Introduction of musical knowledge and qualitative analysis in chord extraction and prediction tasks with machine learning

Tristan Carsault

INTRODUCTION OF MUSICAL KNOWLEDGE AND QUALITATIVE ANALYSIS IN CHORD EXTRACTION AND PREDICTION TASKS WITH MACHINE LEARNING

APPLICATION TO HUMAN-MACHINE CO-IMPROVISATION

by

Tristan Carsault

Ph.D. Thesis in Computer Science
École Doctorale Informatique, Télécommunications et Électronique (EDITE)
Sorbonne University/Université Pierre et Marie Curie/Paris 6, FRANCE
Sciences et Technologie de la Musique et du Son (UMR 9912)
Institut de Recherche et Coordination Acoustique/Musique (IRCAM)
Defended on December 17th, 2020

Supervised by Philippe Esling and Jérôme Nika
Directed by Gérard Assayag

Tristan Carsault: *Introduction of musical knowledge and qualitative analysis in chord extraction and prediction tasks with machine learning*: Application to human-machine co-improvisation, Ph.D. Thesis © December 2020 – v1.1

ABSTRACT

This thesis investigates the impact of introducing musical properties in machine learning models for the extraction and inference of musical features. Furthermore, it discusses the use of musical knowledge to perform qualitative evaluations of the results. In this work, we focus on *musical chords* since these mid-level features are frequently used to describe harmonic progressions in Western music. Hence, amongst the variety of tasks encountered in the field of Music Information Retrieval (MIR), the two main tasks that we address are the Automatic Chord Extraction (ACE) and the inference of symbolic chord sequences.

In the case of musical chords, there exists inherent strong hierarchical and functional relationships. Indeed, even if two chords do not belong to the same class, they can share the same harmonic function within a chord progression. Hence, we developed a specifically-tailored analyzer that focuses on the functional relations between chords to distinguish *strong* and *weak* errors. We define *weak* errors as a misclassification that still preserves the relevance in terms of harmonic function. This reflects the fact that, in contrast to *strict* transcription tasks, the extraction of high-level musical features is a rather subjective task. Moreover, many creative applications would benefit from a higher level of harmonic understanding rather than an increased accuracy of label classification. For instance, one of our application case is the development of a software that interacts with a musician in real-time by inferring expected chord progressions. In order to achieve this goal, we divided the project into two main tasks: a *listening module* and a *symbolic generation module*. The listening module extracts the musical structure played by the musician, whereas the generative module predicts musical sequences based on the extracted features.

In the first part of this thesis, we target the development of an ACE system that could emulate the process of musical structure discovery, as performed by musicians in improvisation contexts. Most ACE systems are built on the idea of extracting features from raw audio signals and, then, using these features to construct a chord classifier. This entail two major families of approaches, as either *rule-based* or *statistical* models. In this work, we identify drawbacks in the use of statistical models for ACE tasks. Then, we propose to introduce prior musical knowledge in order to account for the inherent relationships between chords directly inside the loss function of learning methods.

In the second part of this thesis, we focus on learning higher-level relationships inside sequences of extracted chords in order to develop models with the ability to generate potential continuations of chord sequences. In order to introduce musical knowledge in these models, we propose both new architectures, multi-label training methods and novel data representations.

Overall, we show that the introduction of musical knowledge allows to improve our results, in both a quantitative and qualitative manner, for the tasks of chord extraction and constrained prediction. In order to furnish consistent qualitative analysis, we rely on custom libraries adapted to the evaluation of chord-based models. Along with other musical applications, co-improvisation systems have been developed and are presented along this thesis. These applications allow to test our models in a creative context and reinforce our vision of developing MIR systems for real musical use cases. Forthwith, this study strengthens our belief that going toward a refinement of established MIR tasks could allow to produce better quantitative and qualitative outputs.

RÉSUMÉ

Cette thèse étudie l'impact de l'introduction de propriétés musicales dans les modèles d'apprentissage machine pour l'extraction et l'inférence de structures musicales. De plus, elle traite de l'utilisation des connaissances musicales pour effectuer des évaluations qualitatives des résultats. Dans ce travail, nous nous concentrons sur les *accords musicaux* puisque ce sont des structures musicales fréquemment utilisées pour décrire les progressions harmoniques dans la musique occidentale. Ainsi, parmi la variété des tâches rencontrées dans le domaine de la recherche d'informations musicales (MIR), les deux principales tâches que nous abordons sont l'extraction automatique d'accords (ACE) et l'inférence de séquences de label d'accords.

Dans le cas des accords musicaux, il existe de fortes relations inhérentes d'un point de vue hiérarchiques et fonctionnelles. En effet, même si deux accords n'appartiennent pas à la même classe, ils peuvent partager la même fonction harmonique au sein d'une progression d'accords. En outre, de nombreuses applications créatives bénéficieraient d'un niveau plus élevé de compréhension harmonique plutôt que d'une précision accrue dans la tâche de classification. Nous avons donc développé un analyseur spécifiquement adapté qui se concentre sur les relations fonctionnelles entre les accords pour distinguer les erreurs *fortes* et *faibles*. Nous définissons les erreurs *faibles* comme une mauvaise classification qui conserve la pertinence en termes de fonction harmonique. Cela reflète le fait que, contrairement aux tâches de transcription *strict*, l'extraction de caractéristiques musicales de haut niveau est une tâche plutôt subjective.

Un de nos cas d'application est le développement d'un logiciel qui interagit avec un musicien en temps réel en déduisant les progressions d'accords attendues. Pour atteindre cet objectif, nous avons divisé le projet en deux tâches principales : un *module d'écoute* et un *module de génération symbolique*. Le module d'écoute extrait la structure musicale jouée par le musicien, tandis que le module de génération prédit les séquences musicales en fonction des accords extraits.

Dans la première partie de cette thèse, nous visons le développement d'un système ACE qui pourrait émuler le processus de découverte de la structure musicale, tel qu'il est exécuté par les musiciens dans des contextes d'improvisation. La plupart des systèmes ACE sont construits sur l'idée d'extraire des caractéristiques des signaux audio bruts et, ensuite, d'utiliser ces caractéristiques pour construire un classificateur d'accords. Nous distinguons deux grandes familles d'approches, les modèles *basés sur les règles musicales* ou les modèles *statistiques*. Dans ce travail, nous identifions les inconvénients de l'utilisation des modèles statistiques pour les tâches ACE. Ensuite, nous proposons d'introduire les connaissances musicales préalables afin de rendre compte des relations inhérentes entre les accords directement à l'intérieur de la fonction de coût des méthodes d'apprentissage machine.

Dans la deuxième partie de cette thèse, nous nous concentrons sur l'apprentissage de relations de plus haut niveau à l'intérieur de séquences d'accords extraites, en vue de développer des modèles capables de générer des suites potentielles de séquences d'accords. Afin d'introduire des connaissances musicales dans ces modèles, nous proposons à la fois de nouvelles architectures, des méthodes d'apprentissage multi-label et de nouvelles représentations de données.

Dans l'ensemble, nous montrons que l'introduction de la connaissance musicale permet d'améliorer nos résultats, de manière quantitative et qualitative, pour les tâches d'extraction d'accords et de prédiction de séquences d'accords. Afin de fournir une analyse qualitative cohérente, nous nous appuyons sur des libraires personnalisées adaptées à l'évaluation des modèles portants sur les accords musicaux. Parmis d'autres applications musicales, des systèmes de co-improvisation ont été développés et sont présentés dans le cadre de cette thèse. Ces applications permettent de tester nos modèles dans un contexte créatif et renforcent notre vision du développement de systèmes MIR pour des cas réels d'utilisation musicale. Cette étude nous amène à penser que le perfectionnement d'une définition des tâches MIR pourrait permettre de produire de meilleurs résultats quantitatifs et qualitatifs.

ACKNOWLEDGMENTS

Ce fut un réel plaisir d'effectuer cette thèse au sein de l'IRCAM, mes deux encadrants Jérôme Nika et Philippe Esling m'ont énormément apporté durant ces précieuses années. Ils ont su faire preuve de patience, de soutien inconditionnel et d'une attention toute particulière. Je tiens à les remercier très chaleureusement pour tous ces moments partagés ensemble, toujours dans le bien-être et la bienveillance. Je souhaite également remercier Gérard Assayag qui a dirigé cette thèse en lui donnant la possibilité de s'inscrire dans un contexte solide parmi les recherches innovantes de l'équipe *représentations musicales*. Au sein de cette équipe, et tout particulièrement avec les collègues d'ACIDS, j'ai pu durant cette thèse m'accomplir en terme de recherche et de création. Je tiens donc à remercier avec beaucoup d'amitiés Axel Chemla-Romeu-Santos, Ninon Devis, Mathieu Prang, Adrien Bitton, Léopold Crestel, Constance Douwes, Théis Bazin, Antoine Caillon, Cyran Aouameur, Clement Tabary ainsi que toutes les personnes avec qui j'ai pu de près ou de loin partager des moments de travail ou de détente au sein de l'IRCAM.

Durant les quelques mois passés à Kyoto Université, j'ai eu l'occasion de partager de riches moments avec des gens brillants et passionnés. Je remercie tout particulièrement Kazuyoshi Yoshii de m'avoir accueilli dans son équipe de recherche, de m'avoir fait confiance et de m'avoir apporté une vision différente de la recherche me permettant de développer de nombreux sujets présents dans cette thèse. Je remercie également tous les chercheurs, doctorants post-doctorants et personnels de son laboratoire, notamment Eita Nakamura, Yiming Wu, Koji Inoue, Adrien Ycart, Andrew McLeod, Florian Thalmann, Shinsuke Sakai et Mayumi Abe.

Je remercie infiniment mes amis, ma famille et Anna de m'avoir toujours soutenu et grandi durant cette riche expérience.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

MIR Music Information Retrieval

ACE Automatic Chord Extraction

ML Machine Learning

NN Neural Network

DNN Deep Neural Network

MLP Multi-Layer Perceptron

CNN Convolutional Neural Network

RNN Recurrent Neural Network

LSTM Long Short Term Memory

ED Encoder-Decoder

HMM Hidden Markov Model

STFT Short Term Fourier Transform

CQT Constant Q Transform

CV Computer Vision

NLP Natural Language Processing

MIREX Music Information Retrieval Evaluation eXchange

1

INTRODUCTION

The concept of *musical structure* can be defined as the arrangement and relations between musical elements across time. Furthermore, a piece of music possesses different levels of structure depending on the analyzed *temporal scale*. Indeed, elements of music such as notes (defined by their pitch, duration and timbre) can be gathered into groups like chords, motifs and phrases. Equivalently, these can be combined into larger structures such as chord progressions or choruses and verses. Thus, there can be complex and multi-scaled hierarchical and temporal relationships between different types of musical elements.

Among these different levels of music description, *chords* are one of the most prominent mid-level features in Western music such as pop or jazz music. The chord structure defines, at a high level of abstraction, the musical intention along the evolution of a song. Indeed, in the case of musical improvisation, it is common for musicians to agree upon a sequence of chords beforehand in order to develop notes along this high-level structure. Thus, a musical piece can be roughly described by its chord sequence that is commonly referred to as its *harmonic structure*.

In this thesis, we focus on the extraction and prediction of musical chords sequences from a given musical signal. Existing models in the field of Music Information Retrieval (MIR) for theses tasks can be broadly divided between two major categories: *rule-based* or *statistical* models. The rule-based models use hand-crafted rules from music theory, which allows a good understanding of the models output and a decent level of performance. However, in the last decade, statistical models such as *Deep Neural Networks* (DNN) strongly outperformed rule-based models in nearly every tasks of MIR. These models are often trained with the help of annotated musical data and are built to infer musical properties within this dataset. Nevertheless, most of these DNNs rely quite straightforwardly on methods that have been successfully developed for other fields such as Computer Vision (CV) or Natural Language Processing (NLP). Hence, musicological knowledge and the intrinsic properties of music are rarely taken into account. The core idea of this thesis is to introduce music theory knowledge into learning models in order to improve their performances and bridge the gap between ML and rule-based methods. Furthermore, due to the hierarchical and intrinsic relationships between musical elements, common evaluation techniques are often not fully adapted to measure the performances of MIR models. Hence, beyond the introduction of novel learning methods, this thesis aims to analyze the results of chord-based models in a more musically sound and qualitative way. This has been performed through the development of analysis methods that directly include musical theory concepts.

Some real-time music improvisation systems, such as (Nika et al., 2017), generate music by combining reactivity to the environment and anticipation with respect to a

fixed chord sequence. However, they could take a step forward by being able to infer this sequence dynamically as a musician plays. Therefore, as detailed in Chapter 2, one of our application motivation is the development of an intelligent listening module (performing real-time extraction of underlying chord labels and prediction of future chord progressions) that could be encapsulated in such co-improvisations systems.

Thus, after exposing some properties of musical data in Chapter 3, we put forward problematic that can encounter machine learning models for the extraction and inference of musical features in Chapter 4. Furthermore, we examine the use of musical knowledge to perform qualitative evaluations on subjective results.

Besides, in order to develop the contributions of the thesis that are summarized in Chapter 5, Chapter 6 introduces backgrounds in music theory and Chapter 7 presents basic concepts of machine learning and neural networks.

Therefore, after having defined chord alphabets and qualitative evaluations in Chapter 8, Chapter 9 presents the development of statistical ACE systems that rely on our proposed introduction of musical knowledge. Hence, we introduce the use of prior musical knowledge underlying the labeling alphabets in order to account for the inherent relationships between chords directly inside the loss function of learning methods. By analyzing our results, we uncover a set of related insights on ACE tasks based on statistical models, and also formalize the musical meaning of some classification errors.

Chapter 10 focuses on the use of musical knowledge in machine learning models, in order to predict the continuation of symbolic harmonic progressions. In order to introduce musical knowledge in the training of such statistical models, we propose new architectures, multi-label training methods and novel data representations. Again, the results of these models are analyzed both quantitatively and qualitatively.

Chapter 11, Chapter 12, Chapter 13 and Chapter 14 introduce prototypes of musical applications, such as the architecture of our intelligent listening module performing real-time extraction of underlying chord labels and prediction of future chord progressions. These applications allow to test our models in a creative context and confront our prototypes to real musicians.

Finally, in Chapter 15, we expose a conclusion on our work on the introduction of musical information within statistical models, the evaluation of these models and the development of creative applications. We conclude that the introduction of musical knowledge has an impact on the training of such models in terms of performances and in terms of quality of the outputs. Hence, for the extraction task, as well as the prediction task, the qualitative analysis allows to uncover insights on the introduction of additional musical information.

2

APPLICATIVE MOTIVATION AND RELATED WORKS

The motivation for this work is twofold. On the one hand, we are aiming to develop a system that use real-time musical feature extraction to predict a possible continuation. On the other hand, we would like to evaluate our results in a qualitative way by relying on musical knowledge. Indeed, in the context of creative application, we could be interested by a prediction that has a high level of harmonic coherence instead of a high level of accuracy in label classification. For instance, if we choose chords as inferred labels, each element of the predicted sequence has a functional behaviour. Thus, even if the predicted label is not exactly what was expected, it can contains a high level of harmonic understanding (e.g. by sharing the same harmonic function).

In this chapter, we present interactive music generation systems named *co-improvisation* systems. For these systems, it exists different levels of interaction between the musician and the computer. Some of them are used only in a constraint creative manner. For instance, the musician will define a musical context beforehand (e.g. chord progression) in order to generate new musical materials. Others are used in a more reactive context where the generative system is guided by a listening of the musician. Finally, some hybridization of the two aforementioned system allow to combine a reactive and constraint behaviors. After having presented these three different systems in the next sections, we introduce the applicative motivation of this thesis: the development of an intelligent listening module (performing real-time extraction of underlying chord labels and prediction of future chord progressions) that will be integrated into existing guided co-improvisation systems.

2.1 CO-IMPROVISATION SYSTEMS

Our applicative motivation is co-improvisation processes that stem from the interplay between human and computer agents. That is, a digital system able to play music coherently with a musician in real time. This produces a form of *feedback loop*, where the human listens to the real-time output of the system, itself conditioned by the musician musical stream (see Figure 2.1).

A prominent example of this idea is *Omax* (Assayag et al., 2006), which learns the specific characteristics of the musician style in real-time, and then plays along with him from this learned model. This technology is based on *Factor Oracle* automatons (Assayag and Bloch, 2007), allowing to generate stylistically coherent music using an online or offline audio database.

In recent years, another family of software that provides musical control, while being style-sensitive has been proposed by (Nika, 2016). In this work, human-machine co-improvisation processes are further improved through the concept of

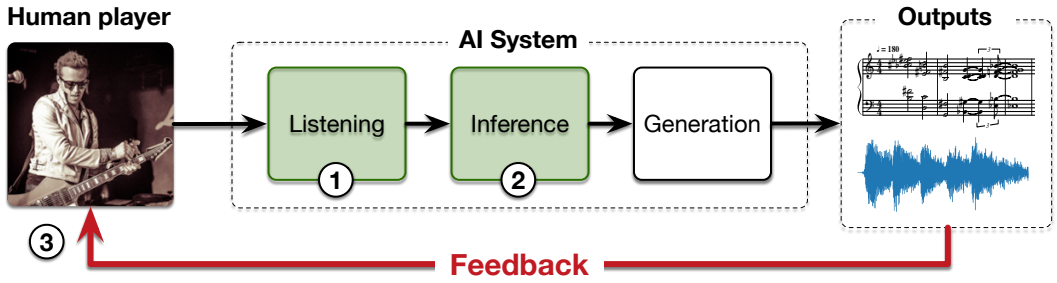

Figure 2.1: General scheme of an interactive improvisation system with a feedback loop.

guidance. In this field of human-machine co-improvisation, the notion of *guided* music generation has two different meanings. On the one hand, *guiding* can be seen as a purely reactive and gradual process. This approach offers rich possibilities for interaction but cannot take advantage of prior knowledge of a temporal structure to introduce anticipatory behaviour. On the other hand, *guiding* can use temporal structures (named *scenario*) to drive the generation process of the entire musical sequence. These *scenario-based* systems are capable of introducing anticipatory behaviors, but require a prior knowledge of the musical context (a predefined *scenario*).

2.1.1 *Guiding step-by-step*

The step-by-step process aims to produce automatic accompaniment by using purely reactive mechanisms without relying on prior knowledge. Hence, the input signal from the musician is analyzed in real-time and the system compares this information to its corpus, allowing to select the most relevant parts to generate accompaniments. For instance, *SoMax* (Bonnasse-Gahot, 2014) uses a pre-annotated corpus and extracts multimodal observations of the input musical stream in real-time. Then, it retrieves the most relevant slices of music from the corpus to generate an accompaniment.

Other software such as *VirtualBand* (Moreira, Roy, and Pachet, 2013) or *Reflexive Looper* (Pachet et al., 2013), also rely on feature extraction (e.g. spectral centroid or chroma) to selects the audio accompaniment from the database. Furthermore, improvisation software such as *MASOM* (Tatar and Pasquier, 2017) use specific listening modules to extract higher-features like eventfulness, pleasantness, and timbre.

2.1.2 *"Guiding" with formal temporal structures*

Another approach to co-improvisation is to introduce different forms of guidance as constraints for the generation of musical sequences. On the one hand, constraints can be used to preserve certain structural patterns already present in the database. In this line of thought, (Herremans et al., 2015) built a system to generate bagana

music, a traditional Ethiopian lyre based on a first-order Markov model. A similar approach using Markov models was proposed by (Eigenfeldt and Pasquier, 2010), producing corpus-based generative electronic dance music.

On the other hand, some research projects have introduced temporal specifications to guide the process of music generation. For example, (Donzé et al., 2014) applied this concept to generate monophonic solo lines similar to a given training melody based a given chord progression. Recently, the notion of *musical scenario* has been proposed by (Nika et al., 2014) as the specification of high-level musical structures that define hard constraints on the generated sequences. This approach allows to define a global orientation of the music generation process. For instance, if it is applied to tonal music, constraining the model with a given temporal structure makes it possible to direct the generation towards a harmonic resolution (e.g. by going to the tonic).

SCENARIO: In this work, a scenario stands for a formalized temporal structure guiding a music generation process.

An example of scenario-based system is *ImproteK* (Nika, Chemillier, and Assayag, 2016), which uses pattern-matching algorithms on symbolic sequences. The musical inputs and the memory are compared to predefined scenarios at a symbolic level. ImproteK is also reactive, as the scenario can be modified via a set of pre-coded rules or with parameters controlled in real-time by an external operator (Nika et al., 2015).

2.2 HYBRIDIZATION OF THE REACTIVE AND SCENARIO-BASED GUIDING SYSTEMS

In 1969, (Schoenberg and Stein, 1969) formulated a fundamental distinction between *progression* and *succession*. From their point of view, a progression is goal-oriented and future-oriented, whereas a succession only specifies a step-by-step process.

Systems using formal temporal structures include this notion of progression. In addition, they introduce a form of anticipatory behaviour. However, the concept of scenario is limited to a predefined configuration of a system, which limits the algorithm ability to predict future movements or changes within musical streams. Furthermore, Huron (Huron, 2006) formalized the following chain:

$$expectation \rightarrow prediction \rightarrow anticipation \tag{2.1}$$

Thus, the scenario is a temporal specification that replaces prediction and expectation. In other words, this chain becomes "specification $\rightarrow$ anticipation" in scenario-based systems.

The DYCI2 library (Nika et al., 2017), has been developed to merge the "free", "reactive" and "scenario-based" music generation paradigms. Thus, it proposes an adaptive temporal guidance of generative models. Based on a listening module that extract musical feature from the musician's input or manually informed by an

operator, the system will propose at each step a scenario. Even if each step-scenario has a size which is more than one step, each queries is dynamically killed, merged and / or reordered, and launched in due time. Therefore, the DYCI2 library opens a promising path for the use of short-term scenarios in order to obtain the entire chain of Equation 2.1. Nevertheless, the short-term scenarios inferred by DYCI2 are currently generated from simple musical heuristics. Thus, in this thesis, the motivation in term of application is to infer short-term scenario based on a high-level feature discovering module (e.g. chord sequence extraction) and a high-level feature predictive modules (e.g. chord sequence prediction).

2.3 INTELLIGENT LISTENING AND INFERENCE OF SHORT-TERM SCENARIOS

Structures are crucial elements of music, and their inference becomes decisive in the process of improvisation involving several musicians. Indeed, although improvisation is associated with spontaneous reactions, it is largely based on rules and structures that allow different musicians to play together properly. If we focus on blues or jazz improvisation, these are usually based on fixed progression of chords that define structuring guidelines for the whole performance. This chord progression is an high-level structure that musicians will follow to develop their improvisations while playing with others. Therefore, in a collective improvisation, it is essential to understand underlying musical structures.

STRUCTURE: In this work, we define a structure as a sequence of symbols in a chosen alphabet (e.g. chord progression) , which describes the temporal evolution of a musical sequence. Therefore, our notion of structures not only stands for a high-level segmentation of an entire piece of music (Paulus, Müller, and Klapuri, 2010; Peeters, 2007).

Our objective in term of application is then to design an intelligent listening module capable of real-time discovery of the structures existing within a musical flow (here chord progression discovery). Therefore, an inference of short-term structures will be made from this discovery in order to feed scenario-based generative systems.

Figure 2.2 illustrates the general workflow of our such co-improvisation system. First, the musical signal is processed to obtain a time-frequency representation. Second, we extract from this representation high-level abstract musical characteristics. Here, we call high-level abstractions any musical characteristics that summarizes the original signal without a precise level of description, but with a high level of understanding of the musician's intent (akin to chord progressions). Finally, a prediction system receives this abstract representation and proposes a possible continuation of this sequence structure.

Multiple music generation applications could already benefit from real-time prediction of musical structures, especially scenario-based generation software such as in the aforementioned DYCI2 library (Nika et al., 2017). Based on this

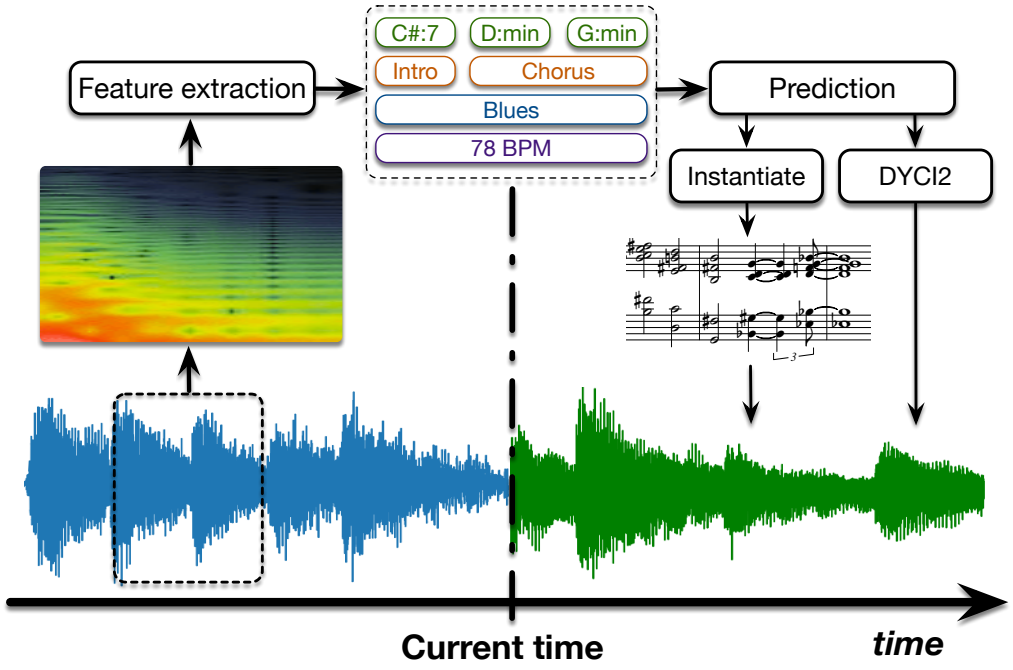

Figure 2.2: Extracting structures and prediction.

inference, music generation processes could fully combine the advantages of both forms of "guidance" mentioned above. Therefore, our motivation is to develop an application that could be reactive (in the real-time sense), while maintaining a long-term temporal vision, capable of anticipating and generating improvisation from an inferred (rather than solely predefined) underlying structure.

OUR MUSICAL MOTIVATION is to infer future musical structures to improve human-computer co-improvisation. In this thesis, we present our proposed approaches for extracting structures from audio streams and for predicting symbolic music based on this information.

OUR APPLICATION CASE is that of musical chords. Indeed, chords are medium-level musical elements that concisely describe the harmonic content of a piece. Moreover, chord sequences are often sufficient for musicians to play in an improvisation context. Thus, the models presented here mainly concern the field of Automatic Chord Extraction (ACE) and the generation of musical chord sequences.

3

METHODOLOGY

3.1 DATA REPRESENTATION

The data representation used for training machine learning models usually strongly influences their performances. Indeed, we can represent data in different ways and this has an impact in terms of the sparsity, repetition or even relationships between elements. For music, the data representation could be divided into two families: the signal and the symbolic. The signal representation of the music is the waveform or any time-frequency representation whereas a symbolic representation of music is a more abstract concept.

3.1.1 *Waveform and time-frequency transforms*

An acoustic signal is a variation of atmospheric pressure taken at a specific point of space. The visualization of the amplitude's variation through time is called a waveform. In order to store this continuous signal, it has to be reduced to a discrete signal. This operation is called a *sampling* and the *sampling rate* defines the amount of points per second.

SAMPLING RATE: It is commonly defined in Hertz (Hz) and stands for the number of samples contained in a second of recording. The human range of hearing is between 20 and 20000 Hz. The Nyquist-Shannon theorem states that the sampling rate requirement should be the double of the maximum frequency. Indeed, (Shannon, 1949) asserts that: "If a function $x(t)$ contains no frequencies higher than B hertz, it is completely determined by giving its ordinates at a series of points spaced $1/(2B)$ seconds apart." Then, the sampling rate of audio waveforms is often at 44.1 kHz.

Even if the sampling of an acoustic signal allows to obtain a finite set of points, the dimentionality of this representation is often too high for the use with informatics. Indeed, only recent Deep Learning models handle waveform data (Mehri et al., 2016; van den Oord et al., 2016). For this model, the training time and the dimentionality of the network is important and even if upgrades have been proposed (Oord et al., 2018; Paine et al., 2016), it could be useful to work with a time-frequency representation of the waveform signal.

3.1.1.1 *Short Term Fourier Transform*

The Short-Term Fourier Transform (STFT) is a spectral transform which is used to extract the sinusoidal frequency and phase content of local sections of a signal

(Bracewell, 1986). In the STFT, we divide the pressure signal, into shorter segments of equal length (by performing a *windowing* operation) and then compute the Fourier transform separately on each segment.

$$X_{STFT}(m,k) = \sum_{n=0}^{N-1} x(n).w(n-m).e^{-i\frac{2\pi k}{N}n} \tag{3.1}$$

where $x(m)$ is the $m-th$ sample of the signal and $w(n)$ is the window function, commonly a Hann or Gaussian window centered around zero. The variation of the window length N is a trade-off between time and frequency resolution.

3.1.1.2 *Constant-Q Transform*

The STFT gives a linear resolution of the frequency bands along a definite range. However, the pitches in western music are rather based on a logarithmic scale organization along the spectrum of audible frequency. Thus, the STFT might not be the most optimized representation for the study of musical chords. The Constant-Q Transform (CQT) is also a spectral transform of audio signals, akin to the Fourier transforms (Brown, 1991). However, this transform can be thought of as a series of logarithmically spaced filters. Thus, in the CQT, the Q value, which is the ratio between the central f_k frequency to the bandwidth δf_k is constant :

$$Q = \frac{f_k}{\delta f_k} \tag{3.2}$$

The windows length N becomes a function of the bin number $N(k)$ and the windowing function $W(k, n-m)$ becomes a function of the window length in order to maintain the alignment across the different analysis frequencies by centering the windows appropriately:

$$X_{CQT}(m,k) = \frac{1}{N(k)} \sum_{n=0}^{N(k)-1} x(n).W(k,n-m).e^{-i\frac{2\pi Q}{N(k)}n} \tag{3.3}$$

Therefore, an appropriate choice in the bank of filters allows us to have a correspondence between the calculated bins and the music notes. This property is also very helpful to simplify the implementation of audio transpositions.

3.1.1.3 *Spectogram*

The CQT or the STFT transform an audio track into a spectrogram. Thus, a spectogram is a time-frequency representation of an audio signal, such as a musical song. Similarly to images, it can be described as a two-dimensional matrix of points. However, each point of a spectrogram contains the intensity of a frequency band over a specific time step (see Figure 3.1). Thus, the two axes of the spectogram are not equivalent since they do not represent the same physical information. Unlike an image, a rotation of the spectogram will completely change its meaning. A translation on the frequency axis will also have a significant impact on the nature

of the resulting audio signal. Indeed, as each note is associated with a whole set of frequencies, the translation operation can change the tone of the song. Furthermore, this set of harmonic frequencies are strongly related and define a cyclical behavior on the frequency axis. Moreover, unlike an image, the spectrogram of a song can be very scattered, as song may be tuned on various tonalities, which will most likely be observed as missing frequencies. Finally, if two instruments are played simultaneously, they will not necessarily obscure each other, but rather become a mixture across both dimensions. This mixing property, also sum up in a non-linear way in terms of local energy, due to the phase of each harmonic in the calculation of the resulting intensity.

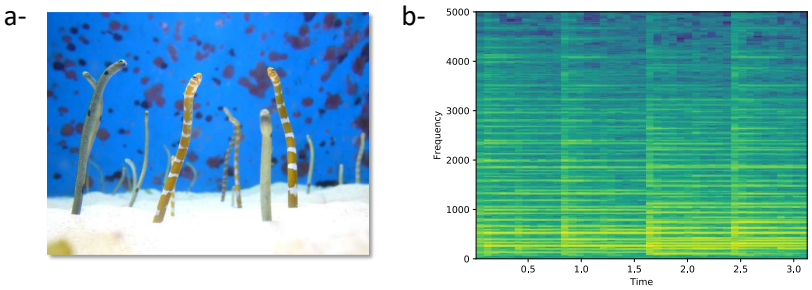

Figure 3.1: a- Picture of sea snakes. b- Time-frequency representation of a sequence of chords.

IMAGE An image can be seen as a two-dimensional matrix of pixels that contains one or more values describing the intensity of different color components at each point. The two axes x and y of the image have an equivalent importance and also feature the same overall properties. Therefore, slight rotations or translations of the image will most likely not change its meaning. Furthermore, images are mostly continuous in terms of neighboring pixels, but also features a property of *stationarity* (where local statistical properties are similar across the whole image). In general, the information contained in an image is not specific to a particular area. In addition to the above properties, objects in an image can live in different configurations of occlusion between foreground and background.

Based on all these observations, images and spectrograms are different on many aspects. Indeed, spectogram and images seem to only share the fact that they can be represented as two-dimensional matrices. Therefore, blindly applying methods that were designed specifically for computer vision tasks might be a perilous endeavor.

3.1.2 *Symbolic music*

In contrast to acoustic signals, symbolic music stands for the different representations that can describe music through a set of symbols. Thus, the MIDI notation, scores or musical labels are considered as symbolic music notation. Although these

labels could be described as words in fixed dictionaries, their intrinsic relationships are different from that of natural language (see Figure 3.2).

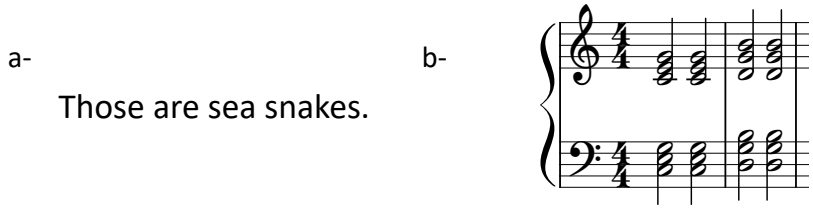

Figure 3.2: a- Sentence describing a picture. b- Score of piano.

3.1.2.1 *Musical score*

A musical score is composed of notes, where each note is defined by its pitch, duration and velocity. A combination of such notes played simultaneously creates a chord, and the complete set of chords will usually define the tonality of a song. Notes, chords and keys are musical entities that have their own specific alphabet. For each entity, the amount of elements in the vocabulary is small, but each element can have a large number of occurrences in a musical song. Indeed, the duration, occurrences and repetitions of various musical elements are strongly correlated to the rhythm of a song. Moreover, music is often polyphonic where different instruments can be played simultaneously.

3.1.2.2 *Piano roll*

The piano-roll notation is often used in machine learning models. In that case, the model receives a vector for each time step containing the activation of all the notes played at this step. This is a very sparse representation, and depending of the quantization, a lot of repetitions is observed between two successive piano-roll vectors.

3.1.2.3 *Midi encoding*

Other representations such as the *MIDI-like* (Moog, 1986) encoding are also frequently used for machine learning models. Unlike the piano-roll representation, a MIDI file does not contain a vector for each time step but a list of MIDI events. An event is added to the list for each new note played (symbol *NOTE_ON*). This event contains information about the pitch and velocity of the played note. When the note is released, an event containing the pitch and release velocity is added to the list (symbol *NOTE_OFF*).

TEXT In ML, text is often processed by considering each word as a unique element from a dictionary. Hence, the corresponding vocabulary is quite large and most

words will have a very low amount of occurrences. In addition to this vocabulary, each language is based on a set of grammatical rules. These rules impose the ordering of words in a sentence and might also force some transformations on the words themselves. Furthermore, in most languages the dictionary elements are *polysemic*, and the system must itself understand the different meanings of a single word. Finally, it is interesting to note that we usually have an extremely low repetition rate of words in a sentence.

Once again, even if notes and other musical features can be considered as words in specific alphabets, their relationships, grammar and syntax are very different from that of natural languages.

3.2 RULE-BASED AND STATISTICAL APPROACHES

Informatic systems in Music Information Retrieval (MIR) could be divided into two distinct families, namely *rules-based* and *stochastic* approaches. The rule-based approaches tackle automatic composition by calculating novel pieces using given sets of formal rules, usually derived from music theory. Oppositely, stochastic approaches produce the generation of pieces based on given statistical processes. Indeed, these systems usually attempt to extract their own understanding of music from a large amount of data. This learning process allows the system to learn its own rules, which can then be used to generate novel musical pieces in a deterministic or stochastic way. Deep Learning models now consistently provide better results than the previous rule-based methods. However, rule-based models are usually very understandable and provide interpretable results, which can often be directly associated with existing rules of music theory. Oppositely, current deep models are often considered as complicated black boxes. These two families of approaches should not be thought as incompatible and can be combined, as is sometimes the case with *Artificial Intelligence* (AI) systems.

4

USING MUSICAL RELATIONSHIPS BETWEEN CHORD LABELS TO INTRODUCE QUALITATIVE ANALYSIS OF CLASSIFICATION RESULTS

As aformentioned in Chapter 2, the application case of this thesis is to develop an intelligent listening and predictive module that discover the musical intent of a live performer at a high level of abstraction in order to predict short-term scenarios. In this chapter, we want to highlight the existing problems that statistical models might encounter when applied to MIR tasks such as chord extraction or chord sequence prediction.

Deep learning methods and modern neural networks have witnessed tremendous success in Computer Vision (CV) (Guo et al., 2016) and Natural Language Processing (NLP) (Young et al., 2018). In the field of Music Information Retrieval (MIR), the number of published papers using Deep Learning (DL) models has also exploded in recent years and now represents the majority of the state-of-the-art models (Choi et al., 2017). However, most of theses works rely straightforwardly on successful methods that have been developed in other fields such as CV or NLP. Hence, musicological knowledge and the intrinsic properties of music are rarely taken into account.

The Convolutional Neural Networks (CNN) (LeCun, Bengio, et al., 1995) were developed based on the observation of the mammalian visual cortex, while recurrent neural networks (Cho et al., 2014; Mikolov et al., 2010) and attention-based models (Luong, Pham, and Manning, 2015; Vaswani et al., 2017) were first developed with a particular emphasis on language properties for NLP. In MIR, these models are often used directly to classify or infer information from sequences of musical events, and have shown promising success in various applications, such as music transcription (Hawthorne et al., 2017; Sturm et al., 2016), chord estimation (Humphrey and Bello, 2012; McVicar et al., 2014), orchestration (Crestel and Esling, 2017; Hadjeres, Pachet, and Nielsen, 2017) or score generation (Dong et al., 2018).

Nevertheless, DL models are often applied to music without considering its intrinsic properties. However, images and time-frequency representations - and similarly scores and texts - do not share the same properties. With this is mind, it is relevant to propose DL methods that take into account musical relationships and attempt to introduce known musical properties directly inside the learning process. Indeed, introducing knowledge of music theory into learning models could improve their performances and bridge the gap between ML and rule-based methods.

4.1 CONSIDER THE INHERENT RELATIONSHIPS BETWEEN MUSICAL OBJECTS

4.1.1 *Reflect musical properties through data representation*

Musical signal or its time-frequency representation could have easily recognizable intrinsic relationships on predefined geometries. Indeed, each instrument has its specific timbre and all the instruments blend together to create musical structures and textures. In addition, most of the instruments used to compose Western music are harmonic instruments. That is, the fundamental frequencies will have harmonics at proportional frequencies. For instance, the frequency repetition of note harmonics shows a cyclic behaviour along the frequency axis (see Figure 4.1). However, as pointed out by recent studies (Bronstein et al., 2017), some machine learning models such as convolutional neural networks are not well adapted to particular cases of geometries (e.g. non-Euclidean).

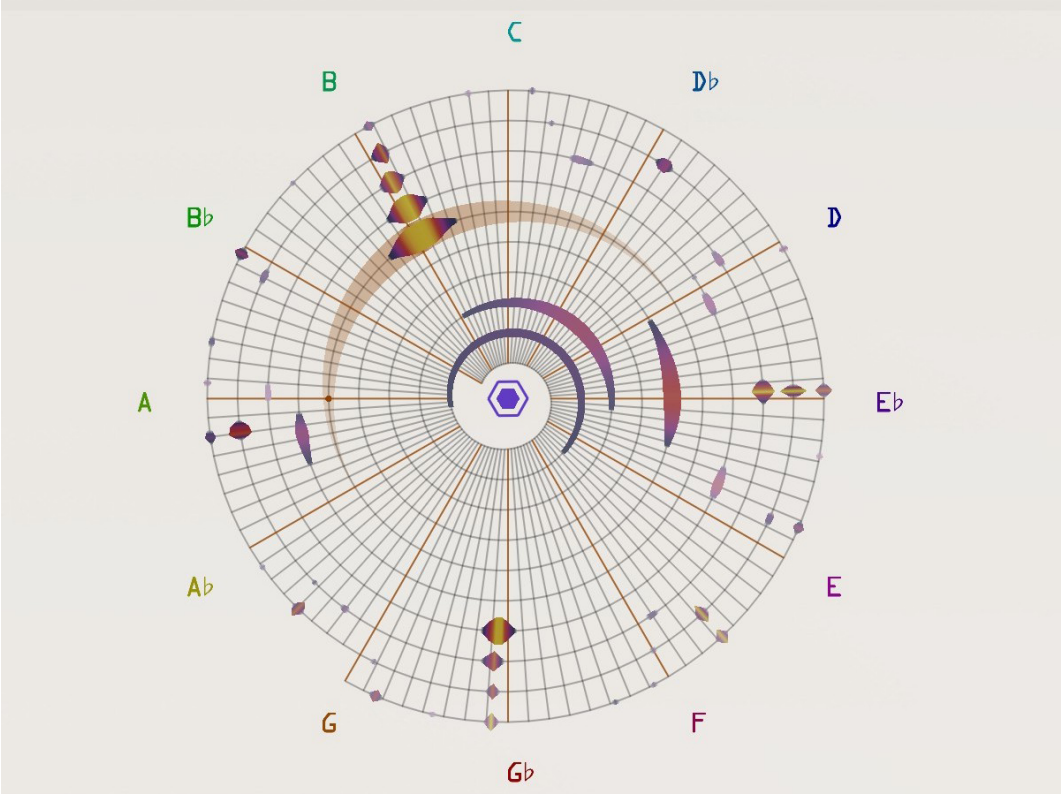

Figure 4.1: Cyclical visualization of an acoustic signal through the SNAIL, a real-time software developed at IRCAM (Hélie and Picasso, 2017).

Similarly, some high-level musical labels have strong relationships if we represent them in a carefully selected geometry. For instance, the relations between three-note chords can be very efficiently visualized on a Tonnetz space (see Figure 4.2). This toric geometry is based on the harmonic relations between chords, each chord having as neighbours major or minor chords that have two pitches in common with

it. In the case of piano-roll, MIDI or score representation, repetitions or patterns along the score are very frequent. Indeed, quantization implies that the most common event is repetition, leading to an ill-defined problem (Crestel and Esling, 2017). In this sense, we need to think about the representation of the data and perhaps use other representations that directly contain the repetition or duration information, or even develop the score of the song in a temporal hierarchical notation.

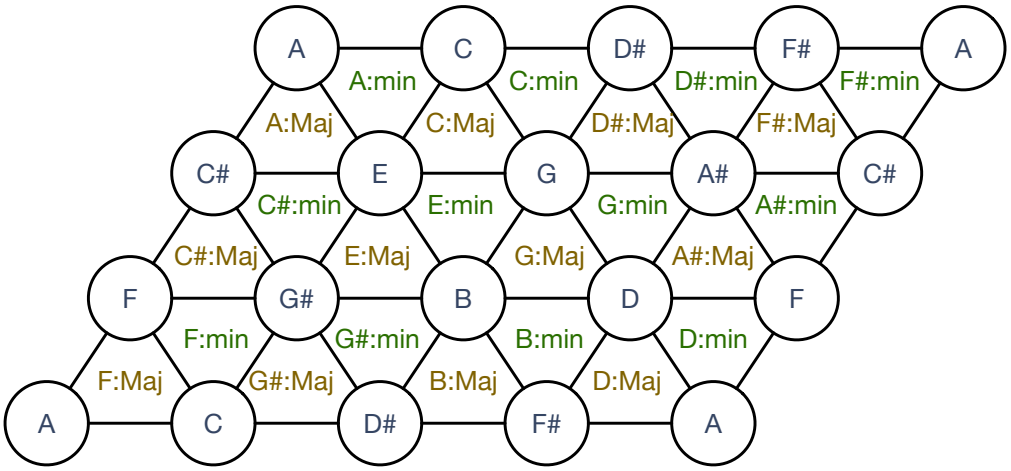

Figure 4.2: Tonnetz representation of major and minor chords. Each node corresponds to a note. The combination of three nodes create a chord (major or minor). Each juxtaposed chords share two pitches.

4.1.2 *Take advantage of the hierarchical behaviour of musical objects*

Music has a hierarchical temporal organization, in terms of symbolic representation, from note to harmonic progression as well as for the raw signal in terms of its frequency representation.

Recent models such as Wavenet (van den Oord et al., 2016) or Sample-RNN (Mehri et al., 2016) use respectively dilated convolutional neural network or hierarchical recurrent neural network in order to use as input the raw audio signal. The hierarchical architecture of these models allows them to understand the structure of music at different time-scales and then generate impressive new materials.

This idea of multi-scale architectures has also been successfully applied to images. For instance, the architecture defined in (Karras et al., 2017) makes it possible to drive the network with increasing input resolution thanks to a progressive growing of Generative Adversarial Network (GAN). The resulting image generation shows a good resolution as well as a good overall image consistency.

In music, the multi-scale organisation of musical structures is inherent. Indeed, notes, chords or tonalities are labels describing music at different time scales. Thus,

it is interesting to consider the design of models that use all available information from a corpus to classify or generate music.

4.2 CONSIDER CHORD LABELS AS FUNCTIONS RATHER THAN NATURES

In Computer Vision, neural networks are commonly used for classification tasks, where a model is trained to identify specific objects, by providing a probability distribution over a set of predefined classes. For instance, we can train a model on a dataset containing images of ten classes of numbers. After training, this network can predict the probabilities over unseen images. Since a digit belongs to only one class, there is supposedly no ambiguity for the classification task. However, in the case of music, high levels of abstraction such as chords can be associated with multiple classes (e.g. C:Maj, C:Maj7, C:Maj9). Indeed, one can choose to describe a chord with a different level of detail by deleting or adding a note and, thus, changing its precise qualities. On the other hand, an ambiguity could be present if a note is only played furtively or is an anticipation of the next chord. Indeed, in contrast with images, music is a discourse and has a temporal development. An audio section could have notes that do not belong to the associated chord label. Therefore, strong relationships exist between the different chord classes depending on the musical discourse and the hierarchical behaviour of chord labels. Hence, as underlined in recent studies (Humphrey and Bello, 2015), even expert human annotators do not agree on the precise classification of each chord (see Figure 4.3).

A "high-level of abstraction" defines any music label that can give a musically understanding of pieces without the ability to precisely reconstruct them. For example, chords and keys can be considered as a higher level of abstraction than musical notes. In the case of our intelligent listening module, the most important objective is to discover the underlying harmonic progression and not a succession of precisely annotated chords. Hence, even a wrongly predicted chord label could actually be an adequate substitute for the original chord and, therefore, still give useful information about the harmonic progression. In some cases, we could even prefer to have "equivalent" classes of chords instead of a very precise level of description. Therefore, we prefer to consider a chord as a function that will be used for a piece of music rather than an entity that belongs solely to a specific class (see Section 6.3.5).

It is important to note that these previous observations are specific to a high-level of abstraction of the musical characteristics. Oppositely, in the case of piano transcription, a note will always be associated with its given class. Therefore, in the next section, we attempt to give different definitions of transcription in the field of MIR.

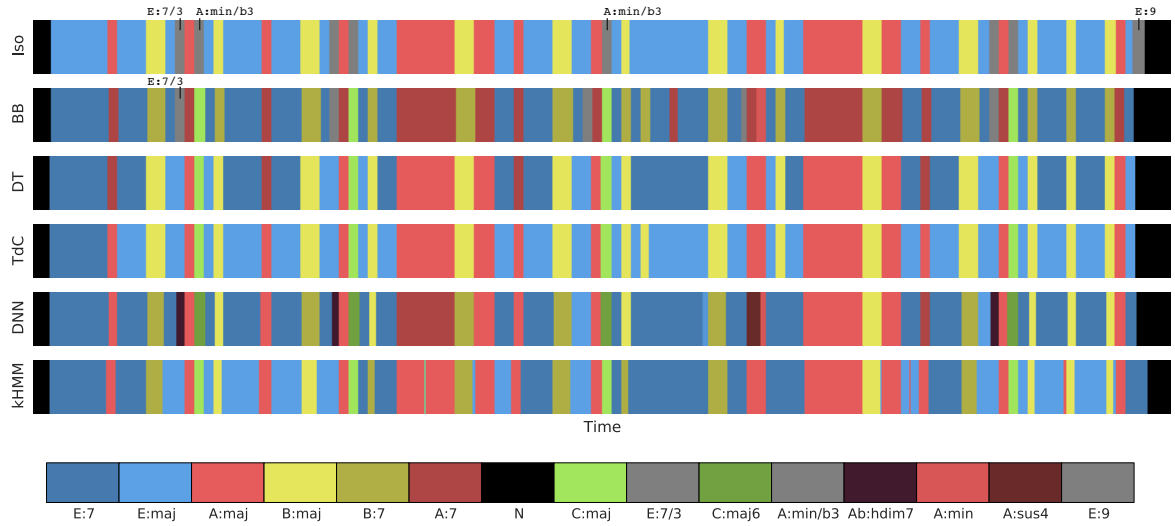

Figure 4.3: Six perspectives on "I Saw Her Standing There", by The Beatles, according to Isophonics (Iso), Billboard (BB), David Temperley (DT), Trevor deClercq (TdC), the Deep Neural Network (DNN), and the k-stream HMM (kHMM). Image taken from (Humphrey and Bello, 2015).

4.2.1 *Differentiate strict and high-level transcriptions in MIR*

Music Information Retrieval (MIR) is a rapidly growing research field, that relies on a wide variety of domains such as signal processing, psychoacoustics, computer science, machine learning, optical music recognition and all fields of music analysis and composition. In this section, we present the specific MIR task called *automatic music transcription*. However, a state-of-the-art for Automatic Chord Extraction (ACE) will be presented in Section 9.2.2.

In (Klapuri and Davy, 2007), authors state that "*music transcription refers to the analysis of an acoustic musical signal so as to write down the pitch, onset time, duration, and source of each sound that occurs in it. [...] Besides the common musical notation, the transcription can take many other forms, too. For example, a guitar player may find it convenient to read chord symbols which characterize the note combinations to be played in a more general manner.*" In addition to this definition, (Humphrey and Bello, 2015) describes the chord transcription task as "*an abstract task related to functional analysis, taking into consideration high-level concepts such as long term musical structure, repetition, segmentation or key.*" With the help of these definitions, we propose two different kinds of transcription named *strict transcription* and *high-level transcription*

STRICT TRANSCRIPTION: We define as *strict* a transcription that objectively associates a unique label for each section of a musical signal. Labels are defined on the basis of a purely physical analysis and must not have any ambiguity in their definition. For example, the extraction of notes from a piano recording

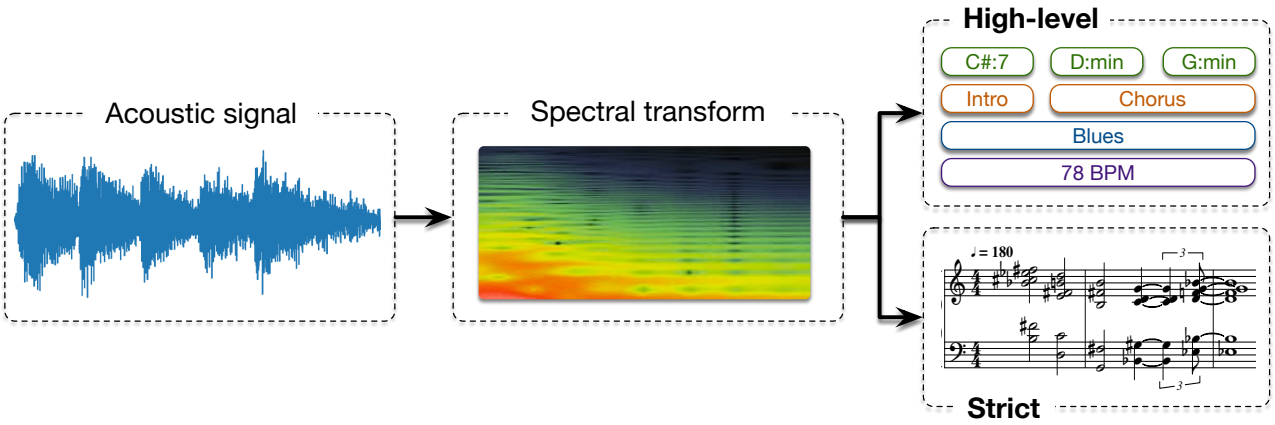

Figure 4.4: Workflow of a transcription task. The signal is first represented by its time-frequency representation. Then, musical labels or notes can be extracted from this representation. We define two kind of transcription named *strict* and *abstract*, depending of the nature of the extracted musical labels.

is a strict transcription. There is only one set of notes that defines perfectly the recording.

HIGH-LEVEL TRANSCRIPTION: We define a *high-level* transcription as the labelling of a musical signal into abstract musical labels. Abstract musical labels include all musical information that could be subjective, such as different chords in an harmonic context, the tonality of a song, the structural division of a track, the musical genres, or any human-annotated labels based on perceptual properties.

In general, the automatic transcription of an acoustic musical signal follows a specific workflow (see Figure 4.4). First, the acoustic signal is usually transformed into a time-frequency representation, which reduces the large dimensionnality of the corresponding waveform to a smaller set of frequency bands at discrete time steps. Then, this representation is used to extract musical information such as notes for producing a score (*strict* transcription), or more abstract features such as chords, key or genre (*high-level* transcription). This transcription step (from a spectogram to musical labels) could be realized through *rule-based* and *statistical* methods.

4.3 QUALITATIVELY IMPROVE ML MODELS FOR EXTRACTION AND PREDICTIVE MIR TASKS

In Machine Learning (ML), we rely on a loss function to optimize the models. The purpose of this function is to calculate a value that informs on the accuracy of the classification, prediction or reconstruction. Thus, a loss function has to be defined according to the aimed task, and is highly dependent on the data representation of the model output. The input representations, as well as the model architecture that will process them, could also have a strong impact on the model performances.

In the previous section we proposed two different kinds of transcription named *strict* transcription and *high-level* transcription. In this thesis, we focus on the transcription and the prediction of *high-level* musical labels (i.e. chord labels). We argued that, oppositely to *strict* transcription, the estimation of musical chords, key or genre can never result in a perfectly accurate score, as the labeled datasets used are inherently biased by the subjective nature of the annotation. Thus, innovative evaluation methods are very important to measure the performances of these systems (Feisthauer et al., 2020). Moreover, many creative applications would benefit from a higher level of harmonic understanding rather than an increased accuracy of label classification. Reciprocally, it seems very complex to establish a single criterion for the evaluation of the generative models in music (Sturm et al., 2019). Thus, for the extraction and prediction tasks of high-level labels, we are convinced that we cannot rely solely on quantitative analysis, but we must also carry out a qualitative analysis, by studying equivalently the nature of the predictions along with the nature and type of errors. Indeed, errors could be considered as weak or strong depending on the nature of the misclassified chords. For example, the misclassification of a *C:Maj* into *A:min* or *C#:Maj*, will be considered equivalently wrong for usual loss function. However, *C:Maj* and *A:min* share two pitches in common whereas *C:Maj* and *C#:Maj* have completely different pitch vectors.

Based on these observations, we present a tentative list of challenges for the development of MIR applications. As depicted on Figure 4.5, we divide these challenges into four different categories that are not orthogonal, since changes in one aspect could affect the others.

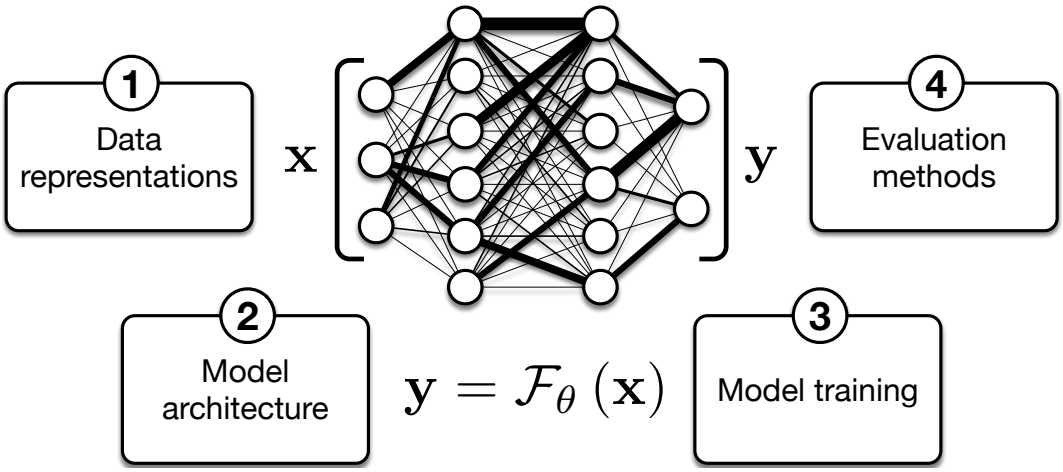

Figure 4.5: Identification of four entities within Neural Network methods where musical knowledge could be used to improve classification or inference of musical features.

5

OVERVIEW OF THE CONTRIBUTIONS

5.1 LIST OF PUBLICATIONS

Carsault, Tristan, Axel Chemla-Romeu-Santos, and Philippe Esling (2018). "Learning spectral transforms to improve timbre analysis." In: *Timbre is a Many-Splendored Thing*.

Carsault, Tristan, Andrew Mcleod, Philippe Esling, Jérôme Nika, Eita Nakamura, and Kazuyoshi Yoshii (2019). "Multi-step chord sequence prediction based on aggregated multi-scale encoder-decoder network." In: *Proceedings of the IEEE conference on Machine Learning for Signal Proccessing*.

Carsault, Tristan, Jérôme Nika, and Philippe Esling (2018). "Using musical relationships between chord labels in automatic chord extraction tasks." In: *Proceedings of ISMIR*.

Esling, Philippe, Theis Bazin, Adrien Bitton, Tristan Carsault, and Ninon Devis (2020). "Ultra-light deep MIR by trimming lottery tickets." In: *Proceedings of ISMIR*.

Wu, Yiming, Tristan Carsault, Eita Nakamura, and Kazuyoshi Yoshii (2020). "Semi-supervised Neural Chord Estimation Based on a Variational Autoencoder with Latent Chord Labels and Features." In: *IEEE Transactions on Audio, Speech and Language Processing*.

Wu, Yiming, Tristan Carsault, and Kazuyoshi Yoshii (2019). "Automatic Chord Estimation Based on a Frame-wise Convolutional Recurrent Neural Network with Non-Aligned Annotations." In: *27th European Signal Processing Conference (EUSIPCO)*. IEEE.

5.2 LIST OF ASSOCIATED GITHUB REPOSITORIES

- https://github.com/carsault/ismir2018, Automatic Chord Extraction system of "Using musical relation-ships between chord labels in automatic chord extraction tasks", ISMIR 2018.
- https://github.com/jnika/ACE_Analyzer, ACE analyzer of "Using musical relation-ships between chord labels in automatic chord extraction tasks", ISMIR 2018.
- https://github.com/carsault/MLSP19, Chord Sequence Predictive system of "Multi-step chord sequence prediction based onaggregated multi-scale encoder-decoder network", MLPS 2019.

5.3 LIST OF PRESENTATIONS

- Collegium Musicæ "Analyse | Création", "Extraction automatique d'accords et prédiction de structures musicales par des méthodes d'apprentissage machine" [1], IRCAM Paris, March 2018.
- Workshop on Artificial Intelligence applied to Music Composition, "Automatic chord extraction and structure prediction through machine learning, application to human-computer improvisation", Tokyo University of the Arts, June 2019.

5.4 RESEARCH EXCHANGES

Two research exchanges for a total of 8 months in *the Speech and Audio Processing Group*, Kyoto University, JAPAN, under the supervision of Kazuyoshi Yoshii.

5.5 ORGANIZATION OF CONFERENCES

Creation and organization of the first *Workshop on Artificial Intelligence applied to Music Composition*[2] held in *Tokyo University of the Arts* with the help of Suguru Goto. This workshop had a second edition thanks to Adrien Bitton [3].

5.6 SUPERVISION OF STUDENTS

- Supervision of three groups of students (master ATIAM at IRCAM) for their Machine Learning project (3 years). Subject: Introduction of musical distances in the learning of chord sequence predictive models.
- Supervision of a student from Columbia University for a three-month internship. Subject: Changing the data representation by integrating chord repetition information, application to chord sequence predictive models.

1 https://medias.ircam.fr/x98a226
2 https://tcmml.github.io/Outline/
3 https://adrienchaton.github.io/seminar_geidai_AI_Music/

Part I

BACKGROUND

6

BACKGROUND IN MUSIC THEORY

6.1 INTRODUCTION

The concept of musical chords is at the heart of our co-improvisation application, as it defines the structuring framework of live improvisation. Therefore, in this work, we will mainly focus on the extraction and prediction of chord labels. This abstract notation allows to describe an acoustic musical signal at high-level of abstraction, but it also has a musical semantic that represents musical functions. In order to fully understand the theory behind musical chords, this chapter will go through the successive layers of musical abstractions, by starting from the acoustic signal.

ACOUSTIC SIGNAL: Following the seminal analysis of Joseph Fourier in 1822, it has been shown that any acoustic signal could be represented as an infinite sum of different frequencies whose amplitude varies over time (Sneddon, 1995). In order to have a more complete view on musical signals, as stated by (Roads et al., 2013), *"the sound produced by acoustic musical instruments is caused by physical vibration of a resonating structure. This vibration can be described by signals that correspond to the evolution in time of the acoustic pressure generated by the resonator"*.

MUSICAL SCORE: At the beginning of the 7^{th} century, Isodorus of Seville declared in his book *Etymologiae*, that *"unless sounds are held by the memory of man, they perish, because they cannot be written down"*. Thus, one could describe a score as a symbolic representation of a song aimed at transcribing the musical intent of a composer as precisely as possible.

Different levels of abstraction can be noticed between the existing music representations, akin to the musical score already patently providing a higher level of abstraction than the signal. But the score itself is composed of other notations such as notes and chords, whose abstraction and precision are themselves variable. If musical chords are constituted of notes, it is still possible to approximately describe some pieces through their progressions. This is for example the case of jazz standards, that can be defined by series of notes *theme* and an harmonic progression in which each chord has a specific function. Therefore, a chord label will have a different role depending on the context.

6.2 DEFINING MUSICAL SCALES FROM PHYSICAL PHENOMENA

The definition of notes are highly dependent on the culture. Nevertheless, most of the songs that can be found in Western popular music are built on established musical scales that come from the physical analysis of vibrating bodies.

6.2.1 *Standing waves in a homogeneous string*

In order to clearly define the notion of musical scale, we study the resonance frequencies of a string attached to two knots. A string can be represented as a chain of oscillators of finite length, which can be excited by an external perturbation that propagates through the string and reaches the end of the chain. At this point, the wave is reflected and interferes with the initial wave to generate a standing wave. The peculiarity of this standing wave is that each peak of amplitude $u(t,x)$ will be stationary along the axis of the string x. The physical time equation that satisfies the string is related to the linear mass density μ and the string tension T.

$$\frac{\mu}{T}\frac{\partial^2 u(t,x)}{\partial t^2} = \frac{\partial^2 u(t,x)}{\partial x^2} \tag{6.1}$$

The solutions of this equation are a family of sinusoidal functions:

$$u(t,x) = \sum_{n=1}^{\infty} B_n sin(\sqrt{\frac{\mu}{T}}\omega_n x)cos(\omega_n t) \tag{6.2}$$

In Equation 6.2, ω_n is called the *pulse*. Each solution $u(t,x)_n$ is an oscillating motion defined by ω_n. We display in Figure 6.1 the first three oscillation modes of our chain. Here, it can be observed that the ratios between the first mode (*the fundamental*) and the following ones (*the harmonics*) are respectively 2 and 3. Thus, if we take a frequency f_1 corresponding to the oscillation frequency of the first mode, then the first harmonic has a frequency of $f_2 = 2 * f_1$ and the second one a frequency of $f_3 = 3 * f_1$. This first harmonic f_2 corresponds to the *octave* in music.

As we want to define the scale of notes on a single octave, we reduce the frequency of the third mode in the range of the octave. We get $f_3' = f_3/2$, which correspond to the normalized frequency of f_3. The tone defined by the frequency f_3' is called the perfect fifth and has a ratio of $3/2$ compared to f_1. We can extend this concept repeatedly to obtain new perfect fifths. After scaling these frequency ratios so that they are contained within the interval of an octave, we obtain a division of the octave scale into a series of perfect fifth ratios. The ratios of the generated note to the fundamental frequency (here *C*), are depicted in the "*Perfect fifth*" column of Table 6.1.

This iteration of fifths is infinite if we use perfect ratios of harmonic frequencies. However, the twelfth ratio value (line *Octave* on Table 6.1) of the circle of fifth is close to 2. Thus, in order to divide the octave into an appropriate number of intervals, it is customary to consider these twelve values as a natural division of the octave.

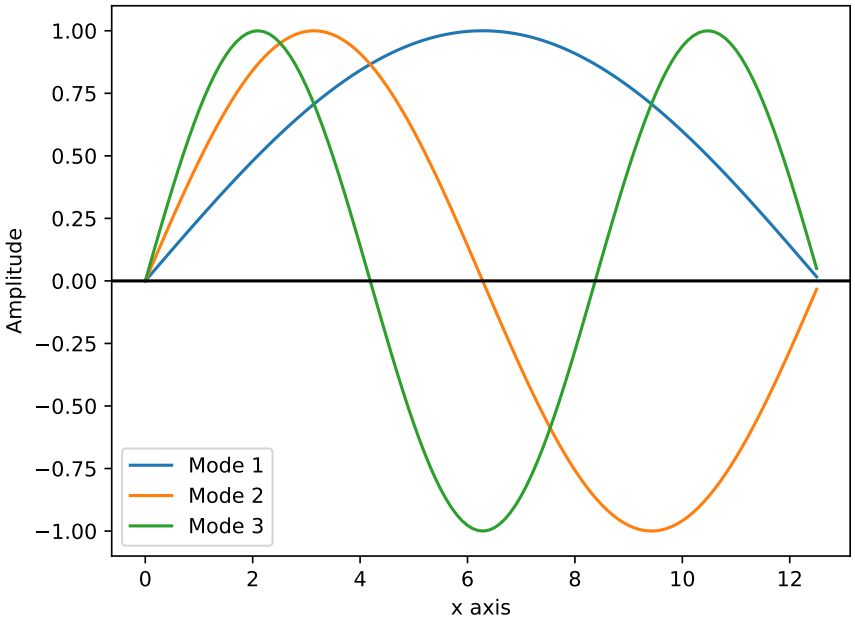

Figure 6.1: Three first modes of a vibrating chord.

Name	Perfect fifth	Tempered scale	Difference
Unison (C)	$(\frac{3}{2})^0 = 1$	$2^{\frac{0}{12}} = 1$	0
Minor second (C♯/D♭)	$(\frac{3}{2})^7 * 2^{-4} \approx 1.07$	$2^{\frac{1}{12}} \approx 1.06$	1.96
Major second (D)	$(\frac{3}{2})^2 * 2^{-1} \approx 1.13$	$2^{\frac{2}{12}} \approx 1.12$	3.91
Minor third (D♯/E♭)	$(\frac{3}{2})^9 * 2^{-5} \approx 1.20$	$2^{\frac{3}{12}} \approx 1.19$	5.87
Major third (E)	$(\frac{3}{2})^4 * 2^{-2} \approx 1.27$	$2^{\frac{4}{12}} \approx 1.26$	7.82
Perfect fourth (F)	$(\frac{3}{2})^{11} * 2^{-6} \approx 1.35$	$2^{\frac{5}{12}} \approx 1.33$	9.78
Tritone (F♯/G♭)	$(\frac{3}{2})^6 * 2^{-3} \approx 1.42$	$2^{\frac{6}{12}} \approx 1.41$	11.73
Perfect fifth (G)	$(\frac{3}{2})^1 = 1.5$	$2^{\frac{7}{12}} \approx 1.50$	13.69
Minor sixth (G♯/A♭)	$(\frac{3}{2})^8 * 2^{-4} \approx 1.60$	$2^{\frac{8}{12}} \approx 1.59$	15.64
Major sixth (A)	$(\frac{3}{2})^3 * 2^{-1} \approx 1.69$	$2^{\frac{9}{12}} \approx 1.68$	17.60
Minor seventh(A♯/B♭)	$(\frac{3}{2})^{10} * 2^{-5} \approx 1.80$	$2^{\frac{10}{12}} \approx 1.78$	19.55
Major seventh (B)	$(\frac{3}{2})^5 * 2^{-2} \approx 1.90$	$2^{\frac{11}{12}} \approx 1.89$	21.51
Octave (C)	$(\frac{3}{2})^9 * 2^{-5} \approx 2.03$	$2^{\frac{12}{12}} = 2$	23.46

Table 6.1: Names of the notes obtained by dividing the octave into twelve tones, along with their frequency ratio according to the perfect fifth scale and the tempered scale as well as the differences between the two scales in term of cents (a logarithmic unit of measure dividing the octave into 1200).

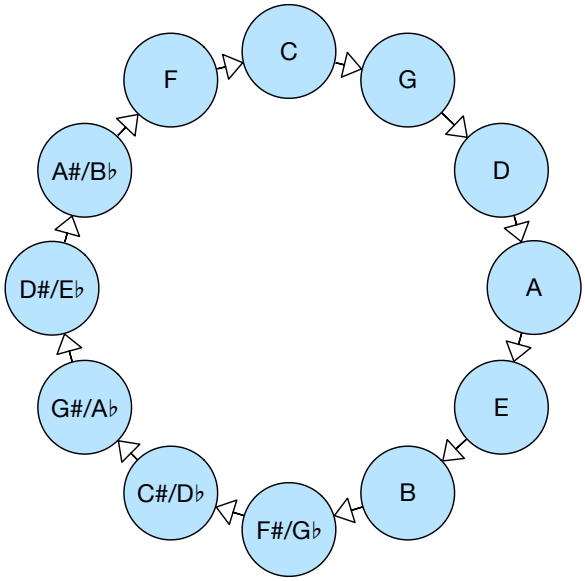

Figure 6.2: Circle of fifth. Each arrow defines the fifth degree of the previous note.

CIRCLE OF FIFTH: This division of the octave into twelve values allows us to define the *circle of fifth*, a geometrical representation of the iteration of the fifths over twelve notes (see Figure 6.2).

As a result of these observations, the *Pythagorean Scale* (based on perfect fifth ratio) was proposed as the reference musical scale and has been used by musicians until the Middle Ages (Sandresky, 1979). However, since this approach of splitting an octave does not result in equal intervals, this does not allow a simple transposition to another tonic.

6.2.2 *The tempered scale*

In order to simplify the Pythagorean scale and obtain an easily transposable one, it has been proposed to develop the *tempered scale*, in which each octave is divided into 12 equal semitones. (see column *Tempered scale* of Table 6.1). It is used in almost all types of Western music and its most common notation is depicted in Figure 6.3. Each note of this scale has a name that depends on its position in the scale (see column *Name* of Table 6.1).

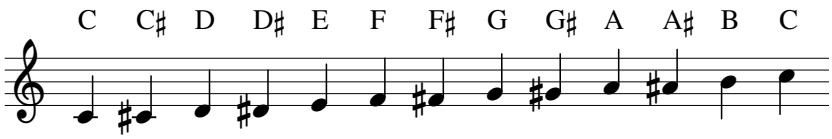

Figure 6.3: One octave of the tempered scale.

6.3 FROM NOTES TO HARMONY

In this section, we introduce musical *scales* that are commonly used in the composition or analysis of music, when performed with tempered instruments.

MUSICAL SCALE: A musical scale is a set of notes, beginning with a root note and following a specific pattern of intervals selecting a specific subset of the tempered notes defined previously. A scale is equivalent from one octave to another.

After presenting the *major scale* and the different *minor scales*, we introduce the musical chords that are obtained through harmonization. Finally, we introduce the concept of *tonalities*, the *functional* behaviour of chords, and present some basic notions of harmony.

6.3.1 *The major scale*

The major scale is composed of seven notes, in addition to the octave note. These eight notes can be divided into two series of four notes, following the same sequence of interval: whole, whole and half. A *whole* interval corresponds to an entire tone and a *half* to a semitone. The interval between these two groups of notes being a whole tone, we call the interval between the 4^{th} and 7^{th} a *triton*. All the major scales taken at the 12 different tonics of the tempered scale will follow this interval pattern.

In order to describe the construction of the major scale, we introduce the notion of triads and major/minor chords.

TRIAD CHORD: As defined by (Pen, 1992), "*a triad is a set of notes consisting of three notes built on successive intervals of a third. [...] the note that forms the foundation pitch is called the root, the middle tone of the triad is designated the third (because it is separated by the interval of a third from the root), and the top tone is referred to as the fifth (because it is a fifth away from the root).*"

MAJOR/MINOR CHORD: The upper tone is always a perfect fifth for major or minor chords. Hence, the upper tone is separated from the root by seven semitones. The middle tone of a triad, the third, has an interval above the root that can be three semitones (*minor third*) or four semitones (*major third*). Thus, a *major chord* is defined by the combination of a major third and a minor third. Equivalently, a *minor chord* is the combination of a minor third and a major third.

The set of major or minor chords is then defined by taking the two natures of the chords for each root notes.

$$Maj\_min\_chord = \{P \times Maj, min\} \tag{6.3}$$

where P in Equation 6.3 represents the 12 pitch classes. For instance, the major A# chord is denoted A#:Maj and the minor A# chord is denoted A#:min.

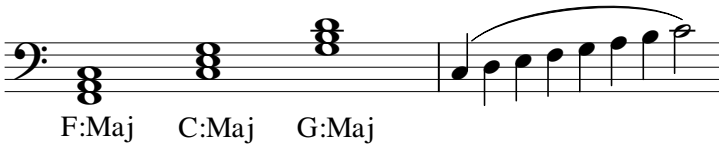

Figure 6.4: Major scale in C.

The major scale can also be defined by joining three major chords (see Figure 6.4). First, we choose a fundamental note and generate its major chord (here C:Maj). Then we choose the major chord whose fifth is the root of C:Maj (here F:Maj). Finally, we select the major chord whose root is the fifth of C:Maj (here G:Maj). By taking all the notes generated by these three major chords, we obtain the major scale of C. All the notes of this scale have a specific name and a related function, as detailed in Table 6.2.

Position in the scale	Function of the note
1^{st}	Tonic
2^{nd}	Supertonic
3^{rd}	Mediant
4^{th}	Subdominant
5^{th}	Dominant
6^{th}	Submediant
7^{th}	Leading tone
8^{th}	Tonic

Table 6.2: Name of the notes depending of the position on the scale

Some of the notes in the major scale have specific relationships with each other. For instance, the 1^{st} note (the *tonic*) of the scale and the 5^{th} note (the *dominant*) have a large set of common harmonics. Furthermore, the 1^{st} note is also the 5^{th} note of the scale having the 4^{th} note (the *subdominant*) as tonic. The aforementioned relationship can be seen using the circle of fifths (see Figure 6.2). These three notes are strongly linked and usually hint to define the tonality of a song. As those notes (C, F, G) are present in both major scale of C and major scale of F, we need the 7^{th} note (the *leading tone*) to remove any uncertainty and precisely define the key of a song. This leading tone is specific to the major scale of C and has only a semitone interval with the tonic.

6.3.2 *The minor scales*

Unlike the major scale, there are various types of minor scales. In this section, we present three different minor scales: the *natural minor scale*, the *harmonic minor scale* and the *melodic minor scale*.

NATURAL MINOR SCALE: This scale is generated by minor chords, which are composed of a root, a minor third and a perfect fifth. Similarly to the major scale, the minor scale is constructed by taking notes from three minor chords that share some specific notes (see Figure 6.5). Importantly, the notes in the natural minor scale of C are the same as those in the major E♭ scale.

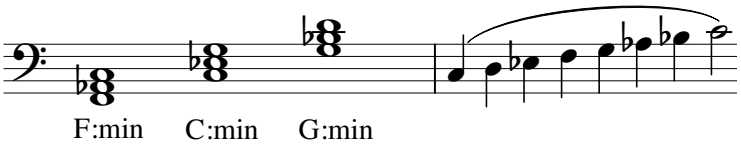

Figure 6.5: Minor scale in C.

HARMONIC MINOR SCALE: This scale is the natural minor scale where the 7^{th} note is increased by one semitone. Thus, the 7^{th} note regains its function as a leading tone (see Figure 6.6). In this scale, the interval between the sixth and seventh is three semitones. It is higher than all other intervals on the scale.

MELODIC MINOR SCALE: In order to reduce the interval between the 6^{th} and the 7^{th}, the 6^{th} note is increased by one semitone, leading to the melodic minor scale (see Figure 6.7).

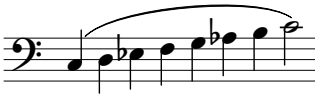

Figure 6.6: Harmonic minor scale in C.

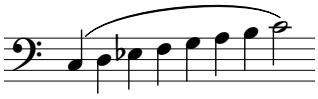

Figure 6.7: Melodic minor scale in C.

Although the natural minor scale has been commonly used in the classical music corpus, it is considered as a mode of the major scale. However, the harmonic minor scale and melodic minor scale are still frequently used in Western music.

6.3.3 *Three-note chords*

In Section 6.3.1, we introduced major and minor chords that are obtained by combining a major third with a minor third. However, other three-note chords can

be obtained with triads: two minor thirds create a *diminished chord*, while two major thirds lead to an *augmented chord*.

The chords generated by the notes of the major scale are called *diatonic*. In Figure 6.8, we represent the triad chords generated by taking each position of the scale as a root note.

The column named *position in the scale* in Table 6.3 gives the degree of each chord in the C:Maj scale.

DEGREE: It is written with a Roman character and corresponds to the position of a note inside a defined scale. The interest of using degrees is that the notation is independent of the pitch of the tonic. A degree is also assigned for each chord obtained by the harmonization of the scale, and is defined by the position of the root note inside the scale.

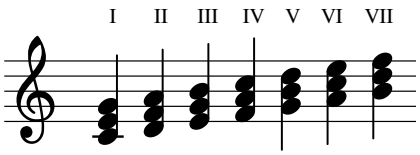

Figure 6.8: Triad chords in C:Maj scale.

Position in the scale	Intervals in semitone	Name of the chord
I	(4, 7)	C:Maj
II	(3, 7)	D:min
III	(3, 7)	E:min
IV	(4, 7)	F:Maj
V	(4, 7)	G:Maj
VI	(3, 7)	A:min
VII	(3, 6)	B:dim

Table 6.3: Harmonization of the major scale with three-note chords. Position of the chord in the scale, name and intervals that constitute the chord.

In Table 6.3, three different qualities of chords are determined by the nature of the thirds present in the chord: *Major* (Maj), *Minor* (min) and *Diminished* (dim). Note that the *Augmented* (aug) chord does not appear on the harmonization of the major scale.

In addition, we define another family of chords composed of three notes:

- The *Suspended* chords, which do not possess a third degree. Thus, we introduce *sus2* and *sus4*, two chords for which we replace the third degree by a second or a fourth degree.

6.3.4 *Four-note chords*

As with three-note chords, we generate four-note diatonic chords by combining triads (see Figure 6.9).

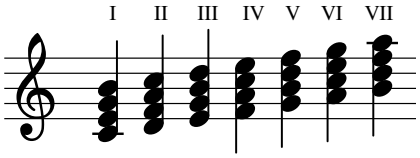

Figure 6.9: Tetrachords in C:Maj scale.

Table 6.4 indicates the nature of the intervals composing the diatonic chords and their denomination. We obtain four different chord qualities: *Major Seventh* (Maj7), *Minor Seventh* (min7), *Dominant Seventh* (7) and *Half-Diminished* (hdim7).

Position in the scale	Intervals	Name of the chord
I	(4, 7, 11)	C:Maj7
II	(3, 7, 10)	D:min7
III	(3, 7, 10)	E:min7
IV	(4, 7, 11)	F:Maj7
V	(4, 7, 10)	G:7
VI	(3, 7, 10)	A:min7
VII	(3, 6, 10)	B:hdim7

Table 6.4: Harmonization of the major scale with four-note chords. Position of the chord in the scale, name and intervals that constitute the chord.

However, there are other types of four-note chords that can be found by stacking triads. For example, a chord of *tenth diminished* (dim7) is composed of three minor triads. This chord is also obtained by reducing the seventh of the *half-diminished seventh* by one semitone, which then becomes the sixth note of the scale. Another chord frequently used in Western music is the *major minor seventh chord* (minmaj7). It is obtained by increasing the seventh of a minor seventh chord by a semitone. Thus, this chord is a combination of one minor third and two major thirds.

In addition to the previous chords, we introduce another three-note chord which is not explained by the superposition of triads :

- 6^{th} chords: We obtain a sixth chord by adding the sixth note of a scale to a major or a minor chord. Then, we obtain a *Major Sixth* (Maj6) or a *Minor Sixth* (min6) chord.

All the different chords presented in the previous sections are listed in Table 6.5. In this work, we do not aim to use a complete list of enriched chords. On the con-

Name of the chord	Intervals	Notation
Major	(4, 7)	Maj
Minor	(3, 7)	min
Diminished	(3, 6)	dim
Augmented	(4, 8)	aug
Suspended 2	(2, 7)	sus2
Suspended 4	(5, 7)	sus4
Major sixth	(4, 7, 9)	Maj6
Minor sixth	(3, 7, 9)	min6
Major Seventh	(4, 7, 11)	Maj7
Minor Seventh	(3, 7, 10)	min7
Minor Major Seventh	(3, 7, 11)	minmaj7
Dominant Seventh	(4, 7, 10)	7
Half-Diminished	(3, 6, 10)	hdim7
Diminished Seventh	(3, 6, 9)	dim7

Table 6.5: List of the qualities of chords with three and four notes. Intervals that constitute the chord and notations.

trary, as we explain in the following sections, we focus on the *functional* behaviour of the chords. Thus, it exists more chords that ones listed in Table 6.5.

6.3.5 *Harmonic progressions and keys*

After defining the concept of musical scales and degrees, the notion of musical tonality must be developed. Indeed, the relations between the different degrees of a scale are dictated by musical rules. Therefore, we often refer to the *tonality* to describe a song. In the context of popular music, we often reduce the tonality of a song to its musical key.

MUSICAL KEY: A key is defined by a tonic and a scale. We often rely on the major scale and the minor scale. By considering all the different pitches existing in tonal music, we obtain a key alphabet of 24 elements (12 major and 12 minor).

However, the tonality of a song also implies strong relationships between the musical notes defined by the scale of the key. Thus, (Berry, 1987) describes the tonal system as "*a hierarchic ordering of pitch class factors, with the tonic the ultimate point of relationship which tonal successions are contrived to expect. [...] Tonality may be thus broadly conceived as a formal system in which pitch content is perceived as functionally related to a specific pitch class*".

Consequently, the function of a chord within a scale is related to its degree.

(Anger-Weller, 1990) describes a chord by three factors:

1. Its sound quality, that depends on the intervals that constitute the chord.
2. Its degree on the scale. Sometimes different chords can share the same degree. We can substitute them in order to break a potential monotony.
3. Its direct entourage. As aforementioned in Section 6.3.1, relationships exist between notes that lead to attraction or repulsion between them. The juxtaposition of chords in a music piece is called an harmonic progression and follows some rules that have proved their efficiency through observing and understanding existing corpus.

In his book *Structural functions of harmony*, (Schoenberg and Stein, 1969) put forward that a chord should be seen as a function in a progression instead of a set of notes:

> "*A triad standing alone is entirely indefinite in its harmonic meaning; it may be the tonic of one tonality or one degree of several others. The addition of one or more other triads can restrict its meaning to a lesser number of tonalities. A certain order promotes such a succession of chords to the function of a progression. A succession is aimless; a progression aims for a definite goal. Whether such a goal may be reached depends on the continuation. It might promote this aim; it might counteract it. A progression has the function of establishing or contradicting a tonality. The combination of harmonies of which a progression consists depends on its purpose - whether it is establishment, modulation, transition, contrast, or reaffirmation.*"
>
> - Schoenberg and Stein, 1969, Structural functions of harmony

Therefore, a chord progression is often independent of the key. It is more common to refer to the chord degrees within the key in order to define the musical cadence.

MUSICAL CADENCE: A cadence is a chord progression that gives a sense of resolution within a musical phrase. This feeling is due to acoustic reasons (unstable chords in the key tend to lead to stable chords) and cultural reasons (we are used to harmonic progressions where one chord leads to another).

The classification of chords according to their degree in a key allows to separate diatonic chords into three families:

- *Tonic chords*: diatonic chords that do not contain a fourth degree note. Thus, if the key is C major, the tonic chords are C:Maj, C:Maj6 and C:Maj7. In addition to these chords, there exists four other chords that are not first degrees of the scale but can substitute for tonic chords. The four diatonic chords that can be substituted for tonic chords are E:min, E:min7, A:min and A:min7.

- *Subdominant chords*: diatonic chords that contain a fourth degree note but where we have no tritone relations between its notes (see Section 6.3.1). Thus, the following chords are switchable and have the same function: D:min, D:min7, F:Maj, F:Maj6, F:Maj7.
- *Dominant chords*: diatonic chords that contain a tritone relationships between its notes. For instance, there exists different dominant chords for the key of C major: G:Maj, G:7, B:dim and D:min6.

The different chords and their function are summarized in the Table 6.6.

Chords	in C major key	Function
I:Maj, I:Maj6, I:Maj7	C:Maj, C:Maj6, C:Maj7	Tonic
III:min, III:min7	E:min, E:min7	Tonic subs.
VI:min, VI:min7	A:min, A:min7	Tonic subs.
II:min, II:min7	D:min, D:min7	Subdominant
IV:Maj, IV:Maj6, IV:Maj7	F:Maj, F:Maj6, F:Maj7	Subdominant
V:Maj, V:Maj7	G:Maj, G:7	Dominant
VII:dim, II:min6	B:dim, D:min6	Dominant Subs.

Table 6.6: Function of chords depending of its degree and qualities. Name of the chords if taken in the C major scale

In the theory of musical harmony, tonic chords and their substitutions are characterized as *stable* chords. Contrariwise, dominant chords and their substitutes are defined as *unstable* chords (Anger-Weller, 1990). Therefore, most musical cadences are based on these aspects of *tension* and *resolution*. For example, dominant chords tend to be resolved by a tonic chord. This movement ($V \rightarrow I$) is called the *perfect cadence*. Similarly, dominant chords could be followed by a submediant or mediant chords (VI-th and III-rd degree). This resolution is not perfect as these chords are only substitutes of the tonic. These progressions, $V \rightarrow III$ and $V \rightarrow VI$, are called *evaded cadence*

There are many other cadences that can be explained mainly by the cycle of fifths. To conclude, musical harmony generally deals with the contextual relationships between chords and has a strong temporal behaviour.

In this chapter, we started from standing waves in a homogeneous string to define tempered scale. We then presented the usual major and minor scales. From the harmonization of the major scale, we introduced three-note and four-note chords. Then, we introduced the notion of degree within a tonality. The main interest of using degree resides in the possibility of withdrawing from the key approach while moving forward the notion of chord labels. Therefore, the progression of chords within an harmonic progression enables to consider chords as functions. Hence, the classification of chords according to their degree in the key allows to separate diatonic chords into different families depending on their functions. This

classification makes possible to substitute a chord to another without changing the nature of the harmonic progression. All these observations could be taken into account for the chord labeling of a musical piece. Furthermore, the accuracy of chord sequence prediction might be interpreted using harmonic analysis.

7

BACKGROUND IN MACHINE LEARNING

7.1 GENERAL APPROACH OF MACHINE LEARNING

The main idea of Machine Learning (ML) is to develop algorithms able to learn from the observation of sets of examples. Thus, ML algorithms use computational methods to learn directly from the data by adapting the parameters of a predefined family of functions. The overarching goal of ML is that these models could generalize their understanding from the given (training) set to unseen data. In this section, we introduce the basic aspects and theoretical definitions required.

7.1.1 *Function approximation*

In most ML problems, we start from a given dataset $X = \{x_1, ..., x_N\}$, usually defined in a high-dimensional space $x_i \in \mathbb{R}^d$. The goal of *supervised learning* is to relate these examples to a set of corresponding information in a given target space $Y = \{y_1, ..., y_N\}; y_i \in \mathbb{R}^t$. This target space defines the goal of the learning system. In most cases, going from one space to the other can be seen as a transform $f : \mathbb{R}^d \to \mathbb{R}^t$, such that

$$f(x_i) = y_i, \forall i \in [1, N]$$

Hence, the aim of machine learning is to find this function $f \in \mathcal{F}$ that produces the desired solution when applied on any input data from these spaces. However, the family of functions $\mathcal{F}$ is usually not defined in a straightforward manner and cannot be used directly. Therefore, across the set of all possible functions, we usually restrain the algorithm to consider a given family of functions $\mathcal{F}^*$, that we define with the aim of being as close to $\mathcal{F}$ as possible. Finally, to approximate the ideal solution, we train a parametric function $f_\theta \in \mathcal{F}^*$, providing the estimation

$$f_\theta(x_i) = \tilde{y}_i, \forall i \in [1, N]$$

By modifying the parameters $\theta \in \Theta$ of this function, the algorithm learns to predict solutions that should be as close to the real solutions as possible.

7.1.2 *Loss function*

The function f_θ requires a set of parameters θ that are defined as a multi-dimensional vector containing the values of each parameter of the function. In order to find this vector, the loss function $\mathcal{L}$ allows to quantify the difference between the correct solution and the current approximation.

The goal of machine learning is then to minimize $L(\mathbf{x} \mid \Theta)$ (see Equation 7.1) by iteratively adjusting the parameters of the function f_θ. This error minimization process evaluates the training examples and then updates the parameters based on the derivative of the error.

$$L(\mathbf{x} \mid \Theta) = \sum_{i=1}^{N} \mathcal{L}(F(x^i), f_\theta(x^i)) \tag{7.1}$$

7.1.3 *Gradient descent*

Learning the optimal parameters of a ML model is obtained by optimizing the loss function $L(\mathbf{x} \mid \Theta)$. The relations between all model parameters and this loss function define a multidimensional space where we want to find the minimum.

GRADIENT DESCENT This is an iterative optimization algorithm aimed at finding a local minimum of the error function. Each step consists of moving the set of parameters in a direction that maximally lowers the value of the loss function. Therefore, gradient descent is defined by starting from a random set of parameters Θ_0 and then repeatedly selecting the direction of steepest descent (with respect to the loss function) in order to update the parameters.

The minimum of $L(\mathbf{x} \mid \Theta)$ corresponds to the optimal setting of model parameters that will provide a function $f_\theta(x) = \tilde{y}$, as close as possible to the desired function $F(x) = y$. The optimization of the parameters θ is iterative. We compute the gradient of the current f_θ by modifying the parameters in order to reduce the objective function.

Thus, at each step we update the parameters such that

$$\Theta_{t+1} \leftarrow \Theta_t - \eta * \nabla_\Theta L(\Theta_t) \tag{7.2}$$

where $\nabla_\Theta L(\Theta_t)$ represents the gradient of the loss function with respect to the parameters and η is an *hyperparameter* called the *learning rate*. This hyperparameter allows to control the magnitude of the step that we are taking at each parameter update.

These ideas are exemplified in Figure 7.1. This figure represents the shape of a two parameters loss function $J(\theta_0, \theta_1)$. The two paths on this figure start in different areas, depending on the random initialization of the model. Then, each path is constructed by choosing the largest gradient descent at each step, finally ending in a local minima (colored in blue).

Relying on too high learning rates could accelerate the convergence but can also cause to "jump" over the minima as the update steps are too large. Conversely, training with a too low learning rate will take a very large number of iterations before reaching the minima as depicted on Figure 7.2.

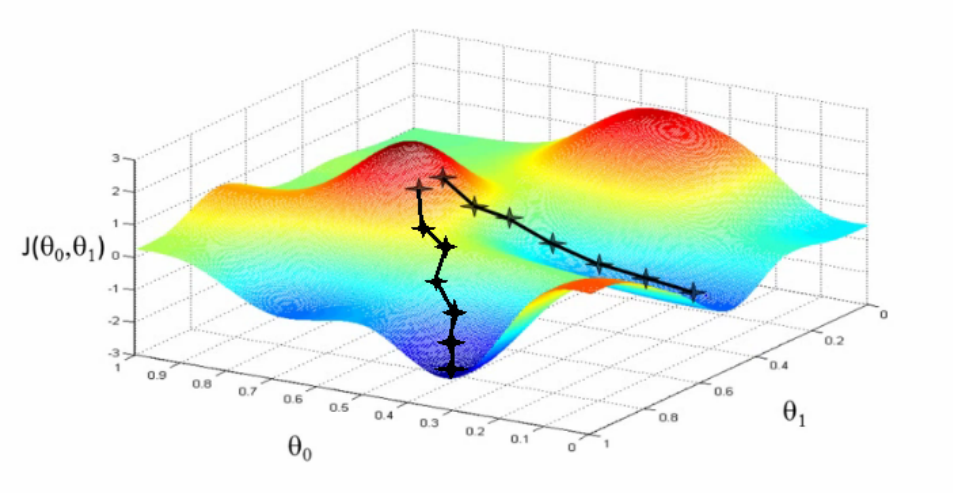

Figure 7.1: Gradient of a two-parameters function with two local minima and example of the use of gradient descent based on two different random initializations, image taken from ("Coursera Machine Learning, Lecture 2 Slides").

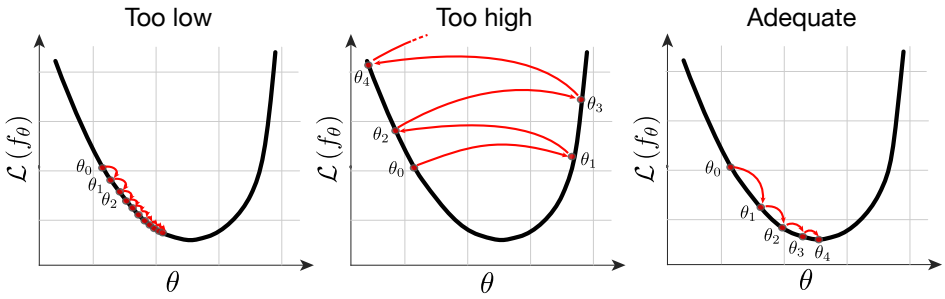

Figure 7.2: Impact of the learning rate on the convergence of the gradient descent algorithm to a local minimum.

7.1.4 *Training procedure*

As discussed earlier, the overarching goal of ML is to obtain models that provide a good *generalization*. This generalization performance refers to the capacity to make accurate predictions for independent (unseen) test data. Thus, the training set should adequately reflect the characteristics of the entire sample space.

The *overfitting* phenomenon appears when a model overemphasizes the details of the training set. For instance, Figure 7.3 shows the effect of overfitting for a task of classification between two classes inside a two-dimensional feature space. When a model is *underfitted*, it does not fit to the data distributions, (as depicted on the left side). A model is *overfitted* when it is over adapted to the training data.

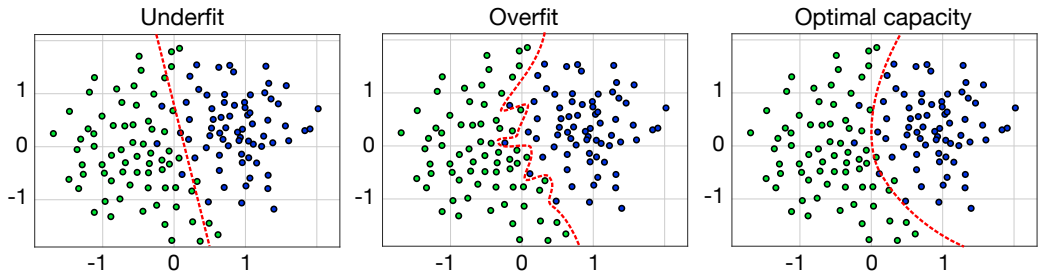

Figure 7.3: Three different models showing either underfitting, overfitting or good generalization in a two-dimensional feature space with two classes.

The overfitting problem can come from the complexity of the model. Indeed, if it is excessively complex, it might possess too many parameters when compared to the number of observations. Among other solutions, we can avoid overfitting and obtain better generalization by slightly corrupting the inputs. Overfitting is also related to the number of training epochs. Thus, we have to stop the optimization process at one point in order to avoid this effect. This step can be seen as finding a tradeoff between the data that we use in our training set and the behavior for future unseen data. One widely used method to avoid overfitting is to validate the model with a supplementary dataset which is independent from the training and the test set.

7.1.4.1 *Train, valid and test datasets*

The dataset that is used to train a model is often divided into three subsets:

- THE TRAINING SET that is used to train the model. It contains the samples with which we adjust the parameters.
- THE VALIDATION SET is used to avoid overfitting. As seen in Figure 7.4, the validation error decreases and then increases during the training process. We must stop the training when the validation error is the lower in order to have the better generalization.

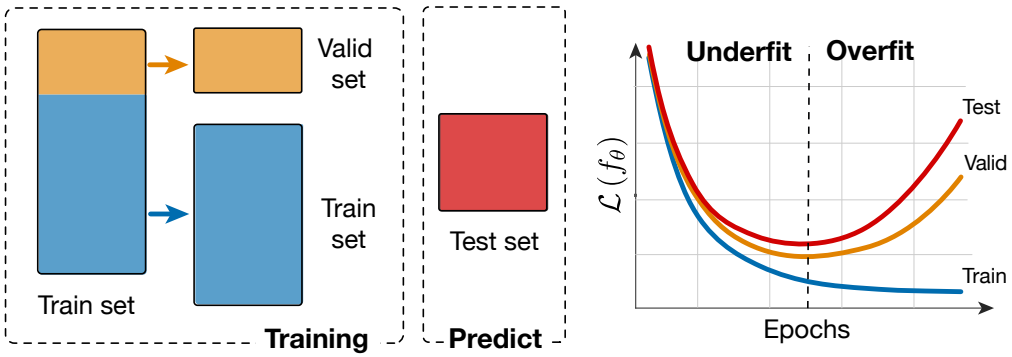

Figure 7.4: Separation of the dataset and evolution of the corresponding errors versus training time.

- THE TEST SET is used to evaluate the final performance of the model.

The training of a model should be stopped when the validation loss is at its lowest point. Thus, we apply *early stopping* when this validation loss does not show any improvement for a defined amount of epochs.

7.1.5 *Types of learning algorithms*

The different approaches to machine learning can be broadly divided into three major types: supervised, unsupervised and reinforcement learning. The choice between these approaches depends principally on the information that we have on the data and the seeking output for a given task. Here, we focus on detailing the supervised and unsupervised approaches as they will both be used in our subsequent methods, in the form of semi-supervised learning.

7.1.5.1 *Supervised learning*

Supervised learning aims at classifying a set of examples by relying on another set of corresponding labels. Hence, learning is done on a complete dataset $\{x(i), y(i)\}_{1 \leq i \leq N}$ containing N samples. Here, $x(i)$ is an input data and $y(i)$ its associated label. We also define an alphabet C that contains all the different labels. In this way, we want the system to extract enough information from $x(i)$ to classify it in the right class $c \in C$.

For instance, in the field of ACE, a well-studied supervised task is to use a spectrogram frame $x(t)$ as an input and develop a model that could find the associated chord label $y(t)$. Thus, we need a training set associating different frames to their chord labels. An example dataset is depicted in Figure 7.5.

For training an ACE model that could perform chord labeling, a potential choice of loss function is the distance between the predicted chord labels and the annotated chord labels. Following our previously defined notation of Equation 7.1, we have the desired output of the classification task given by $f(x(t)) = y(t)$ and

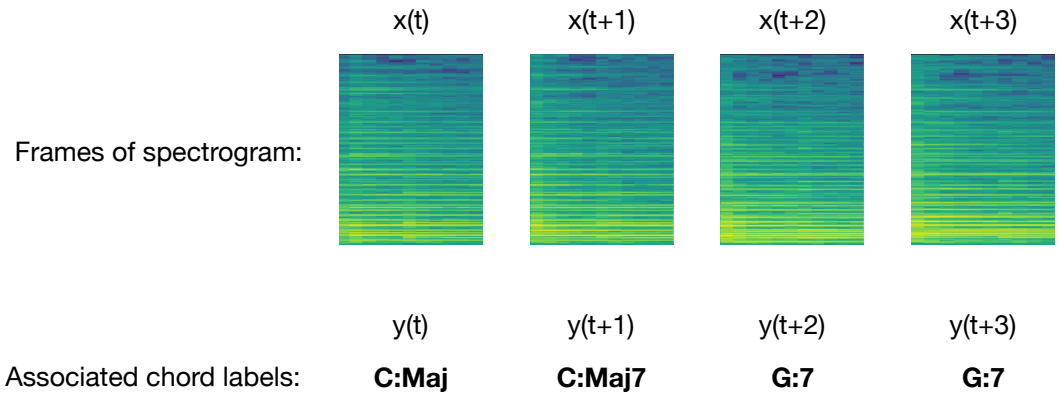

Figure 7.5: Example of elements inside a chord labeling dataset based on spectrogram windows.

the approximation of our model $f_\Theta(x(t)) = \tilde{y}(t)$. By comparing these two values through the loss function $\mathcal{L}(y(t), f_\Theta(x(t)))$, we can assess the errors made by the model in order to improve it by changing its parameters.

7.1.5.2 *Unsupervised learning*

Most of the time, the data that are available for a task are not labeled. Thus, we only have a dataset composed by M elements $\{x(j)\}_{1 \leq j \leq M}$ and we might still want to understand their underlying structure. The most well-known type of unsupervised learning is clustering algorithms. It is used for exploratory data analysis to find patterns or groups inside a dataset. In other words, we extract the most salient features from the inputs. Then, we cluster the data by applying a distance between these features. Since the model has no information about the features we want to focus on, the unsupervised learning works without any assumptions.

Even though supervised learning has demonstrated impressive results in the past years, the unsupervised learning approach is increasingly growing within the machine learning field.

> "*We expect unsupervised learning to become far more important in the longer term. Human and animal learning is largely unsupervised: we discover the structure of the world by observing it, not by being told the name of every object.*"
>
> – LeCun, Bengio, and Hinton, 2015

7.1.5.3 *Semi-supervised learning*

In most applications, we usually only have access to a scarce number of labeled data $\{x_t, y_t\}_{1 \leq t \leq N}$ and, comparatively, to a lot of unlabeled data $\{x_t\}_{N+1 \leq t \leq M}$ (e.g. music tracks without annotations). Consequently, the labeled data is scarce whereas the unlabeled data is plentiful ($N \ll M$). The main idea is to assign probabilistic

labels to unlabeled data in order to use them in the training. By modeling $P(x \mid y)$ as clusters, unlabeled data affects the shape and size of clusters (Zhu and Goldberg, 2009). Technically, we can use unsupervised clustering to assign a label at each cluster, by relying on the labeled dataset.

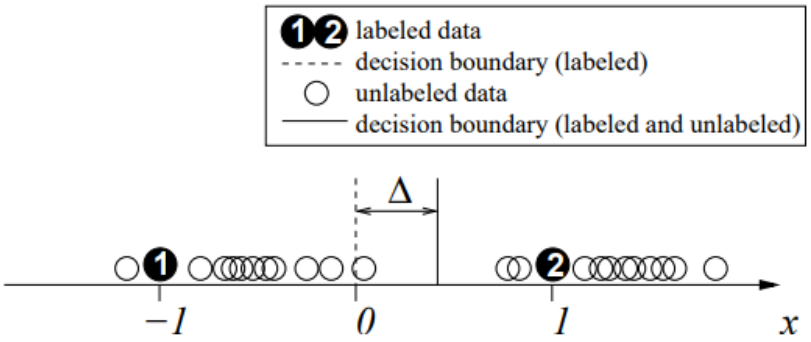

Figure 7.6: Example of a two-class cluster classification dataset where two samples are labeled and others are unlabeled. The addition of unlabeled data modifies the cluster sizes and shapes. (Zhu and Goldberg, 2009).

Figure 7.6 depicts a two-class cluster classification dataset where two samples are labeled and others are unlabeled. Thus, the addition of unlabeled data shifts the decision boundary and gives a more accurate classification on unseen data. Nevertheless, note that semi-supervised learning does not always provide better results depending on the task at hand (Singh, Nowak, and Zhu, 2009).

7.2 NEURAL NETWORK BASICS

7.2.1 *Artifical neuron*

An artificial neuron can be compared to a biological neuron where dendrites are connected to a cell body. Each dendrite transmits incoming electrical impulsion to the body. When the electrical charge exceeds a threshold the cell body sends an electrical impulsion through its axon. In an artificial neuron, as depicted in Figure 7.7, the dendrites are modeled by the inputs $\{x_i\}_{1 \leq i \leq n}$, the activation is a non-linear function σ with a threshold b and the axon is the output h.

A neuron is then defined by its parameters, namely the bias b, weights w_i, and an activation function σ. Formally, the input x_i is linked to the output h through

$$h = \sigma(\sum_{j=1}^{m} w_j x_j + b) \tag{7.3}$$

Many activation functions σ can be used, which are all defined as non-linear transformation. Here, we introduce two widely used functions

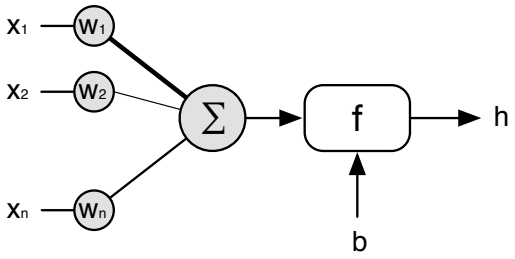

Figure 7.7: Schema of an artificial neuron.

THE SIGMOÏD FUNCTION has a very simple derivative, which facilitates the computation of gradient descent in a neural network:

$$\sigma(x) = \frac{1}{(1 + exp(-x))} \tag{7.4}$$

RELU that stands for Rectified Linear Unit, has been introduced (Hahnloser et al., 2000) with strong biological and mathematical motivations. This is currently one of the most popular activation functions used in deep neural architecture:

$$\sigma(x) = max(0, x) \tag{7.5}$$

7.2.1.1 *Interpretation*

Here we provide two ways of interpreting Neural Networks (NN). By looking at Equation 7.3, we can see that if the activation function is a threshold function, this equation defines an hyperplane.

On the one hand, networks with a single processing layer can be interpreted as separating the input space into two regions with a straight line (as seen in Figure 7.8). Here, the blue and red lines represent two data distributions, while the colored region represents the value of the output neuron. Therefore, we can note that a single-layered network cannot properly separate these two distributions (as this dataset is *non linearly separable*). Hence, we must use a non-linear activation function for this classification task.

On the other hand, a way of interpreting these networks is to see the successive layers as performing *space transformation* operations. As we can see in Figure 7.9, the space is gradually transformed by the successive layers, where the coordinates in each successive space is defined by the output values of the different neurons. Hence, for complex data we must use neural networks with multiple layers, to exploit this property of *compositionality*.

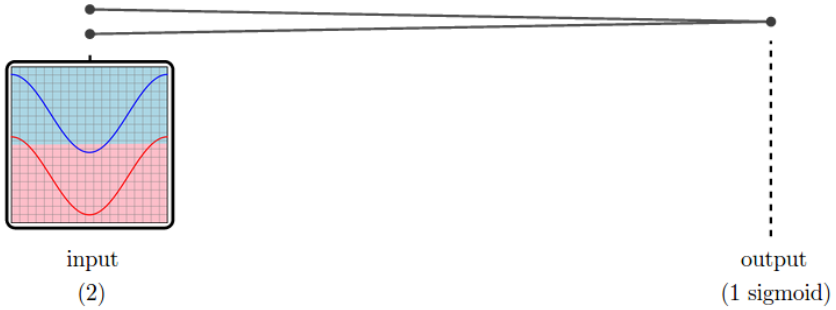

Figure 7.8: Space separation with one linear output layer. Image from (Bengio, Goodfellow, and Courville, 2015).

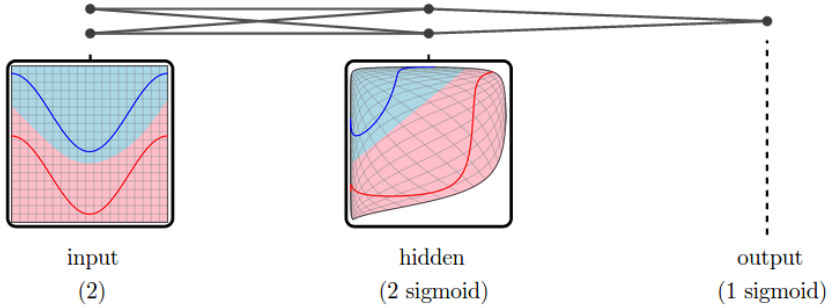

Figure 7.9: Space transform operated by successive hidden layer. Image from (Bengio, Goodfellow, and Courville, 2015).

7.2.2 *Multi-layer perceptron*

The *Multi-Layer Perceptron* (MLP) is a feed-forward model organized as a succession of layers containing neurons. Each layer receives the output of neurons in the previous layer as input and then applies a (linear) affine transform and a non-linear transfer function to these values. Therefore, a MLP is a fully-connected network of depth L. Here, we denote the output value of the l-th layer by vector $y^l \in \mathbb{R}^{N_l}$, where N_l is the number of neurons contained in this layer (see Figure 7.10). For $1 \leq l \leq L$, the parameters of a layer are defined by a weight matrix $W^l \in \mathbb{R}^{N_l \times N_{l-1}}$ and a bias vector $b^l \in \mathbb{R}^{N_l}$. Therefore, the activation of neuron i in layer l is computed with the following equation:

$$h_i^l = \sigma(\sum_{j=1}^{N_{l-1}} (W_{i,j}^l . h_j^{l-1}) + b_i^l), \tag{7.6}$$

where $h_i^0 = x_i$. We can rewrite Equation 7.6 such that $b_i^l = W_{i,0}^l . h_0^{l-1}$.

$$h_i^l = \sigma(\sum_{j=0}^{N_{l-1}} (W_{i,j}^l . h_j^{l-1})) \tag{7.7}$$

Finally, the output of the network is defined by $y_i = h_i^L$.

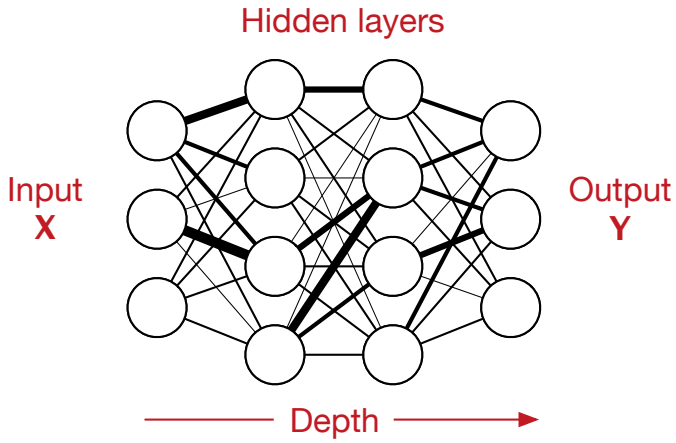

Figure 7.10: Multi-Layer Perceptron.

7.2.3 *Numerical optimization and training*

As discussed previously, we need to find an efficient way to adjust the parameters of a model in order to increase its performance. This performance corresponds to the opposite of the error function. One solution for an efficient training is to use the derivative of the error function that depends directly on the parameters of the model. Hence, at each training iteration, the aim is to find the direction to explore.

7.2.3.1 *Backpropagation*

In order to train a NN model, it would seem that we should define a procedure taking all layers into account at once, leading to a very complicated formulation. However, the introduction of the *backpropagation* method (Rumelhart, Hinton, and Williams, 1986) allows to decompose this into a set of simple operations. The main idea is to see that adding layers amounts to adding a function computed on the previous output. Therefore, in the *forward pass*, this output is updated by relying on the weights and biases of the neurons in this layer. Thus, when trying to obtain the contribution of a given neuron to the final error value, we can use the *chain rule* of derivations, to separate its contribution inside the network. The error values obtained after a forward pass can be simply propagated backwards, starting from the output, and computing the derivative of each neuron output given its parameters. This procedure is repeated until all the weights of the network have been updated.

The forward pass allows to compute the loss $\mathcal{L}$ between the desired output δ_i^L and the output of the network h_i^L. By using Equation 7.6, we define $\mathcal{L}$ on the output of layer $L-1$, the weight matrix W^L and σ the activation function (see Equation 7.8).

Thus, we can define the total loss as an equation containing only the set of weight of the models, the inputs x_j and the desired output

$$\mathcal{L}(\delta_i^L, h_i^L) = \mathcal{L}(\delta_i^L, \sigma(\sum_{j=0}^{N_{L-1}} W_{i,j}^L . h_j^{L-1})) \tag{7.8}$$

$$= \mathcal{L}(\delta_i^L, \sigma(\sum_{j=0}^{N_{L-1}} W_{i,j}^L [...] \sigma(\sum_{j=0}^{N_0} W_{i,j}^1 . x_j))) \tag{7.9}$$

We denote e_i^l the derivative of the error for neuron i of layer l and $o_i^l = \sum_{j=0}^{N_{l-1}} W_{i,j}^l . h_j^{l-1}$, the output of layer $l-1$ before its activation function
The error for the last layer is defined based on the loss function

$$e_i^L = \frac{\delta \mathcal{L}(\delta_i^L, h_i^L)}{\delta h_i^L} \tag{7.10}$$

The chain rule allows to compute the errors e_i^l for all other layers of the NN. First, the chain rule is used to compute the derivative of $\mathcal{L}(\delta_i^L, h_i^L)$ with respect to δo_i^L. In Equation 7.11, the first term corresponds to e_i^L, whereas the second term is the derivative of the activation function.

$$\frac{\delta \mathcal{L}(\delta_i^L, h_i^L)}{\delta o_i^L} = \frac{\delta \mathcal{L}(\delta_i^L, h_i^L)}{\delta h_i^L} * \frac{\delta h_i^L}{\delta o_i^L} \tag{7.11}$$

$$= e_i^L * \sigma'(o_i^l) \tag{7.12}$$

Second, the chain rule is used another time to compute the derivative of $\mathcal{L}(\delta_i^L, h_i^L)$ with respect to the contributions of neurons in the previous layer h_i^{L-1}.

$$\frac{\delta \mathcal{L}(\delta_i^L, h_i^L)}{\delta o_i^L} * \frac{\delta o_i^L}{\delta h_i^{L-1}} = \frac{\delta \mathcal{L}(\delta_i^L, h_i^L)}{\delta o_i^L} * W_{i,j}^L \tag{7.13}$$

$$= e_i^L * \sigma'(o_i^l) * W_{i,j}^L \tag{7.14}$$

Thus, by taking into account all the connections emanating from the last layer

$$e_i^{L-1} = \sigma'(o_i^l) \sum_{j=0}^{M_L} W_{i,j}^L . e_j^L \tag{7.15}$$

Therefore, we obtain a relationship between the errors of two successive layers. This error can therefore be back propagated throughout the network, thanks to the generalization of Equation 7.15

$$e_i^l = \sigma'(o_i^l) \sum_{j=0}^{M_{l+1}} W_{i,j}^{l+1} . e_j^{l+1} \quad ; \quad l \in [1...L-1] \tag{7.16}$$

Finally, the weights are updated in each layer by an amount proportional to the derivative of the error with respect to the associated weight

$$W_{ij}^{l} = W_{ij}^{l} + \eta . e_i^l h_j^{l-1} \tag{7.17}$$

Where η is the learning rate.

7.2.3.2 *Usual loss functions*

The choice of loss function depends on many factors such as the task at hand, but also the distribution of the data used for the training. Here, we present three different losses used for regression or classification tasks.

MSE LOSS The most used loss function for regression task. It is defined as

$$\ell(\tilde{y}, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = (\tilde{y}_n - y_n)^2 \tag{7.18}$$

Due to its quadratic behavior, the predictions that are far from the labeled data are more strongly penalized.

L_1 LOSS It is also often used for regression task and is defined as

$$\ell(\tilde{y}, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = |\tilde{y}_n - y_n| \tag{7.19}$$

This loss is more robust to outliers as it computes errors in a linear way in each dimension.

CROSS ENTROPY LOSS The most common loss for classification tasks, defined as

$$\ell(\tilde{y}, class) = -\log\left(\frac{\exp(\tilde{y}[class])}{\sum_j \exp(\tilde{y}[j])}\right) = -\tilde{y}[class] + \log\left(\sum_j \exp(\tilde{y}[j])\right) \tag{7.20}$$

This loss can be split in two terms. The first one is directly linked to the correct class (where $\tilde{y}$ is the output probability vector). The second one takes into account the probability over other classes. This loss heavily penalizes wrong predictions that have a high level of confidence.

7.2.3.3 *Regularization and initialization to improve generalization*

Simple neural networks, such as MLPs, are composed by few layers and require relatively low computational costs. However, such shallow architectures would require an infinite number of computational elements (Bengio et al., 2009) to solve complex tasks. Conversely, deep architectures exploit non-linear hierarchies to solve this issue. Thus, we use deep learning when we need higher levels of abstractions. This approach allows to construct abstractions by selecting the important variations at each layer, providing information at different scales. Nevertheless, adding several layers leads to a *gradient diffusion* problem during back-propagation. Thus, many *regularization* or *initialization* techniques have been proposed to improve the learning process giving better generalization capacities to NNs.

INITIALIZATION OF WEIGTHS: The weights of NNs can be initialized in ways that take into account the specificity of the architecture. Indeed, instead of initializing the weights with uniform sampling, specific methods such as the *Xavier initialization* (Glorot and Bengio, 2010) initialize the weights from different Gaussian in order to have a variance that remains the same for each layer. This will help to prevent the signal from exploding into a high value (exploding gradient) or fading to zero (vanishing gradient).

BATCH NORMALIZATION: It consists in normalizing the hidden values computed at each layer of NN models. Thus, we compute

$$h_i \longleftarrow \gamma \frac{h_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta \tag{7.21}$$

where μ_B and σ_B^2 are the mean and variance across batches. This reduces the covariance shift of hidden unit values and also accelerates the training process (Ioffe and Szegedy, 2015) by allowing to increase the learning rate.

DROPOUT: During the training and for each forward pass, a given ratio of hidden units is randomly masked. Hence, this forces the networks to optimize the connections between neurons in order to avoid interdependence within a set of neurons (Srivastava et al., 2014).

DATA AUGMENTATION: The augmentation of the dataset size inherently improves generalization. Even if the best way to benefit from this would be to construct bigger datasets, several methods allow to artificially augment the dataset. One such way is to slightly corrupt the inputs by adding random noise. In this way, the network never process exactly the same inputs during training. Other methods exist that are specific to the attributes of the input. For instance, images can be rotated or translated without loosing their meaning. Some more music-specific data augmentation are presented in Section 9.3.2.1.

7.3 ADVANCED NEURAL NETWORKS

7.3.1 *Encoder-Decoder*

An encoder/decoder model is aimed at deconstructing an object and reconstructing it by learning its structure. In the case of supervised learning, the encoder/decoder model can be found in many tasks such as *sequence-to-sequence* translation tasks. The main goal of this architecture is to learn a low dimensional representation of the inputs. Thus, the encoder is acting as a function $E_\phi(x) = z$, where x are the inputs and z their latent codes. The latent representation z is then decoded through the decoder to obtain the output of the model $D_\theta(z) = y$ (see Figure 7.11).

In the case of an Auto-Encoder, the training objective is to get the output y to be as close as possible to x. Hence, the output of the model should be equal to

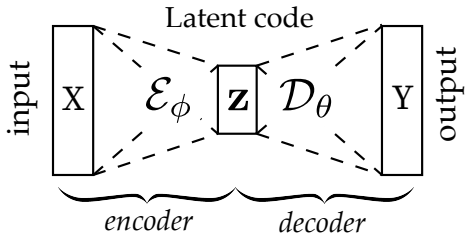

Figure 7.11: Encoder-Decoder architecture. The input **x** are first encoded to a latent representation **z**, which is then decoded to the output **y**.

its input. In this case, as the model does not need labeled data to be trained, the learning is unsupervised.

7.3.2 *Generative models*

The Auto-Encoder (AE) presented in Section 7.3.1 compresses input data into a latent code. This can be seen as a reduced space of the initial data space. However, in classical AE, this latent space is often not well organized. Indeed, taking a point randomly in the latent space and trying to decode it could give meaningless output. To tackle this issue, the Variational Auto-Encoder is defined as an Auto-Encoder that will be trained with a regularization on this latent space.

7.3.2.1 *Variational Inference*

Latent variable models consider that observed data **x** are generated based on some unobserved *latent* random variables **z** (Blei, Kucukelbir, and McAuliffe, 2017). Thus, the complete data distribution is expressed through the marginal distribution $p(\mathbf{z})$ and the probability of generating **x** with given latent variable **z**

$$p(\mathbf{x}) = \int p(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z}) d\mathbf{z} \tag{7.22}$$

Nevertheless, the two terms in the integral of Equation 7.22 are often complicated distributions. Variational Inference (VI) allows to approximate these distributions by relying on a parametrized family of distribution $\mathcal{Q}$. Thus, the conditional density $p(\mathbf{z}|\mathbf{x})$ can be approximated by a distribution $q(\mathbf{z}|\mathbf{x}) \in \mathcal{Q}$. The aim is then to minimize the difference between this approximation and the exact posterior distribution. Thus, the KullBack-Leibler Divergence ($\mathcal{D}_{KL}$) is used to measure the distance between the two distributions:

$$\mathcal{D}_{KL}[q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})] = \mathbb{E}_{q(\mathbf{z})}[\log q(\mathbf{z}|\mathbf{x}) - \log p(\mathbf{z}|\mathbf{x})] \tag{7.23}$$

The Bayes' rule allow us to replace $p(\mathbf{z}|\mathbf{x})$ in Equation 7.23 giving

$$\mathcal{D}_{KL}\left[q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})\right] = \mathbb{E}_{q(\mathbf{z})}\left[\log q(\mathbf{z}|\mathbf{x}) - \log p(\mathbf{x}|\mathbf{z}) - \log p(\mathbf{z}) + \log p(\mathbf{x})\right] \tag{7.24}$$

Since $p(\mathbf{x})$ is independent on $q(\mathbf{z})$, we can rewrite this equation as:

$$\log p(\mathbf{x}) - \mathcal{D}_{KL}\big[q(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z}|\mathbf{x})\big] = \mathbb{E}_{\mathbf{z}}\big[\log p(\mathbf{x}|\mathbf{z})\big] - \mathcal{D}_{KL}\big[q(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})\big] \quad (7.25)$$

In order to perform an optimization of the divergence of the two distribution, we rely on parametric functions $q_\phi(\mathbf{z})$ with $\phi \in \Phi$ and $p_\theta(\mathbf{z})$ with $\theta \in \Theta$. Furthermore, since $\log p(\mathbf{x})$ is a constant value , the optimization is finally defined by:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \underbrace{\mathbb{E}_{q_\phi(\mathbf{z})}\big[\log p_\theta(\mathbf{x}|\mathbf{z})\big]}_{\text{reconstruction}} - \underbrace{\mathcal{D}_{KL}\big[q_\phi(\mathbf{z}|\mathbf{x}) \parallel p_\theta(\mathbf{z})\big]}_{\text{regularisation}} \quad (7.26)$$

The two terms of this optimization stand for two different aspects of the training. The first term is the reconstruction error of the input data $\mathbf{x}$, whereas the second term is acting as a regularisation on the latent space.

Based on this theory, the Variational Auto-Encoder has been proposed to extend encoder-decoder models with this probabilistic background (Kingma and Welling, 2013). At first glance, the sampling operation of this optimization would render the whole network non-differentiable. However, the *reparameterization* trick is proposed to tackle this issue. The main idea is to move the sampling operation out of the network in order to preserve its derivability.

7.3.3 *Convolutional neural network*

A Convolutional Neural Network (CNN) is a specific type of feed-forward neural network that is currently amongst the best performing systems for image processing tasks (Ciresan et al., 2011; Krizhevsky, Sutskever, and Hinton, 2012). The architecture of CNNs were inspired by the organization of the animal visual cortex (LeCun, Bengio, et al., 1995). The main idea of CNNs is to introduce invariance properties (such as translation, rotation, perspective distortion) into the transform computed by neural networks. Compared to MLPs, the hidden layers of a CNN compute different operations, by relying on *convolution* and *pooling* operators. In the next parts, we describe these operators and the overall behavior of CNNs.

7.3.3.1 *Convolutional Layers*

A convolutional layer is defined by a set of convolution kernels that are applied in parallel to the inputs to produce a set of output *feature maps*. These are defined by a three-dimensional tensor $h \in \mathbb{R}^{M \times I \times J}$ where M is the number of kernels, I is the height and J the width of each kernel.

If we denote the input as matrix X, then the output feature maps are defined as $Y = X * h_m$ for every kernels, where $*$ is a 2D discrete convolution operation

$$(A * B)_{i,j} = \sum_{r=1}^{T} \sum_{s=1}^{F} A_{r,s} B_{r+i-1,s+j-1} \quad (7.27)$$

for $A \in \mathbb{R}^{T \times F}$ and $B \in \mathbb{R}^{I \times J}$ with $1 \leq T \leq I - 1$ and $1 \leq F \leq J - 1$.

For instance, as depicted in Figure 7.12, we start from the top left region and move the kernel until it reaches the bottom right border of the input matrix. A zero-padding operation is also performed by adding zeros to the border of the original input. This allows to avoid loosing border information, but it also increases the size of the output representation. Finally, an activation function is applied after each convolutional layer.

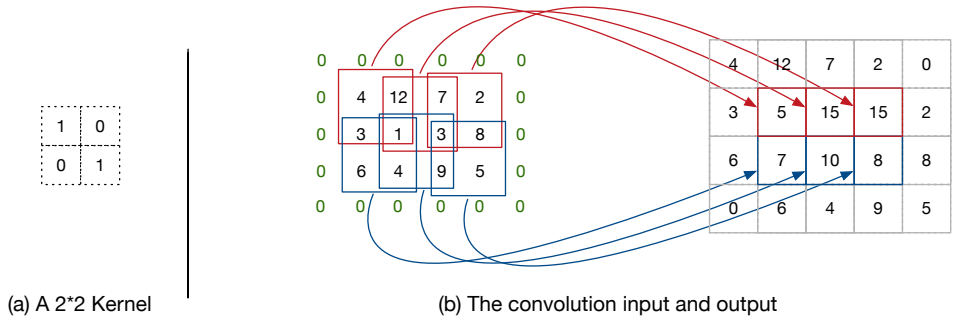

Figure 7.12: Illustration of the convolution operation.

7.3.3.2 *Pooling Layer*

Each convolutional layer significantly increases the dimensionality of the data. Therefore, a *pooling* layer is often placed between convolutional layers in order to reduce the size of the feature maps. This downsampling layer can either perform an *average* pooling, L_2*-norm* pooling or *max* pooling. For instance, the max pooling (as depicted in Figure 7.13) operation only keeps the maximum value in each region of a partition of the input. The pooling size refers to the size of these regions.

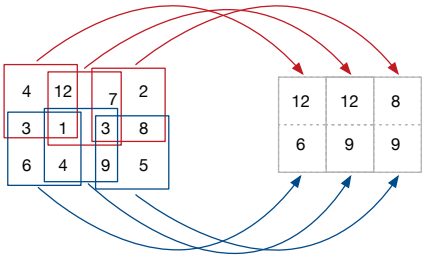

Figure 7.13: Example of a 2×2 max pooling.

7.3.3.3 *Fully-connected layer*

The overall structure of a CNN usually consists of an alternated succession of convolution, activation and pooling layers. Then, in order to perform classification, this architecture is typically followed by one or many fully-connected layers (see

Figure 7.14). This fully-connected layer is a standard MLP, where each neuron in layer l is connected to all neurons in layer $l+1$. Thus, for this model, the last layer produces a probability vector of the same size as the number of classes.

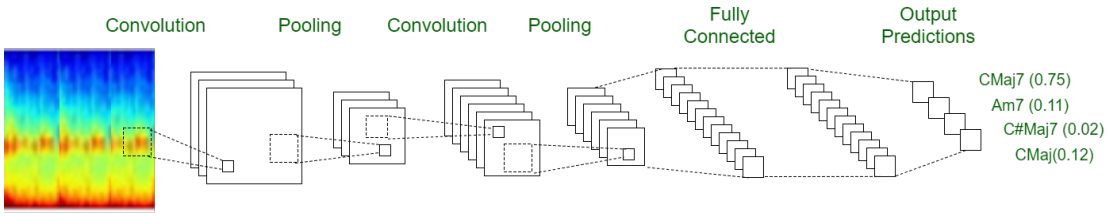

Figure 7.14: Example of a convolutional neural network for a task of ACE.

7.3.4 *Temporal modeling*

One of the largest issue in NNs is their incapacity to store temporal information. However, for our applicative case, we aim to learn and predict the underlying structure of a music signal. Therefore, we need to define a model that must be able to understand the temporal structure of inputs.

Recurrent Neural Networks (RNN) are NN models designed to infer information from time series. However, RNNs do not have a strong temporal reminiscence because of the non-linear transformation at each time step. Thus, Long Short-Term Memory (LSTM) models have been proposed to alleviate this problem thanks to a gated mechanism.

These two kinds of recurrent neural network are presented in the following. Furthermore, the *sequence-to-sequence* mechanism is introduced. These methods allow to efficiently infer a sequence of symbols from an initial one, and is widely used in translation tasks or for the prediction of sequences of elements.

7.3.4.1 *Recurrent neural network*

Recurrent Neural Networks (RNN) have provided promising results in several application fields (Cho et al., 2014), including musical structure generation (Graves, 2013). To define a recurrent network, we usually need to add some loops inside the network, where the output of a neuron is directly linked to itself. However, as this renders backpropagation impossible, the RNN can also be interpreted as a neural network with lateral connections, as seen in Figure 7.15.

The inputs x_i are processed by the RNN, which outputs a value h_i. The transformation is operated by a module $A(x_i, h_{i-1})$ that processes both the input x_i and the output of the previous step h_{i-1}. Therefore, the mathematical formulation of a RNN can be written as

$$h_i = A(x_i, h_{i-1}) \tag{7.28}$$

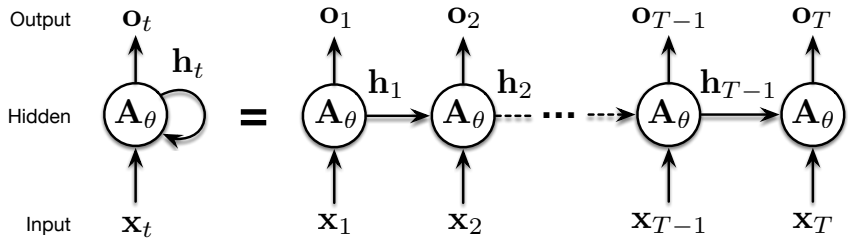

Figure 7.15: Loops of a RNN unfolded through time.

In a standard RNN, the function A has a very simple structure such as a linear layer followed by a *tanh* layer. Nevertheless, standard RNNs tend to forget things quickly along the time steps. Gated memory mechanisms solved this problem by adding to standard RNNs few trainable gates that allow to select, stock and transport the most important information for the task at hand. The two most used gated RNN structures are the Long Short-Term Memory (LSTM) and the Gated Recurrent Unit (GRU).

7.3.4.2 *LSTM*

Generic RNN are limited in their capacity to learn long-term dependencies and they are hard to train (Pascanu, Mikolov, and Bengio, 2013). An elegant variant, called LSTM, has been introduced by (Hochreiter and Schmidhuber, 1997). It has been shown to provide good results in several tasks (Sturm et al., 2016).

Similarly to classic RNN, LTSM networks perform a transformation of input x_i by taking into account the transformations of previous time steps. The major difference lies in the structure of module A. In classic recurrent neurons, the output of the previous step h_{i-1} is simply passed through a non-linearity. Thus, the reminiscence of information between distant time steps is relatively poor.

In LSTMs, an additional internal connection is added, which carries information through time steps with only linear interactions. Thus, the information between distant time steps can remain unchanged. In addition to this memory cell, LSTMs are composed by three gates which control how to update and use the memory cell. The gates are principally the combination of a sigmoïd layer followed by a point-wise multiplication layer.

1. The *forget* gate controls which part of the memory cell to forget. It relies on h_{i-1} and x_i to select which dimension we have to keep or forget and then update the memory cell.

2. The *update* gate decides which values to update, then compute a new memory vector and add it to the previous vector.

3. The *output* gate controls which dimensions of the memory must be passed to the next time step.

As a recurrent network is composed by several layers of recurrent units, it allows to detect patterns at different abstraction levels. This hierarchical vision is reminiscent of the multi-scale structure found in audio tracks.

7.3.4.3 *Sequence-to-sequence models*

RNNs can be used in the context of translation or sequence prediction. Thus, the model has to predict not only one element but a whole sequence. Hence, the *sequence-to-sequence* architecture has been proposed to tackle these situations (Sutskever, Vinyals, and Le, 2014). This architecture is decomposed in two parts such as an *encoder/decoder* (similar to Section 7.3.1), as depicted on Figure 7.16.

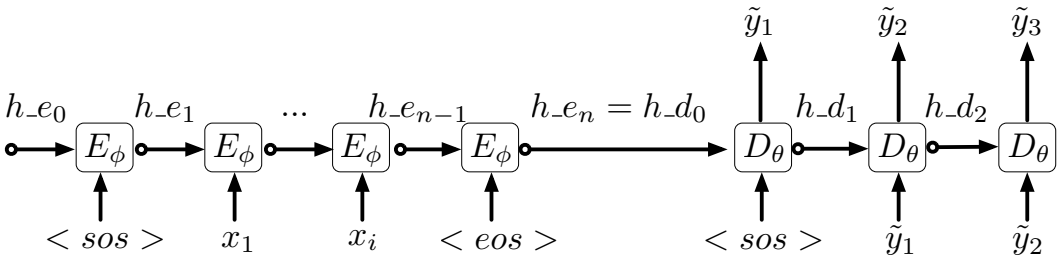

Figure 7.16: Sequence to sequence architecture. The input are given iteratively to E_ϕ. A the end of the input sequence, we start decoding with D_θ

.

After having initialized the hidden parameters of network with the value of h_e_0, the NN receives the *start of sequence* value ($< sos >$). Then, we iteratively input the elements of the sequence in the encoder with each previous hidden states

$$out\_e_{n+1}, h\_e_{n+1} = E_\phi(x_n, h\_e_n) \tag{7.29}$$

At the end of the sequence, we inform the network with the *end of sequence* value ($< eos >$). Then, h_e_n, which is the latent representation of the encoder network, is given to the decoder in order to generate the output sequence.

The second part of the process is the decoding part. Thus, we predict the *len_pred* elements of the sequences by iteratively applying the decoder

$$out\_d_{n+1}, h\_d_{n+1} = D_\theta(\tilde{y}_n, h\_d_n) \tag{7.30}$$

In Equation 7.30 $\tilde{y}_n$ is the input value given to the decoder, defined as

$$\tilde{y}_n = \begin{cases} < sos > & \text{if } n = 0 \\ y_n & \text{if } teacher\_forcing == true \\ arg\_max(out\_d_{n-1}) & \text{if } teacher\_forcing == false \end{cases} \tag{7.31}$$

If the *teacher forcing* algorithm is used for the training (Williams and Zipser, 1989), we use the ground truth data y_n to compute the predicted output at time $n + 1$.

Otherwise, the free training algorithm is used and the input of the decoder is the decoded output at step $n - 1$.

The training of these temporal models is performed with Backpropagation Through Time (BPTT). This algorithm works by fixing the amount of time steps, unrolling the RNN in order to obtain a classic NN, and training it with a standard backpropagation algorithm.

Part II

MUSIC INFORMATION RETRIEVAL CONTRIBUTIONS

8

CHORDS ALPHABETS AND QUALITATIVE EVALUATION METHODS

In this chapter, we present the chord alphabets and the evaluations methods that we use for the chord extraction and the chord sequence prediction tasks. Indeed, the methodology that we introduce here can be used for any chord-based MIR model.

8.1 DEFINITION OF CHORD ALPHABETS

Chord annotations from referenced datasets are very precise and include extra notes (in parenthesis) and bass (after the slash) (Harte et al., 2005). With this notation, we would obtain more than one thousand chord classes with a very sparse distribution. However, we do not use these extra notes and bass in our classification, therefore, we can remove this information.

$$F : maj7(11)/3 \rightarrow F : maj7 \tag{8.1}$$

Even with this reduction, the number of chord qualities (eg. *maj7, min, dim*) is extensive and we usually do not aim for such a degree of precision. Thus, we propose three alphabets named A_0, A_1 and A_2 with a controlled number of chord qualities. The level of precision of the three alphabets increases gradually (see Figure 8.1). The black lines symbolize chord reductions, and chord symbols that do not fit in a given alphabet are either reduced to the corresponding standard triad, or replaced by the no chord symbol *N.C*.

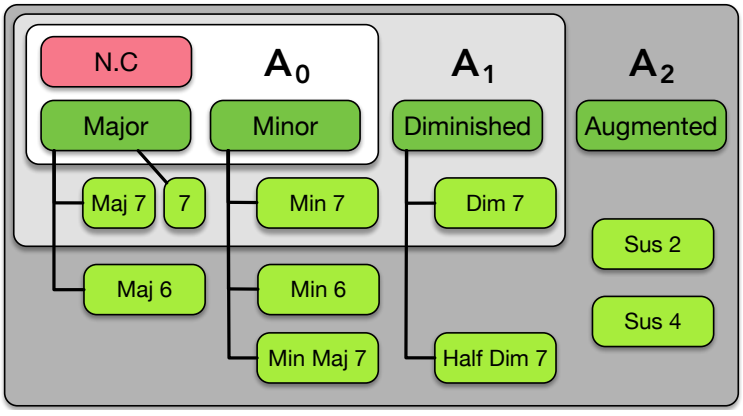

Figure 8.1: The three chord vocabularies A_0, A_1, and A_2 we use in the next chapters are defined as increasingly complex sets. The standard triads are shown in dark green.

The first alphabet A_0 contains all the major and minor chords, which defines a total of 25 classes:

$$A_0 = \{N.C\} \cup \{P \times maj, min\} \tag{8.2}$$

where P represents the 12 pitch classes.

Nevertheless, usual notation of harmony in jazz music contains chords that are not listed in alphabet A_0. Therefore, we propose an alphabet that contains all the four-note chords present in the harmonization of the major scale (see Section 6.3.4). This corresponds to the chord qualities annotated in the A_1 area on Figure 8.1 and their parents. The chord qualities without any heritage are included in the no-chord class N, leading to 73 classes:

$$A_1 = \{N.C\} \cup \{P \times maj, min, dim, dim7, maj7, min7, 7\} \tag{8.3}$$

Finally, the alphabet A_2 is inspired from the large vocabulary alphabet proposed by (McFee and Bello, 2017). This most complete chord alphabet contains 14 chord qualities and 169 classes:

$$\begin{aligned} A_2 = \{N.C\} \cup \{P \times maj, min, dim, aug, maj6, min6, \\ maj7, minmaj7, min7, 7, dim7, hdim7, sus2, sus4\} \end{aligned} \tag{8.4}$$

8.2 CHORD ANALYZER

We propose to analyze ACE results from a qualitative point of view with a *functional* approach. In tonal music, the *harmonic functions* qualify the roles and the tonal significance of chords, and the possible equivalences between them within a sequence (Rehding, 2003; Schoenberg and Stein, 1969). We developed an *ACE Analyzer* including two analysis modules to discover musical relationships between the chords predicted by ACE models and the target chords. Both modules are generic and independent from the classification model, and available online[1]. This analyzer is intended to be used for future ACE research to develop a better understanding of the reasons behind the success or failure of ACE systems.

8.2.1 *Substitution rules*

The first module detects the errors corresponding to hierarchical relationships or usual *chord substitutions* rules, which is the use of a chord instead of another in a chord progression (usually substituted chords have two pitches in common with the triad that they are replacing).

1 http://repmus.ircam.fr/dyci2/ace_analyzer

8.2.2 *Harmonic degrees*

The second module of our *ACE Analyzer* focuses on harmonic degrees. First, by using the annotations of key in the dataset in addition to the chords, this module determines the harmonic degrees of the predicted chord and of the target chord: *e.g.* in C, if the reduction of a chord on A_0 is *C* it will be considered as "I", if the reduction of a chord on A_0 is *D:min* it will be considered as "ii", etc. Then, it counts the substitutions of harmonic degrees when it is possible (*e.g.* in C, if the reduction of a chord on A_0 is *C#* it does not correspond to any degree).

9

AUTOMATIC CHORD EXTRACTION TASKS

9.1 INTRODUCTION

Automatic Chord Extraction (ACE) is a topic that has been widely studied by the Music Information Retrieval (MIR) community over the past years. However, the trend in recent results obtained in chord classification seem to indicate that ACE have reached a glass ceiling (McVicar et al., 2014). Recently, a part of the MIR community pointed out the need to rethink the experimental methodologies. Indeed, current evaluation methods do not account for the intrinsic relationships between different chords (Humphrey and Bello, 2015). This work is built on these questions and is aimed to give some insights on the impact of introducing musical relationships between chord labels in the development of ACE methods.

In this chapter we present two methods that use different techniques to improve the quality of chord extraction system. The first proposed method studies the introduction of musical knowledge through the learning process via musical distances computed between the predicted and the labeled chords. The second method combines discriminative and generative models, and allows to use unlabeled musical data to perform semi-supervised learning.

9.2 RELATED WORKS

Automatic Chord Extraction (ACE) is defined as the task of labeling each segment of an audio signal using an alphabet of musical chords. In this task, the chords are seen as the concomitant or successive combination of different notes played by one or many instruments.

9.2.1 *Considerations on ACE task*

While most MIR tasks have benefited continuously from the recent advances in deep learning, the ACE field appears to face a stagnation in improving the classification score. (Humphrey and Bello, 2015) highlighted the need to rethink the whole ACE methodology by giving four insights on the task.

First, several songs from the reference annotated chord datasets (Isophonics, RWC-Pop, McGill Billboard) are not always tuned to 440Hz and may vary up to a quarter-tone. This leads to multiple misclassification on the concomitant semi-tones. Moreover, they underline the fact that chord labels are not always well suited to describe every song in these datasets.

Second, the chord labels are related and some subsets have hierarchical organizations. Therefore, the one-to-K assessment where all errors are equivalently weighted appears widely incorrect. For instance, the misclassification of a *C:Maj* as a *A:min* or *C#:Maj*, will be considered equivalently wrong. However, *C:Maj* and *A:min* share two pitches in common whereas *C:Maj* and *C#:Maj* have totally different pitch vectors.

Third, the very definition of the ACE task is also not entirely clear. Indeed, there is a frequent confusion between two different tasks: the literal *recognition* of a local audio segment using a chord label and its extensive extensions, and the *transcription of an underlying harmony*, taking into account the functional aspect of the chords and the long-term structure of the song. Therefore, even the notion of context windows used for the classification might be wrong, as it classifies a single chord but still relies on the information of neighboring chords. Finally, the labeling process involves the subjectivity of the annotators. For instance, even for expert annotators, it is hard to agree on possible chord inversions.

Therefore, this prompts the need to focus on other aspects such as the introduction of musical knowledge in the representation, the improvement of the models towards more complex chord alphabets and the development of more adapted evaluation methods.

9.2.2 *Workflow of ACE systems*

Due to the complexity of this task, ACE systems are usually divided into four main modules performing *feature extraction*, *pre-filtering*, *pattern matching* and *post-filtering* (Cho, Weiss, and Bello, 2010).

First, the *pre-filtering* may apply low pass filters or harmonic-percussive source separation methods on the raw signal (Jiang, Li, and Wu, 2017; Zhou and Lerch, 2015). This optional step allows to remove noise or other percussive information that are irrelevant for the chord extraction task. Then, the audio signal is transformed into a time-frequency representation such as the Short-Time Fourier Transform (STFT) or the Constant-Q Transform (CQT) that provides a logarithmically-scaled frequencies. These representations are sometimes summarized in a pitch bin vector called *chromagram* (Harte and Sandler, 2005). Then, successive time frames of the spectral transform are averaged in context windows. This allows to smooth the extracted features and account for the fact that chords are longer-scale events. It has been shown that this could be done efficiently by feeding STFT context windows to a CNN in order to obtain a clean chromagram (Korzeniowski and Widmer, 2016b).

Then, these extracted features are classified by relying on either a rule-based chord template system or a statistical model. Rule-based methods give fast results and a decent level of accuracy (Oudre, Grenier, and Févotte, 2009). With these methods, the extracted features are classified using a fixed dictionary of chord profiles (Cannam et al., 2015) or with a collection of decision trees (Jiang, Li, and Wu, 2017). However, these methods are usually brittle to perturbations of the spectral distribution in the input signal and do not generalize well.

Statistical models aim to extract the relations between pre-calculated features and chord labels based on a training dataset in which each temporal frame is associated to a label. The optimization of the model is then performed by using gradient descent algorithms to find an adequate configuration of its parameters. Several probabilistic models have obtained good performances in ACE, such as multivariate Gaussian Mixture Model (Cho, 2014) and Neural Networks (NN) either convolutional (Humphrey and Bello, 2012; Korzeniowski and Widmer, 2016a) or recurrent (Boulanger-Lewandowski, Bengio, and Vincent, 2013; Wu, Feng, and Li, 2017).

Finally, *post-filtering* is applied to smooth out the classified time frames. This is usually based on a study of the transition probabilities between chords by a Hidden Markov Model (HMM) optimized with the Viterbi algorithm (Lou, 1995) or with Conditional Random Fields (Lafferty, McCallum, and Pereira, 2001).

9.3 INTRODUCING MUSICAL KNOWLEDGE THROUGH THE LEARNING PROCESS

In this section, we aim to target the gap between rule-based and statistical models by introducing musical information directly in the training process of statistical models. To do so, we propose to use prior knowledge underlying the labeling alphabet and to account for the inherent relationships between chords directly inside the loss function of learning methods. Due to the complexity of the ACE task and the wealth of models available, we choose to rely on a single Convolutional Neural Network (CNN) architecture, that provides the current best results in ACE (McFee and Bello, 2017). First, we study the impact of chord alphabets and their relationships by using a specific hierarchy of alphabets (see Section 8.1). We show that some of the reductions performed by previous research might be inadequate for learning algorithms. We also show that relying on more finely defined and extensive alphabets allow to grasp more interesting insights on the errors made by ACE systems, even though their accuracy is only marginally better or worse. Then, we introduce two novel chords distances based on musical relationships found in the *Tonnetz-space* or directly between chord components through their categorical differences. These distances can directly be used as loss functions for learning algorithms. We show that these new loss functions improves current ACE results even without relying on any pre- or post-filtering. Finally, we perform an extensive analysis of our approaches in order to extract insights on the methodology required for ACE. To do so, we use specifically-tailored analyzer that focuses on the functional relations between chords to distinguish "strong" and "weak" errors (see Section 8.2).

9.3.1 *Our proposal*

9.3.1.1 *Definition of chord distances*

In previous CNN classification systems, the model does not take into account the distance between the output classes by considering the nature of each class. This distance D_0 which we called categorical distance is binary

$$D_0(chord_1, chord_2) = \begin{cases} 0 & \text{if } chord_1 = chord_2 \\ 1 & \text{if } chord_1 \neq chord_2 \end{cases} \tag{9.1}$$

However, we want here to include in our model some relationships between chords. For instance, a *C:maj7* is nearer to a *A:min7* than a *C#:maj7*. Therefore, we propose more refined distances to be used as loss functions through the representation of chords in an harmonic space or in a pitch space to describe more precisely the chord labels.

TONNETZ DISTANCE The *Tonnetz-space* is a geometric representation of the tonal space based on harmonic relationships between chords (Cohn, 1997) (see Figure 4.2 and Section 4.1.1 for details on Tonnetz). Representing chords in this space have already showed promising results for classification on the A_0 alphabet (Humphrey, Cho, and Bello, 2012). We choose to work in the Tonnetz-space generated by three transformations applied to major and minor chords (reduction on alphabet A_0) and changing only one of the three notes of the triad. The R, P, L transformations respectively exchange a chord for its *relative* (relative major / minor), *parallel* (same root but major instead of minor or inversely), *leading-tone exchange* (in a major triad the root moves down by a semitone, in a minor triad the fifth moves up by a semitone).

To calculate the distance between two chords, we iterate over all the possible paths in the Tonnetz-space to find the least expensive transformation path between them, applying the same cost for each transformation. Furthermore, an extra cost is added if the chords have been reduced beforehand in order to fit the alphabet A_0, and give a cost of zero if the two chords share exactly the same pitches. We note C^∞ the set of the costs of all the possible paths from $chord_1$ to $chord_2$ using a combination of the R, P, L. Then, our distance D_1 is defined by:

$$D_1(chord_1, chord_2) = min(C^\infty) \tag{9.2}$$

EUCLIDEAN DISTANCE ON PITCH CLASS VECTORS In some works, pitch class vectors are used as an intermediate representation for ACE tasks (Lee and Slaney, 2006). Here, we use these pitch class profiles to calculate the distances between chords according to their harmonic content.

Each chord from the dictionary is associated to a a 12-dimensional binary pitch vector with 1 if the pitch is present in the chord and 0 otherwise (for instance

C:maj7 becomes $(1,0,0,0,1,0,0,1,0,0,0,1)$). The distance between two chords is defined as the Euclidean distance between the two binary pitch vectors.

$$D_2(chord_1, chord_2) = \sqrt{\sum_{i=0}^{11}(chord_1^i - chord_2^i)^2} \tag{9.3}$$

Hence, this distance allows to account for the number of pitches that are shared by two chords.

The D_0, D_1 or D_2 distance is used as loss function for training the CNN classification model.

9.3.1.2 *Introducing the relations between chords*

Based on our three alphabets A_0, A_1, A_2 with controlled amount of chord qualities and hierarchical relationships, we reduce the original labels before the training to fit one of these alphabet.

In parallel, one of three distances D_0, D_1, D_2 are introduced during the training through the loss function that will be minimized with gradient descent. These loss functions can take different forms but are always defined by a comparison between ground truth annotations y_{true} and the corresponding output y_{pred}. The ground truth annotation y_{true} is a one-hot vector where each bin corresponds to a chord in a chosen alphabet A_i. The output y_{pred} is a vector of probabilities over all the chords in the same alphabet A_i. For the distance D_0, the loss function only takes into account the probability of the corresponding bin in y_{true}.

However, for our proposed distance, we introduce a similarity matrix M that associates each couple of chords to a similarity ratio.

$$M_{i,j} = \frac{1}{D_k(chord_i, chord_j) + C} \tag{9.4}$$

C is an arbitrary constant to avoid division by zero. The matrix M is symmetric and we normalize it with its maximum value to obtain $\bar{M}$. Afterwards, we define a new y_{true} which is the matrix multiplication of the old y_{true} and the normalized matrix $\bar{M}$.

$$y_{true} = y_{true}\bar{M} \tag{9.5}$$

The loss function for D_1 and D_2 is now a weighted multi-label classification between the new y_{true} and y_{pred}.

9.3.2 *Experiments*

9.3.2.1 *Dataset*

We perform our experiments on the *Beatles* dataset as it provides the highest confidence regarding the ground truth annotations (Harte, 2010b). This dataset is

composed by 180 songs annotated by hand. For each song, we compute the CQT by using a window with a maximum size of 4096 samples and a hop size of 2048 for a sampling rate equals to 44.1 kHz. The transform is mapped to a scale of 3 bins per semi-tone over 6 octaves ranging from C1 to C7. We augment the available data by performing all transpositions from -6 to +6 semi-tones and modifying the labels accordingly. Finally, to evaluate our models, we split the data into a training (60%), validation (20%) and test (20%) sets.

9.3.2.2 *Models*

We use the same CNN model for all test configurations, but change the size of the last layer to fit the size of the selected chord alphabet. We apply a batch normalization and a Gaussian noise addition on the inputs layer. The architecture of the CNN consists of three convolutional layers followed by two fully-connected layers. The architecture is very similar to the first CNN that has been proposed for the ACE task (Humphrey and Bello, 2012). However, we add dropout between each convolution layer to prevent over-fitting.

For the training, we use the ADAM optimizer with a learning rate of 2×10^{-5} for a total of 1000 epochs. We reduce the learning rate if the validation loss has not improved during 50 iterations. Early stopping is applied if the validation loss has not improved during 200 iterations and keep the model with the best validation accuracy. For each configuration we perform a 5-cross validation, by repeating a random split of the dataset.

9.3.3 *Results*

The aim of this chapter is to study the impact of inherent musical relationships carried by parameters described in the previous section rather than obtaining the best classification scores. Therefore, we do not use pre- or post-filtering methods and analyze our results through two evaluators: the *mireval* library (Raffel et al., 2014), and the Python *ACE Analyzer* (described in Section 8.2) in order to reveal the musical meaning of classification errors and, therefore, understand their qualities.

9.3.3.1 *Quantitative analysis: MIREX evaluation*

Regarding the MIREX evaluation, the efficiency of ACE models is assessed through classification scores over different alphabets (Raffel et al., 2014). The MIREX alphabets for evaluation have a gradation of complexity from Major/Minor to Tetrads. In our case, for the evaluation on a specific alphabet, we apply a reduction from our training alphabet A_i to the MIREX evaluation alphabet. Here, we evaluate on three alphabet : Major/Minor, Sevenths, and Tetrads. These alphabets correspond roughly to our three reduction alphabets (Major/Minor $\sim A_0$, Sevenths $\sim A_1$, Tetrads $\sim A_2$).

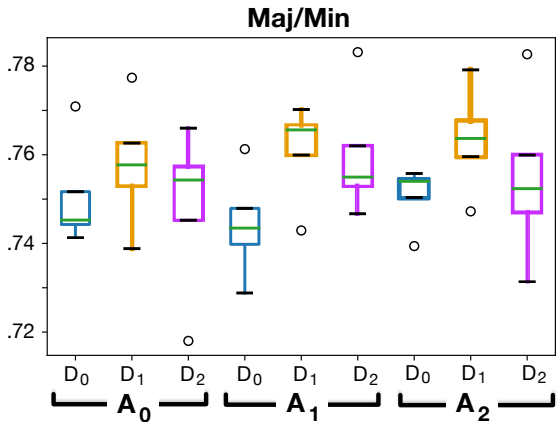

Figure 9.1: Results of the 5-folds: evaluation on MIREX Maj/Min ($\sim$ reduction on A_0).

9.3.3.2 *MIREX Major/minor*

Figure 9.1 depicts the average classification scores over all frames of our test dataset for different distances and alphabets. We can see that the introduction of the D_1 or the D_2 distance improves the classification scores compared to D_0. With these distances, and even without pre- or post-filtering, we obtain classification scores that are superior to that of similar works (75.9% for CNN with post-filtering in (Humphrey and Bello, 2012) versus 76.3% for $A_2 - D_1$). Second, the impact of working first on large alphabets (A_1 and A_2), and then reducing on A_0 for the test is negligible on MIREX Maj/Min (at least from a quantitative point of view, see Section 10.5.4.2).

MIREX SEVENTHS With more complex alphabets, the classification score is lower than for MIREX Maj/Min (see Figure 9.2. This result is not surprising since we observe this behavior on all ACE systems. Moreover, the models give similar results and we can not observe a particular trend between the alphabet reductions or with the different distances. The same result is observed for the evaluation with MIREX tetrads ($\sim$ reduction on A_2). Nonetheless, the MIREX evaluation uses a binary score to compare chords. Because of this approach, the qualities of the classification errors cannot be evaluated.

9.3.3.3 *Qualitative analysis: understanding the errors*

We propose to analyze ACE results from a qualitative point of view with a *functional* approach.

SUBSTITUTION RULES Table 9.1 presents: *Tot.*, the total fraction of errors that can be explained by the whole set of substitution rules we implemented in our chord analyzer (see Section 8.2), and $\subset$ *Maj* and $\subset$ *min*, the errors with inclusions in the correct triad (*e.g. C:maj* instead of *C:maj7*, *C:min7* instead of *C:min*). Table 9.2

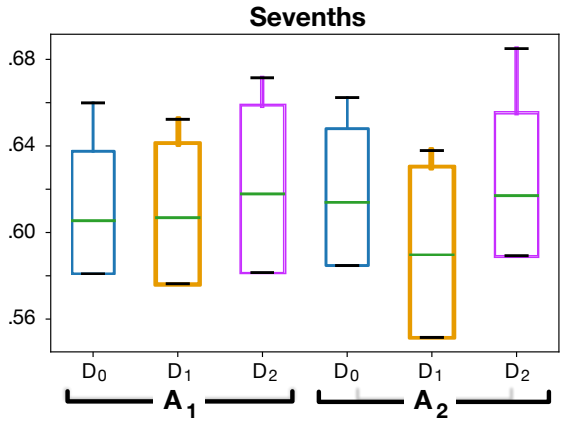

Figure 9.2: Results of the 5-folds: evaluation on MIREX Sevenths ($\sim$ reduction on A_1).

Model	*Tot.*	$\subset$ *Maj*	$\subset$ *min*
A_0-D_0	34.93	–	–
A_0-D_1	36.12	–	–
A_0-D_2	35.37	–	–
A_1-D_0	52.40	23.82	4.37
A_1-D_1	57.67	28.31	5.37
A_1-D_2	55.17	25.70	4.21
A_2-D_0	55.28	26.51	4.29
A_2-D_1	60.47	31.61	6.16
A_2-D_2	55.45	25.74	4.78

Table 9.1: Left: total percentage of errors corresponding to inclusions or chords substitutions rules, right: percentage of errors with inclusion in the correct triad (% of the total number of errors).

presents the percentages of errors corresponding to widely used substitution rules: *rel. m* and *rel. M*, relative minor and major; *T subs. 2*, tonic substitution different from *rel. m* or *rel. M* (*e.g. E:min7* instead or *C:maj7*), and the percentages of errors $m \rightarrow M$ and $M \rightarrow m$, same root but major instead of minor (or inversely) after reduction to triad. The tables only show the categories representing more than 1% of the total number of errors, but other substitutions (that will not be discussed here) were analyzed: tritone substitution, substitute dominant, and equivalence of some *dim7* chords modulo inversions.

First, *Tot.* in Table 9.1 shows that a huge fraction of errors can be explained by usual substitution rules. This percentage can reach 60.47%, which means that numerous classification errors nevertheless give useful indications since they associate chords with equivalent harmonic function. For instance, Table 9.2 shows that a significant amount of errors (up to 10%) are relative major/minor substitutions. Besides, for the three distances, the percentage in *Tot.* (Table 9.1) increases with the

Model	*rel. M*	*rel. m*	*T subs. 2*	*m→M*	*M→m*
A_0-D_0	4.19	5.15	2.37	7.26	12.9
A_0-D_1	4.40	5.20	2.47	7.66	13.4
A_0-D_2	5.13	4.87	2.26	8.89	10.89
A_1-D_0	2.63	3.93	1.53	4.46	8.83
A_1-D_1	3.05	3.36	1.58	5.53	7.52
A_1-D_2	3.02	4.00	1.62	5.84	8.07
A_2-D_0	2.54	4.15	1.51	4.96	8.54
A_2-D_1	2.79	2.97	1.54	5.29	7.46
A_2-D_2	3.11	4.26	1.63	5.34	7.59

Table 9.2: Left: percentage of errors corresponding to usual chords substitutions rules, right: percentage of errors "major instead of minor" or inversely (% of the total number of errors).

size of the alphabet: bigger alphabets imply more errors that preserve harmonic function.

Second, for A_0, A_1, and A_2, using D_1 instead of D_0 increases the fraction of errors attributed to categories in Table 9.2 (and in almost all the configurations when using D_2). This shows a qualitative improvement since all these operations are usually considered as valid chord substitutions. Finally, we can note that a large amount of errors (between 28.19% and 37.77%) corresponds to inclusions in major or minor chords ($\subset$ *Maj* and $\subset$ *min*, Table 9.1) for A_1 and A_2.

HARMONIC DEGREES This section shows an analysis of the results using the second module of the analyzer presented in Section 8.2. First, it determines if the target chord is diatonic (*i.e.* belongs to the harmony of the key), as presented in Table 9.3. If it is the case, the notion of incorrect degree for the predicted chord is relevant and the percentage of errors corresponding to substitutions of degrees is computed (Table 9.4).

A first interesting fact presented by Table 9.3 is that 37.99% to 45.87% of the errors occur when the target chord is non-diatonic. It also shows, for the three alphabets, that using D_1 or D_2 instead of D_0 makes the fraction of errors corresponding to non-diatonic predicted chords decreases (Table 9.3, particularly A_0), which means that the errors are more likely to stay in the right key.

High percentages of errors are associated to errors I∼V (up to 14.04%), I∼IV (up to 17.41%), or IV∼V (up to 4.54%) in Table 9.4. These errors are not usual substitutions, and IV∼V and I∼IV have respectively 0 and 1 pitch in common. In most of the cases, these percentages tend to decrease on alphabets A_1 or A_2 and when using more musical distances (particularly D_2). Conversely, it increases the amount of errors in the right part of Table 9.4 containing usual substitutions:

Model	Non-diat. targ.	Non-diat. pred.
A_0-D_0	37.96	28.41
A_0-D_1	44.39	15.82
A_0-D_2	45.87	17.60
A_1-D_0	38.05	21.26
A_1-D_1	37.94	20.63
A_1-D_2	38.77	20.23
A_2-D_0	37.13	30.01
A_2-D_1	36.99	28.41
A_2-D_2	37.96	28.24

Table 9.3: Errors occurring when the target is non-diatonic (% of the total number of errors), non-diatonic prediction errors (% of the errors on diatonic targets).

Model	I∼IV	I∼V	IV∼V	I∼vi	IV∼ii	I∼iii
A_0-D_0	17.41	14.04	4.54	4.22	5.41	2.13
A_0-D_1	17.02	13.67	3.33	4.08	6.51	3.49
A_0-D_2	16.16	13.60	3.08	5.65	6.25	3.66
A_1-D_0	17.53	13.72	3.67	5.25	4.65	3.5
A_1-D_1	15.88	13.82	3.48	4.95	6.26	3.46
A_1-D_2	16.73	13.45	3.36	4.70	5.75	2.97
A_2-D_0	16.90	13.51	3.68	4.45	5.06	3.32
A_2-D_1	16.81	13.60	3.85	4.57	5.37	3.59
A_2-D_2	16.78	12.96	3.84	5.19	7.01	3.45

Table 9.4: Errors ($> 2\%$) corresponding to degrees substitutions (% of the total number of errors on diatonic targets).

once again the more precise the musical representation is, the more the harmonic functions tend to be correct.

9.3.4 *Conclusion*

In this section, we presented a novel approach taking advantage of musical prior knowledge underlying the labeling alphabets into ACE statistical models. To this end, we applied reductions on different chord alphabets and we used different distances to conduct a quantitative and qualitative analysis of the classification results.

First, we conclude that training the model using distances reflecting the relationships between chords improves the results both quantitatively (classification scores) and qualitatively (in terms of harmonic functions). Second, it appears that working

first on large alphabets and reducing the chords during the test phase does not improve the classification scores but provides an improvement in the quality of the errors.

Finally, ACE could be improved by moving away from its binary classification paradigm. Indeed, MIREX evaluations focus on the nature of chords but a large amount of errors can be explained by inclusions or usual substitution rules. Our evaluation method therefore provides an interesting notion of *musical quality* of the errors, and prompts to adopt a functional approach or even to introduce equivalences. It could be adapted to the ACE problem downstream and upstream: in the classification processes as well as in the methodology for labeling the datasets.

9.4 SEMI-SUPERVISED CHORD ESTIMATION BASED ON VARIATIONAL AUTO-ENCODER

In this section we first aim at using variational learning techniques to chord extraction task. Variational learning have been previously investigated in image processing and have shown promising results. Hence, we focus on unsupervised representation learning algorithms, such as Variational Auto-Encoders (VAE), which aim to model the distribution of input data through latent spaces. By combining discriminative and generative learning, these methods can reconstruct explanatory latent spaces in an unsupervised way. The core idea behind this technique is to use the reconstructed (lower-dimensional) code as a probability distribution from which we can perform sampling (generation of examples following the data distribution). Hence, the lower-dimensional spaces obtained provides a straightforward mechanism for generating new data that follows the original distribution.

In a second time, we target structure detection in these complex audio recordings by extending the previously developed approaches with semi-supervised approaches. By combining a discrimination (supervised) task with a reconstruction (unsupervised) task, these approaches allow to provide extremely high accuracy scores while relying on only few labeled examples. In our case, this alleviate the relative scarcity of available labeled multi-track recordings, as compared to the very large amount of audio data freely available.

9.4.1 *Our proposal*

This proposal has been developed with the help of Yiming Wu under the supervision of Kazuyoshi Yoshii. The contributions of the proposals are extensively developed in our paper (Wu et al., 2020).

In this section, we present a brief overview of the methods, however we redirect interested reader to the aforementioned paper for more details and results.

The main contribution in (Wu et al., 2020) is to draw the potential of deep discriminative model for ACE, by integrating it into the principled statistical inference formalism of the generative approach. A key feature of our VAE approach is the use of a Markov model to define the prior probability of chord label sequences. Thus, it works as a chord language model and prevents frequent frame-level transitions of chord labels, such as in the post-filtering process of usual ACE model (see Section 9.2.2).

As depicted in figure 9.3, the inputs of the classification and recognition models are acoustic feature X. The classification model is aimed to extract a probability of chord sequence from the acoustic features (such as chroma vectors). The recognition model is extracting latent features from the acoustic properties. This specific latent features stand for the deviation from basic chroma patterns (partially induced by the timbre of the different instruments present in the associated input). The chord labels are assumed to follow a first order Markov model, hence we introduce a Markovian prior on latent variables S whose role is to favor self-transitions. Besides,

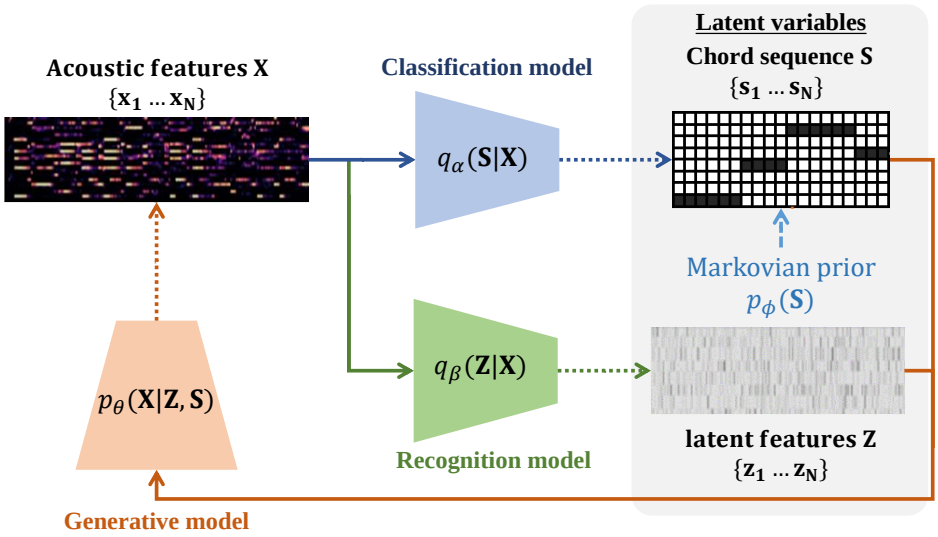

Figure 9.3: The proposed variational autoencoder consisting of a deep generative model of chroma vectors, a deep classification model of chord labels, and a deep recognition model of latent features. Dashed arrows indicate stochastic relations. These three models are trained jointly in a semi-supervised manner by using annotated and non-annotated music signals. Image taken from (Wu et al., 2020).

the latent features Z are assumed to follow a standard Gaussian distribution. The generative and discriminative models can be trained in a supervised or in a semi-supervised ways.

9.4.2 *Experiments*

The used dataset for the experiments is composed of 1200 annotated tracks taken from Isophonics (Harte, 2010a), RWC-MDB-P-2001 (Goto et al., 2002), uspop2002 (Berenzweig et al., 2004)[1], and McGill Billboard dataset (Burgoyne, Wild, and Fujinaga, 2011). We also collected 700 *non-annotated* popular songs composed by Japanese and American artists.

In order to evaluate these methods, five different kinds of models have been trained:

- ACE-SL: the baseline, only composed with the discriminative model trained in a supervised manner.
- VAE-UN-SL: the proposed architecture without the Markovian prior trained in a supervised manner.
- VAE-MR-SL: the proposed architecture with the Markovian prior trained in a supervised manner.

1 The annotations for RWC-MDB-P-2001 and uspop2002 has been provided by the Music and Audio Research Lab at NYU.

- VAE-UN-SSL: the proposed architecture without the Markovian prior trained in a semi-supervised manner.
- VAE-MR-SSL: the proposed architecture with the Markovian prior trained in a semi-supervised manner.

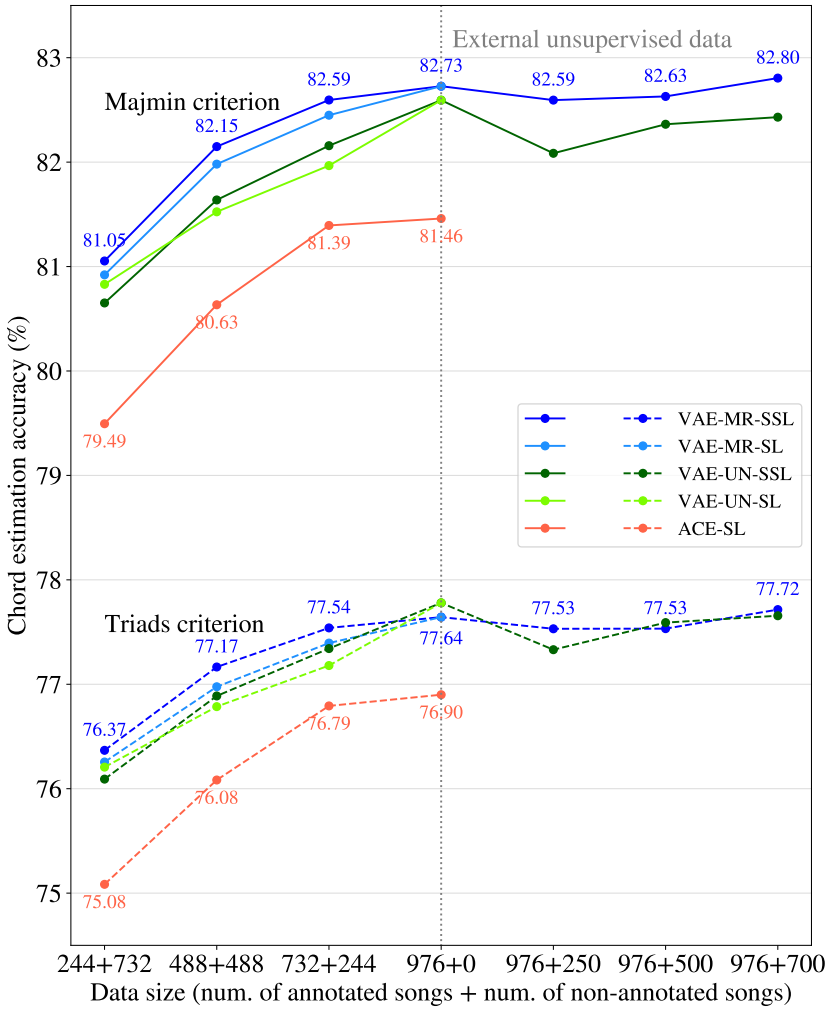

Figure 9.4: The experimental results of the five-fold cross validation using the 1210 annotated songs and the 700 external non-annotated songs. Image taken from (Wu et al., 2020).

9.4.3 *Results and discussion*

All the models have been trained and have been evaluated with a five-fold cross validation. The experimental results regarding the accuracy are presented in Figure 9.4. The first observation is that the VAE-based regularised training improved the accuracy scores of ACE models. Secondly, it can be noticed that the semi-supervised do

not improve dramatically the accuracy score. Indeed, the performance once dropped with few non-annotated songs (right half of Figure 9.4), then barely recovered the score without non-annotated songs. We hypothesize that the large difference in the musical and acoustic characteristics of the annotated and non-annotated songs is considered to have hindered the semi-supervised learning. Nevertheless, performing other tests with a larger non-annotated dataset containing multiple types of acoustic signals could be interesting. Further analyzes have been also realized by observing matrices with the respect of chord types. It appears that while the accuracies on the *maj*, *min*, *aug* and *dim* types were improved by **VAE-MR-SSL**, the accuracies on the other uncommon chord types were significantly degraded. Rare chords tended to be wrongly classified to the *maj* and *min* triads. Here again, a more functional analyze of the chord labels allows to better understand the errors.

10

CHORD SEQUENCE PREDICTION TASKS

10.1 INTRODUCTION

Real-time music improvisation system, such as (Nika et al., 2017), crucially need to be able to predict chords in real time along with a human musician at a long temporal horizon (as evoked in Chapter 2). A long-term horizon is thus necessary since these structures carry more than the step-by-step conformity of the music to a local harmony.

We define the multi-step chord generation as the prediction of successive beat-aligned chords given an initial chord sequence. Most existing systems for multi-step chord sequence generation only target the prediction of the next chord symbol given a sequence, disregarding repeated chords and ignoring timing (Scholz, Vincent, and Bimbot, 2009; Tsushima et al., 2018). However, exact timing is important for our use case, and such models cannot be used without retraining them on sequences including repeated chords. Indeed, since the "harmonic rhythm" (frequency at which the harmony changes) is often 2, 4, or even 8 beats in the music of our study area, such models inherently cannot generalize to real-life scenarios, and can be outperformed by a simple identity function (Crestel and Esling, 2017). Moreover, such predictive models can suffer from error propagation if used to predict more than a single chord at a time. Since we want to use our chord predictor in a real-time improvisation system (Nika, 2016; Nika et al., 2017), the ability to predict coherent long-term sequences is of utmost importance.

In the following, we study the prediction of a sequence of 8 beat-aligned chords given the 8 previous beat-aligned chords. The majority of chord extraction and prediction studies rely on a fixed chord alphabet of 25 elements (major and minor chords for every root note (i.e. c, c#, d, d#, etc.), along with a *no chord* symbol), whereas some studies perform an exhaustive treatment of every unique set of notes as a different chord (Eigenfeldt and Pasquier, 2010; Paiement, Eck, and Bengio, 2005; Yoshii and Goto, 2011). Here, we investigate the effect of using chord alphabets of various precision described in Section 8.1. In this chapter, we present different techniques and study their impact on the quality of chord sequence predictions. First, we present an aggregation approach, summarizing input chords at different time scales. Then, after having underlined the importance of using downbeat position information in the development of future chord sequence prediction models, we introduce two different approach based on the introduction in the training of high-level metadata (such as key or downbeat information) and new data representations.

10.2 PREVIOUS WORK

Most works in chord sequence prediction focus on chord transitions (eliminating repeated chords), and does not include the duration of the chords. Such models include Hidden Markov Models (HMMs) and N-Gram models (Scholz, Vincent, and Bimbot, 2009; Tsushima et al., 2018; Yoshii and Goto, 2011). Here, we use a 9-gram model, trained at the beat level, as a baseline comparison. HMMs (Eigenfeldt and Pasquier, 2010) have also been used for chord sequence estimation based on the melody or bass line, sometimes by including a duration component. However, they rely on the underlying melody to generate an accompanying harmonic progression, rather than predicting a future chord sequence. Recently, neural models for audio chord sequence estimation have also been proposed, but these similarly rely on the underlying audio signal during estimation (Boulanger-Lewandowski, Bengio, and Vincent, 2013; Korzeniowski and Widmer, 2018).

Long Short-Term Memory (LSTM) networks have shown some promising results in chord sequence generation. For instance, (Eck and Schmidhuber, 2002) describes an LSTM which can generate a beat-aligned chord sequence along with an associated monophonic melody. Similarly, in (Choi, Fazekas, and Sandler, 2016), a text-based LSTM is used to perform automatic music composition. The authors use different types of Recurrent Neural Networks (RNNs) to generate beat-aligned symbolic chord sequences. They focus on two different approaches, each with the same basic LSTM architecture: a *word-RNN*, which treats each chord as a single symbol, and a *char-RNN*, which treats each character in a chord's text-based transcription as a single symbol (in that case, `A:min` is a sequence of 5 symbols). In this chapter, we re-implemented the same word-RNN model as a baseline for comparison. However, we aim to improve the learning by embedding specific multi-step prediction mechanisms, in order to reduce single-step error propagation.

10.2.1 *Multi-step prediction*

It has been observed that using an LSTM for multi-step prediction can suffer from error propagation, where the model is forced to re-use incorrectly predicted steps (Cheng et al., 2006). Indeed, at inference time, the LSTM cannot rely on the ground-truth sequence and is forced to rely on samples from its previous output distribution. Thus, the predicted sequences gradually diverge as the error propagates and gets amplified at each step of the prediction. Another issue is that the dataset of chord sequences contains a large amount of repeated symbols. Hence, the easiest error minimization for networks would be to approximate the identity function, by always predicting the next symbol as repeating the previous one. In order to mitigate this effect, previous works (Choi, Fazekas, and Sandler, 2016) introduce a diversity parameter that re-weights the LSTM output distribution at each step in order to penalizes redundancies in the generated sequence. Instead, we propose to minimize this repetition, as well as error propagation, by feeding the

LSTM randomly ground and non-ground truth chords during training time using teacher forcing (Williams and Zipser, 1989).

10.3 BASELINE MODELS FOR EVALUATIONS

In order to evaluate our proposed model, we compare it to several state-of-the-art methods for chord predictions. In this section, we briefly introduce these models and the different parameters used for our experiments.

NAIVE BASELINES. We compare our models against two naive baselines: predicting a *random* chord at each step; and predicting the *repetition* of the most recent chord.

N-GRAMS. The N-gram model estimates the probability of a chord occurring given the sequence of the previous $n-1$ chords. Here, we use $n = 9$ (a 9-gram model), and train the model using the Knesser-Ney smoothing (Heafield et al., 2013) approach. For training, we replace the padded N.C. symbols with a single start symbol. Since an n-gram model does not require a validation set, we combine this with the training set for training the n-gram. During decoding, we use a beam search, saving only the top 100 states (each of which contains a sequence of 9 chords and an associated probability) at each step. The probability of a chord at a given step is calculated as the sum of the (normalized) probabilities of the states in the beam at that step which contain that chord as their most recent chord.

MLP-VAN We compare our proposed models to a vanilla architecture, we propose a MLP Encoder Decoder. We observed that adding a bottleneck between the encoder and the decoder slightly improved the results compared to the classical MLP. All encoder and decoder blocks are defined as fully-connected layers with ReLU activation. We performed a grid search to select the most adapted network with a variation on four different parameters: the number of layers ($nb\_l \in [1, 2, 3]$), the size of the hidden layers ($nb\_h \in [100, 200, 500, 1000]$), the size of the bottleneck ($nb\_bn \in [10, 20, 50, 100]$) and the dropout ratio ($r\_do \in [0, 0.2, 0.4, 0.6, 0.8]$). Some of these parameters are depicted on Figure 10.1.

The results of this grid search is detailed in Annex (see Section A.1). Regarding the accuracy scores and the number of parameters of these models, the MLP-Van that we selected is defined by $[nb\_l = 1; nb\_h = 500; nb\_bn = 50, r\_do = 0.4]$

LSTM. We use the sequence to sequence architecture to build our model (see Section 7.3.4.3). Thus, our network is divided into two parts (encoder and decoder). The encoder extracts useful information of the input sequences and gives this hidden representation to the decoder, which generates the output sequence. The encoder transforms the input chord sequence into a latent representation that will be fed to the decoder at each step. We propose to introduce a bottleneck to compress the latent variable between the encoder and the decoder. Besides, we use

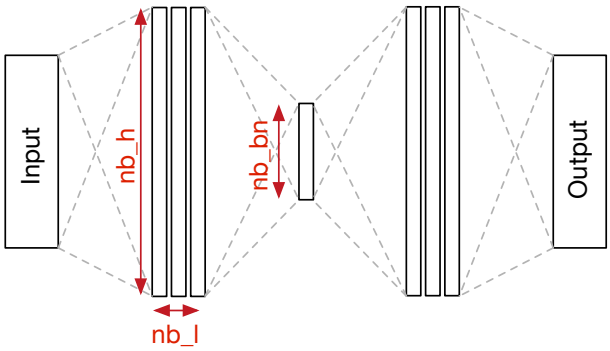

Figure 10.1: Varying parameters for the grid search of an Encoder Decoder neural networks. *nb_l* is number of layers of the encoder and the decoder, *nb_h* is the number of hidden units for the encoder and decoder layers, *nb_bn* is the size of the bottleneck. We also use set different values for the dropout ratio (*r_do*).

an attention mechanism for the decoder (Graves, 2013). Thus, the decoder generates the prediction for the next chords for each step of the predicted sequence. This process of generating in an iterative way the chords of the predicted sequence by using *LSTM* units has been described in previous paper as a *Seq2Seq* model (Sutskever, Vinyals, and Le, 2014). In order to mitigate the problem of the decoder error propagating across time steps of the prediction, we train our model with the teacher forcing algorithm (Williams and Zipser, 1989). Thus, the decoder uses randomly the ground-truth label instead of his own preceding prediction for the multi-step prediction. We decrease the teacher forcing ratio from 0.5 to 0 along the training. Once again, we did a grid search to find correct model parameters (see Table A.5 for results of the grid search). Regarding the accuracy scores and the number of parameters of these models, the LSTM that we selected has a bottleneck and is defined by $[nb\_l = 1; nb\_h = 200; nb\_bn = 50, r\_do = 0.4]$

10.3.1 *Chord sequence prediction loss*

In the case of the temporal classification of chord classes, neural networks models are trained with the help of a temporal binary cross entropy loss. For each chord in predicted sequence we apply a Cross Entropy (CE) loss between the target chord and the predicted chord at this position.

$$L_{prediction} = \sum_{t=0}^{T} CE\left(pred(t), target(t)\right) \tag{10.1}$$

Thus, $L_{prediction}$ is then the sum of the CE for each time-step of the multi-step prediction.

10.4 AGGREGATED MULTI-SCALE ENCODER-DECODER NETWORKS

In this section, we propose a multi-scale model which predicts the next 8 chords directly, eliminating the error propagation issue which inherently exists in single-step prediction models. In order to provide a multi-scale modeling of chord progressions at different levels of granularity, we introduce an aggregation approach, summarizing input chords at different time scales. First, we train separate encoder-decoders to predict the aggregated chords sequences at each of those time scales. Finally, we concatenate the bottleneck layers of each of those pre-trained encoder-decoders and train the multi-scale decoder to predict the non-aggregated chord sequence from the concatenated encodings. This multi-scale design allows our model to capture the higher-level structure of chord sequences, even in the presence of multiple repeated chords.

To evaluate our system, we compare its chord prediction accuracy to a set of the models presented in Section 10.3. We also introduce a musical evaluation process, which uses a musically informed distance metric presented in Section 9.3.1.1 to analyze the predicted chord sequences.

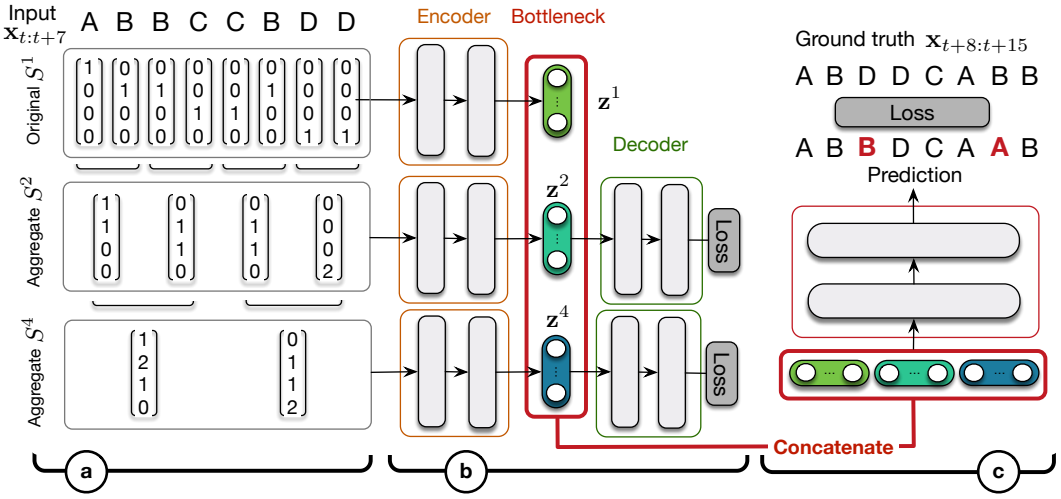

Figure 10.2: The architecture of the aggregated multi-scale encoder-decoder network.

10.4.1 *Proposed Method*

Our proposed approach is based on a sequence-to-sequence architecture with a combination of Encoder-Decoder (ED) networks and an aggregation mechanism for pre-training the models at various time scales. Here, we define *aggregation* as an increase of the temporal span covered by one point of chord information in the input/target sequences (as depicted in Figure 10.2-a). In order to train our whole architecture, we use a two-step training procedure. During the first step, we train separately each network with inputs and targets aggregated at different ratios

(Figure 10.2-b). The second step performs the training of the whole architecture where we concatenate the output of the previously trained encoders (Figure 10.2-c).

10.4.2 *Multi-scale aggregation*

In this work, each chord is represented by a categorical one-hot vector. We compute aggregated inputs and targets by repeatedly increasing the temporal step of each sequence by a factor of two, and computing the sum of the input vectors within each step. This results in three input/output sequence pairs: S^1 and T^1, the original one-hot sequences; S^2 and T^2, the sequences with timestep 2; and S^4 and T^4, the sequences with timestep 4. Formally, for each timestep greater than 1, S^n_i (the ith vector of S^n, 0-indexed) is calculated as shown in Equation 10.2. This aggregation is illustrated in Figure 10.2-a.

$$S^n_i = S^{n/2}_{2i} + S^{n/2}_{2i+1} \tag{10.2}$$

The Multi-Scale ED (MLP-MS) is composed of the same encoder and decoder parts as the vanilla MLP (MLP-Van) in terms of number of layers, hidden units and bottleneck size.

10.4.2.1 *Pre-training networks on aggregated inputs/targets*

First, we train two ED networks: one for each of the aggregated input/target pairs. In order to obtain informative latent spaces we create a bottleneck between the *encoder* and the *decoder* networks, which forces the network to compress the input data. Hence, we first train each ED network independently with aggregated inputs and targets at different ratio. Our loss function for this training is the Mean Squared Error between S^n and T^n. Then, from each encoder, we obtain the latent representation z^n of its input sequence S^n.

10.4.2.2 *Second training of the whole architecture*

For the full system, we take the latent representations of the pre-trained ED networks, and concatenate them with the latent vector of a new ED network whose input is the original sequence S^1. From this concatenated latent vector, we train a decoder through the cross-entropy loss to the target T^1. During this full-system training, the parameters of the pre-trained independent encoders are frozen and we optimize only the parameters of the non-aggregated ED.

10.4.3 *Experiments*

10.4.3.1 *Dataset*

In our experiments, we use the *Realbook* dataset (Choi, Fazekas, and Sandler, 2016), which contains 2,846 jazz songs based on band-in-a-box files[1]. All files come in a *xlab* format and contain time-aligned beat and chord information. We choose to work at the beat level, by processing the xlab files in order to obtain a sequence of one chord per beat for each song. We perform a 5-fold cross-validation by randomly splitting the song files into training (0.6), validation (0.2), and test (0.2) sets with 5 different random seeds for the splits. We report results as the average of the resulting 5 scores. We use all chords sub-sequences of 8 elements throughout the different sets, beginning at the first No-chord symbol (padding this input, and the target being chords 2 to 9), and ending where the target is the last 8 chords of each song.

10.4.3.2 *Alphabet reduction*

The dataset is composed by a total alphabet of 1259 chord labels. This great diversity comes from the precision level of the chosen syntax. Here, we apply a hierarchical reduction of the original alphabet into three smaller alphabets of varying levels of specificity as presented in Section 8.2, containing triads and tetrachords commonly used to write chord progressions.

10.4.4 *Results*

10.4.4.1 *Quantitative analysis*

We trained all models on the three alphabets. In order to evaluate our models, we compute the mean prediction accuracy over the output chord sequences (see Table 10.1). The first two lines represent the accuracy over increasingly complex alphabets for the *random* and *repeat* models. Interestingly, the *repeat* classification score remains rather high, even for the most complex alphabets, which shows how common repeated chords are in our dataset. The last four lines show the accuracy of the more advanced models, where we can observe that the score decreases as the alphabet becomes more complex.

First, we can observe that our MLP-MS obtains the highest results in most cases, outperforming the LSTM in all scenarios. However, the score obtained with a 9-Gram on A_2 is higher than the MLP-MS. We hypothesize that this can be partly explained by the distribution of chord occurrences in our dataset. Many of the chords in A_2 are very rare, and the neural models may simply need more data to perform well on such chords. The 9-gram, on the other hand, uses smoothing to

1 http://bhs.minor9.com/

Model	A_0	A_1	A_2
Random	4.00	1.37	0.59
Repeat	34.2	31.6	31.1
9-Gram	40.4	37.8	**36.9**
MLP-Van	41.8	37.0	35.2
LSTM	41.8	37.3	36.0
MLP-MS	**42.3**	**38.0**	36.5

Table 10.1: Mean prediction accuracy for each method over the different chord alphabets.

Measure	Perplexity			Rank		
Alphabet	A_0	A_1	A_2	A_0	A_1	A_2
9-Gram	7.93	13.3	**15.7**	4.13	8.05	10.3
MLP-Van	7.45	13.5	16.7	3.98	8.08	10.6
LSTM	7.60	13.3	16.0	4.02	7.94	10.2
MLP-MS	**7.40**	**12.9**	**15.7**	**3.94**	**7.74**	**9.99**

Table 10.2: Left side: Perplexity of each model over the test dataset; Right side: Mean of the rank of the target chords in the output probability vectors.

estimate probabilities of rare and unseen chords (though it is likely that it would not continue to improve as much as the neural models, given more data).

We also compare our models in terms of perplexity and rank of the correct target chord in the output probability vector (see Table 10.2). Our proposed model performs better or equal to all other models on all alphabets with these metrics, which are arguably more appropriate for evaluating performance on our task.

10.4.4.2 *Musical analysis*

EUCLIDEAN DISTANCE In order to compare different models, we evaluate errors through a musically-informed distance described in paragraph 9.3.1.1. In this distance, each chord is associated with a binary pitch class vector. Then, we compute the Euclidean distance between the predicted vectors and target chords.

The results, presented in Table 10.3, show two different approaches of using this distance. The left side of the table represents the Euclidean distances between the contribution of all the chords in each model's output probability vector (weighted by their probability) and the target chord vectors. The right side shows the Euclidean distance between the single most likely predicted chord at each step and the target chord. We observe that the MLP-MS always obtains the best results, except on a

Level	Probabilistic			Binary		
Alphabet	A_0	A_1	A_2	A_0	A_1	A_2
9-Gram	1.66	1.61	1.57	1.33	1.30	**1.28**
MLP-Van	1.61	1.61	1.58	1.28	1.31	1.31
LSTM	1.60	1.59	**1.54**	1.29	1.30	1.29
MLP-MS	**1.59**	**1.58**	1.55	**1.28**	**1.29**	**1.28**

Table 10.3: Mean Euclidean distance between (left) contribution of all the chords in the output probability vectors, and (right) chords with the highest probability score in the output vectors.

single case (A_2 and probabilistic distance), where the LSTM performs best by a small margin.

INFLUENCE OF THE DOWNBEAT POSITION IN THE SEQUENCE Figure 10.3 shows the prediction accuracy of the MLP-MS on A_0 at each position of the predicted sequence, depending on the position of the downbeat in the input sequence. Prediction accuracy significantly decreases across each bar line, likely due to bar-length repetition of chords. The improvement of the score for the first position when the downbeat is in position 2 can certainly be explained by the fact that the majority of the RealBook tracks have a binary metric (often 4/4). We also see that the prediction accuracy of chords on downbeats is lower than that of the following chords in the same bar. It can be assumed that this is due to the fact that chords often change on the downbeat, and that the following target chords can sometimes have the same harmonic function as the predicted chords but without being exactly the same. Both trends are observed over all models and alphabets. This underlines the importance of using downbeat position information in the development of future chord sequence prediction models.

10.4.5 *Conclusion*

In this section, we studied the prediction of beat-synchronous chord sequences at a long horizon. We introduced a novel architecture based on the aggregation of multi-scale encoder-decoder networks. We evaluated our model in terms of accuracy, perplexity and rank over the predicted sequence, as well as by relying on musically-informed distances between predicted and target chords.

We showed that our proposed approach provides the best results for simpler chord alphabets in term of accuracy, perplexity, rank and musical evaluations. For the most complex alphabet, existing methods appear to be competitive with our approach and should be considered. Our experiments on the influence of the downbeat position in the input sequence underlines the complexity of predicting

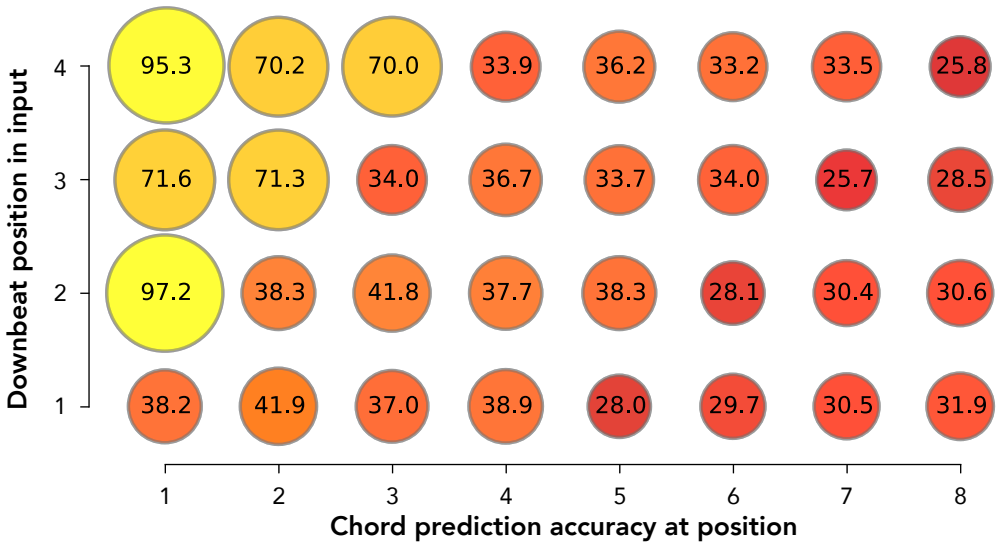

Figure 10.3: Chord prediction accuracy at each position of the output sequence depending on the position of the downbeat in the input sequence.

chords across bar lines. We conclude that it might be useful to investigate the use of the downbeat position in chord sequence prediction systems.

10.5 IMPACT OF USING MULTI-LEVEL OF MUSICAL KNOWLEDGE FOR PREDICTION

In the previous section we pointed out the importance of taking the global musical coherence of the generated sequences into account. This is why we decided to perform multi-step chord sequence prediction over long horizons given a multi-step input sequence of beat-aligned chords by using structural high-level metadata.

To study the relevance of taking advantage of this structural high-level metadata, we trained several instances of MLP Encoder Decoder model using the chord alphabets (defined in Section 8.1) for the input data, and including information on key and downbeat position. We studied the consequences of simplifying the alphabet of chord labels before or after training the model, and we compared the evolution of the prediction accuracy score. After obtaining better classification score by models using metadata, we split the dataset into diatonic and non-diatonic chords and perform additional prediction accuracy evaluation. Finally, we analyze qualitatively the errors by using musical relationships between the predicted and targeted chords.

10.5.1 *Proposed Method*

The information on key is highly correlated to the nature of chords that will be present in a song. For each key (minor or major), we have a specific set of musical notes called the scale. All the chords composed by notes of this scale are called *diatonic chords*. The other chords are called *non-diatonic chords*. The *downbeat position* is defined as the first beat of a measure. Depending of its time signature, a measure is often composed by four beats. In Section 10.4, we have shown that beat-aligned chords tends to be repeated intensively and that chord changes mainly occur on the downbeat position.

In order to analyze the impact of using these two musical information in the prediction of chord sequences, we give information on beat and key as inputs and analyze independently the improvements in a quantitative and in a qualitative way. Furthermore, we propose to define the learning of the key and the downbeat position as an optimization criteria.

10.5.1.1 *Key and downbeat as inputs*

In western music, we can define the dominant key of a song by studying the notes that compose the chords of the track. The key will then defined a scale as a series of notes. The first tonality of this scale will be named the *tonic* and the others will be instantiated following a pattern of musical semi-tone intervals (see Section 6.3). We consider two different scales for each tonic the *major* and the *minor* scale.

Considering all the different existing key we obtain a key alphabet of 25 elements.

$$Dict\_Key = \{N\} \cup \{P \times maj, min\} \tag{10.3}$$

where P represents the 12 pitch classes and N is the No-key symbol.

On the other hand, chord transitions are highly correlated with the beginning of new musical sentences, which are themselves linked to the metric and the downbeat position. The information on downbeat is introduced by numerating each beat in each measure of the input chord sequence.

This information of key or downbeat will be given as input of our neural networks along with the initial chord sequence (see Figure 10.4).

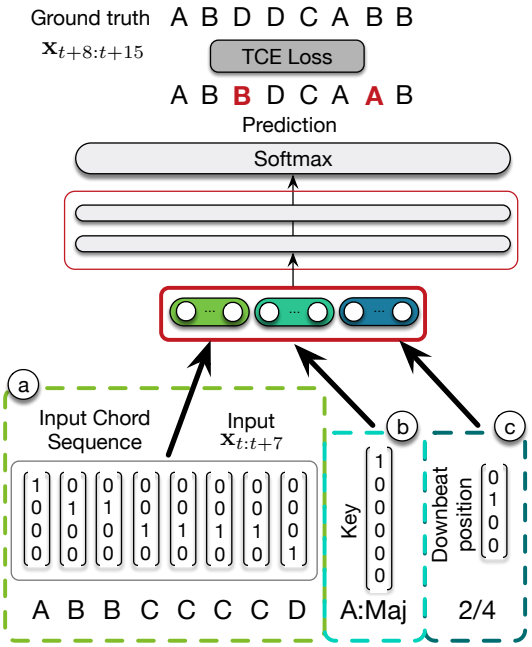

Figure 10.4: The architecture of our proposed models. MLP-Vanilla takes only the part *a* as inputs, MLP-Key takes the parts *a* and *b*, MLP-Beat takes the parts *a* and *c*, MLP-KeyBeat takes the three parts *a*, *b* and *c*.

Thus, in the vanilla MLP-ED, the inputs of the models are only the initial chord sequences (see Figure 10.4, input *a*). In order to study the impact of adding extra information for the prediction of chord sequences, we add two other musical information. For MLP-Key, the inputs of our model is the initial chord sequence and the key (Figure 10.4 part *a* and *b*). For MLP-Beat we use the downbeat position but not the key (Figure 10.4 part *a* and *c*). Finally, for MLP-KeyBeat, we use both metadata as inputs (Figure 10.4 part *a*, *b* and *c*). The architecture of these three models are similar to the MLP-Vanilla, expect for the first layers that will be shaped to fit the size of the inputs.

10.5.1.2 *Learning the key and downbeat information*

We also propose to train the network to recognize the key or the downbeat position of an input chord sequence. Hence, in order to perform the training of our model, the complete loss is defined as the sum of the predictive loss and the additional losses $\mathcal{L}_{downbeat}$ for downbeat and $\mathcal{L}_{key}$ for key detection.

$$\mathcal{L}_{total} = \lambda_p \mathcal{L}_{prediction} + \mathcal{L}_{downbeat} + \mathcal{L}_{key} \tag{10.4}$$

We first train our models in order to improve the prediction of the downbeat and the key, by setting $\lambda_p = 0$. Thus, we freeze the two sub-networks and train the network by introducing the prediction loss on the chord sequence. The overall architecture is depicted on Figure 10.5.

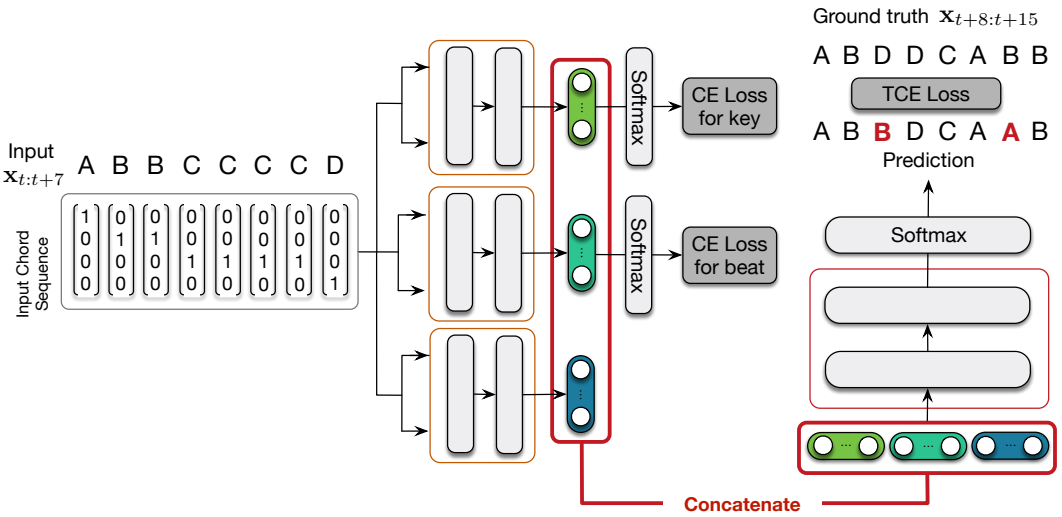

Figure 10.5: The architecture of our proposed models that learns the key and the downbeat position in order to improve the chord sequence predictions.

10.5.2 *Experiments*

DATASET AND CHORD REDUCTIONS In our experiments, we rely on songs taken from the same data set that in Section 10.4 (the *Realbook* dataset (Choi, Fazekas, and Sandler, 2016)) . We also apply the reduction on four different chord alphabets. Three alphabets, A_0, A_1 and A_2, are already described in Section 8.1. As aforementioned in Chapter 6, the 25 chords present in the alphabet A_0 has not the same functional behaviour when they are instantiated within a song. Hence, we propose a new alphabet A_R, that is composed of only 12 classes plus the no-chord ($N.C$) class and that contains for each class a major chord and its relative minor (see Figure 10.6)

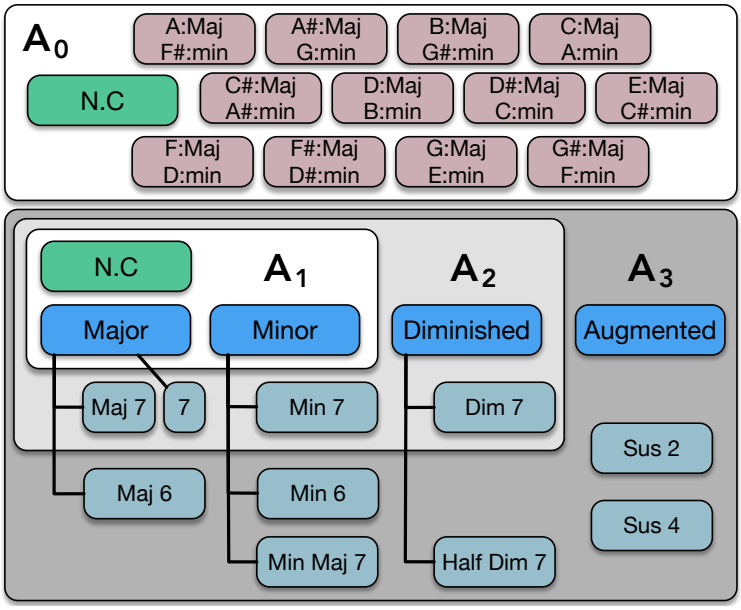

Figure 10.6: The three chord vocabularies A_0, A_1, and A_2 we use in this chapter are defined as increasingly complex sets. The standard triads are shown in turquoise blue. The A_R alphabet contains the 12 families of relative chords. For all the alphabets, $N.C$ stands for No-Chord.

However, in order to clean the dataset we removed all files that contain A_2 chords repeated for more than 8 bars, leading to a total of 78 discarded songs. For all the remaining songs, we pad the beginning of the song with 7 beat-aligned $N.C$ symbols and extract all chord sub-sequences that contain 16 beat aligned chords. Since the dataset is not exactly the same than in Section 10.4, the accuracy score between the two sections are not comparable. Thus, in the following, we present the score of the previous models on this cleaned dataset.

10.5.3 *Models and training*

All of our neural network models are trained with the ADAM optimizer with a decreasing learning rate starting at 10^{-3} and divided by a factor of 2 when the validation accuracy does not decrease for 10 epochs. We follow a 5-fold cross-validation training procedure, by randomly splitting the dataset into train (0.8), valid (0.1) and test (0.1) sets. In order to evaluate the efficiency of our proposed methods, we also train state of the art baseline models for chord prediction describe in Section 10.3.

10.5.4 *Results*

10.5.4.1 *Quantitative analysis*

GENERAL CLASSIFICATION ACCURACY Our first experiment is the evaluation of the different models on the three alphabets A_0, A_1 and A_2. We compute the mean prediction accuracy over the output chord sequence (see Table 10.4). As a baseline, we present on the first two lines the classification score for the *random* and *repeat* models. We note that the *repeat* models obtain a rather high accuracy even for the most complex alphabets. This can probably be explained by the fact that in this type of corpus, the underlying harmony often varies every 2 or 4 beats.

For all the models, we observe that the accuracy decreases when using more complex chord alphabets. On the quantitative side, we see that the introductions of key and/or downbeat position improve the accuracy score of the MLP on every alphabet, and that the highest accuracy is reached when both are introduced in the MLP model. As detailed in the previous section, the MLP-MS models shows globally an improvement comparing to MLP-Van models.

Model	A_0	A_1	A_2
Random	4.00	1.37	0.59
Repeat	32.68±.75	29.86±.46	29.37±.52
9-Gram	37.87±.64	34.45±.65	33.76±.53
LSTM	40.82±.63	37.22±.44	35.29±.43
MLP-Van	41.49±.63	37.04±.40	35.47±.62
MLP-MS	41.45±.70	37.32±.32	35.91±.52
MLP-K	43.85±.62	37.72±.32	36.06±.57
MLP-B	42.06±.61	38.15±.37	36.65±.55
MLP-KB	44.86±.66	39.33±.23	37.87±.70
MLP-Aug	41.38±.72	37.50±.33	36.10±.56

Table 10.4: Mean prediction accuracy for each method over the chord alphabets A_0, A_1 and A_2

The model that learn also the information of key and downbeat position, denoted MLP-Aug, shows better accuracy scores that MLP-Van or MLP-MS.

ACCURACIES FOR THE LEARNING OF THE KEY AND THE DOWNBEAT We observe on Table 10.5 that the more the alphabet is complex the best accurate the extraction will be for the key and the downbeat position. Indeed, adding chords with enriched notes will obviously helps the network to recognized the key. Some of these chords could also be passing chords and will be located at specific positions within the bar, helping the network to determine the downbeat position.

	A_0	A_1	A_2
Key	53.57±1.73	59.89±3.35	61.12±1.97
Downbeat position	88.15±.60	89.56±.75	89.76±.82

Table 10.5: Mean prediction accuracy for the learning of the key and the downbeat information on A_0, A_1 and A_2

REDUCING THE OUTPUTS ON GIVEN ALPHABETS We introduced the alphabet A_R, grouping relative chords together, as a first step towards the introduction of harmonic equivalence classes in the task of predicting multi-step chords. The main idea is here to observe whether the loss of precision could in return bring an increase in the accuracy of the harmonic role of a chord.

The scores obtained for A_R (first column in Table 10.6) are higher than that of A_0, A_1, and A_2 (Table 10.4). This can be simply explained by its lower number of chord classes.

The columns 2 to 4 in Table 10.6 show that training the models on more complex alphabets, and reducing the outputs on A_R in a second time always lead to better results. Similarly, the scores increase if we reduce complex alphabets to A_0 for the models MLP-Vanilla and MLP-Beat. Besides, the models using the information on key for the prediction obtain worse or similar results in the case of a reduction from complex alphabets to A_0.

Model	A_R	$A_0 \to A_R$	$A_1 \to A_R$	$A_2 \to A_R$	A_0	$A_1 \to A_0$	$A_2 \to A_0$
MLP-Van	43.57±.62	44.87±.61	45.53±.53	44.86±.64	41.49±.63	42.32±.54	41.66±.64
MLP-MS	43.59±.56	44.85±.65	45.90±.45	45.30±.56	41.45±.70	42.69±.48	42.08±055
MLP-K	45.91±.67	47.34±.58	46.30±.60	45.52±.64	43.85±.62	43.06±.63	42.28±.67
MLP-B	44.43±.63	45.51±.59	46.88±.52	46.35±.61	42.06±.61	43.69±.51	43.12±.62
MLP-KB	46.92±.57	48.37±.64	48.11±.57	47.55±.68	44.86±.66	44.83±.55	44.31±.71
MLP-Aug	43.60±.74	44.77±.66	46.19±.46	45.57±.54	41.38±.72	42.96±.49	42.36±.52

Table 10.6: Mean prediction accuracy for each method on alphabet A_R (left) and A_0 (right), with different alphabet reductions.

DISTINGUISHING BETWEEN DIATONIC AND NON-DIATONIC TARGETS Table 10.7 shows the accuracy scores and the number of correct predictions depending of the nature of the targeted chords for the different models over the different alphabets. We observe that adding information on downbeat position (MLP-Beat) improves the prediction scores for both diatonic and non-diatonic chords. The model taking the key into account (MLP-Key) shows an important improvement for the diatonic chords, nevertheless we observe a decrease of the classification score for the non-diatonic targets. Finally, the model using both key and downbeat position (MLP-KB) presents a better score for the diatonic targets but loses accuracy

on the non-diatonic targets. Even if the classification scores of MLP-KB are slightly better for every chord alphabet compared to MLP-Key, we note that the classification score for the non-diatonic targets are better if we do not take the key into account. The model that learns key and downbeat information, MLP-Aug, obtain better results than MLP-Van on every alphabets for diatonic and non-diatonic chord targets. Thus, learning the key instead of using it does not decrease the accuracy score for the non-diatonic chord targets.

Alphabet	A_0		A_1		A_2	
Chord tgt.	D. (1.44M)	N-D. (490K)	D. (1.37M)	N-D. (559K)	D. (1.33M)	N-D. (602K)
MLP-Van	45.88(660K)	28.58(139K)	41.96(574K)	24.98(139K)	40.80(541K)	23.70(142K)
MLP-MS	45.78(659K)	28.73(140K)	42.27(579K)	25.20(140K)	41.25(547K)	24.10(145K)
MLP-K	50.53(727K)	24.21(118K)	44.29(606K)	21.63(121K)	42.95(570K)	20.83(125K)
MLP-B	46.44(668K)	29.21(143K)	43.11(590K)	25.99(145K)	42.08(558K)	24.63(148K)
MLP-KB	51.48(741K)	25.39(124K)	45.89(628K)	23.26(130K)	44.95(596K)	22.22(133K)
MLP-Aug	45.91(660K)	28.05(137K)	42.55(582K)	25.14(140K)	41.68(553K)	23.78(143K)

Table 10.7: Mean prediction accuracy for each method over the different chord alphabets on diatonic (D.) and non-diatonic (N-D.) chord targets. The amount of associated items in the test dataset is in parenthesis.

.

These first observations lead to the conclusion that the use of metadata could be envisaged in different ways depending on the framework of use of a chord sequence prediction module, as well as on the repertoire and the corpus used. For example, in the context of popular music, the accuracy of diatonic chords, and therefore the use of the metadata about key, could be privileged. Conversely, in a jazz context where modulations, borrowing chords from other keys, and chromatisms are more frequent, one could prefer to gain in precision on non-diatonic chords thanks to the metadata about downbeat position only, even if it means losing some accuracy on diatonic chords. This conclusion is strengthened if, as studied below, the loss in accuracy is compensated by an improvement in the quality of classification errors in terms of harmonic function.

10.5.4.2 *Qualitative analysis: understanding the errors*

In this section, we propose to analyze the chord sequence predictions from a qualitative point of view. Our goal is twofold: to understand what causes the errors in the first place, and to distinguish "weak" from "strong" errors with a *functional* approach.

In tonal music, the *harmonic functions* qualify the roles and the tonal significances of chords, and the possible equivalences between them within a sequence (Rehding, 2003; Schoenberg and Stein, 1969). To do so, we use the *ACE Analyzer* library of Section 8.2 that includes two modules discovering some formal musical relationships between the target chords and the predicted chords.

SUBSTITUTION RULES AND FUNCTIONAL EQUIVALENCES We first study the errors corresponding to usual *chord substitutions* rules: using a chord in place of another within a chord progression (usually substituted chords have two pitches in common with the triad that they are replacing); and *hierarchical relationships*: prediction errors included in the correct triads (*e.g. C:maj* instead of *C:maj7*, *C:min7* instead of *C:min*). In Table 10.8, we present the percentage of errors explained by hierarchical relationships (columns $\subset$ *Maj* and $\subset$ *min*). The three other columns of the right part show the percentages of errors corresponding to widely used substitution rules: *rel. m* and *rel. M*, relative minor and major; *T subs. 2*, tonic substitution different from *rel. m* or *rel. M* (*e.g. E:min7* instead or *C:maj7*). Other substitutions (that are not discussed here) were analyzed: same root but major instead of minor or conversely, tritone substitution, substitute dominant, and equivalence of *dim7* chords modulo inversions. In the next sections, we call "explainable" the mispredicted chords that can be related to the target chords through one these usual substitutions.

Model	$\subset$*Maj*	$\subset$*min*	*rel.M*	*rel.m*	*T s.2*
A_0-MLP-Van	_	_	4.14	1.64	1.14
A_0-MLP-MS	_	_	4.14	1.68	1.17
A_0-MLP-K	_	_	4.49	1.72	1.17
A_0-MLP-B	_	_	4.33	1.62	1.12
A_0-MLP-KB	_	_	4.67	1.69	1.12
A_0-MLP-Aug	_	_	4.18	1.61	1.12
A_1-MLP-Van	6.96	0.98	3.52	1.58	1.32
A_1-MLP-MS	7.07	1.02	3.44	1.68	1.31
A_1-MLP-K	7.04	1.12	3.58	1.62	1.29
A_1-MLP-B	7.45	1.07	3.58	1.57	1.25
A_1-MLP-KB	7.46	1.17	3.72	1.68	1.29
A_1-MLP-Aug	7.22	1.06	3.47	1.68	1.32
A_2-MLP-Van	7.69	1.14	3.34	1.61	1.23
A_2-MLP-MS	7.72	1.19	3.3	1.71	1.29
A_2-MLP-K	7.88	1.25	3.42	1.64	1.30
A_2-MLP-B	8.30	1.19	3.42	1.68	1.20
A_2-MLP-KB	8.40	1.36	3.53	1.67	1.29
A_2-MLP-Aug	7.96	1.17	3.27	1.76	1.29

Table 10.8: Percentage of errors corresponding to inclusions or usual chords substitutions rules.

We observe that introducing metadata increases the fraction of errors attributed to the categories presented in Table 10.8. This shows a qualitative improvement since all these operations are considered as valid chord substitutions.

Alphabet	A_0		A_1		A_2	
Chord tgt.	D. (1.44M)	N-D. (490K)	D. (1.37M)	N-D. (559K)	D. (1.33M)	N-D. (602K)
MLP-Van *expl.*	14.87(115K)	30.74(107K)	26.40(209K)	32.79(137K)	28.42(223K)	31.02(142K)
Tot. (OK or expl.)	52.70(776K)	37.36(247K)	53.04(784K)	33.17(277K)	52.40(765K)	31.05(285K)
MLP-MS *expl.*	14.86(116K)	30.61(106K)	26.59(210K)	32.79(137K)	28.73(224K)	30.96(141K)
Tot. (OK or expl.)	52.58(775K)	37.52(247K)	53.51(789K)	33.46(278K)	53.10(771K)	31.56(286K)
MLP-K *expl.*	13.79(98K)	30.76(114K)	26.14(199K)	32.17(141K)	28.58(216K)	30.47(145K)
Tot. (OK or expl.)	57.5(825K)	31.66(232K)	55.87(806K)	28.59(262K)	55.23(786K)	27.17(270K)
MLP-B *expl.*	15.05(116K)	31.12(107K)	27.06(210K)	33.17(137K)	29.51(226K)	31.66(143K)
Tot. (OK or expl.)	53.43(784K)	38.30(250K)	54.78(801K)	34.62(282K)	54.50(785K)	32.43(291K)
MLP-KB *expl.*	13.84(96K)	30.59(111K)	27.11(200K)	32.39(139K)	29.55(215K)	30.83(144K)
Tot. (OK or expl.)	58.61(837K)	33.16(236K)	58.33(829K)	30.79(269K)	58.24(812K)	29.07(277K)
MLP-Aug *expl.*	14.62(113K)	30.78(108K)	26.61(209K)	32.71(137K)	28.85(223K)	31.23(143K)
Tot. (OK or expl.)	52.62(774K)	36.69(245K)	53.87(792K)	33.36(277K)	53.70(776K)	31.21(286K)

Table 10.9: For each model: the first line corresponds to the percentage of errors explained by substitution on diatonic (D.) and non-diatonic (N-D.) chord targets, the second line is the total percentage of correct prediction and explainable errors. The amount of associated items in the test dataset is in parenthesis.

DISTINGUISHING BETWEEN DIATONIC AND NON-DIATONIC TARGETS In addition to the harmonic functions, we analysed the different ways in which the subsets of diatonic and non-diatonic targets were affected by the use of musical metadata. Table 10.9 presents the amount of "explainable" errors (as described in paragraph 10.5.4.2) depending on this criterion.

The first line of the table shows the cumulative percentage of explainable errors for the diatonic (D.) and the non-diatonic (N-D.) chords. For all the models we observe more explainable errors when the alphabets are getting more complex. The lines *expl.* show that using information on key and downbeat makes the amount of explainable errors increase for the diatonic chords; the information on downbeat improves the results for the non-diatonic chords; the information on key does not improve the prediction for the non-diatonic chords. Finally, the lines *Tot.* in Table 10.9 present the sum of the correct predictions and the explainable errors for each model. We see here that extending the study to relevant (and not only correct) predictions, the conclusions of the quantitative study in the paragraph 10.5.4.1 are confirmed: information on key benefits to diatonic chords to the disadvantage of non-diatonic chords, and information on downbeat benefits to non-diatonic chords.

FOCUS ON DIATONIC TARGETS On left part of Table 10.10, we observe that the non-diatonic predictions for diatonic targets (*N-D.p.*) tend to decrease when using the information on key (MLP-Key and MLP-KB) or when we learned it (MLP-Aug), which corresponds to more correct harmonic functions when the target is diatonic.

Furthermore, the ratios of errors corresponding on to usual degree substitutions augment most of the time when we are using the key information (see right part of Table 10.10). Conversely, we see that adding the information on downbeat does not change significantly the harmonic function of the errors comparing to the vanilla MLP model.

Model	*N-D.p.*	I∼IV	I∼V	IV∼V	I∼vi	IV∼ii	I∼iii
A_0-MLP-Van	22.53	15.54	18.77	4.58	4.53	2.61	2.93
A_0-MLP-MS	22.28	15.33	18.66	4.61	4.54	2.68	2.98
A_0-MLP-K	13.09	16.24	22.96	2.63	6.85	2.07	4.15
A_0-MLP-B	22.3	15.67	18.19	4.77	4.68	2.79	3.04
A_0-MLP-KB	12.75	16.61	22.1	2.87	6.96	2.23	4.36
A_0-MLP-Aug	21.31	15.80	19.04	4.54	4.66	2.60	3.01
A_1-MLP-Van	24.89	13.60	19.46	3.55	4.63	2.08	3.14
A_1-MLP-MS	24.37	13.5	19.44	3.57	4.66	2.15	3.15
A_1-MLP-K	16.59	14.52	22.52	2.79	5.86	1.89	3.66
A_1-MLP-B	23.90	13.66	19.42	3.66	4.83	2.15	3.28
A_1-MLP-KB	16.37	14.29	21.48	2.86	6.05	2.10	3.99
A_1-MLP-Aug	22.81	13.86	19.57	3.48	4.84	2.15	3.24
A_2-MLP-Van	26.08	13.45	19.91	3.5	4.49	2.10	2.90
A_2-MLP-MS	25.84	13.47	19.48	3.61	4.52	2.19	2.96
A_2-MLP-K	19.11	13.75	22.29	2.97	5.50	2.02	3.44
A_2-MLP-B	25.24	13.59	19.38	3.6	4.71	2.19	3.06
A_2-MLP-KB	17.02	14.10	22.02	2.99	5.79	2.13	3.72
A_2-MLP-Aug	24.49	13.61	19.69	3.44	4.60	2.22	3.01

Table 10.10: Left: percentage of Non-Diatonic prediction (N-D.p.) over the classification errors when the target is diatonic, right: errors ($> 2\%$) corresponding to degrees substitutions (% of the total number of errors on diatonic targets).

FOCUS ON NON-DIATONIC TARGETS Finally, we carried a last qualitative study focusing on the evolution of the non-diatonic chords identification. We realised this analysis on the major songs only since they are most represented in the dataset, and using the alphabet of triads A_0 to represent the chords. We extracted the non-diatonic chords from each song to study the corresponding predictions resulting from each of the sub-models. The results are shown in Table 10.11: all these observations are aggregated thanks to a representation on a relative scale starting from the tonic of the key (0) and graduated in semitones (in the key of *C Major*, 0-min = *C:min*, 1-Maj = *C#:Maj*, etc.).

The table shows the total amount of instances of each non-diatonic chord class in the corpus compared to the amount of correct predictions of the models.

	Tot.	MLP-Van	MLP-MS	MLP-K	MLP-B	MLP-KB	MLP-Aug
0-min	15K	27.99	28.63	30.21	28.65	31.07	27.56
1-Maj	18K	26.81	27.47	23.67	28.43	24.88	26.61
1-min	2K	12.48	13.57	10.18	13.37	10.33	13.27
2-Maj	**90K**	36.77	36.28	30.74	**38.13**	32.16	36.02
3-Maj	21K	35.29	35.13	30.14	35.59	31.43	35.59
3-min	6K	11.02	10.89	9.41	11.11	9.19	11.14
4-Maj	**48K**	25.97	26.17	22.53	**26.36**	24.43	24.66
5-min	33K	18.89	20.0	16.16	19.74	16.91	18.95
6-Maj	8K	29.21	29.79	17.93	29.57	25.0	29.59
6-min	6K	20.72	25.02	15.7	23.05	16.59	18.65
7-min	26K	15.79	15.6	12.36	15.46	12.97	15.35
8-Maj	23K	27.51	27.35	27.3	27.69	28.21	27.43
8-min	2K	14.04	13.62	13.52	14.28	13.85	13.48
9-Maj	**75K**	29.41	29.99	26.41	**30.4**	27.78	28.74
10-Maj	44K	32.75	32.54	25.14	33.06	25.11	32.3
10-min	7K	19.75	19.96	14.94	20.36	15.71	19.4
11-Maj	19K	21.69	21.15	19.53	21.49	20.46	21.09
11-min	7K	16.35	16.67	12.31	15.83	13.7	15.11

Table 10.11: For each class of non-diatonic chord (relative scale starting from the tonic of the key): total amount of instances in the corpus (Tot.) and percentage of correct predictions for each of the models.

First of all, we notice that the non-diatonic chord classes most represented in the data correspond to well-known passage chords, secondary dominants frequently used to add color to otherwise purely-diatonic chord progressions or to emphasize the transition towards a local tonality or key. Indeed, the class 2:Maj corresponds to the secondary dominant *V/V* (fifth degree of the fifth degree of the key), 9:Maj to *V/ii*, 4:Maj to *V/vi*, and finally 10:Maj corresponds to *bVII* which is frequently used as a substitution of the fifth degree *V*. Our results show that taking the information on downbeat position into account (MLP-Beat) improves the prediction accuracy score on the most represented classes (2:Maj, 9:Maj, 4:Maj). We can assume that this can be explained by the function of these transition chords. Indeed, as passing chords, they are often used at the same positions in turnarounds, cadences, and other classical sequences, which may explain why the information on downbeat helps identify them.

Finally, the models using the metadata about key (MLP-Aug, MLP-Key and MLP-KB) present a lower amount of correct predicted chords on these same classes. This is in line with the results presented earlier, i.e. the deterioration in the quality of errors concerning non-diatonic chords when this information is used.

10.5.5 *Conclusion*

In this section, we studied the introduction of musical metadata in the learning of multi-step chord sequence predictive models. To this end, we compared different state of the art models and choose the most accurate one to run our analysis on different chord alphabets. First, we concluded that using information on key globally improves the classifications score as well as the quality of the errors in term of harmonic functions. Secondly, after distinguishing between diatonic and non-diatonic chords, we found that using the metadata about key only improves the classification score of diatonic chords. In parallel, we observed that introducing the information on downbeat improves both the precision of the diatonic and the non-diatonic predicted chords. A finer analysis of the non-diatonic chords revealed that the non-diatonic chords that are the most represented in the data correspond to passing chords and secondary dominants. We showed that the introduction of information on downbeat helps the model to identify these most relevant non-diatonic chords, which can be explained by their usual positions within the cadences, and thus within the measures. Finally, we conclude that introducing the downbeat position in the prediction of multi-step chord sequences always improve the results in a qualitative and quantitative ways. However, the introduction of the key should be considered in the light of the corpus used and of the musical repertoire. Indeed, using the information on key could be privileged in mostly diatonic contexts since it improves the quality of the predicted harmonic function. Conversely, it may not be suitable for repertoires where modulations and non-diatonic chords are frequents if their precise identification is important.

10.6 INTEGRATION OF A TEMPORAL INFORMATION IN THE DATA REPRESENTATION

During this thesis, I had the opportunity to co-supervise an internship of a student from Columbia University (Gupta, Nika, and Carsault, 2020). The aim of this internship was to study new kinds of data representations for the task of chord sequence prediction. Thus, we proposed to integrate the number of repetitions for each chord along with its chord label (e.g. "`C:Maj 3`" means "`C:Maj`" repeated for `3` beats). Our intuition was that by manipulating repeated chords the network could learn usual chord progressions or known musical cadences, and then improve the predictions. Our first attempt was to build a new dictionary based on those introduced by Section 8.2 but with a number of repetitions contained between 1 and a arbitrary maximum number of repetitions R_{max}. Thus the new dictionary for A_0 is defined by:

$$A_{0temporal} = \{N.C\} \cup \{P \times maj, min \times R\} \tag{10.5}$$

where P represents the 12 pitch classes and $R \in [1, ..., R_{max}]$.

Hence, this new dictionary has a length R_{max} times bigger that the initial one. Unfortunately the results do not conclude on a significant improvement in term of classification score. Indeed, we observed an average decrease (by 1 or 2%) of the accuracy score comparing to our baseline (MLP-Van). Nevertheless, we are convinced that new data representations that include repetitions of chords is an interesting way to improve chord sequence predictive systems.

Part III

APPLICATIONS

11

TOWARDS A LISTENING AND A PREDICTION MODULE: APPLICATION TO CO-IMPROVISATION

11.1 GLOBAL ARCHITECTURE OF THE SYSTEM

In this section we combine the chord extraction module presented in Section 9.3 and the prediction module detailed in Section 10.5. The general workflow is depicted on Figure 11.1.

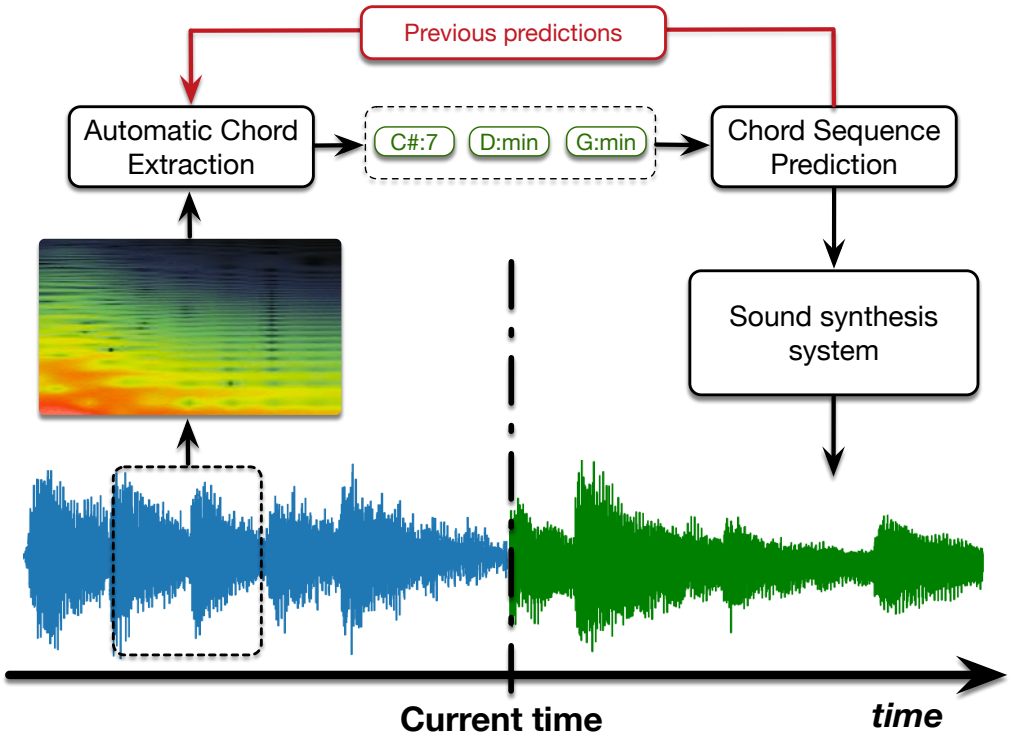

Figure 11.1: General architecture of our listening and predictive prototype. For each beat, a sequence of extracted chord is sent to the chord sequence predictive system. The predicted chord sequence could be sent to a sound synthesis system in order to generate an audio feedback. We also add a retro-active loop in order to take advantage of the previous chord predictions to enhance the local extraction.

Firstly, the acoustical signal is converted into a time-frequency representation. Then we use the ACE system developed in Chapter 9 to extract local chords. This NN model initially handles as input one dataframe composed of 15 successive frames of spectrogram. Thus, the length of an input dataframe is approximately 0.7 second. As we would like to work at beat level, a time step of 0.7 second between

two local chord extraction might be to long. Hence, in order to have an architecture which is tempo independent, each audio section between two consecutive beats will be composed of 15 frames. The 15 frames are computed by taking the mean of each frequency band contribution of the audio section. We end up with an input composed of 15 identical frames of spectrogram.

Secondly, we predict a chord sequence from the locally extracted chords. We use the models developed in Section 10.5 to infer a possible continuation of the chord sequence. Thus, we use the previous extracted chords ($\{chord_{n-MaxStep}, ..., chord_{n-1}\}$, where *MaxStep* is the size of the input sequence for the prediction) to compute the probability vector of the next upcoming chord at step n.

Finally, this predicted sequence of chords is given to a sound synthesis model in order to generate the musical signal.

11.1.1 *Using the temporal prediction to enhance the local prediction*

In order to enhance the local prediction, we propose to use the predictive system (red feedback loop on Figure 11.1). The two systems that we are using output probability vectors. Thus, for the local extraction task, $p_{local}(n)$ is the probability vector output by the ACE system for the dataframe n. In the same way, $p^0_{pred}(n)$ is the probability of the first predicted chord of the sequence at time step n.

$$p_{enhanced}(n) = p_{local}(n) + \alpha * p^0_{pred}(n) \tag{11.1}$$

Therefore, the resulting vector is a linear combination of the probability vector $p_{local}(n)$ and the predictive probability vector of chord at step n, $p^0_{pred}(n)$. In Equation 11.1.1, α is an arbitrary value to fix in order to ponder $p_{enhanced}(n)$ between the local and predictive probability vectors. In our experiments, we choose a value of α around 0.5 in order to give more strength to the local chord estimation.

11.2 IMPLEMENTATION AND INTEGRATION

The implementation of our intelligent listening module (performing real-time extraction of underlying chord labels and prediction of future chord progressions) is realized with a two-side software linked together with an OSC communication. Firstly, the input of the musician is processed and saved with the help of *Max/MSP*. In python, the listening module extracts a chord for each beat. The beat information is given manually or with the help of a metronome. Then, with the help of the previously extracted chords, the continuation of the chord progression is inferred. Secondly, this symbolic chord sequence is sent back to *Max/MSP* where the user visualizes the prediction and the extracted chords.

Finally, this chord sequence is sent to the *DYCI2* library (Nika et al., 2017) which is used to generate the concatenated audio synthesis. Indeed, this software is able to generate audio that corresponds to short-term scenario with the possibility to

rewrite its anticipation. Therefore, the scenario is redefined at each beat based on the prediction of the intelligent listening module. Then, the *DYCI2* adapts the generation at each time step (see Figure 11.2).

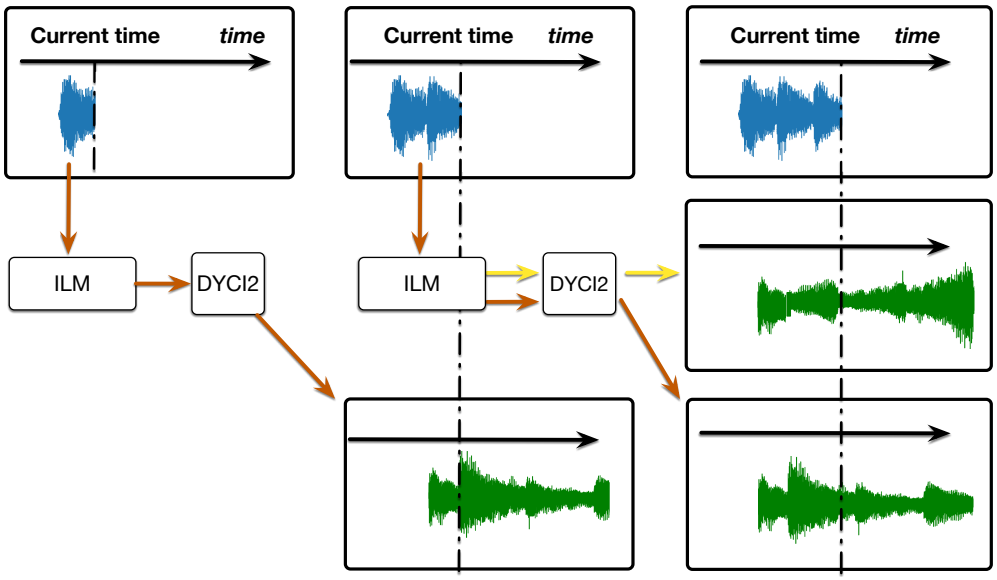

Figure 11.2: The Intelligent Listening Module (ILM) sent a predicted chord sequence at each beat. The DYCI2 library handle the short-term scenario for each beat. Depending of the scenario (if the prediction at **t-1** is coherent with the prediction at **t** (in orange) or not (in yellow)), DYCI2 will adapt the generation of the audio concatenated synthesis.

11.3 EXPERIMENTS

In order to test our intelligent listening module (performing real-time extraction of underlying chord labels and prediction of future chord progressions), we used it in different experimental contexts. Thereafter, we present our dataset used by the *DYCI2* library for the concatenative synthesis and some use of the global system.

11.3.1 *Creation of a dataset*

Since our predictive models are trained on *Realbook* songs, we selected jazz songs to build our dataset. We manually annotated three songs from the *Aebersold* corpus by associating a chord for each bar, based on a given score.

11.3.2 *Use of the software*

The software can be used by the musician in different ways as the reaction to the output can be diverse. First, by following exactly the chord sequences predicted by the module, we then observe a continuity in the scenarios sent to the *DYCI2* library.

Afterwards, we ask the musician to play music in an interactive context. In this case, the musician is not following exactly the predicted chord sequences but use the output of the co-improvisation system to build his play.

The two experiments are still under progress and it is still too early to conclude on the musician's' feedback.

12

LEARNING SPECTRAL TRANSFORMS TO IMPROVE TIMBRE ANALYSIS

One of the improvement that could be proposed for our listening module would be to identify the different instruments that are played during a co-improvisation. As an attempt to integrate instrument identification with the automatic chord extraction task, we propose to replace the CQT transformation of the input signal with a *learned* CQT-like feature specific to the timbre of each instrument.

Therefore, we argue that one of the key aspect for understanding timbre derives directly from the spectral transforms used to analyse its characteristics. Almost all timbre studies rely on the computation of a Fourier transform. However, the success of further analyses is highly dependent on the type of spectral representation. Here, we show that transforms can be learned depending on the specific timbre of each instrument by relying on recent advances in statistical learning. We propose different approaches to perform relevant feature extraction by relying directly on the signal waveform information. The transforms are learned through an instrument-specific objective (presence/absence in an acoustical mixture) allowing to adapt the computation of the transform to the timbre of each instrument. We compare the various transforms obtained and evaluate their properties as a novel proxy to timbre analysis.

12.1 BACKGROUND INFORMATION

The complexity of musical signal results from the acoustical mixture of different instruments having their own timbre characteristics. Almost all works in timbre analysis research rely on a time-frequency representation (Burgoyne, 2007) of the sound. However, the Fourier Transform might not always be the optimal representation for all analyses. This remark appears more clearly in other research fields, such as machine learning, where classification problems use preferentially the Constant-Q transforms (CQT). Recently, several breakthroughs in unsupervised generative models based on neural networks (Oord, 2016) have obtained astounding results in speech synthesis both qualitatively and quantitatively (Mehri, 2016), while learning directly from the raw audio waveform information. The large dimensionality of the inputs is handled by a hierarchical organization within the network. In the case of WaveNet (Oord, 2016), the complexity of the waveform signal is handled by first computing an organized hierarchy of dilated convolutions at different scales. Regarding the SampleRNN model (Mehri, 2016), the idea is to use as a first layer an auto-regressive process to address the specificity of waveforms and, then, to learn on the coefficients of this process. Both models are able to learn an intermediary representation that allows to generate speech

that is perceptually very close to human production. Here, we hypothesize that if those models are able to obtain such results, there might inherently learn a more adapted transform as an intermediary step of the learning process. Therefore, we introduce a model that is built of two complementary learning systems. The first part, trains a deep dilated convolution network akin to WaveNet to learn a given spectral transform in a supervised way. The second part then builds a classifier on top of this learned transform and learns a supplementary supervised instrument recognition task. However, the computation of the original transform (first part) is still allowed to be modified. Hence, we aim to make the system learn its own spectral representation, that is built specifically for a given task and the different timbre of the inputs. By analysing how the original transform layers are adapted depending on different orchestral instruments, we study how we could use the transform as a proxy for timbre analysis. To do so, we require the model that we learned to modify its weights towards an extremely specific transform by learning the instrument recognition task with a very low capacity classifier above the transform network. Therefore, we ensure that the discrimination has to be made by the spectral transform part rather than by the classifier layer on top. We study how the original transform layers are adapted depending on different orchestral contents, by analysing how the representation accomplish this particular task. Finally, we study how each transform could impact further timbre analysis by comparing the saliency and weights of the different learned transforms.

12.2 METHODOLOGY

We rely on Studio On Line (SOL) to obtain a dataset for 12 instruments (Piano, Flute, Clarinet, Trombone, English-Horn, French-Horn, Oboe, Saxophone, Trumpet, Violin, Violoncello, Bassoon), with 10 playing styles for each. We use samples of clean (isolated) instruments that are annotated with the same intensity (to remove effects from the pitch and loudness and incentivize the networks to focus on the timbre). Then, we extract all 12ms windows of signals with a Blackman window from each sample. For each window, we compute the magnitude of the CQT that will be used as a target for the pre-training of the networks. We normalize the transforms so that they have zero mean and unit variance. Regarding the models, we use a specifically designed architecture. First, we use two layers of hierarchical dilated convolutions as defined in the WaveNet system (Oord, 2016). Then, 2 layers of convolutions are used to increase the capacity of the model and reduce the dimensionality of the data. Finally, we add one fully-connected layer, to map to the dimensionality of the magnitude of the CQT. As our goal is to learn more adapted spectral transforms, we start by learning a common model on the whole dataset of orchestral instrument samples. We provide the magnitude of the CQT as the supervised target that the network must approximate. The model is learned with the SGD optimization algorithm with a learning rate $\eta = 10^{-3}$. Once the model converges, we add the low-capacity classifier composed by a single fully-connected layer in order to learn the instrument discrimination task. For these complete

models, we learn one different supervised task for each instrument, in order to study how the transform learned by the first layers may vary. To do so, we create mixtures with potential appearance of a given instrument to obtain a labelled dataset that associates every frame with the presence of specific instruments. Thus, the whole network must learn to infer the timbre properties of that instrument.

12.3 RESULTS

COMPARING DIFFERENT TRANSFORMS AND THEIR WEIGHTINGS: after learning to approximate the CQT, the global network converges to a very high accuracy (mean L_2 error of 0.055), which confirms that we are able to learn the transform. After training on instrument recognition, the very high variance of the weights and final representation obtained confirms that the transform adapts to the timbre of each instrument.

COMPARING THE EFFICIENCY IN MUSICAL INSTRUMENT RECOGNITION: by applying the transform learned for one instrument to recognize another, we obtain very significant drops in accuracy that depends on the classified instrument (e.g. Δacc(Bassoon) = $11 \pm 2\%$; Δacc(Violin) = $32 \pm 1\%$), which confirms that the transforms learned are highly specific to each instrument (see Figure 12.1).

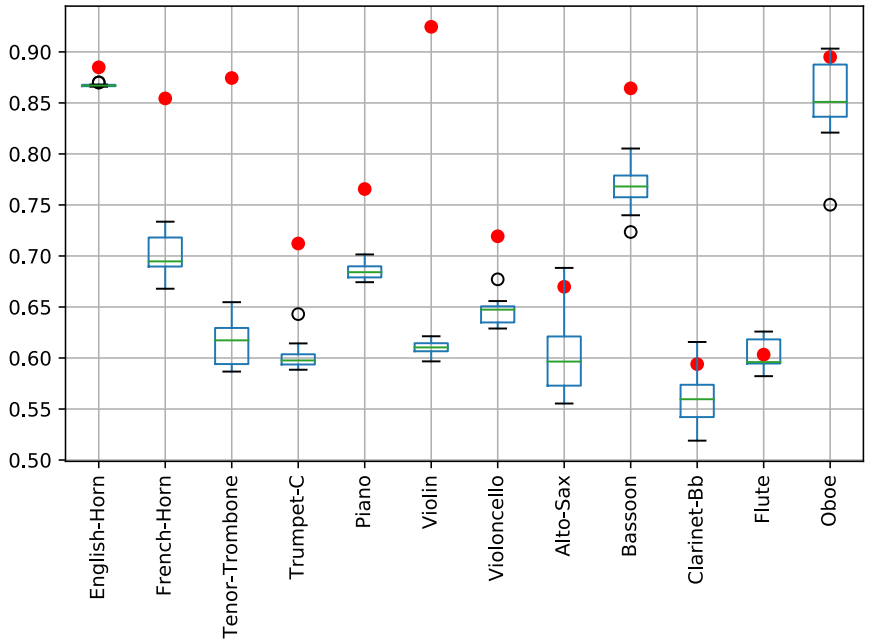

Figure 12.1: Red dots correspond to the classification accuracy scores for the instrument with its own model. Box plots are the classification accuracy scores for the instrument with other instrument models

USING TRANSFORMS AS A PROXY TO TIMBRE ANALYSIS: when using the saliency to determine the specificities of each transform, the variability and complexity of the results precluded a straightforward result. However, specific tools should be developed for using these transforms as a new way of performing timbre analysis.

12.4 CONCLUSION

We have shown that we could learn spectral transforms that are specific to the timbre of each orchestral instrument. Various analyses confirmed that these were highly variable and specific, which indicates that they have to adapt to instrumental timbre (shown through variance analysis and classification accuracies). However, the very high variability and complexity of the first layers precluded a direct understanding of their inner mechanisms, calling for the development of more specific analysis tools. These timbre-specific transforms still open very intriguing avenues of research as a novel proxy for timbre understanding.

13

GENERATING SCORES WITH VARIATIONAL AUTO-ENCODER

Our assumption in this application is that music style own their specific development for note sequences based on a particular harmonic path. Thus, our aim is twofold, first we want to generate a multi-dimensional space that will musically organize a set of existing chord sequences. Secondly, we want to generate multiple sequences of notes based on a chosen chord sequence. For the generation of note sequences, we want to be able to instantiate a note sequence which is specific to one musical style or an interpolation of different musical styles. In those terms we divide our work in three parts. The first part is the creation of a chord sequence space through the frameworks of Beta Variational Auto-Encoder (B-VAE) (Higgins et al., 2016). The second part is the generation of note sequences based on a Conditional VAE. Finally, the last part will stand as the adaptation of the generated note sequence for the generated chord sequence.

We propose to train a Variational Auto-Encoder to generate a multi-dimensional space that will musically organize a set of existing chord sequences. Thus, we train our models on the Realbook dataset, which is a set of 2,846 jazz songs converted into chord labels (see Section 10.4.3.1 for more details on the dataset). Our sequences are 16 beat-aligned chords, that corresponds to 4 bars of music. All the chords are reduced to two different chord alphabets A_0 and A_2 (described in Section 8.1). The architecture of our VAE follows the MLP-Van architecture detailed in Section 10.3. Therefore, the encoder/decoder of our VAE is composed with 2 fully-connected layers composed of 500 hidden units with ReLU activation. The latent space of this VAE has 16 dimensions. All the models have been trained with the ADAM optimizer for a total of 2000 epochs. We end up with a reconstruction accuracy of 95.70% for the alphabet A_0 and 85, 84% for the alphabet A_2.

In order to generate note sequences, we rely on a continuous and ordered latent space obtained through the variational learning method. To that aim, we train a recurrent VAE on a large set of Jazz midi files which have been sliced in small chunk of 4 consecutive beats and encoded with the Piano-roll representation. The resulting space have shown some interesting metric properties such as the dependency of the positions of the note sequences on the musical style. The assumption here is that a note sequence is written for a specific musical style depending in its intrinsic properties like its pitch range, its musical scale or its rhythm.

In that way, for the conditioning of the note generation, we simply limit our system to generate from a certain region of the space corresponding to the chosen musical style. Because of the continuous nature of such spaces, we argue that by moving along a region to another we could generate some sequences resulting from the interpolation of different styles.

In order to adapt the generated note sequence to fit the chord sequence, for each note of the sequence we take the closest note that is contained within the set of notes presents into the current beat-aligned chord taken at every octave. An example of four different note scores that follow the same chord sequence but conditioned with four music styles is presented on Figure 13.1.

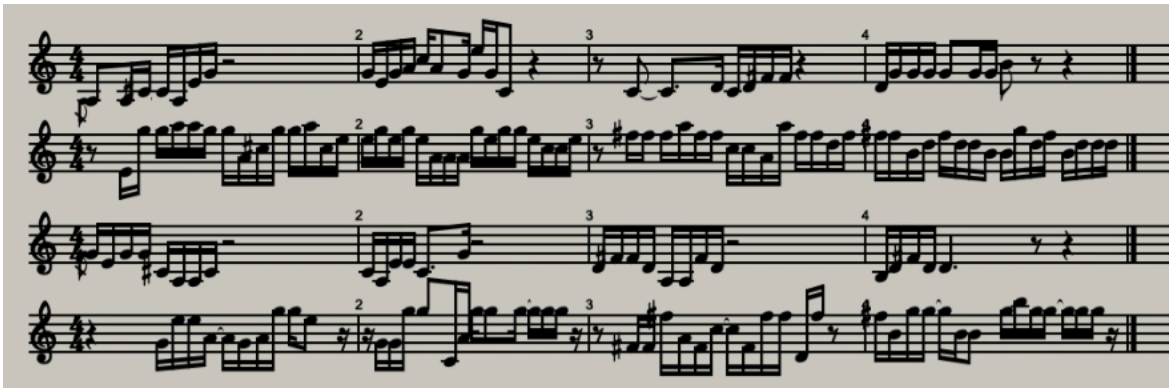

Figure 13.1: Four different note scores for the same generated chord sequence (A:7 A:min7 D:7 G:Maj7) conditioned with four different music styles.

14

CREATING HARMONIC SPACES FOR "LULLABY EXPERIENCE"

For this application we proposed to use our local chord extraction system to organize audio events in a 3D space. This application has been presented for a sound installation project named *Lullaby Experience* [1]. This musical creation is a participatory project imagined by the composer Pascal Dusapin. Many chorals have been collected around the world and provide the sonic material to generate new chorals with the help of the *DYCI2* framework (Nika, 2016). The sound spatialization of these chorals would eventually be realized with an harmonic criteria by creating paths into an harmonic space. Our aim was to create this harmonic space with the help of our ACE system.

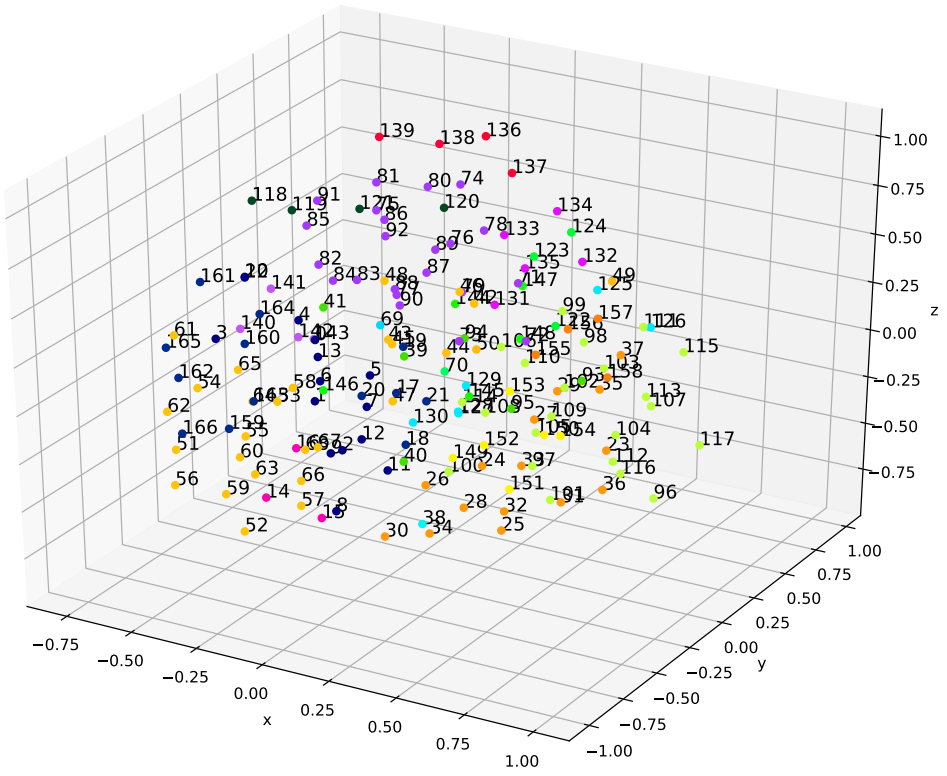

Figure 14.1: Harmonic path of 25 chorales, each chorale is split with a time step of 2 seconds, every part of the same chorale has the same color.

Thus, we used our trained ACE network with 25 output dimensions (corresponding to the Major/minor chords and No-chord categories). Then, we removed the

1 https://www.ircam.fr/agenda/lullaby-experience/detail/

last layer to get the output of the last hidden layer (200 output bins). Then, for each track we slid the network over the entire track. At the end we applied three operations (mean, max and L2 norm) over all the collected output vectors. We obtained a unique 600-bins vector for each track. In order to reduce the dimensions of the space to fit a 3-D representation we used the TSNE algorithm (Maaten and Hinton, 2008).

At the end, we obtained a space that takes into account the harmonic similarity between the corpus tracks. By splitting the tracks and applying the same processing for each parts, we also obtained a path in the space for each track (see Figure 14.1).

Part IV

CONCLUSION AND FUTURE WORKS

15

CONCLUSION AND FUTURE WORKS

In this thesis, we focused on the automatic chord extraction from audio and the prediction of symbolic chord sequences. We defined musical chords as *high-level* labels since they represent for us equivalent classes of functions. The motivation in term of application was to propose an intelligent listening module performing real-time extraction of underlying chord labels and prediction of future chord progressions, that can inform and guide generative models.

For the automatic chord extraction task, we proposed two methods to improve quantitatively and qualitatively chord extraction systems. The first method proposes to introduce musical knowledge through the learning process via a musical distance computed between the predicted and the labeled chords. The second method combines discriminative and generative models, and allows to use unlabeled musical data to perform semi-supervised learning.

For the chord sequence prediction task, we first presented an aggregation approach, summarizing input chords at different time scales. Then, after having underlined the importance of using downbeat position information in the development of chord sequence prediction models, we introduced two different approach based on the introduction in the training of high-level metadata (such as key or downbeat information) and new data representations.

Furthermore, we introduced a specifically-tailored chord analyzer to measure the performances of our models in term of *error quality*. Thus, for both tasks, we proposed more adapted evaluation techniques in order to exhibit *strict* transcription errors which nevertheless reflect correct *high-level* transcription functions. Thus, we developed an *ACE Analyzer* including two analysis modules to discover musical relationships between the chords predicted by chord-based models and the target chords. The first module detects the errors corresponding to hierarchical relationships or usual *chord substitutions* rules, whereas the second module of focuses on harmonic degrees.

15.1 MUSIC INFORMATION RETRIEVAL

15.1.1 *Chord extraction task*

In an attempt to bridge the gap between rule-based an statistical models in ACE field, we experimented the training of Deep Learning based models with the help of musical distances reflecting the relationships between chords. To do so, we trained CNN models on different chord alphabets and with different distances. In order to evaluate our models with a musically oriented approach, we proposed a specifically tailored ACE analyzer. This analyzer is aimed to take into account the *harmonic*

functions that qualify the roles and the tonal significance of chords and the possible equivalences between them within a sequence. We concluded that training the models using distances reflecting the relationships between chords (e.g. within a Tonnetz-space or with pitch class profiles) improves the results both quantitatively (in terms of classification scores) and qualitatively (in terms of harmonic functions).

In future works, we would like to propose functional chord alphabets or equivalent chord classes that could alleviate the problem of subjectivity, inherent within chord labels. Indeed, high levels of abstraction such as chords can be associated with multiple classes, and even expert human annotators do not agree on the precise classification of each chord of a song. We also plan to investigate the impact of the instrument recognition tasks along with the chord extraction task. Our hypothesis is that each instrument own their specific development and knowing them could improve the discovering of the harmonic path. Finally, in order to obtain a multi-scale discovering module, we would like to propose a system that can treat mutually-dependent musical elements such as keys, beats, and notes.

15.1.2 *Chord sequence prediction task*

We studied the prediction of successive beat-aligned chords given an initial chord sequence. In the chord prediction task, many existing systems only target the prediction of the next chord symbol given a sequence, disregarding repeated chords and ignoring timing. Thus, when used in a iterative way to predict a chord sequence, these systems suffer from errors propagation along the prediction steps. Therefore, we proposed a model which predicts the entire chord sequence in one step. In order to provide a multi-scale modeling of chord progressions at different levels of granularity, we introduced an aggregation approach, summarizing input chords at different time scales. We showed that this kind of approach outperform existing systems in a qualitative and a quantitative way. However, we observed that the prediction accuracy significantly decreases across each bar line, likely due to bar-length repetition of chords. Hence, we underlined the importance of using downbeat position information in the development of future chord sequence prediction models.

In a second attempt, we investigated the relevance of taking advantage of structural high-level metadata (such as key or downbeat information) in the training of chord sequence predictive models. To this end, we trained several instances of a neural network model using the different chord alphabets. Then, we included information on key and downbeat position during the prediction or as an additional learning task. After having observed a quantitative improvement of the predictions, we analyzed qualitatively the errors by using musical relationships between the predicted and targeted chords. We concluded, that introducing the information of downbeat or key has an impact on the performances in term of accuracy but also in terms of harmonic functions. Indeed, using the information on key could be privileged in mostly diatonic contexts since it improves the quality of the predicted harmonic function. Conversely, it may not be suitable for repertoires where

modulations and non-diatonic chords are frequents if their precise identification is important.

We are planning to improve our chord sequence predictive model in different ways. First of all, we are convinced that the data representation that we are using could not be adapted to the task. Indeed, the large ratio of repetitions within beat aligned chord labels could be avoided by introducing a additional temporal information. Furthermore, a functional alphabet or an alphabet based on equivalent classes could give a better understanding of the underlying harmony. However, the length of our predicted chord sequence could have an impact on the performances. Thus, we will investigate the chord sequence prediction at different temporal horizon.

15.1.3 *Transverse analysis*

After these studies on chord extraction task and chord sequence prediction task, we ended up with considerations that could push to rethink some of MIR tasks. Firstly, the use of classifying models widely used in Computer Vision could not be perfectly adapted to high-level labels such as chord, key, genre or any other subjective labels. Indeed, high-level musical labels handle intrinsic and hierarchical relationships that should be taken into account in the developments of such models. Thus, a solution could be to move from strict labels to functional annotations by taking into account the nature of the musical elements. Secondly, the integration of additional musical information in Deep Learning models can improve the results in quantitative and a qualitative ways. Furthermore, the qualitative analysis of the results should be realized by taking into account the applicative case at hand. Indeed, depending of the information that we want to extract (e.g. the harmonic progression), it could be interesting to gain a better high-level understanding at the cost of a lower classification accuracy. Furthermore, it appears that adding contextual information (such as key for a chord progression) could constraint the model in the prediction. Thus, a model can obtain better performances in term of accuracy or perplexity, but output biased results. Once again, performing a functional analyzes or any high-level tests allow a better understanding of the results and a more adapted design of MIR models.

In a more general context, our work led to the design of a theoretical framework which makes possible to generalize our approach to other musical alphabets if followed. On the one hand, one could analyze the data in order to define annotations that take into account the nature of the elements. Thus, the annotations should have hierarchical and intrinsic relationships. Otherwise, it could be constituted by equivalence classes that represent specific functions. On the other hand, the analyses have to be performed with tailored evaluation methods.

15.2 CREATIVE APPLICATIONS

We presented the architecture of different applicative prototypes. Indeed, as aforementioned in Section 15.1.3, we designed MIR models in the perspective of using them in musical applications. Thus, we combined the chord extraction and the chord sequence prediction models to infer short-term chord sequences from real-time discovery of chords in an co-improvisation context.

In order to improve this intelligent listening module (performing real-time extraction of underlying chord labels and prediction of future chord progressions), we developed a model that learns an instrument-oriented CQT in order to identify the different instruments in an audio flux by relying on their associated timbre.

We also used generative model to regularize the latent space of the learned chord sequences. Therefore, we created multi-dimensional space that musically organizes a set of existing chord sequences and allow to generate multiple sequences of notes based on a chosen chord sequence.

Finally, we used our chord extraction system to obtain a space that takes into account the harmonic similarity between tracks of a corpus. This system allows to spatialize tracks by creating paths into an harmonic space.

To conclude, we proposed several prototypes that are aimed to be used in a creative context. Having feedback from musicians helped us to design more adapted musical tools and brought a new perspective to our systems.

As future works, we aim to integrate an instrument recognition module in order to perform co-improvisation with different instruments and musicians. At a final stage of development, we would like to integrate an on-line learning module that could learn along with a musician and then adapt itself to the played style.

Part V

APPENDIX

A

APPENDIX TEST

A.1 GRID SEARCH FOR MLP APPLIED TO CHORD SEQUENCE PREDICTION

Hidden layer=1		nb_h=100	nb_h=200	nb_h=500	nb_h=1000
nb_bn=10	r_do=0	40.05(63K)	40.75(166K)	40.80(716K)	40.40(2432K)
	r_do=0.2	39.60(63K)	40.97(166K)	41.30(716K)	41.37(2432K)
	r_do=0.4	38.30(63K)	40.24(166K)	41.66(716K)	41.62(2432K)
	r_do=0.6	36.87(63K)	38.79(166K)	40.90(716K)	41.84(2432K)
	r_do=0.8	28.65(63K)	36.44(166K)	39.29(716K)	40.64(2432K)
nb_bn=20	r_do=0	40.43(65K)	41.02(170K)	41.04(726K)	41.24(2452K)
	r_do=0.2	39.99(65K)	41.31(170K)	41.79(726K)	41.57(2452K)
	r_do=0.4	38.61(65K)	40.71(170K)	**42.10(726K)**	41.94(2452K)
	r_do=0.6	36.85(65K)	39.13(170K)	41.62(726K)	42.31(2452K)
	r_do=0.8	30.14(65K)	36.39(170K)	39.21(726K)	41.01(2452K)
nb_bn=50	r_do=0	40.28(71K)	40.55(182K)	40.96(756K)	41.46(2512K)
	r_do=0.2	39.97(71K)	41.69(182K)	41.72(756K)	41.68(2512K)
	r_do=0.4	38.67(71K)	40.86(182K)	42.26(756K)	42.09(2512K)
	r_do=0.6	36.94(71K)	39.13(182K)	41.60(756K)	42.37(2512K)
	r_do=0.8	30.05(71K)	36.35(182K)	39.07(756K)	41.05(2512K)
nb_bn=100	r_do=0	40.63(81K)	40.77(202K)	41.19(806K)	41.17(2612K)
	r_do=0.2	40.33(81K)	41.57(202K)	41.92(806K)	41.67(2612K)
	r_do=0.4	38.85(81K)	40.90(202K)	42.25(806K)	42.09(2612K)
	r_do=0.6	37.01(81K)	39.41(202K)	41.78(806K)	42.28(2612K)
	r_do=0.8	28.11(81K)	36.31(202K)	39.39(806K)	41.16(2612K)

Table A.1: Accuracy score (on the test dataset for the alphabet A_0 and for one split), and in parenthesis the associated number of network parameters for the MLP-Van with one hidden layer for the encoder and the decoder ($nb\_l = 1$). *nb_h* is the number of hidden units for the encoder and decoder layers, *nb_bn* is the size of the bottleneck. *r_do* is the dropout ratio (*r_do*).

Hidden layers=2		nb_h=100	nb_h=200	nb_h=500	nb_h=1000
nb_bn=10	r_do=0	40.28(84K)	40.72(247K)	40.79(1219K)	40.57(4438K)
	r_do=0.2	39.01(84K)	40.90(247K)	41.81(1219K)	41.17(4438K)
	r_do=0.4	37.00(84K)	39.13(247K)	41.32(1219K)	42.06(4438K)
	r_do=0.6	34.09(84K)	36.61(247K)	39.67(1219K)	41.32(4438K)
	r_do=0.8	26.11(84K)	32.53(247K)	36.15(1219K)	38.21(4438K)
nb_bn=20	r_do=0	40.44(86K)	40.47(251K)	40.78(1229K)	41.08(4458K)
	r_do=0.2	39.23(86K)	41.01(251K)	42.13(1229K)	41.29(4458K)
	r_do=0.4	36.88(86K)	39.27(251K)	41.66(1229K)	42.35(4458K)
	r_do=0.6	34.18(86K)	36.77(251K)	39.70(1229K)	41.47(4458K)
	r_do=0.8	26.96(86K)	33.07(251K)	36.05(1229K)	38.24(4458K)
nb_bn=50	r_do=0	40.44(92K)	40.71(263K)	40.58(1259K)	41.22(4518K)
	r_do=0.2	39.15(92K)	41.05(263K)	41.71(1259K)	41.57(4518K)
	r_do=0.4	36.97(92K)	39.31(263K)	41.70(1259K)	42.27(4518K)
	r_do=0.6	33.84(92K)	36.74(263K)	39.84(1259K)	41.59(4518K)
	r_do=0.8	25.95(92K)	32.37(263K)	35.85(1259K)	38.36(4518K)
nb_bn=100	r_do=0	40.62(102K)	40.79(283K)	40.88(1309K)	41.08(4618K)
	r_do=0.2	39.33(102K)	41.25(283K)	41.77(1309K)	41.72(4618K)
	r_do=0.4	36.96(102K)	39.51(283K)	41.83(1309K)	42.31(4618K)
	r_do=0.6	34.09(102K)	36.78(283K)	39.88(1309K)	41.55(4618K)
	r_do=0.8	27.97(102K)	32.63(283K)	35.84(1309K)	38.38(4618K)

Table A.2: Accuracy score (on the test dataset for the alphabet A_0 and for one split), and in parenthesis the associated number of network parameters for the MLP-Van with two hidden layers for the encoder and the decoder ($nb\_l = 2$). *nb_h* is the number of hidden units for the encoder and decoder layers, *nb_bn* is the size of the bottleneck. *r_do* is the dropout ratio (*r_do*).

Hidden layers=3		nb_h=100	nb_h=200	nb_h=500	nb_h=1000
nb_bn=10	r_do=0	40.53(104K)	40.68(329K)	40.64(1722K)	40.83(6444K)
	r_do=0.2	38.64(104K)	40.41(329K)	41.81(1722K)	41.42(6444K)
	r_do=0.4	34.9(104K)	38.23(329K)	40.86(1722K)	41.93(6444K)
	r_do=0.6	31.90(104K)	34.66(329K)	38.60(1722K)	40.52(6444K)
	r_do=0.8	22.72(104K)	29.44(329K)	34.45(1722K)	36.72(6444K)
nb_bn=20	r_do=0	40.43(106K)	40.38(333K)	40.70(1732K)	40.89(6464K)
	r_do=0.2	38.72(106K)	40.58(333K)	42.07(1732K)	41.67(6464K)
	r_do=0.4	35.12(106K)	38.27(333K)	40.94(1732K)	41.96(6464K)
	r_do=0.6	32.05(106K)	34.86(333K)	38.66(1732K)	40.54(6464K)
	r_do=0.8	21.15(106K)	29.34(333K)	34.18(1732K)	36.77(6464K)
nb_bn=50	r_do=0	40.35(112K)	40.59(345K)	40.87(1762K)	41.37(6524K)
	r_do=0.2	38.44(112K)	40.55(345K)	42.06(1762K)	41.50(6524K)
	r_do=0.4	35.05(112K)	38.34(345K)	40.90(1762K)	41.96(6524K)
	r_do=0.6	32.41(112K)	34.86(345K)	38.56(1762K)	40.58(6524K)
	r_do=0.8	22.77(112K)	29.38(345K)	34.27(1762K)	36.75(6524K)
nb_bn=100	r_do=0	40.58(122K)	40.62(365K)	40.81(1812K)	40.65(6624K)
	r_do=0.2	38.65(122K)	40.66(365K)	42.00(1812K)	41.82(6624K)
	r_do=0.4	35.55(122K)	38.36(365K)	41.00(1812K)	42.10(6624K)
	r_do=0.6	32.13(122K)	34.66(365K)	38.59(1812K)	40.41(6624K)
	r_do=0.8	21.66(122K)	29.13(365K)	34.38(1812K)	36.57(6624K)

Table A.3: Accuracy score (on the test dataset for the alphabet A_0 and for one split), and in parenthesis the associated number of network parameters for the MLP-Van with three hidden layers for the encoder and the decoder ($nb\_l = 3$). *nb_h* is the number of hidden units for the encoder and decoder layers, *nb_bn* is the size of the bottleneck. *r_do* is the dropout ratio (*r_do*).

A.2 GRID SEARCH FOR LSTM APPLIED TO CHORD SEQUENCE PREDICTION

Without bottleneck		nb_h=200	nb_h=500	nb_h=1000
r_do=0.4	at=True	40.49(492K)	41.23(2880K)	41.60(11261K)
	at=False	40.53(371K)	41.62(2129K)	41.47(8259K)
r_do=0.6	at=True	40.77(492K)	40.99(2880K)	41.39(11261K)
	at=False	40.69(371K)	41.25(2129K)	41.21(8259K)
r_do=0.8	at=True	40.54(492K)	40.82(2880K)	41.45(11261K)
	at=False	40.52(371K)	41.07(2129K)	41.29(8259K)
With bottleneck		**nb_h=200**	**nb_h=500**	**nb_h=1000**
r_do=0.4	at=True	41.80(573K)	41.86(3082K)	41.44(11665K)
	at=False	40.69(371K)	41.10(2129K)	41.32(8259K)
r_do=0.6	at=True	**42.03(573K)**	41.73(3082K)	41.38(11665K)
	at=False	41.02(371K)	41.05(2129K)	41.18(8259K)
r_do=0.8	at=True	41.44(573K)	41.58(3082K)	41.72(11665K)
	at=False	40.29(371K)	41.09(2129K)	41.22(8259K)

Table A.4: Accuracy score (on the test dataset for the alphabet A_0 and for one split), and in parenthesis the associated number of network parameters for the LSTM with one hidden layer for the encoder and the decoder ($nb\_l = 1$) with or without a bottleneck of 50 hidden units. *nb_h* is the number of hidden units for the encoder and decoder layers, *r_do* is the dropout ratio (*r_do*) and *at* is the use of attention mechanism.

Without bottleneck		nb_h=200	nb_h=500	nb_h=1000
r_do=0.4	at=True	41.49(1135K)	41.32(6888K)	41.03(27277K)
	at=False	41.11(1015K)	41.21(6137K)	41.31(24275K)
r_do=0.6	at=True	41.07(1135K)	41.80(6888K)	41.68(27277K)
	at=False	40.70(1015K)	41.68(6137K)	41.35(24275K)
r_do=0.8	at=True	41.43(1135K)	41.63(6888K)	41.31(27277K)
	at=False	41.14(1015K)	41.62(6137K)	41.39(24275K)
With bottleneck		nb_h=200	nb_h=500	nb_h=1000
r_do=0.4	at=True	41.17(1216K)	41.11(7090K)	41.01(27681K)
	at=False	40.94(1015K)	40.96(6137K)	41.24(24275K)
r_do=0.6	at=True	41.52(1216K)	41.07(7090K)	40.21(27681K)
	at=False	41.00(1015K)	41.37(6137K)	41.62(24275K)
r_do=0.8	at=True	41.58(1216K)	41.08(7090K)	41.13(27681K)
	at=False	41.15(1015K)	41.19(6137K)	41.57(24275K)

Table A.5: Accuracy score (on the test dataset for the alphabet A_0 and for one split), and in parenthesis the associated number of network parameters for the LSTM with two hidden layers for the encoder and the decoder ($nb\_l = 1$) with or without a bottleneck of 50 hidden units. *nb_h* is the number of hidden units for the encoder and decoder layers, *r_do* is the dropout ratio (*r_do*) and *at* is the use of attention mechanism.

BIBLIOGRAPHY

Anger-Weller, Jo (1990). *Clés pour l'harmonie: a l'usage de l'analyse, l'improvisation, la composition: 2ème édition revue et augmentée*. HL music.

Assayag, Gérard and Georges Bloch (2007). "Navigating the oracle: A heuristic approach." In: *International Computer Music Conference'07*, pp. 405–412.

Assayag, Gérard, Georges Bloch, Marc Chemillier, Arshia Cont, and Shlomo Dubnov (2006). "Omax brothers: a dynamic yopology of agents for improvization learning." In: *Proceedings of the 1st ACM workshop on Audio and music computing multimedia*. ACM, pp. 125–132.

Bengio, Yoshua, Ian J Goodfellow, and Aaron Courville (2015). "Deep learning." In: *Nature* 521, pp. 436–444.

Bengio, Yoshua et al. (2009). "Learning deep architectures for AI." In: *Foundations and trends® in Machine Learning* 2.1, pp. 1–127.

Berenzweig, A., B. Logan, D. Ellis, and B. Whitman (2004). "A large-scale evaluation of acoustic and subjective music-similarity measures." In: *Computer Music Journal* 28.2, pp. 63–76.

Berry, Wallace (1987). *Structural functions in music*. Courier Corporation.

Blei, David M, Alp Kucukelbir, and Jon D McAuliffe (2017). "Variational inference: A review for statisticians." In: *Journal of the American statistical Association* 112.518, pp. 859–877.

Bonnasse-Gahot, Laurent (2014). "An update on the SOMax project." In: *Ircam-STMS, Internal report ANR project Sample Orchestrator* 2.

Boulanger-Lewandowski, Nicolas, Yoshua Bengio, and Pascal Vincent (2013). "Audio Chord Recognition with Recurrent Neural Networks." In: *ISMIR*. Citeseer, pp. 335–340.

Bracewell, Ronald Newbold (1986). *The Fourier transform and its applications*. Vol. 31999. McGraw-Hill New York.

Bronstein, Michael M, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst (2017). "Geometric deep learning: going beyond euclidean data." In: *IEEE Signal Processing Magazine* 34.4, pp. 18–42.

Brown, Judith C (1991). "Calculation of a constant Q spectral transform." In: *The Journal of the Acoustical Society of America* 89.1, pp. 425–434.

Burgoyne, John Ashley, Jonathan Wild, and Ichiro Fujinaga (2011). "An Expert Ground Truth Set for Audio Chord Recognition and Music Analysis." In: *Proceedings of the International Society for Music Information Retrieval (ISMIR)*, pp. 633–638.

Cannam, C., E. Benetos, M. Mauch, M. E. P. Davies, S. Dixon, C. Landone, K. Noland, and D. Stowell (2015). "MIREX 2015: VAMP Plugins from the Centre for Digital Music." In: *In Proceedings of the Music Information Retrieval Evaluation eXchange (MIREX)*.

Cheng, Haibin, Pang-Ning Tan, Jing Gao, and Jerry Scripps (2006). "Multistep-ahead time series prediction." In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, pp. 765–774.

Cho, Kyunghyun, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio (2014). "Learning phrase representations using RNN encoder-decoder for statistical machine translation." In: *arXiv preprint arXiv:1406.1078*.

Cho, T. (2014). "Improved techniques for automatic chord recognition from music audio signals." PhD thesis. New York University.

Cho, T., R. J. Weiss, and J. P. Bello (2010). "Exploring common variations in state of the art chord recognition systems." In: *Proceedings of the Sound and Music Computing Conference (SMC)*, pp. 1–8.

Choi, Keunwoo, George Fazekas, and Mark Sandler (2016). "Text-based LSTM networks for automatic music composition." In: *arXiv preprint arXiv:1604.05358*.

Choi, Keunwoo, György Fazekas, Kyunghyun Cho, and Mark Sandler (2017). "A tutorial on deep learning for music information retrieval." In: *arXiv preprint arXiv:1709.04396*.

Ciresan, Dan Claudiu, Ueli Meier, Luca Maria Gambardella, and Jurgen Schmidhuber (2011). "Convolutional neural network committees for handwritten character classification." In: *Document Analysis and Recognition (ICDAR), 2011 International Conference on*. IEEE, pp. 1135–1139.

Cohn, R. (1997). "Neo-riemannian operations, parsimonious trichords, and their" tonnetz" representations." In: *Journal of Music Theory* 41.1, pp. 1–66.

Crestel, Léopold and Philippe Esling (2017). "Live Orchestral Piano, a system for real-time orchestral music generation." In: *14th Sound and Music Computing Conference*, p. 434.

Dong, Hao-Wen, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang (2018). "Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment." In: *Thirty-Second AAAI Conference on Artificial Intelligence*.

Donzé, Alexandre, Rafael Valle, Ilge Akkaya, Sophie Libkind, Sanjit A Seshia, and David Wessel (2014). "Machine improvisation with formal specifications." In: *ICMC*.

Eck, Douglas and Juergen Schmidhuber (2002). "Finding temporal structure in music: Blues improvisation with LSTM recurrent networks." In: *12th IEEE workshop on neural networks for signal processing*, pp. 747–756.

Eigenfeldt, Arne and Philippe Pasquier (2010). "Realtime generation of harmonic progressions using controlled Markov selection." In: *Proceedings of ICCC-X-Computational Creativity Conference*, pp. 16–25.

Feisthauer, Laurent, Louis Bigo, Mathieu Giraud, and Florence Levé (2020). "Estimating keys and modulations in musical pieces." In: *17th Sound and Music Computing Conference*.

Glorot, Xavier and Yoshua Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks." In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256.

Goto, Masataka, Hiroki Hashiguchi, Takuichi Nishimura, and Ryuichi Oka (2002). "RWC music database: Popular, classical, and jazz music databases." In: *Proceedings of the International Society for Music Information Retrieval (ISMIR)*, pp. 287–288.

Graves, Alex (2013). "Generating sequences with recurrent neural networks." In: *arXiv preprint arXiv:1308.0850*.

Guo, Yanming, Yu Liu, Ard Oerlemans, Songyang Lao, Song Wu, and Michael S Lew (2016). "Deep learning for visual understanding: A review." In: *Neurocomputing* 187, pp. 27–48.

Gupta, Max, Jérôme Nika, and Tristan Carsault (2020). "Multi-Step Chord Prediction: Time-Separated Chord Progressions across MLP, LSTM and Auto-Encoder Architectures." In:

Hadjeres, Gaëtan, François Pachet, and Frank Nielsen (2017). "Deepbach: a steerable model for bach chorales generation." In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, pp. 1362–1371.

Hahnloser, Richard HR, Rahul Sarpeshkar, Misha A Mahowald, Rodney J Douglas, and H Sebastian Seung (2000). "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit." In: *Nature* 405.6789, pp. 947–951.

Harte, C. (2010a). "Towards automatic extraction of harmony information from music signals." PhD thesis. Queen Mary University of London.

Harte, C., M. B. Sandler, S. A. Abdallah, and E. Gómez (2005). "Symbolic Representation of Musical Chords: A Proposed Syntax for Text Annotations." In: *International Symposium on Music Information Retrieval*. Vol. 5, pp. 66–71.

Harte, Christopher (2010b). "Towards automatic extraction of harmony information from music signals." PhD thesis.

Harte, Christopher and Mark Sandler (2005). "Automatic chord identifcation using a quantised chromagram." In: *Audio Engineering Society Convention 118*. Audio Engineering Society.

Hawthorne, Curtis, Erich Elsen, Jialin Song, Adam Roberts, Ian Simon, Colin Raffel, Jesse Engel, Sageev Oore, and Douglas Eck (2017). "Onsets and frames: Dual-objective piano transcription." In: *arXiv preprint arXiv:1710.11153*.

Heafield, Kenneth, Ivan Pouzyrevsky, Jonathan H Clark, and Philipp Koehn (2013). "Scalable modified Kneser-Ney language model estimation." In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*. Vol. 2.

Hélie, Thomas and Charles Picasso (2017). "The Snail: a real-time software application to visualize sounds." In: *Proc. DAFx*.

Herremans, Dorien, Stéphanie Weisser, Kenneth Sörensen, and Darrell Conklin (2015). "Generating structured music for bagana using quality metrics based on Markov models." In: *Expert Systems with Applications* 42.21, pp. 7424–7435.

Higgins, Irina, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner (2016). "beta-vae: Learning basic visual concepts with a constrained variational framework." In:

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long short-term memory." In: *Neural computation* 9.8, pp. 1735–1780.

Humphrey, Eric J and Juan P Bello (2012). In: *Rethinking automatic chord recognition with convolutional neural networks*. Vol. 2. IEEE, pp. 357–362.

Humphrey, Eric J and Juan Pablo Bello (2015). "Four Timely Insights on Automatic Chord Estimation." In: *ISMIR*. Vol. 10, pp. 673–679.

Humphrey, Eric J, Taemin Cho, and Juan P Bello (2012). "Learning a robust tonnetz-space transform for automatic chord recognition." In: *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*. IEEE, pp. 453–456.

Huron, David Brian (2006). *Sweet anticipation: Music and the psychology of expectation*. MIT press.

Ioffe, Sergey and Christian Szegedy (2015). "Batch normalization: Accelerating deep network training by reducing internal covariate shift." In: *arXiv preprint arXiv:1502.03167*.

Jiang, J., W. Li, and Y. Wu (2017). "Extended abstract for mirex 2017 submission: Chord recognition using random forest model." In: *MIREX evaluation results*.

Karras, Tero, Timo Aila, Samuli Laine, and Jaakko Lehtinen (2017). "Progressive growing of gans for improved quality, stability, and variation." In: *arXiv preprint arXiv:1710.10196*.

Kingma, Diederik P and Max Welling (2013). "Auto-encoding variational bayes." In: *arXiv preprint arXiv:1312.6114*.

Klapuri, Anssi and Manuel Davy (2007). *Signal processing methods for music transcription*. Springer Science & Business Media.

Korzeniowski, Filip and Gerhard Widmer (2016a). "A fully convolutional deep auditory model for musical chord recognition." In: *Machine Learning for Signal Processing (MLSP), 2016 IEEE 26th International Workshop on*. IEEE, pp. 1–6.

– (2016b). "Feature learning for chord recognition: the deep chroma extractor." In: *arXiv preprint arXiv:1612.05065*.

– (2018). "Improved Chord Recognition by Combining Duration and Harmonic Language Models." In: *Proceedings of ISMIR*.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "Imagenet classification with deep convolutional neural networks." In: *Advances in neural information processing systems*, pp. 1097–1105.

Lafferty, J., A. McCallum, and F. C. N. Pereira (2001). "Conditional random fields: Probabilistic models for segmenting and labeling sequence data." In:

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep learning." In: *Nature* 521.7553, pp. 436–444.

LeCun, Yann, Yoshua Bengio, et al. (1995). "Convolutional networks for images, speech, and time series." In: *The handbook of brain theory and neural networks* 3361.10, p. 1995.

Lee, Kyogu and Malcolm Slaney (2006). "Automatic Chord Recognition from Audio Using a HMM with Supervised Learning." In: *ISMIR*, pp. 133–137.

Lou, H-L. (1995). "Implementing the Viterbi algorithm." In: *IEEE Signal Processing Magazine* 12.5, pp. 42–52.

Luong, Minh-Thang, Hieu Pham, and Christopher D Manning (2015). "Effective approaches to attention-based neural machine translation." In: *arXiv preprint arXiv:1508.04025*.

Maaten, Laurens van der and Geoffrey Hinton (2008). "Visualizing data using t-SNE." In: *Journal of Machine Learning Research* 9.Nov, pp. 2579–2605.

McFee, B. and J. P. Bello (2017). "Structured training for large-vocabulary chord recognition." In: *Proceedings of the 18th International Society for Music Information Retrieval Conference (ISMIR'2017). ISMIR*.

McVicar, Matt, Raúl Santos-Rodríguez, Yizhao Ni, and Tijl De Bie (2014). "Automatic chord estimation from audio: A review of the state of the art." In: *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)* 22.2, pp. 556–575.

Mehri, Soroush, Kundan Kumar, Ishaan Gulrajani, Rithesh Kumar, Shubham Jain, Jose Sotelo, Aaron Courville, and Yoshua Bengio (2016). "SampleRNN: An unconditional end-to-end neural audio generation model." In: *arXiv preprint arXiv:1612.07837*.

Mikolov, Tomas, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur (2010). "Recurrent neural network based language model." In: *Interspeech*. Vol. 2, p. 3.

Moog, Robert A (1986). "Midi: Musical instrument digital interface." In: *Journal of the Audio Engineering Society* 34.5, pp. 394–404.

Moreira, Julian, Pierre Roy, and François Pachet (2013). "Virtualband: Interacting with Stylistically Consistent Agents." In: *ISMIR*, pp. 341–346.

Ng, Andrew. "Coursera Machine Learning, Lecture 2 Slides." In: ().

Nika, Jérôme (2016). "Guiding human-computer music improvisation: introducing authoring and control with temporal scenarios." PhD thesis. Paris 6.

Nika, Jérôme, Dimitri Bouche, Jean Bresson, Marc Chemillier, and Gérard Assayag (2015). "Guided improvisation as dynamic calls to an offline model." In: *Sound and Music Computing (SMC)*.

Nika, Jérôme, Marc Chemillier, and Gérard Assayag (2016). "ImproteK: introducing scenarios into human-computer music improvisation." In: *Computers in Entertainment (CIE)* 14.2, p. 4.

Nika, Jérôme, Ken Déguernel, Axel Chemla, Emmanuel Vincent, and Gérard Assayag (2017). "DYCI2 agents: merging the" free"," reactive", and" scenario-based" music generation paradigms." In: *Proceedings of ICMC*.

Nika, Jérôme, José Echeveste, Marc Chemillier, and Jean-Louis Giavitto (2014). "Planning human-computer improvisation." In: *ICMC*.

Oord, Aaron, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George Driessche, Edward Lockhart, Luis Cobo, Florian Stimberg, et al. (2018). "Parallel wavenet: Fast high-fidelity speech synthesis." In: *International conference on machine learning*. PMLR, pp. 3918–3926.

Oudre, Laurent, Yves Grenier, and Cédric Févotte (2009). "Template-based Chord Recognition: Influence of the Chord Types." In: *ISMIR*, pp. 153–158.

Pachet, François, Pierre Roy, Julian Moreira, and Mark d'Inverno (2013). "Reflexive loopers for solo musical improvisation." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, pp. 2205–2208.

Paiement, Jean-François, Douglas Eck, and Samy Bengio (2005). "A probabilistic model for chord progressions." In: *Proceedings of the Sixth International Conference on Music Information Retrieval (ISMIR)*. EPFL-CONF-83178.

Paine, Tom Le, Pooya Khorrami, Shiyu Chang, Yang Zhang, Prajit Ramachandran, Mark A Hasegawa-Johnson, and Thomas S Huang (2016). "Fast wavenet generation algorithm." In: *arXiv preprint arXiv:1611.09482*.

Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio (2013). "On the difficulty of training recurrent neural networks." In: *ICML (3)* 28, pp. 1310–1318.

Paulus, Jouni, Meinard Müller, and Anssi Klapuri (2010). "State of the Art Report: Audio-Based Music Structure Analysis." In: *ISMIR*, pp. 625–636.

Peeters, Geoffroy (2007). "Sequence Representation of Music Structure Using Higher-Order Similarity Matrix and Maximum-Likelihood Approach." In: *ISMIR*, pp. 35–40.

Pen, Ronald (1992). *Introduction to Music. New York: McGraw-Hili*.

Raffel, C., B. McFee, E. J. Humphrey, J. Salamon, O. Nieto, D. Liang, D. P. W. Ellis, and C. C. Raffel (2014). "mir_eval: A transparent implementation of common MIR metrics." In: *Proceedings of the 15th International Society for Music Information Retrieval Conference, ISMIR*.

Rehding, A. (2003). *Hugo Riemann and the birth of modern musical thought*. Vol. 11. Cambridge University Press.

Roads, Curtis, Stephen Travis Pope, Aldo Piccialli, and Giovanni De Poli (2013). *Musical signal processing*. Routledge.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1986). "Learning representations by back-propagating errors." In: *nature* 323.6088, pp. 533–536.

Sandresky, Margaret Vardell (1979). "The continuing concept of the Platonic-Pythagorean system and its application to the analysis of fifteenth-century music." In: *Music Theory Spectrum* 1, pp. 107–120.

Schoenberg, Arnold and Leonard Stein (1969). *Structural functions of harmony*. 478. WW Norton & Company.

Scholz, Ricardo, Emmanuel Vincent, and Frédéric Bimbot (2009). "Robust modeling of musical chord sequences using probabilistic N-grams." In: *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*. IEEE, pp. 53–56.

Shannon, Claude Elwood (1949). "Communication in the presence of noise." In: *Proceedings of the IRE* 37.1, pp. 10–21.

Singh, Aarti, Robert Nowak, and Xiaojin Zhu (2009). "Unlabeled data: Now it helps, now it doesn't." In: *Advances in neural information processing systems*, pp. 1513–1520.

Sneddon, Ian Naismith (1995). *Fourier transforms*. Courier Corporation.

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov (2014). "Dropout: a simple way to prevent neural networks from overfitting." In: *The journal of machine learning research* 15.1, pp. 1929–1958.

Sturm, Bob L, Oded Ben-Tal, Úna Monaghan, Nick Collins, Dorien Herremans, Elaine Chew, Gaëtan Hadjeres, Emmanuel Deruty, and François Pachet (2019). "Machine learning research that matters for music creation: A case study." In: *Journal of New Music Research* 48.1, pp. 36–55.

Sturm, Bob L, João Felipe Santos, Oded Ben-Tal, and Iryna Korshunova (2016). "Music transcription modelling and composition using deep learning." In: *arXiv preprint arXiv:1604.08723*.

Sutskever, Ilya, Oriol Vinyals, and Quoc V Le (2014). "Sequence to sequence learning with neural networks." In: *Advances in neural information processing systems*, pp. 3104–3112.

Tatar, Kıvanç and Philippe Pasquier (2017). "MASOM: A musical agent architecture based on self organizing maps, affective computing, and variable Markov models." In: *Proceedings of the 5th International Workshop on Musical Metacreation (MUME 2017). Atlanta, Georgia, USA*.

Tsushima, Hiroaki, Eita Nakamura, Katsutoshi Itoyama, and Kazuyoshi Yoshii (2018). "Generative statistical models with self-emergent grammar of chord sequences." In: *Journal of New Music Research* 47.3, pp. 226–248.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin (2017). "Attention is all you need." In: *Advances in Neural Information Processing Systems*, pp. 5998–6008.

Williams, Ronald J and David Zipser (1989). "A learning algorithm for continually running fully recurrent neural networks." In: *Neural computation* 1.2, pp. 270–280.

Wu, Y., X. Feng, and W. Li (2017). "Mirex 2017 submission: Automatic audio chord recognition with miditrained deep feature and blstm-crf sequence decoding model." In: *MIREX evaluation results*.

Wu, Yiming, Tristan Carsault, Eita Nakamura, and Kazuyoshi Yoshii (2020). "Semi-supervised Neural Chord Estimation Based on a Variational Autoencoder with Latent Chord Labels and Features." In: *IEEE Transactions on Audio, Speech and Language Processing*.

Yoshii, Kazuyoshi and Masataka Goto (2011). "A Vocabulary-Free Infinity-Gram Model for Nonparametric Bayesian Chord Progression Analysis." In: *Proceedings of ISMIR*.

Young, Tom, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria (2018). "Recent trends in deep learning based natural language processing." In: *ieee Computational intelligenCe magazine* 13.3, pp. 55–75.

Zhou, X. and A. Lerch (2015). "Chord detection using deep learning." In: *Proceedings of the 16th International Symposium on Music Information Retrieval Conference*. Vol. 53.

Zhu, Xiaojin and Andrew B Goldberg (2009). "Introduction to semi-supervised learning." In: *Synthesis lectures on artificial intelligence and machine learning* 3.1, pp. 1–130.

van den Oord, Aaron, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu (2016). "Wavenet: A generative model for raw audio." In: *arXiv preprint arXiv:1609.03499*.

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede and Ivo Pletikosić.