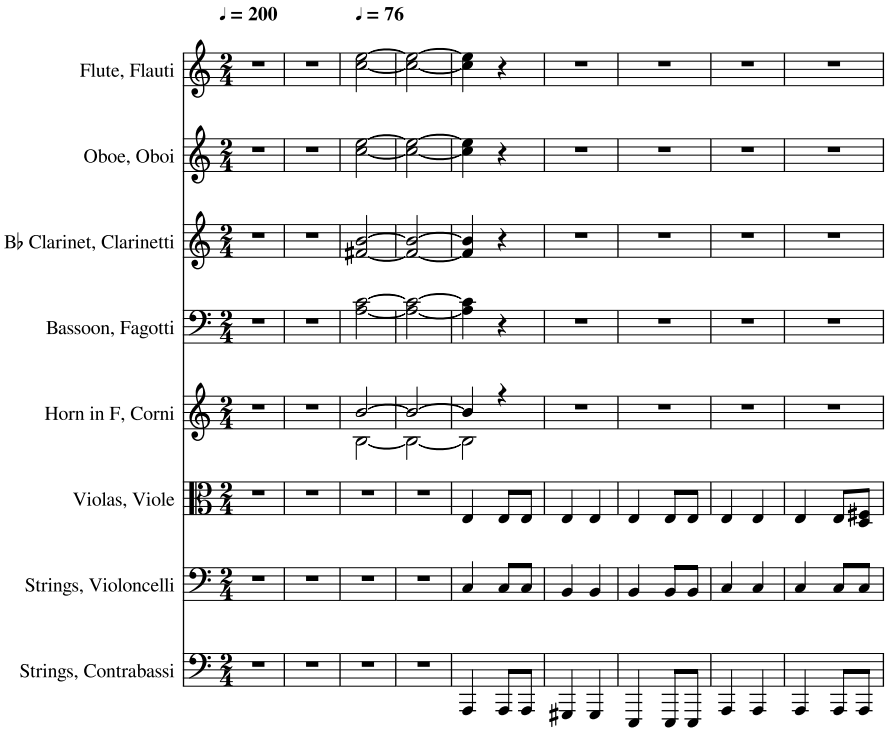

Figure D.1: Beginning of the second movement of the 7th Symphony written by Ludwig van Beethoven

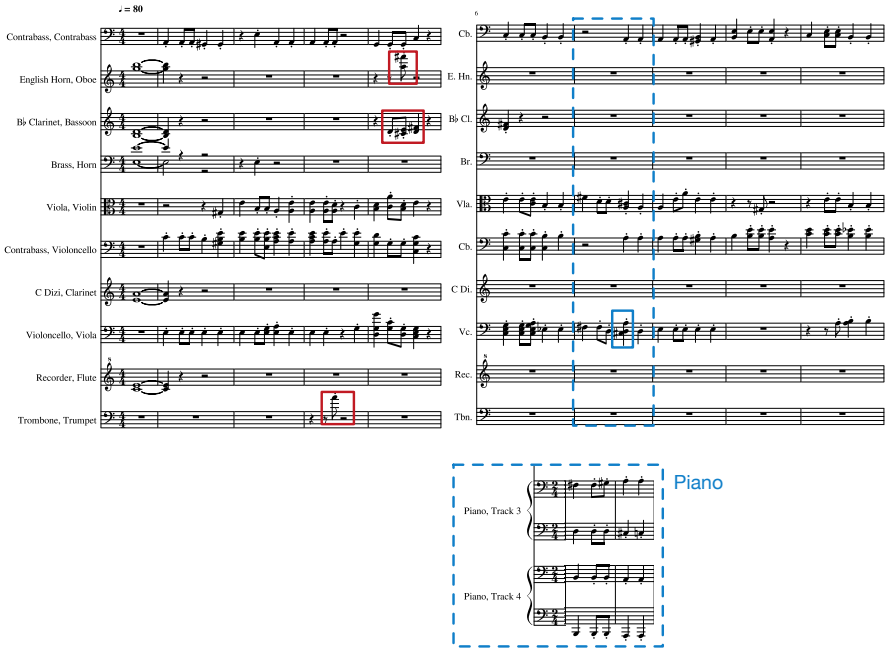

Figure D.2: Orchestration generated by a *RNN* model using the per-note weighted binary cross-entropy. One can notice the crowded orchestration compared with the original. The original piano score was the reduction by Franz Liszt of the second movement of the 7^{th} symphony of Ludwig von Beethoven. The melodic lines of the Oboe, Bassoon and Trombone are particularly fragmented (red frame). The system also makes obvious mistakes, such as the presence of a D in the cello section played over an A major chord (blue frame). The presence of the fourth (D) is problematic here, first because the piano score (blue dotted frame) contains only the root and fifth, and second because the fourth is extremely rarely played over a major chord. Besides, the presence of a semi-tone in the same instrumental section (the cello) in such consonant harmony is out of context.

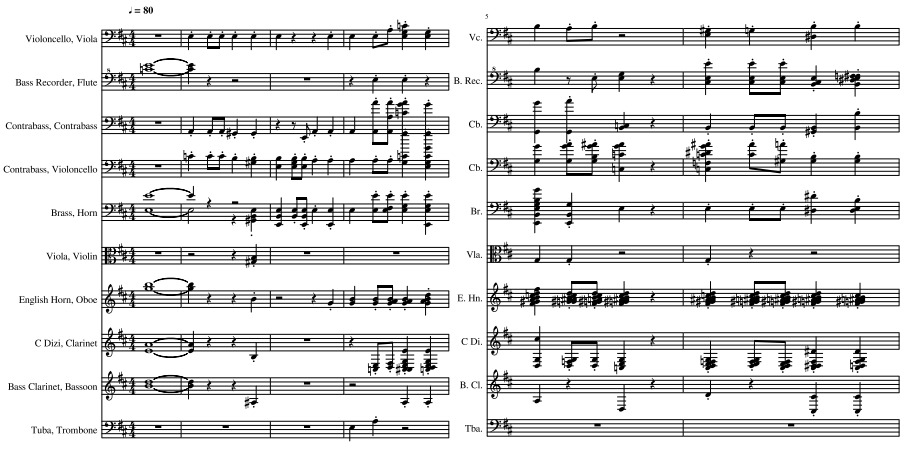

Figure D.3: Orchestration generated by a *RNN* model using the weighted binary cross-entropy with a negative weight value of $\lambda = 0.1$. The reduce impact of the false positive is extremely tangible here, as the orchestration becomes rapidly extremely crowded. The original piano score was the reduction by Franz Liszt of the second movement of the 7^{th} symphony of Ludwig von Beethoven.

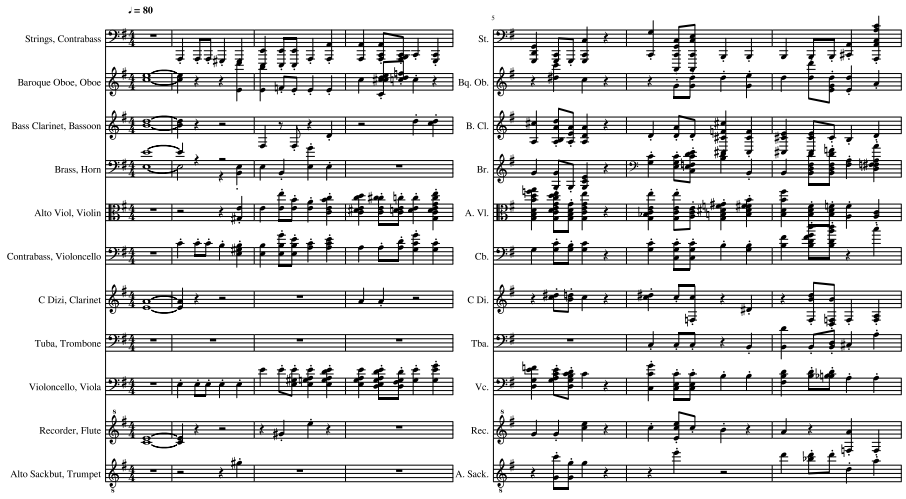

Figure D.4: Orchestration generated by a *RNN* model using the per-note weighted binary cross-entropy as a training and testing criterion. A lot of notes are out of harmony. The original piano score was the reduction by Franz Liszt of the second movement of the 7^{th} symphony of Ludwig von Beethoven.

BIBLIOGRAPHY

[1] Manuel Alfonseca, Manuel Cebrian, and Alfonso De la Puente. *Evolving computer-generated music by means of the normalized compression distance*. Aug. 2005.

[2] Moray Allan and Christopher K. I. Williams. *Harmonising Chorales by Probabilistic Inference*. Jan. 2004.

[3] Torsten Anders. *Compositions created with constraint programming*. This chapter surveys music constraint programming systems, and how composers have used them. The chapter motivates and explains how users of such systems describe intended musical results with constraints. This approach to algorithmic composition is similar to the way declarative and modular compositional rules have successfully been used in music theory for centuries as a device to describe composition techniques. In a systematic overview, this survey highlights the respective strengths of different approaches and systems from a composer's point of view, complementing other more technical surveys of this field. This text describes the music constraint systems PMC, Score-PMC, PWMC (and its successor Cluster Engine), Strasheela and Orchidée – most are libraries of the composition systems PWGL or OpenMusic. These systems are shown in action by discussing the composition process of specific works by Jacopo Baboni-Schilingi, Magnus Lindberg, Örjan Sandred, Torsten Anders, Johannes Kretz and Jonathan Harvey. Oxford University Press. ISBN: 9780190226992. URL: http://hdl.handle.net/10547/622708.

[4] Christof Angermueller, Tanel Pärnamaa, Leopold Parts, and Oliver Stegle. "Deep learning for computational biology." In: *Molecular Systems Biology* 12.7 (2016). DOI: 10.15252/msb.20156651. eprint: http://msb.embopress.org/content/12/7/878.full.pdf. URL: http://msb.embopress.org/content/12/7/878.

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate." In: *CoRR* abs/1409.0473 (2014). arXiv: 1409.0473. URL: http://arxiv.org/abs/1409.0473.

[6] C. Baily and C. Bogey. "An overview of numerical methods for acoustic wave propagation." In: *ECCOMAS CFD 2006: Proceedings of the European Conference on Computational Fluid Dynamics*. 2006.

[7] Mattia Giuseppe Bergomi. "Dynamical and topological tools for (modern) music analysis." Theses. Université Pierre et Marie Curie - Paris VI, Dec. 2015. URL: https://tel.archives-ouvertes.fr/tel-01293602.

[8] J. Bergstra, D. Yamins, and D. D. Cox. "Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures." In: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. ICML'13.

Atlanta, GA, USA: JMLR.org, 2013, pp. I–115–I–123. URL: http://dl.acm.org/citation.cfm?id=3042817.3042832.

[9] H. Berlioz. *Grand traité d'instrumentation et d'orchestration modernes, Op.10*. Paris: Schonenberger, n.d.(1855). Plate S. 996., 1855.

[10] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. "The Million Song Dataset." In: *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*. 2011.

[11] Greg Bickerman, Sam Bosley, Peter Swire, and Robert Keller. "Learning to Create Jazz Melodies Using Deep Belief Nets." In: (Jan. 2010).

[12] Louis Bigo, Antoine Spicher, Daniele Ghisi, and Moreno Andreatta. "Spatial Transformations in Simplicial Chord Spaces." In: *Proceedings ICMC | SMC | 2014*. cote interne IRCAM: Bigo14b. Athens, Greece, Sept. 2014, pp. 1112–1119. URL: https://hal.archives-ouvertes.fr/hal-01161081.

[13] John Biles. "GenJam: A genetic algorithm for generating jazz solos. In: Proceedings of the 19th international computer music conference (ICMC)." In: (Sept. 1994), pp. 131–137.

[14] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[15] Kaan M Biyikoglu. "A Markov model for chorale harmonization." In: *Preceedings of the 5 th Triennial ESCOM Conference*. 2003, pp. 81–84.

[16] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. "Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription." In: *arXiv preprint arXiv:1206.6392* (2012).

[17] Gregoire Carpentier. "Approche computationnelle de l'orchestration musicale : optimisation multi-critères sous contraintes de combinaisons instrumentales dans de grandes banques de sons." PhD thesis. IRCAM, 2009.

[18] Travers Ching et al. "Opportunities And Obstacles For Deep Learning In Biology And Medicine." In: *bioRxiv* (2017). DOI: 10.1101/142760. eprint: https://www.biorxiv.org/content/early/2017/05/28/142760.full.pdf. URL: https://www.biorxiv.org/content/early/2017/05/28/142760.

[19] Sumit Chopra, Raia Hadsell, and Yann LeCun. "Learning a similarity metric discriminatively, with application to face verification." In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Vol. 1. IEEE. 2005, pp. 539–546.

[20] John M. Chowning. "The Synthesis of Complex Audio Spectra by Means of Frequency Modulation." In: *J. Audio Eng. Soc* 21.7 (1973), pp. 526–534. URL: http://www.aes.org/e-lib/browse.cfm?elib=1954.

[21] Hang Chu, Raquel Urtasun, and Sanja Fidler. "Song From PI: A Musically Plausible Network for Pop Music Generation." In: *CoRR* abs/1611.03477 (2016). arXiv: 1611.03477. URL: http://arxiv.org/abs/1611.03477.

[22] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling." In: *CoRR* abs/1412.3555 (2014). arXiv: 1412.3555. URL: http://arxiv.org/abs/1412.3555.

[23] Liam Connor and Joeri van Leeuwen. "Applying Deep Learning to Fast Radio Burst Classification." In: *arXiv preprint arXiv:1803.03084* (2018).

[24] J. Cookerly. *Complete orchestration system*. US Patent 7,718,883. May 2010. URL: http://www.google.ch/patents/US7718883.

[25] Pedro P. Cruz-Alcázar and Enrique Vidal-Ruiz. "Learning regular grammars to model musical style: Comparing different coding schemes." In: *Grammatical Inference*. Ed. by Vasant Honavar and Giora Slutzki. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 211–222. ISBN: 978-3-540-68707-8.

[26] G. Cybenko. "Approximation by superpositions of a sigmoidal function." In: *Mathematics of Control, Signals and Systems* 2.4 (Dec. 1989), pp. 303–314. ISSN: 1435-568X. DOI: 10.1007/BF02551274. URL: https://doi.org/10.1007/BF02551274.

[27] Roberto De Prisco, Gianluca Zaccagnino, and Rocco Zaccagnino. *EvoBassComposer: a Multi-objective Genetic Algorithm for 4-voice Compositions*. Jan. 2010.

[28] Hao-Wen Dong and Yi-Hsuan Yang. "Convolutional Generative Adversarial Networks with Binary Neurons for Polyphonic Music Generation." In: *CoRR* abs/1804.09399 (2018). arXiv: 1804.09399. URL: http://arxiv.org/abs/1804.09399.

[29] Vincent Dumoulin, Ethan Perez, Nathan Schucher, Florian Strub, Harm de Vries, Aaron Courville, and Yoshua Bengio. "Feature-wise transformations." In: *Distill* (2018). https://distill.pub/2018/feature-wise-transformations. DOI: 10.23915/distill.00011.

[30] Kemal Ebcioğlu. "An Expert System for Harmonizing Four-Part Chorales." In: *Computer Music Journal* 12.3 (1988), pp. 43–51. ISSN: 01489267, 15315169. URL: http://www.jstor.org/stable/3680335.

[31] Douglas Eck and Jasmin Lapamle. "Learning Musical Structure Directly from Sequences of Music." In: 2006.

[32] Douglas Eck and Juergen Schmidhuber. *A First Look at Music Composition Using LSTM Recurrent Neural Networks*. Tech. rep. 2002.

[33] D. Effron, D. Hunsberger, S. Adler, P. Hesterman, and S. Adler. *The study of orchestration*. Editions Henry Lemoine, 2002.

[34] Nakamura Eita and Yoshii Kazuyoshii. "Statistical piano reduction controlling performance difficulty." In: *APSIPA Transactions on Signal and Information Processing*. 2018.

[35] Philippe Esling. "Multiobjective time series matching and classification." PhD thesis. IRCAM, 2012.

[36] Philippe Esling, Grégoire Carpentier, and Carlos Agon. "Dynamic Musical Orchestration Using Genetic Algorithms and a Spectro-Temporal Description of Musical Instruments." In: *Applications of Evolutionary Computation* (2010), pp. 371–380.

[37] Benoit Fabre, Joël Gilbert, Avraham Hirschberg, and Xavier Pelorson. "Aeroacoustics of Musical Instruments." In: *Annual Review of Fluid Mechanics* 44.1 (2012), pp. 1–25. DOI: 10.1146/annurev-fluid-120710-101031. eprint: https://doi.org/10.1146/annurev-fluid-120710-101031. URL: https://doi.org/10.1146/annurev-fluid-120710-101031.

[38] Mary Farbood and Bernd Schöner. "Analysis and Synthesis of Palestrina-Style Counterpoint Using Markov Chains." In: *ICMC*. 2001.

[39] Asja Fischer and Christian Igel. "An Introduction to Restricted Boltzmann Machines." In: *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*. Ed. by Luis Alvarez, Marta Mejail, Luis Gomez, and Julio Jacobo. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 14–36. ISBN: 978-3-642-33275-3.

[40] A Freitas and Frederico Guimarães. *Melody Harmonization in Evolutionary Music Using Multiobjective Genetic Algorithms*. Jan. 2011.

[41] Thomas Funkhouser, Nicolas Tsingos, and Jean-Marc Jot. "Survey of Methods for Modeling Sound Propagation in Interactive Virtual Environment Systems." In: (Jan. 2003).

[42] Andrew Gartland-Jones. "Can a Genetic Algorithm Think Like a Composer?" In: (Sept. 2018).

[43] Jon Gillick, Kevin Tang, and Robert Keller. "Learning Jazz Grammars." In: 34 (Jan. 2010), pp. 56–66.

[44] Meghan Goodchild and Stephen McAdams. "Perceptual Processes in Orchestration." In: ed. by Emily Dolan and Alexander Rehding. May 2018.

[45] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[46] Maarten Grachten, Martin Gasser, Andreas Arzt, and Gerhard Widmer. "Automatic alignment of music performances with structural differences." In: (Jan. 2013), pp. 607–612.

[47] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. "LSTM: A Search Space Odyssey." In: *CoRR* abs/1503.04069 (2015). arXiv: 1503.04069. URL: http://arxiv.org/abs/1503.04069.

[48] Gaëtan Hadjeres and François Pachet. "DeepBach: a Steerable Model for Bach chorales generation." In: *CoRR* abs/1612.01010 (2016). arXiv: 1612.01010. URL: http://arxiv.org/abs/1612.01010.

[49] Gaëtan Hadjeres, Jason Sakellariou, and François Pachet. "Style Imitation and Chord Invention in Polyphonic Music with Exponential Families." In: *CoRR* abs/1609.05152 (2016). arXiv: 1609.05152. URL: http://arxiv.org/abs/1609.05152.

[50] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. "A Fast Learning Algorithm for Deep Belief Nets." In: *Neural Computation* 18.7 (2006). PMID: 16764513, pp. 1527–1554. DOI: 10.1162/neco.2006.18.7.1527. eprint: https://doi.org/10.1162/neco.2006.18.7.1527. URL: https://doi.org/10.1162/neco.2006.18.7.1527.

[51] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory." In: *Neural Computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735. eprint: https://doi.org/10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[52] S. R. Holtzman. "Using Generative Grammars for Music Composition." In: *Computer Music Journal* 5.1 (1981), pp. 51–64. ISSN: 01489267, 15315169. URL: http://www.jstor.org/stable/3679694.

[53] Kurt Hornik. "Approximation capabilities of multilayer feedforward networks." In: *Neural Networks* 4.2 (1991), pp. 251 –257. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(91)90009-T. URL: http://www.sciencedirect.com/science/article/pii/089360809190009T.

[54] Jiun-Long Huang, Shih-Chuan Chiu, and Man-Kwan Shan. "Towards an Automatic Music Arrangement Framework Using Score Reduction." In: 8 (Feb. 2012), p. 8.

[55] Ronald J. Williams and David Zipser. "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks." In: 1 (Sept. 1998).

[56] Bruce Jacob. "Composing with genetic algorithms." In: (1995).

[57] Natasha Jaques, Shixiang Gu, Dzmitry Bahdanau, Jose Miguel Hernandez Lobato, Richard E. Turner, and Doug Eck. "Tuning Recurrent Neural Networks With Reinforcement Learning." In: 2017. URL: https://openreview.net/pdf?id=Syyv2e-Kx.

[58] Nicholas Jillings, David Moffat, Brecht De Man, and Joshua D. Reiss. "Web Audio Evaluation Tool: A browser-based listening test environment." In: *12th Sound and Music Computing Conference*. July 2015.

[59] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization." In: *CoRR* abs/1412.6980 (2014). arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980.

[60] Diederik P. Kingma and Max Welling. "Auto-Encoding Variational Bayes." In: *CoRR* abs/1312.6114 (2013). arXiv: 1312.6114. URL: http://arxiv.org/abs/1312.6114.

[61] Charles Koechlin. *Traité de l'orchestration*. Vol. 1. Éditions Max Eschig, 1941.

[62] Hugo Larochelle and Iain Murray. "The Neural Autoregressive Distribution Estimator." In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, Apr. 2011, pp. 29–37. URL: http://proceedings.mlr.press/v15/larochelle11a.html.

[63] Victor Lavrenko and Jeremy Pickens. "Polyphonic Music Modeling with Random Fields." In: *Proceedings of the Eleventh ACM International Conference on Multimedia*. MULTIMEDIA '03. New York, NY, USA: ACM, 2003, pp. 120–129. ISBN: 1-58113-722-2. DOI: 10.1145/957013.957041. URL: http://doi.acm.org/10.1145/957013.957041.

[64] Yann LeCun, Y Bengio, and Geoffrey Hinton. "Deep Learning." In: 521 (May 2015), pp. 436–44.

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

[66] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." In: *Proceedings of the IEEE* 86.11 (Nov. 1998), pp. 2278–2324. ISSN: 0018-9219. DOI: 10.1109/5.726791.

[67] Hao-Min Liu and Yi-Hsuan Yang. "Lead Sheet Generation and Arrangement by Conditional Generative Adversarial Network." In: *arXiv preprint arXiv:1807.11161* (2018).

[68] *Logic Pro 9 User Manual*. http://www.music.mcgill.ca/dcs/Manuals/Software/Logic_Pro_9_User_Manual.pdf.

[69] Huanru Henry Mao, Taylor Shin, and Garrison W. Cottrell. "DeepJ: Style-Specific Music Generation." In: *CoRR* abs/1801.00887 (2018). arXiv: 1801.00887. URL: http://arxiv.org/abs/1801.00887.

[70] Stephen McAdams. "Timbre as a structuring force in music." In: *Proceedings of Meetings on Acoustics*. Vol. 19. 1. Acoustical Society of America. 2013, p. 035050.

[71] Stephen McAdams and Albert Bregman. "Hearing Musical Streams." In: *Computer Music Journal* 3.4 (1979), pp. 26–60. ISSN: 01489267, 15315169. URL: http://www.jstor.org/stable/4617866.

[72] Jon McCormack. "Grammar Based Music Composition." In: 3 (Apr. 1996).

[73] Stephen Mcadams and Bruno Giordano. *The perception of musical timbre*. Jan. 2008.

[74] Warren Mcculloch and Walter Pitts. "A Logical Calculus of Ideas Immanent in Nervous Activity." In: *Bulletin of Mathematical Biophysics* 5 (1943), pp. 127–147.

[75] Soroush Mehri, Kundan Kumar, Ishaan Gulrajani, Rithesh Kumar, Shubham Jain, Jose Sotelo, Aaron C. Courville, and Yoshua Bengio. "SampleRNN: An Unconditional End-to-End Neural Audio Generation Model." In: *CoRR* abs/1612.07837 (2016). arXiv: 1612.07837. URL: http://arxiv.org/abs/1612.07837.

[76] A. Van Der Merwe and W. Schulze. "Music Generation with Markov Models." In: *IEEE MultiMedia* 18.3 (Mar. 2011), pp. 78–85. ISSN: 1070-986X. DOI: 10.1109/MMUL.2010.44.

[77] Gabriel Meseguer-Brocal, Alice Cohen-Hadria, Gomez, and Peeters Geoffroy. "DALI: a large Dataset of synchronized Audio, LyrIcs and notes, automatically created using teacher-student machine learning paradigm." In: *19th International Society for Music Information Retrieval Conference*. Ed. by ISMIR. Sept. 2018.

[78] Pantelis N. Vassilakis. "Auditory Roughness as a Means of Musical Expression." In: 7 (Perspectives in Musicology) (Jan. 2005), pp. 119–.

[79] Saul B. Needleman and Christian D. Wunsch. "A general method applicable to the search for similarities in the amino acid sequence of two proteins." In: *Journal of Molecular Biology* 48.3 (1970), pp. 443 –453. ISSN: 0022-2836. DOI: http://dx.doi.org/10.1016/0022-2836(70)90057-4. URL: http://www.sciencedirect.com/science/article/pii/0022283670900574.

[80] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. "WaveNet: A Generative Model for Raw Audio." In: *CoRR* abs/1609.03499 (2016). arXiv: 1609.03499. URL: http://arxiv.org/abs/1609.03499.

[81] Ender Özcan and Türker Erçal. "A Genetic Algorithm for Generating Improvised Music." In: *Artificial Evolution*. Ed. by Nicolas Monmarché, El-Ghazali Talbi, Pierre Collet, Marc Schoenauer, and Evelyne Lutton. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 266–277. ISBN: 978-3-540-79305-2.

[82] Jean-François Paiement, Douglas Eck, and Samy Bengio. "A Probabilistic Model for Chord Progressions." In: *ISMIR*. 2005.

[83] T. Painter and A. Spanias. "A review of algorithms for perceptual coding of digital audio signals." In: *Proceedings of 13th International Conference on Digital Signal Processing*. Vol. 1. July 1997, 179–208 vol.1. DOI: 10.1109/ICDSP.1997.628010.

[84] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "Understanding the exploding gradient problem." In: *CoRR* abs/1211.5063 (2012). arXiv: 1211.5063. URL: http://arxiv.org/abs/1211.5063.

[85] Geoffroy Peeters. "A generic system for audio indexing: application to speech/music segmentation and music genre recognition." In: 2007.

[86] Geoffroy Peeters, Bruno L Giordano, Patrick Susini, Nicolas Misdariis, and Stephen McAdams. "The timbre toolbox: Extracting audio descriptors from musical signals." In: *The Journal of the Acoustical Society of America* 130.5 (2011), pp. 2902–2916.

[87] Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron C. Courville. "FiLM: Visual Reasoning with a General Conditioning Layer." In: *CoRR* abs/1709.07871 (2017). arXiv: 1709.07871. URL: http://arxiv.org/abs/1709.07871.

[88] Walter Piston. *Orchestration*. New York: Norton, 1955.

[89] Milad Pourrahmani, Hooshang Nayyeri, and Asantha Cooray. "Lens-Flow: A Convolutional Neural Network in Search of Strong Gravitational Lenses." In: *The Astrophysical Journal* 856.1 (2018), p. 68. URL: http://stacks.iop.org/0004-637X/856/i=1/a=68.

[90] David Powers. "Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness and Correlation." In: 2 (Jan. 2008).

[91] Lutz Prechelt. "Early Stopping - But When?" In: (Mar. 2000).

[92] David Psenicka. "SPORCH: An Algorithm for Orchestration Based on Spectral Analyses of Recorded Sounds." In: (Jan. 2003).

[93] Colin Raffel. "Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching." PhD thesis. Columbia University, 2016.

[94] Jean-Philippe Rameau. *Traité de l'harmonie réduite à ses principes naturels*. Jean-Baptiste-Christophe Ballard, 1722.

[95] Daniele Ravì, Charence Wong, Fani Deligianni, Melissa Berthelot, Javier Andreu, Benny Lo, and Guang-Zhong Yang. "Deep Learning for Health Informatics." In: PP (Dec. 2016).

[96] Ana Rebelo, Ichiro Fujinaga, Filipe Paszkiewicz, Andre R. S. Marcal, Carlos Guedes, and Jaime S. Cardoso. "Optical music recognition: state-of-the-art and open issues." In: *International Journal of Multimedia Information Retrieval* 1.3 (Oct. 2012), pp. 173–190. ISSN: 2192-662X. DOI: 10.1007/s13735-012-0004-6. URL: https://doi.org/10.1007/s13735-012-0004-6.

[97] Nikolay Rimsky-Korsakov. *Principles of Orchestration*. Russischer Musikverlag, 1873.

[98] C. Roads and Paul Wieneke. "Grammars as Representations for Music." In: *Computer Music Journal* 3.1 (1979), pp. 48–55. ISSN: 01489267, 15315169. URL: http://www.jstor.org/stable/3679756.

[99] Curtis Roads. *The Computer Music Tutorial*. Cambridge, MA, USA: MIT Press, 1996. ISBN: 0262680823.

[100] Andres C Rodriguez, Tomasz Kacprzak, Aurelien Lucchi, Adam Amara, Raphael Sgier, Janis Fluri, Thomas Hofmann, and Alexandre Réfrégier. "Fast Cosmic Web Simulations with Generative Adversarial Networks." In: *arXiv preprint arXiv:1801.09070* (2018).

[101] Jose David Fernández Rodriguez and Francisco J. Vico. "AI Methods in Algorithmic Composition: A Comprehensive Survey." In: *CoRR* abs/1402.0585 (2014). arXiv: 1402.0585. URL: http://arxiv.org/abs/1402.0585.

[102] François Rose and James E. Hetrick. "Enhancing Orchestration Technique via Spectrally Based Linear Algebra Methods." In: *Computer Music Journal* 33.1 (2009), pp. 32–41. ISSN: 01489267, 15315169. URL: http://www.jstor.org/stable/40301010.

[103] Matti Ryynänen. "Automatic Transcription of Pitch Content in Music and Selected Applications." PhD thesis. Tampere University of Technology, 2008.

[104] Örjan Sandred, Mikael Laurson, and Mika Kuuskankare. "Revisiting the Illiac Suite - A rule-based approach to stochastic processes." In: 2 (Jan. 2009), pp. 42–46.

[105] Florian Schroff, Dmitry Kalenichenko, and James Philbin. "FaceNet: A Unified Embedding for Face Recognition and Clustering." In: *CoRR* abs/1503.03832 (2015). arXiv: 1503.03832. URL: http://arxiv.org/abs/1503.03832.

[106] Xavier Serra and Julius Smith. "Spectral Modeling Synthesis: A Sound Analysis/Synthesis System Based on a Deterministic Plus Stochastic Decomposition." In: *Computer Music Journal* 14.4 (1990), pp. 12–24. ISSN: 01489267, 15315169. URL: http://www.jstor.org/stable/3680788.

[107] Felix Streichert. "Introduction to evolutionary algorithms." In: *paper to be presented Apr* 4 (2002).

[108] Bob Sturm, João Felipe Santos, and Iryna Korshunova. "Folk music style modelling by recurrent neural networks with long short term memory units." eng. In: *16th International Society for Music Information Retrieval Conference, late-breaking demo session*. Malaga, Spain, 2015, p. 2.

[109] Hirofumi Takamori, Haruki Sato, Takayuki Nakatsuka, and Shigeo Morishima. "Automatic arranging musical score for piano using important musical elements." In: *Proceedings of the 14th Sound and Music Computing Conference*. Aalto, Finland, July 2017. URL: http://smc2017.aalto.fi/media/materials/proceedings/SMC17_p35.pdf.

[110] Pierre Talbot, Carlos Agon, and Philippe Esling. "Interactive computer-aided composition with constraints." In: *43rd International Computer Music Conference (ICMC 2017)*. Shanghai, China, Oct. 2017. URL: https://hal.archives-ouvertes.fr/hal-01577898.

[111] Damien Tardieu and Stephen McAdams. "Perception of dyads of impulsive and sustained instrument sounds." In: *Music Perception* 30.2 (2012), pp. 117–128.

[112] Graham W Taylor and Geoffrey E Hinton. "Factored conditional restricted Boltzmann machines for modeling motion style." In: *Proceedings of the 26th annual international conference on machine learning*. ACM. 2009, pp. 1025–1032.

[113] Charlotte Truchet and Gerard Assayag. *Constraint Programming in Music*. Wiley, 2011.

[114] *Automatic melodic grammar generation for polyphonic music using a classifier system*. Sound and Music Computing Conference. 2018.

[115] Benigno Uria, Marc-Alexandre Côté, Karol Gregor, Iain Murray, and Hugo Larochelle. "Neural Autoregressive Distribution Estimation." In: *CoRR* abs/1605.02226 (2016). arXiv: 1605.02226. URL: http://arxiv.org/abs/1605.02226.

[116] Arthur Jannery Walter Piston Mark DeVoto. *Harmony*. Ed. by Mark DeVoto. 5, illustrated, revised. Norton, 1941.

[117] Eelco van der Wel and Karen Ullrich. "Optical Music Recognition with Convolutional Sequence-to-Sequence Models." In: *CoRR* abs/1707.04877 (2017). arXiv: 1707.04877. URL: http://arxiv.org/abs/1707.04877.

[118] David L. Wessel. "Timbre space as a musical control structure." In: *Computer Music Journal* 3.2 (June 1979).

[119] Geraint Wiggins, George Papadopoulos, Somnuk Phon-Amnuaisuk, and Andrew Tuson. "Evolutionary Methods for Musical Composition." In: (June 2000).

[120] Jacek Wolkowicz, Malcolm Heywood, and Vlado Keselj. "Evolving Indirectly Represented Melodies with Corpus-Based Fitness Evaluation." In: *Proceedings of the EvoWorkshops 2009 on Applications of Evolutionary Computing: EvoCOMNET, EvoENVIRONMENT, EvoFIN, EvoGAMES, EvoHOT, EvoIASP, EvoINTERACTION, EvoMUSART, EvoNUM, EvoSTOC, EvoTRANSLOG*. EvoWorkshops '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 603–608. ISBN: 978-3-642-01128-3. DOI: 10.1007/978-3-642-01129-0_69. URL: http://dx.doi.org/10.1007/978-3-642-01129-0_69.

[121] Peter Worth and Susan Stepney. "Growing Music: Musical Interpretations of L-Systems." In: *Workshops on Applications of Evolutionary Computation*. EvoWorkshops. 2005.

[122] Li-Chia Yang, Szu-Yu Chou, and Yi-Hsuan Yang. "MidiNet: A Convolutional Generative Adversarial Network for Symbolic-domain Music Generation using 1D and 2D Conditions." In: *CoRR* abs/1703.10847 (2017). arXiv: 1703.10847. URL: http://arxiv.org/abs/1703.10847.

[123] Kaisheng Yao, Trevor Cohn, Katerina Vylomova, Kevin Duh, and Chris Dyer. "Depth-Gated LSTM." In: *CoRR* abs/1508.03790 (2015). arXiv: 1508.03790. URL: http://arxiv.org/abs/1508.03790.

[124] Liangrong Yi and Judy Goldsmith. *Automatic Generation of Four-part Harmony*. Jan. 2007.

[125] *https://www.midi.org/*. URL: https://www.midi.org/.

[126] *https://www.musicxml.com/*. URL: https://www.musicxml.com/.

DECLARATION

Put your declaration here.

Paris, October 2018

Léopold Crestel

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede and Ivo Pletikosić. The style was inspired by Robert Bringhurst's seminal book on typography *"The Elements of Typographic Style"*. `classicthesis` is available for both LaTeX and LyX:

https://bitbucket.org/amiede/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

Thank you very much for your feedback and contribution.