DYCI2

Dynamiques créatives de l'interaction improvisée -- Creative Dynamics of Improvised Interaction

ANR-14-CE24-0002-01

Projet DYCI2, WP1 Écoute informée créative, SP1.2 Écoute structurante et prédictive

Rapport de livrable :

L1.2 Écoute structurante et prédictive

Livrable	Date	Contributeurs	Rédacteurs	Contenu
L3.3.1	Septembre 2018	T. Carsault (Ircam-STMS), J. Nika (Univ. La Rochelle / Ircam-STMS), P. Esling (Ircam-STMS)	T. Carsault (Ircam-STMS), J. Nika (Univ. La Rochelle / Ircam-STMS)	Maquette Logicielle, Rapport scientifique

Résumé

Ce document présente les résultats de recherche et le développement de prototypes portant sur l'écoute, la découverte, et la prédiction de structures dans un flux audio en temps réel. Le cas d'application de nos prototypes est l'alphabet des labels d'accords. Des recherches sur des modèles d'extraction d'accords et des modèles d'inférence de séquences d'accords à l'aide d'apprentissage profond ont été menées conjointement et ont abouti à une plateforme de test en langage Python, ainsi qu'un interfaçage avec l'environnement Max pour les modèles les plus performants.

Adresse du livrable logiciel

DYCI2_WP1.2_L.1.2.zip
sur
https://forge.ircam.fr/p/Dyci2/

Notre objectif est ici de concevoir un système permettant de détecter des structures musicales dans un flux audio et de prédire localement les structures futures en temps réel. La première version de ce système s'intéresse à l'extraction automatique de labels d'accords et la génération de séquences de labels d'accords en temps réel pour la prédiction. Le processus se décompose en deux étapes : l'écoute et la prédiction. Le module d'écoute permet d'extraire la structure musicale jouée par le musicien, alors que le module de prédiction est un générateur de séquences musicales symboliques.

Module d'écoute

L'extraction automatique d'accords est étudiée depuis plusieurs décennies et a notamment profité des avancées de l'apprentissage machine. En effet, les systèmes les plus performants utilisent des réseaux de neurones pour extraire les accords à partir du flux audio. Pourtant, ce champ de recherche a été confronté à un effet de plafonnement depuis plusieurs années, ce qui incite à explorer d'autres pistes que l'amélioration intrinsèque des modèles, par exemple la représentation des données et l'utilisation de connaissances a priori sur les labels et les relations entre eux.
Nous avons conçu nos propres détecteurs d'accords en employant des méthodes d'apprentissage machine basées sur la structure de *Convolutional Neural Network*. L'apprentissage de ces réseaux a été réalisée sur un jeu de données où chaque trame audio est annotée par un label d'accord (dataset http://isophonics.org). Afin d'améliorer les résultats de l'état de l'art, nous avons introduit des distances musicales entre les différents labels d'accords dans l'apprentissage. Nous avons réalisé des entraînements en faisant varier différents paramètres comme la distance musicale entre les labels d'accords, ou la précision de l'alphabet d'accords pour étudier les influences croisées de ces différents paramètres. Dans un second temps, nous avons mené une étude inédite des erreurs de classification des modèles les plus répandus d'un point de vue qualitatif en se basant sur la théorie fonctionnelle de l'harmonie. Nos résultats montrent que l'utilisation des distances musicales proposées permet d'améliorer les résultats de l'état de l'art aussi bien qualitativement (équivalence fonctionnelle de certains accords) que quantitativement (score de classification).
Ces recherches ont mené à la publication d'un article pour la conférence ISMIR 2018 joint à la suite de ce rapport. Le code fourni permet de sélectionner un modèle d'apprentissage, une distance musicale ainsi qu'un alphabet, de réaliser l'entraînement du modèle, et de comparer un ensemble de modèles préalablement entraînés.

Module de prédiction

La prédiction de séquences d'accords symboliques a été étudiée de manière comparative en explorant différentes méthodes statistiques dont les modèles de Markov cachés et les *Recurrent Neural Network*s. Dans les deux approches, l'apprentissage a été réalisé sur une base de donnée regroupant des séquences d'accords cohérentes (Dataset comprenant les différents Realbooks https://github.com/keunwoochoi/lstm_real_book). Nos résultats préliminaires indiquent une meilleure qualité de prédiction avec des réseaux de neurones.
Le rapport de stage ATIAM de Tristan Carsault qui développe l'idée générale de la détection d'accords jumelée avec de la prédiction temps-réelle de séquences d'accords symboliques est fourni à la suite de ce rapport.
Le code fourni permet de réaliser l'apprentissage sur les deux grandes familles de modèles. Une partie du code permet de tester la performance des différents modèles.

Associer écoute et prédiction : code source et interface Max

Afin de joindre ces deux modèles, un patch Max offre une interface utilisateur. Tout d'abord, les accords sont extraits d'un flux audio par le module d'écoute à chaque pulsation, et affichés en temps réel. Ces accords extraits sont alors envoyés au module de prédiction à chaque étape, et le patch Max affiche à chaque pulsation la séquence prédite pour les prochaines mesures.

Les codes sources et un fichier README sont fournis pour chacune des taches. */ismir2018_clean* et */HMMLSTM_clean* contiennent respectivement les fichiers d'entraînement des modèles d'extraction d'accords et de génération/prédiction d'accords. Le dossier */OscChordLSTM_clean* contient le patch Max et le serveur OSC permettant la communication avec les modèles développés en Python. Les bases de données sont disponibles sur demande, et des modèles pré-entraînés sont fournis avec le code source (un modèle d'extraction d'accords majeurs/mineurs entraîné avec une distance « tonnetz », et un modèle de prédiction basée sur des LSTM).

USING MUSICAL RELATIONSHIPS BETWEEN CHORD LABELS IN AUTOMATIC CHORD EXTRACTION TASKS

Tristan Carsault[1] **Jérôme Nika**[2,1] **Philippe Esling**[1]
[1]Ircam, CNRS, Sorbonne Université, UMR 9912 STMS, [2]L3i Lab, University of La Rochelle
{carsault, jnika, esling}@ircam.fr

ABSTRACT

Recent research on Automatic Chord Extraction (ACE) has focused on the improvement of models based on machine learning. However, most models still fail to take into account the prior knowledge underlying the labeling alphabets (chord labels). Furthermore, recent works have shown that ACE performances have reached a glass ceiling. Therefore, this prompts the need to focus on other aspects of the task, 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.

In this paper, we propose to exploit specific properties and relationships between chord labels in order to improve the learning of statistical ACE models. Hence, we analyze the interdependence of the representations of chords and their associated distances, the precision of the chord alphabets, and the impact of performing alphabet reduction before or after training the model. Furthermore, we propose new training losses based on musical theory. We show that these improve the results of ACE systems based on Convolutional Neural Networks. 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.

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, recent results seem to indicate that the rate of improvement of ACE performances has diminished over the past years [20].

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 [10]. Our 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.

Most ACE systems are built on the idea of extracting features from the raw audio signal and then using these features to construct a chord classifier [4]. The two major families of approaches that can be found in previous research are *rule-based* and *statistical* models. On one hand, the rule-based models rely on music-theoretic rules to extract information from the precomputed features. Although this approach is theoretically sound, it usually remains brittle to perturbations in the spectral distributions from which the features were extracted. On the other hand, statistical models rely on the optimization of a loss function over an annotated dataset. However, the generalization capabilities of these models are highly correlated to the size and completeness of their training set. Furthermore, most training methods see musical chords as independent labels and do not take into account the inherent relations between chords.

In this paper, we aim to target this gap by introducing musical information directly in the training process of statistical models. To do so, we propose to use prior knowledge underlying the labeling alphabets in order 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, which provides the current best results in ACE [19]. First, we study the impact of chord alphabets and their relationships by introducing a specific *hierarchy* of alphabets. We show that some of the reductions proposed by previous researches might be inadequate for learning algorithms. We also show that relying on more finely defined and extensive alphabets allows 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 be used to define novel loss functions for learning algorithms. We show that these new loss functions improve ACE results with CNNs. Finally, we perform an extensive analysis of our approach and extract insights on the methodology required for ACE. To do so, we develop a specifically-tailored analyzer that focuses on the functional relations between chords to distinguish *strong* and *weak* errors. This analyzer is intended to be used for future ACE research to develop a finer understanding on the reasons behind the success or failure of ACE systems.

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, chords are seen as the concomitant or successive combination of different notes played by one or many instruments.

2.1 Considerations on the ACE task

Whereas most MIR tasks have benefited continuously from the recent advances in deep learning, the ACE field seems to have reached a glass ceiling. In 2015, Humphrey and Bello [10] 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 misclassifications on the concomitant semi-tones. Moreover, chord labels are not always well suited to describe every song in these datasets.

Second, the chord labels are related and some subsets of those 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. First, the literal *recognition* of a local audio segment using a chord label and its precise extensions, and, second, the *transcription of an underlying harmony*, taking into account the functional aspect of the chords and the long-term structure of the song. 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 of chords, the improvement of the models towards more complex chord alphabets and the development of more adapted evaluation methods.

2.2 Workflow of ACE systems

Due to the complexity of the task, ACE systems are usually divided into four main modules performing *feature extraction*, *pre-filtering*, *pattern matching* and *post-filtering* [4].

First, the *pre-filtering* usually applies low-pass filters or harmonic-percussive source separation methods on the raw signal [12, 26]. 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 class vector called *chromagram*. 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 [13].

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 [21]. With these methods, the extracted features are classified using a fixed dictionary of chord profiles [2] or a collection of decision trees [12]. However, these methods are usually brittle to perturbations in the input spectral distribution and do not generalize well.

Statistical models aim to extract the relations between precomputed features and chord labels based on a training dataset in which each temporal frame is associated to a label. The optimization of this 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 [3] and convolutional [9, 14] or recurrent [1, 25] Neural Networks.

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 [17] or with Conditional Random Fields [15].

2.3 Convolutional Neural Network

A Convolutional Neural Network (CNN) is a statistical model composed of layers of artificial neurons that transform the input by repeatedly applying convolution and pooling operations. A convolutional layer is characterized by a set of convolution kernels that are applied in parallel to the inputs to produce a set of output *feature maps*. The convolution kernels are defined as three-dimensional tensors $h \in \mathbb{R}^{M \times U \times V}$ where M is the number of kernels, U is the height and V the width of each kernel. If we note the input as matrix X, then the output feature maps are defined by $Y = X * h_m$ for every kernels, where $*$ is a 2D discrete convolution operation

$$(A * B)_{i,j} = \sum_{r=0}^{(T-1)} \sum_{s=0}^{(F-1)} A_{r,s} B_{i-r,j-s} \qquad (1)$$

for $A \in \mathbb{R}^{T \times F}$ and $B \in \mathbb{R}^{U \times V}$ with $0 \leq i \leq T+U-1$ and $0 \leq j \leq F+V-1$.

As this convolutional layer significantly increases the dimensionality of the input data, a pooling layer is used to reduce the size of the feature maps. The pooling operation reduces the maps by computing local mean, maximum or average of sliding context windows across the maps. Therefore, the overall structure of a CNN usually consists in alternating convolution, activation and pooling layers. Finally, in order to perform classification, this architecture

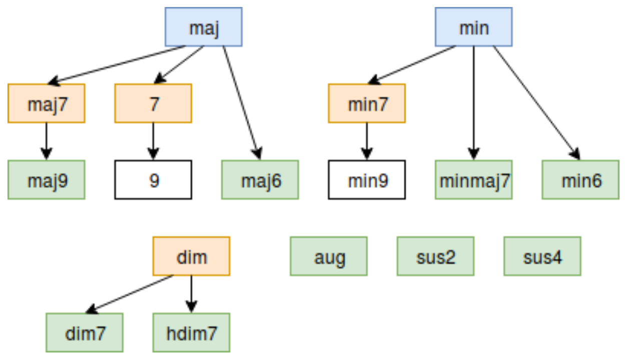


Figure 1. Hierarchy of the chord alphabets (blue: A_0, orange: A_1, green: A_2)

is typically followed by one or many fully-connected layers. Thus, the last layer produces a probability vector of the same size as the chord alphabet. As we will rely on the architecture defined by [9], we redirect interested readers to this paper for more information.

3. OUR PROPOSAL

3.1 Definition of alphabets

Chord annotations from reference datasets are very precise and include extra notes (in parenthesis) and basses (after the slash) [7]. With this notation, we would obtain over a thousand chord classes with very sparse distributions. However, we do not use these extra notes and bass in our classification. Therefore, we can remove this information

$$F : maj7(11)/3 \rightarrow F : maj7 \tag{2}$$

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 1). In order to reduce the number of chord qualities, each one is mapped to a parent class when it exists, otherwise to the *no-chord* class N.

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

$$A_0 = \{N\} \cup \{P \times maj, min\} \tag{3}$$

where P represents the 12 pitch classes.

Here, we consider the interest of working with chord alphabets larger than A_0. Therefore, we propose an alphabet containing all chords present in the harmonization of the major scale (usual notation of harmony in jazz music). This corresponds to the orange chord qualities and their parents in Figure 1. The chord qualities without heritage are included in the no-chord class N, leading to 73 classes

$$A_1 = \{N\} \cup \{P \times maj, min, dim, maj7, min7, 7\} \tag{4}$$

Finally, the alphabet A_2 is inspired from the large vocabulary alphabet proposed by [19]. This most complete chord alphabet contains 14 chord qualities and 169 classes

$$A_2 = \{N\} \cup \{P \times maj, min, dim, aug, maj6, min6, maj7, minmaj7, min7, 7, dim7, hdim7, sus2, sus4\} \tag{5}$$

3.2 Definition of chord distances

In most CNN approaches, the model does not take into account the nature of each class when computing their differences. Therefore, this distance which we called categorical distance D_0 is the binary indicator

$$D_0(chord_1, chord_2) = \begin{cases} 0 & \text{if } chord_1 = chord_2 \\ 1 & \text{if } chord_1 \neq chord_2 \end{cases} \tag{6}$$

However, we want here to include the relationships between chords directly in our model. For instance, a *C:maj7* is closer to an *A:min7* than a *C#:maj7*. Therefore, we introduce more refined distances that can be used to define the loss function for learning.
Here, we introduce two novel distances that rely on the representation of chords in an harmonic space or in a pitch space to provide a finer description of the chord labels. However, any other distance that measure similarities between chords could be studied [8, 18].

3.2.1 *Tonnetz distance*

A *Tonnetz-space* is a geometric representation of the tonal space based on harmonic relationships between chords. We chose a Tonnetz-space generated by three transformations of the major and minor triads [5] changing only one of the three notes of the chords: the *relative transformation* (transforms a chord into his relative major / minor), the *parallel transformation* (same root but major instead of minor or conversely), the *leading-tone exchange* (in a major chord the root moves down by a semitone, in a minor chord the fifth moves up by a semitone). Representing chords in this space has already shown promising results for classification on the A_0 alphabet [11].

We define the cost of a path between two chords as the sum of the succesive transformations. Each transformation is associated to the same cost. Furthermore, an extra cost is added if the chords have been reduced beforehand in order to fit the alphabet A_0. Then, our distance D_1 is:

$$D_1(chord_1, chord_2) = min(C) \tag{7}$$

with C the set of all possible path costs from $chord_1$ to $chord_2$ using a combination of the three transformations.

3.2.2 *Euclidean distance on pitch class vectors*

In some works, pitch class vectors are used as an intermediate representation for ACE tasks [16]. 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 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.

$$D_2(chord_1, chord_2) = \sqrt{\sum_{i=0}^{11}(chord_1^i - chord_2^i)^2} \quad (8)$$

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 to define the loss function for training the CNN classification model.

3.3 Introducing the relations between chords

To train the model with our distances, we first reduce the original labels from the Isophonics dataset [1] so that they fit one of our three alphabets A_0, A_1, A_2. Then, we denote y_{true} as the one-hot vector where each bin corresponds to a chord label in the chosen alphabet A_i. The output of the model, noted y_{pred}, is a vector of probabilities over all the chords in a given alphabet A_i. In the case of D_0, we train the model with a loss function that simply compares y_{pred} to the original label y_{true}. However, for our proposed distances (D_1 and D_2), we introduce a similarity matrix M that associates each couple of chords to a similarity ratio.

$$M_{i,j} = \frac{1}{D_k(chord_i, chord_j) + K} \quad (9)$$

K 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 $y_{t\bar{r}ue}$ which is the matrix multiplication of the old y_{true} and the normalized matrix $\bar{M}$.

$$y_{t\bar{r}ue} = y_{true}\bar{M} \quad (10)$$

Finally, the loss function for D_1 and D_2 is defined by a comparison between this new ground truth $y_{t\bar{r}ue}$ and the output y_{pred}. Hence, this loss function can be seen as a weighted multi-label classification.

4. EXPERIMENTS

4.1 Dataset

We perform our experiments on the *Beatles* dataset as it provides the highest confidence regarding the ground truth annotations [6]. This dataset is composed by 180 songs annotated by hand. For each song, we compute the CQT by using a window size of 4096 samples and a hop size of 2048. 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.

[1] `http://isophonics.net/content/reference-annotations-beatles`

4.2 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 [9]. However, we add dropout between each convolution layer to prevent over-fitting.

For 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 we 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.

5. RESULTS AND DISCUSSION

The aim of this paper is not to obtain the best classification scores (which would involve pre- or post-filtering methods) but to study the impact on the classification results of different musical relationships (as detailed in the previous section). Therefore, we ran 9 instances of the CNN model corresponding to all combinations of the 3 alphabets A_0, A_1, A_2 and 3 distances D_0, D_1, D_2 to compare their results from both a *quantitative* and *qualitative* point of view. We analyzed the results using the *mireval* library [22] to compute classification scores, and a Python *ACE Analyzer* that we developed to reveal the musical meaning of classification errors and, therefore, understand their qualities.

5.1 Quantitative analysis: MIREX evaluation

Regarding the MIREX evaluation, the efficiency of ACE models is assessed through classification scores over different alphabets [22]. 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 alphabets (Major/Minor $\sim A_0$, Sevenths $\sim A_1$, Tetrads $\sim A_2$).

5.1.1 MIREX Major/minor

Figure 2 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 D_2 distance improves the classification 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 but an extended dataset in [10] 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 Maj/Min (only from a quantitative point of view, see 5.2).

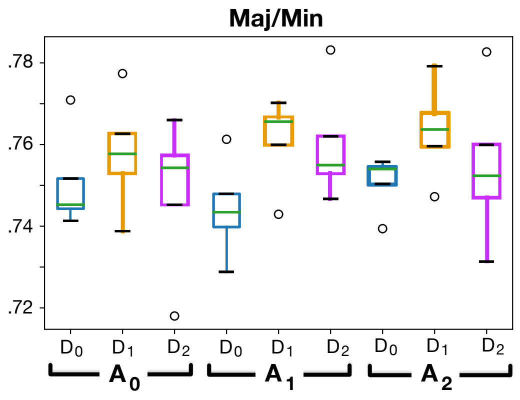


Figure 2. Results of the 5-folds: evaluation on MIREX Maj/Min ($\sim$ reduction on A_0).

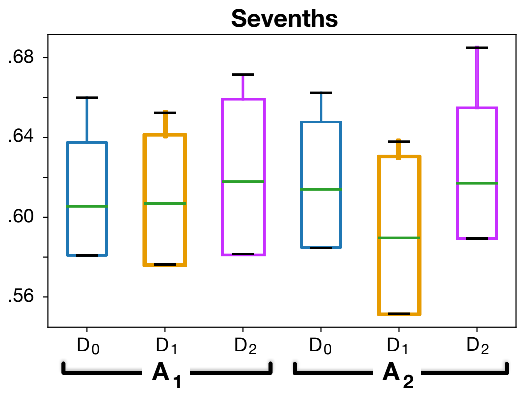


Figure 3. Results of the 5-folds: evaluation on MIREX Sevenths ($\sim$ reduction on A_1).

5.1.2 MIREX Sevenths

With more complex alphabets, the classification score is lower than for MIREX Maj/Min. 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 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.

5.2 Qualitative analysis: understanding the errors

In this section, we propose to analyze ACE results from a qualitative point of view. The aim here is not to introduce new alphabets or distances in the models, but to introduce a new type of evaluation of the results. 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 [23, 24]. Therefore, we developed an *ACE Analyzer* including two modules discovering some formal musical relationships between the target chords and the chords predicted by ACE models. Both modules are generic and independent of the classification model, and are available online. [2]

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 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).

Model	*rel. M*	*rel. m*	*T subs. 2*	$m \to M$	$M \to m$
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 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).

5.2.1 Substitution rules

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

Table 1 presents: *Tot.*, the total fraction of errors that can be explained by the whole set of substitution rules we implemented, and $\subset$ *Maj* and $\subset$ *min*, the errors included in the correct triad (*e.g.* *C:maj* instead of *C:maj7*, *C:min7* instead of *C:min*). Table 2 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 $m \to M$ and $M \to m$ (same root but major instead of minor or conversely). The tables only show the categories representing more than 1% of the total number of errors, but other substitutions (that are not discussed here) were analyzed: tritone substitution, substitute dominant, and equivalence of *dim7* chords modulo inversions.

First, *Tot.* in Table 1 shows that a huge fraction of errors can be explained by usual substitution rules. This percent-

[2] `http://repmus.ircam.fr/dyci2/ace_analyzer`

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 3. Errors occurring when the target is non-diatonic (% of the total number of errors), non-diatonic prediction errors (% of the subset of 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	3.50
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 4. Errors ($> 2\%$) corresponding to degrees substitutions (% of the subset of errors on diatonic targets).

age can reach 60.47%, which means that numerous classification errors nevertheless give useful indications since they mistake a chord for another chord with an equivalent function. For instance, Table 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 1) increases with the size of the alphabet: larger alphabets seem to imply weaker errors (higher amount of equivalent harmonic functions).

We can also note that numerous errors (between 28.19% and 37.77%) correspond to inclusions in major or minor chords ($\subset$ *Maj* and $\subset$ *min*, Table 1) for A_1 and A_2. In the framework of the discussion about *recognition* and *transcription* mentioned in introduction, this result questions the relevance of considering exhaustive extensions when the goal is to extract and formalize an underlying harmony.

Finally, for A_0, A_1, and A_2, using D_1 instead of D_0 increases the fraction of errors attributed to categories in the left part of Table 2 (and in almost all the configurations when using D_2). This shows a qualitative improvement since all these operations are considered as valid chord substitutions. On the other hand, the impact on the (quite high) percentages in the right part of Table 2 is not clear. We can assume that temporal smoothing can be one of the keys to handle the errors *m→M* and *M→m*.

5.2.2 *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 that of chords, this module determines the roman numerals characterizing the harmonic degrees of the predicted chord and of the target chord (*e.g.* in C, if a chord is an extension of *C*, *I*; if it is an extension of *D:min*, *ii*; etc.) when it is possible (*e.g.* in C, if a chord is an extension of *C#* it does not correspond to any degree). Then, it counts the errors corresponding to substitutions of harmonic degrees when it is possible (*e.g.* in C, *A:min* instead of *C* corresponds to *I∼vi*). This section shows an analysis of the results using this second module. First, it determines if the target chord is diatonic (*i.e.* belongs to the harmony of the key), as presented in Table 3. If this is the case, the notion of incorrect degree for the predicted chord is relevant and the percentages of errors corresponding to substitutions of degrees is computed (Table 4).

A first interesting fact presented in Table 3 is that 36.99% 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 non-diatonic errors decrease (Table 3, particularly A_0), which means that the errors are more likely to stay in the correct key. Surprisingly, 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 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 musical distances (particularly D_2). Conversely, it increases the amount of errors in the right part of Table 4 containing usual substitutions: once again we observe that the more precise the musical representations are, the more the harmonic functions tend to be correct.

6. CONCLUSION

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 train the same type of model. Then, we conducted 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 significantly improve the classification scores but provides a qualitative improvement in the type of 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 encourages to adopt a functional approach or even to introduce a notion of equivalence classes. 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.

7. ACKNOWLEDGMENTS

The authors would like to thank the master's students who contributed to the implementation: Alexis Font, Grégoire Locqueville, Octave Roulleau-Thery, and Téo Sanchez. This work was supported by the DYCI2 project ANR-14-CE2 4-0002-01 funded by the French National Research Agency (ANR), the MAKIMOno project 17-CE38-0015-01 funded by the French ANR and the Canadian Natural Sciences and Engineering Reserch Council (STPG 507004-17), the ACTOR Partnership funded by the Canadian Social Sciences and Humanities Research Council (895-2018-1023).

8. REFERENCES

[1] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent. Audio chord recognition with recurrent neural networks. In *International Symposium on Music Information Retrieval*, pages 335–340, 2013.

[2] C. Cannam, E. Benetos, M. Mauch, M. E. P. Davies, S. Dixon, C. Landone, K. Noland, and D. Stowell. Mirex 2015: Vamp plugins from the centre for digital music. *In Proceedings of the Music Information Retrieval Evaluation eXchange (MIREX)*, 2015.

[3] T. Cho. *Improved techniques for automatic chord recognition from music audio signals*. PhD thesis, New York University, 2014.

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

[5] R. Cohn. Neo-riemannian operations, parsimonious trichords, and their" tonnetz" representations. *Journal of Music Theory*, 41(1):1–66, 1997.

[6] C. Harte. *Towards automatic extraction of harmony information from music signals*. PhD thesis, 2010.

[7] C. Harte, M. B. Sandler, S. A. Abdallah, and E. Gómez. Symbolic representation of musical chords: A proposed syntax for text annotations. In *International Symposium on Music Information Retrieval*, volume 5, pages 66–71, 2005.

[8] C-Z. A. Huang, D. Duvenaud, and K. Z. Gajos. Chordripple: Recommending chords to help novice composers go beyond the ordinary. In *Proceedings of the 21st International Conference on Intelligent User Interfaces*, pages 241–250. ACM, 2016.

[9] E. J. Humphrey and J. P. Bello. Rethinking automatic chord recognition with convolutional neural networks. In *Machine Learning and Applications (ICMLA), 2012 11th International Conference on*, volume 2, pages 357–362. IEEE, 2012.

[10] E. J. Humphrey and J. P. Bello. Four timely insights on automatic chord estimation. In *International Symposium on Music Information Retrieval*, pages 673–679, 2015.

[11] E. J. Humphrey, T. Cho, and J. P. Bello. Learning a robust tonnetz-space transform for automatic chord recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 453–456. IEEE, 2012.

[12] J. Jiang, W. Li, and Y. Wu. Extended abstract for mirex 2017 submission: Chord recognition using random forest model. *MIREX evaluation results*, 2017.

[13] F. Korzeniowski and G. Widmer. Feature learning for chord recognition: The deep chroma extractor. *arXiv preprint arXiv:1612.05065*, 2016.

[14] F. Korzeniowski and G. Widmer. A fully convolutional deep auditory model for musical chord recognition. In *Machine Learning for Signal Processing (MLSP), 2016 IEEE 26th International Workshop on*, pages 1–6. IEEE, 2016.

[15] J. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[16] K. Lee. Automatic chord recognition from audio using enhanced pitch class profile. In *International Computer Music Conference*, 2006.

[17] H-L. Lou. Implementing the viterbi algorithm. *IEEE Signal Processing Magazine*, 12(5):42–52, 1995.

[18] S. Madjiheurem, L. Qu, and C. Walder. Chord2vec: Learning musical chord embeddings. In *Proceedings of the Constructive Machine Learning Workshop at 30th Conference on Neural Information Processing Systems (NIPS2016), Barcelona, Spain*, 2016.

[19] B. McFee and J. P. Bello. Structured training for large-vocabulary chord recognition. In *Proceedings of the 18th International Society for Music Information Retrieval Conference (ISMIR2017). ISMIR*, 2017.

[20] M. McVicar, R. Santos-Rodríguez, Y. Ni, and T. De Bie. Automatic chord estimation from audio: A review of the state of the art. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 22(2):556–575, 2014.

[21] L. Oudre, Y. Grenier, and C. Févotte. Template-based chord recognition: Influence of the chord types. In *International Symposium on Music Information Retrieval*, pages 153–158, 2009.

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

[23] A. Rehding. *Hugo Riemann and the birth of modern musical thought*, volume 11. Cambridge University Press, 2003.

[24] A. Schoenberg and L. Stein. *Structural functions of harmony*. Number 478. WW Norton & Company, 1969.

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

[26] X. Zhou and A. Lerch. Chord detection using deep learning. In *Proceedings of the 16th International Symposium on Music Information Retrieval Conference*, volume 53, 2015.

AUTOMATIC CHORD EXTRACTION AND MUSICAL STRUCTURE PREDICTION THROUGH SEMI-SUPERVISED LEARNING

Application to human-computer improvisation

TRISTAN CARSAULT

Supervised by Philippe Esling and Jérôme Nika

IRCAM, UMR STMS 9912
Paris 75004, France
Musical Representations Team

Master's degree
SAR - ATIAM
Faculty of Computer Science
Université Pierre-et-Marie Curie / IRCAM / Télécom ParisTech
February-August 2017

ABSTRACT

Human computer co-improvisation aims to rely on a computer in order to produce a musical accompaniment to a musician's improvisation. Recently, the notion of guidance has been introduced to enhance the process of human computer co-improvisation. Although this concept has already been studied with a step-by-step guidance or by guiding with a formal temporal structure, it is usually only based on a past memory of events. This memory is derived from an annotated corpus which limits the possibility to infer the potential future improvisation structure. Nevertheless, most improvisations are based on long-term structures or grids. Our study intends to target these aspects and provide short term predictions of the musical structures to improve the quality of the computer co-improvisation.

Our aim is to develop a software that interacts in real-time with a musician by inferring expected structures. In order to achieve this goal, we divide 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 these extractions.

In this report, we present a first approach towards this goal by introducing an automatic chord extraction module and a chord label sequence generator. Regarding the structure extraction, as the current state-of-the-art results in automatic chord extraction are obtained with Convolutional Neural Networks (CNN), we first study new architectures derived from the CNNs applied to this task. However, as we underline in our study, the low quantity of audio labeled dataset could limit the use of machine learning algorithms. Hence, we also propose the use of Ladder Networks (LN) which can be trained in a semi-supervised way. This allows us to evaluate the use of unlabeled music data to improve labeled chord extraction. Regarding the chord label generator, many recent works showed the success of Recurrent Neural Networks (RNN) for generative temporal applications. Thus, we use a family of recurrent networks, the Long Short-Term Memory (LSTM) unit, for our generative task.

Here, we present our implementations and the results of our models by comparing to the current state-of-the-art and show that we obtain comparable results on the seminal evaluation datasets. Finally, we introduce the overall architecture of the software linking both modules and propose some directions of future work.

RÉSUMÉ

La co-improvisation homme-machine a comme but l'utilisation d'un ordinateur dans l'accompagnement musical d'un musicien lors d'une improvisation. Les technologies existantes en improvisation homme-machine permettent un guidage pas-à-pas ou un guidage basé sur des formalismes structuraux pré-enregistrés. Ces structures calculées sont issues de corpus annotés, ce qui limite grandement la génération de structures improvisées construites sur un potentiel futur. Ainsi, notre logiciel aspire à combiner ces deux aspects en proposant à chaque pas des séquences de symboles en cohérence avec la progression des structures renseignées par le musicien.

Notre but est de concevoir un programme pouvant interagir en temps-réel avec un musicien en inférant des structures musicales futures. Cette conception peut se décomposer en deux étapes : un *module d'écoute* et un *module d'improvisation*. Le module d'écoute permet d'extraire la structure musicale jouée par le musicien, alors que le module d'improvisation est un générateur de séquences musicales symboliques.

Dans ce rapport, nous présentons les premières étapes du projet : l'extraction automatique d'accords musicaux et la génération de séquences de labels d'accords. A ce jour, les meilleurs algorithmes d'extraction automatique d'accords intègrent un réseau de neurones convolutif. Ainsi, nous avons étudié ces modèles pour la tache d'éxtraction automatique d'accords. Cependant, nous avons souligné durant notre étude la faible quantité de bases de données audio annotées. Nous avons donc étudié le ladder network, un modèle de réseau de neurones pouvant être entraîné de façon semi-supervisée. Cet entraînement semi-supervisé nous permet de paramétrer notre modèle aussi bien avec des données labellisées qu'avec des données non labellisées. D'autre part, plusieurs logiciels de génération musicale ont été récemment développés avec des réseaux de neurones récurrents. C'est pourquoi, nous utilisons une catégorie de réseau de neurones récurrents, les long short term memory.

Ici, nous présentons nos implémentations ainsi que les résultats de nos modèles en les comparant à l'état de l'art actuel et en montrant que nous obtenons des résultats comparables sur les bases de données de références. Finalement, nous introduisons l'architecture globale du logiciel reliant les deux modules et nous proposons des directions pour des études ultérieures.

ACKNOWLEDGEMENTS

This internship at IRCAM within the Music Representations team was a priceless experience. First of all, I would like to express my gratitude to my tutors Philippe Esling and Jérôme Nika. Due to their wisdom on interdependent topics, these two persons were absolutely complementary for a great supervision.

Thanks to the Music Representations team, Gérard Assayag, Carlos Agon, Jean Bresson, Jean-Louis Giavitto, Léopold Crestel, Axel Chemla-Romeu-Santos and Karim Haddad, for their kind advices, their sharing of knowledge and their daily happiness.

Thanks to the ATIAM pedagogical team for their implication in the development of their students.

Thanks to the ATIAM students for this unforgettable moments spent together.

A special thanks to my family and friends for their indefectible support.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

MIR	Music Information Retrieval
MIREX	Music Information Retrieval Evaluation eXchange
ACE	Automatic Chord Extraction
SGD	Stochastic Gradient Descent
RBM	Restricted Boltzmann Machine
DBN	Dynamic Bayesian Networks
HMM	Hidden Markov Model
NN	Neural Network
MLP	Multi-Layer Perceptron
CNN	Convolutional Neural Network
LN	Ladder Network
RNN	Recurrent Neural Network
LSTM	Long Short Term Memory

INTRODUCTION

The concept of *musical structure* could be defined as the arrangement and relations of musical elements through time. Furthermore, a piece of music has different levels of structure depending on the temporal scale that is under scrutiny. Indeed, elements of music such as notes (defined by their pitch, duration and timbre) can be combined into groups like chords, motifs and phrases and these, in turn, combine into larger structures such as chord progressions or choruses and verses. Thus, there can be complex and multi-scaled hierarchical and temporal relationships between the different musical elements. Given these complexities, musical structure have been given different meanings throughout the literature.

STRUCTURE: In this work, we define a structure as a sequence of symbols in a chosen alphabet (e.g. chord progression, profile of audio features), which describes the temporal evolution of a musical sequence. Hence, here we will not focus on other music information retrieval notions such as the high-level segmentation of an entire piece of music [43, 45].

Structures are inherent in music, and their knowledge becomes crucial in the process of improvisation involving multiple musicians. Indeed, even though improvisation is often associated with spontaneity and randomness, it is largely based on rules and structures that allow these different musicians to play together correctly. If we focus on blues or jazz improvisation, we can see that it generally relies on a chord progression defining a guideline for the overall performance. This chord progression is the backbone structure that the musicians will follow to develop their improvisations, while performing with others. Therefore, in a collective improvisation, an understanding of the current musical structures is critical.

Therefore, in this internship, we will target the development of intelligent listening modules that could emulate this process of musical structure discovery, as performed in real-time by a musician. To that end, we separate this task as first the ability to perform automatic chord extraction and then to understand higher-level musical progressions by performing chord prediction.

The remainder of this work is organized as follows. First, we present existing human-computer co-improvisation systems (Section 1) and motivate our choice to use chord extraction as the first step of our musical inference system. Then, we present the state-of-the-art in the Automatic Chord Extraction (ACE) field (Section 2). In the following part, we introduce the basic concepts of machine learning, neural networks and deep learning models (Section 3). Finally, we present the results of our chord extraction models (Section 4) and the architecture of our complete inference and co-improvisation model (Section 5).

1 MUSICAL MOTIVATION AND RELATED WORKS

1.1 GUIDANCE IN HUMAN-COMPUTER CO-IMPROVISATION

The motivation of this work comes from the field of interactive music generation, defined as "*music that is produced by a hardware and software system that importantly includes a computer*" [9]. Over the past years, many works have focused on the idea of relying on a computer algorithm to generate music. Here, we focus on the symbolic music without deep details on the sound synthesis. In these works, the definition of the musical structure underlying the generation is central to the success of these algorithms. Among them, we focus on human-computer co-improvisation processes. That is to say, as system that plays music with a musician in real time.

For instance, Omax [2] is a system that learns features specific to the musician's style in real-time, and then plays along with him interactively. This technology is based on Oracle's theory [1] and generates music with the help of an audio database that has been previously analyzed. Here, we focus on another family of software that retains a control and authoring on the musical output. Indeed, as introduced by Nika [36], human-computer co-improvisation processes can be enhanced through the notion of *guidance*. In this domain of *guided* human-computer co-improvisation, the notion of *guiding* music generation has two different meanings. On the one hand, *guiding* can be seen as a purely reactive and step-by-step process. This approach offers rich interaction possibilities but cannot take advantage of the prior knowledge of a temporal structure to introduce anticipatory behavior. On the other hand, *guiding* can mean defining the temporal structures or descriptions themselves, in order to drive the generation process of the whole musical sequence. These "scenario-based" systems are able to introduce anticipatory behaviours but require some prior knowledge about the musical context (a pre-defined scenario).

1.1.1 *"Guiding" step by step*

The step-by-step process aims to produce an automatic accompaniment using purely reactive mechanisms without prior knowledge. The musician input signal is analyzed in real-time and the system compares it to the corpus and select the most relevant music in order to generate an accompaniment.

For instance, SoMax [6] uses a previously annotated corpus and extracts in real time multimodal observations of the musician's playing. Then it retrieves the most relevant music slices from the corpus to generate an accompaniment.

Other software as VirtualBand [34] or Reflexive Looper [41], also relies on *feature based interaction*. Given the extracted features (e.g. RMS, spectral centroid or chroma) of the musician's audio signal, the software selects the audio musical accompaniment from the database .

1.1.2 *"Guiding" with a Formal Temporal Structure or Description*

Another approach is to see the guidance as constraints for the generation of complete musical sequences.

On the one hand, constraints can be used to preserve some structural patterns present in the database. In 2015 Herremans et al. built a system to generate bagana music [20], a traditional lyre from Ethiopia based on a first order Markov model. Another project using Markovian model is a model for corpus-based generative electronic dance music that has been proposed in 2013 by Eigenfeldt and Pasquier [15].

On the other hand, some research projects introduced a temporal specification to guide the music generation process. For instance, Donzé applies this concept in order to generate a monophonic solo similar to a given training melody upon a given chord progression [14]. Recently, the notion of *musical scenario* has been proposed [38] as the specification of the high-level musical structure that generated sequences should follow. This approach allows to define an overall direction to the music generation process. It introduces such motion or intention at each step of the generation.

SCENARIO: In our work, we define a scenario as a formalized temporal structure guiding the music generation.

An example of scenario-based system is ImproteK [37], which uses pattern-matching algorithms on symbolic sequences. Then, the symbolized inputs from the musician are compared with pre-defined scenarios. ImproteK is also reactive, as the scenario can be modified via a set of pre-coded rules or with parameters controlled in real-time via an external operator [39].

1.1.3 *Intelligent listening and inference of short-term scenarios*

In 1969, Schoenberg and Stein [49] formulated a fundamental distinction between progression and succession. From their point of view, a progression is aimed at a definite goal and is oriented towards its future, whereas a succession is a step-by-step conformity. The systems using a formal temporal structure or description consolidate this notion of progression developed by Schoenberg and Stein. Moreover, they introduce a notion of anticipatory behavior. However, this concept of scenario is limited to a pre-defined setup, which limits the capability of the algorithm to foresee any future movement or change inside the music. Moreover, Huron formalized the chain [25]

$$\text{"}expectation- > prediction- > anticipation\text{"} \tag{1}$$

Thus, the scenario is the temporal specification that takes the place of prediction and expectation. In other words, this chain becomes "specification -> anticipation" in scenario-based systems. Therefore, a promising direction would be the inference of short-term scenarios in order to obtain the entire chain (1).

The over-arching goal of interactive music systems is to emulate the behavior of a trained human musician. Hence, our aim is to design an intelligent listening module able to perform a real-time discovery of the structures existing inside a musical stream. At the end, an inference of short-term structures could be achieved from this discovery in order to feed scenario-based generative systems. (see figure 1)

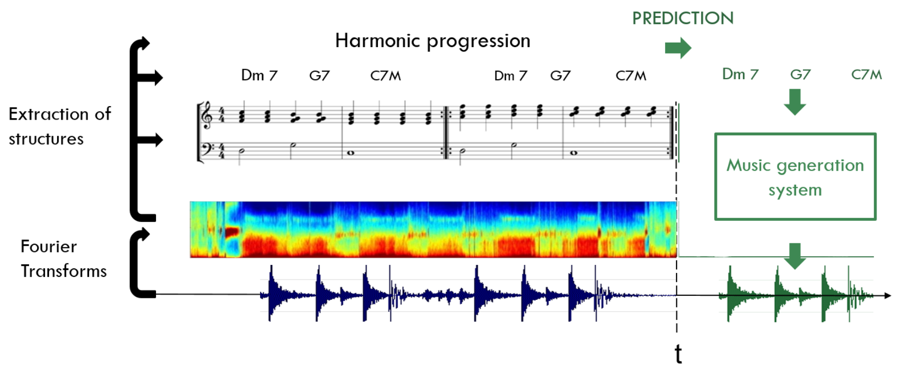


Figure 1: Extracting structures and prediction.

Multiple music generation applications could benefit from a real-time prediction of the musical structures, in particular, scenario-based generative softwares, such as ImproteK [37]. Based on this inference, music generation processes could combine the advantages of the two aforementioned forms of "guidance". Hence, we aim for a system that could be responsive (in a real-time sense), while maintaining a long-term temporal vision, able to anticipate and generate improvisation from an inferred (and not only pre-defined) underlying structure.

OUR MOTIVATION is to infer future musical structures for improving human computer co-improvisation. In this report, we present our approaches for structure extraction in audio streams and for symbolic music prediction based on this information.

OUR APPLICATION CASE is musical chords. Indeed, chords are mid-level musical features which 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 in this report principally concern the Automatic Chord Extraction (ACE) field and the generation of musical chord sequences.

2 STATE OF THE ART IN MACHINE LEARNING

2.1 MACHINE LEARNING

Machine learning aims to empower the computers with the capabilities to perform complex tasks that are innate to humans. The main idea is to develop algorithms able to learn by observing and modeling a set of examples. Thus, machine learning algorithms use computational methods to learn directly from the data by adapting the parameters of a pre-defined family of functions. The overarching goal of machine learning is to produce a model that could be able to generalize its understanding of a given (training) set to unseen data. In this section, we introduce the basic aspects and definitions underlying machine learning. Then, we introduce the different specific models that have been used along this internship.

2.1.1 *Function approximation*

In most machine learning problems, we start from a given dataset $X = \{x_1, ..., x_N\}; x_i \in \mathbb{R}^d$ and want to obtain a set of corresponding information in a given target space $Y = \{y_1, ..., y_N\}; y_i \in \mathbb{R}^t$. This target space usually defines the goal of the learning system. However, 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 would be to find this function $F \in \mathcal{F}$ that applies on the given data to obtain our desired solution. However, this 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 only a given family of functions $\mathcal{F}^*$, that we define, hoping to be as close to $\mathcal{F}$ as possible. Finally, to approximate the ideal solution, we select a parametric function $f_\theta \in \mathcal{F}^*$, in order to obtain an estimation

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

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

2.1.2 *Loss function*

Given this parametric function f_θ, the goal of machine learning is to find the parameters that can provide the highest accuracy possible.

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

$$L(\mathbf{x} \mid \Theta) = \sum_{i=1}^{N} \mathcal{L}(f(x^i), \tilde{f}_\Theta(x^i)) \tag{2}$$

The goal will be, therefore, to minimize this function by adjusting iteratively the parameters of our function f_θ. This error minimization process evaluates a given set of training examples and then performs the parameters update based on the derivative of the error.

2.1.3 *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 output that we seek for a given task. Here, we focus on detailing the supervised 2.1.3.1 and unsupervised 2.1.3.2 approaches as they will both be used in our subsequent methods, in the form of semi-supervised learning 2.1.3.3.

2.1.3.1 *Supervised learning*

The supervised learning approach aims at classifying an unknown dataset with the help of a labeled dataset. The learning is done on a dataset $\{x(i), a(i)\}_{1 \leqslant i \leqslant N}$ containing N samples. Here, $x(i)$ is an input data and $a(i)$ 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 potential supervised task would be to use a frame of the spectrogram $x(t)$ as an input and develop a model that could find the associated chord label $a(t)$. Thus, we need a training set containing associations between frame and chord label (an example dataset is depicted in Figure 2).

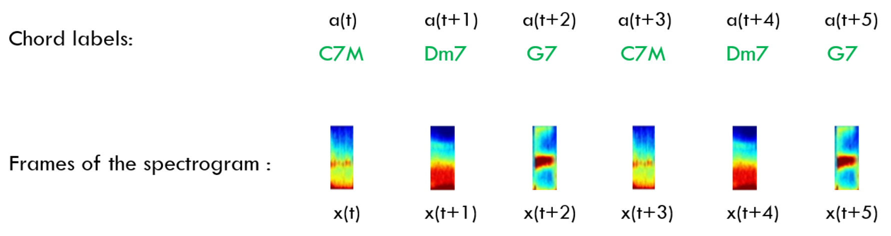


Figure 2: Example of elements inside a chord labeling dataset based on spectrogram windows.

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. Then, training the model can be done as shown in Figure 3. Here, by following our previously defined notation (2), we have the desired output of the classification task given by $f(x(t)) = a(t)$ and the approximation of our model $\tilde{f}_\Theta(x(t)) = y(t)$, which is here composed of several layers of transformations. By comparing these two values through the loss function $\mathcal{L}(f(x(t)), \tilde{f}_\Theta(x(t)))$, we can assess the errors made by the model in order to improve it by changing its parameters.

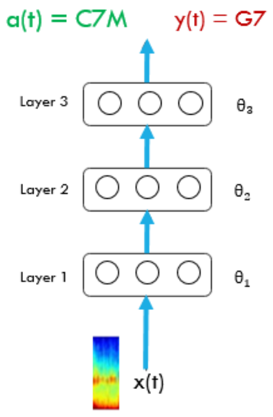


Figure 3: An example of supervised training for an ACE task, where we compare the ground truth annotation $a(t)$ with the approximation of our model $\tilde{f}_\Theta(x(t)) = y(t)$.

2.1.3.2 *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 $\{x(j)\}_{1 \leqslant j \leqslant M}$ and want to discover the structure of those data. The most well-known type of unsupervised learning is clustering algorithms. It is used for exploratory data analysis to find patterns or groupings 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 we don't inform the model on which features we are focusing on, the unsupervised learning works without any assumptions.

Once again, in the context of ACE, an example task would be to present a noisy spectrogram frame $x^*(t)$ as input in order to denoise it. This task is known as a *denoising* operation, as depicted in Figure 4. Here, one part serves to transform the input into a representation, which could then be used to reconstruct a noiseless version of the frame $x^*(t)$. The cost function is defined by the distance between the output $z(t)$ and the initial frame $x(t)$.

Despite the fact that supervised learning has demonstrated impressive results in the past years, unsupervised learning would be an approach widely studied in the next years.

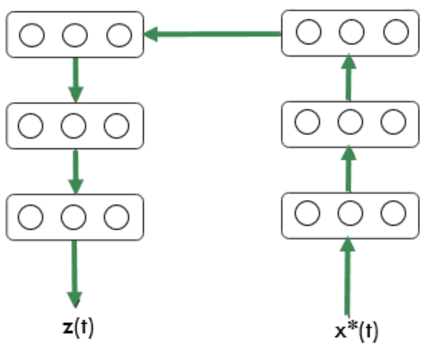


Figure 4: An example of unsupervised training for an ACE task, where we compare a frame of spectrogram $x(t)$ with a denoising corrupt version of it $\tilde{f}_{\Theta}(x^*(t)) = z(t)$.

"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, Hinton, Nature 2015 [29]

2.1.3.3 *Semi-supervised learning*

In most applications, we usually have access to few labeled data $\{x_t, y_t\}_{1 \leqslant t \leqslant N}$ and, comparatively, to a lot of unlabeled data $\{x_t\}_{N+1 \leqslant t \leqslant M}$ (i.e. 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 label to unlabeled data in order to use them in the training. By modeling $P(x|y)$ as clusters, unlabeled data affects the shape and size of clusters [58]. Technically, we use the clustering effect of the unsupervised learning and assign a label at each cluster with the help of the labeled dataset.

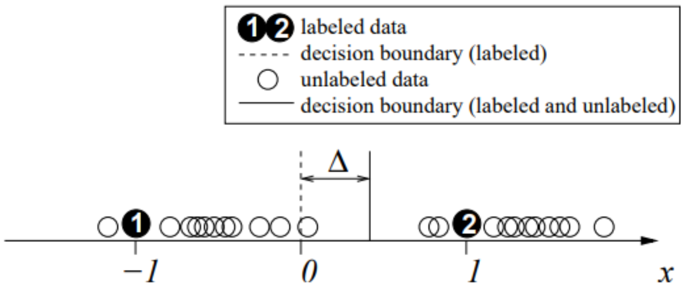

labeled data
decision boundary (labeled)
unlabeled data
decision boundary (labeled and unlabeled)
Δ
−1
0
1
x

Figure 5: Example of a two-class cluster classification dataset where two sample are labeled and others are unlabeled. The addition of unlabeled data modifies the cluster sizes and shapes. [58].

In Figure 5, we represent a two-class cluster classification dataset where two sample are labeled and others are unlabeled. Thus, the addition of unlabeled data shifts the decision boundary and gives a most accurate classification on unseen data.

Nevertheless, the semi-supervised learning does not always give better results and the error convergence depends on the learning task [52].

2.1.4 *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 accuracy. The accuracy is calculated by the error function. One solution for an efficient training is to use its derivative depending on the parameters. Hence, at each training iteration we find which direction to explore, leading to the family of *gradient descent* algorithms.

2.1.4.1 *Gradient descent and learning rate*

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

$$\Theta_{t+1} \leftarrow \Theta_t - \alpha * \nabla_\Theta L(\Theta_t) \tag{3}$$

where $\nabla_\Theta L(\Theta_t)$ represents the gradient of the loss function with respect to the parameters and α 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. A bigger learning rate will accelerate the convergence but can cause to "jump" over the minima as the update steps are too large. These ideas are exemplified in Figure 6. This figure represents the shape of a two parameter loss function $J(\theta_0, \theta_1)$. On the figure we see two paths (in black). The starting points of the paths (in the red area) are set by a random initialization of the model parameters. Then, each path is constructed by choosing the best gradient descent at each step. They naturally end in a local minima (in blue).

Over the past decades, the cardinality and dimensionality of datasets have outpaced the expansion of the speed of processors. As argued by Bottou [7], the capabilities of machine learning has been limited by the computing time rather that the sample size. Hence, stochastic approximation of the gradient optimization method as Stochastic Gradient Descent (SGD) have been proposed [7], which update the parameters of the model after observing each example, rather than relying on the whole dataset.

2.1.4.2 *Overfitting, Generalization*

The goal of learning methods is to obtain a model which provides a good *generalization* over unseen data. The generalization performance

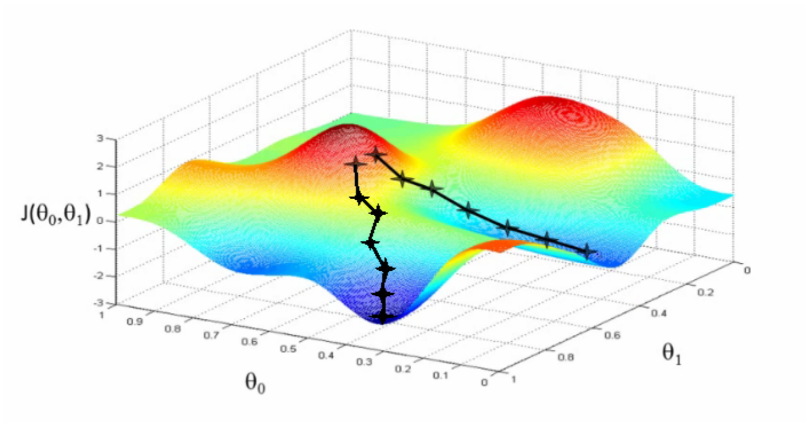


Figure 6: Gradient of a two parameters function with two local minima and example of the use of gradient descent based on two different random initializations, image taken from [35].

of a method refers to its capacity to make good predictions for independent (unseen) test data. Thus, we hope that the training set will reflect the characteristics of the entire sample space.

The overfitting appears when a model attached too many importance to the details of the training set. For instance, we show in Figure 7 the effect of overfitting for a task of classification between two classes inside a two-dimensional feature space. The black line represents a model with a good generalization performance whereas the green line shows a model displaying overfitting.

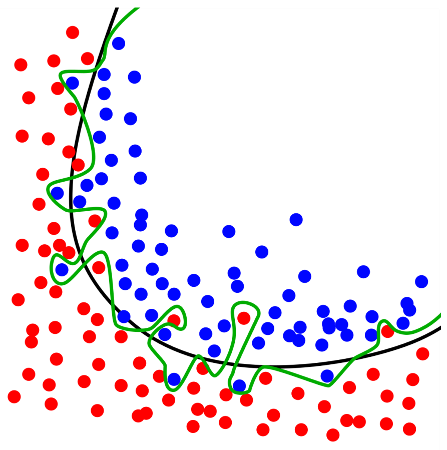


Figure 7: Two different models showing either overfitting (in green) or generalization (in black) for separation in a two-dimensional feature space with two classes, image taken from [40].

The overfitting can come from the complexity of the model. Indeed, if a model is excessively complex, it has too many parameters comparing to the number of observations. Among other solutions we can avoid overfitting and obtain better generalization by corrupting slightly the inputs. The overfitting also depends of the time of training. Thus, when we use learning methods, we have to stop the optimization process in order to avoid this effect. This optimization step can be seen as finding a tradeoff between the data that we use in our training set and the general behavior of future unseen data. One widely used method to avoid overfitting is to validate the model with a dataset which is independent from the training and the test set. Therefore, we divide the data into three subsets.

- THE TRAINING SET is used to train the model. It contains the sample from which we adjust the parameters.
- THE VALIDATION SET is used to avoid overfitting. As seen in Figure 8, 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.
- THE TEST SET is the data set on which the performance of the model is calculated.

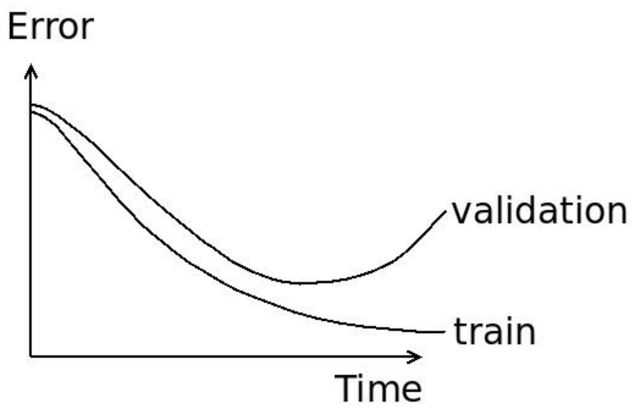


Figure 8: Evolution of train and valid errors versus training time, image taken from [40].

2.2 NEURAL NETWORKS

2.2.1 *Artificial neuron*

An artificial neuron can be compared to a biological neuron. In biological neurons, the dendrites are connected to a cell body. Each dendrite transmits electrical impulsion from the outside to the body. When the electrical charge exceeds a threshold the cell body sends an electrical impulsion through the axon. In an artificial neuron such as Figure 9, the dendrites are modeled by the inputs $\{x_i\}_{1 \leqslant i \leqslant m}$, the activation is a non-linear function f with a threshold b and the axon is the output h.

A neuron is then defined by its parameters, namely the bias b and the weights w_i, and by an activation function f. We have then a mathematic formalization that links the inputs x_i to the output h :

$$h = f(\sum_{j=1}^{m} w_j x_j + b) \tag{4}$$

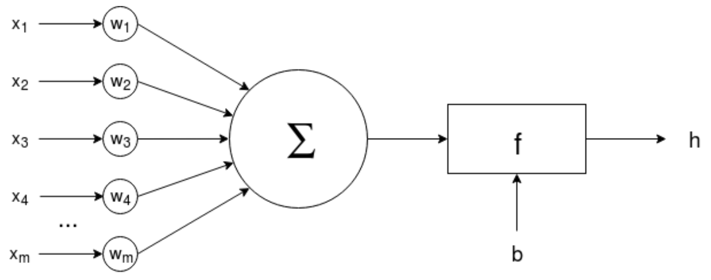
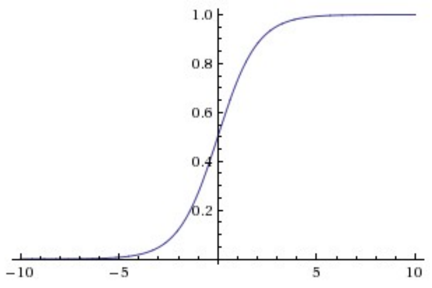


Figure 9: Schema of an artificial neuron.

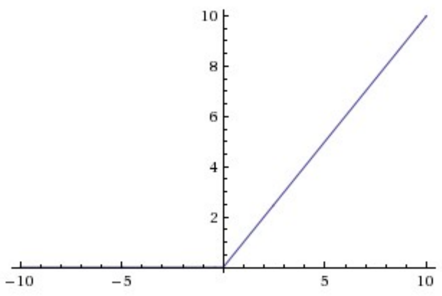
In the literature we find many activation function f. The activation should be a non-linear transformation. Here we present two examples of widely used activation function.

THE SIGMOÏD FUNCTION is a function that has been widely used in the field of machine learning. His derivative is very simple, making easier the chain-rule computations of the back propagation process (see back propagation 2.2.2.1).



$$f(x) = \frac{1}{(1 + \exp(-x))}$$

RELU that stands for rectified linear unit, has been introduced in 2000 with strong biological and mathematical motivation. As of 2015 this is the most popular activation function used in deep neural architecture.



$$f(x) = max(0, x)$$

2.2.1.1 *Interpretation*

Here we provide two ways of interpreting the Neural Networks (NN). First, by looking at Equation 4, we can see that if the transfer function is a threshold function, this defines an hyperplane. On the one hand, network with a single processing layer can be interpreted as separating the input space in two regions with a straight line (as seen in Figure 10). Here, the blue and red lines represents two data distributions, while the colored region represent the value of the output neuron. Therefore, we can note that a single-layered network can not separate properly these two distributions (as this dataset is *non linearly separable*. Hence, we must use a non-linear transfer 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 11, the space is gradually transformed by the successive neuron layers, as the coordinates of the next 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*.

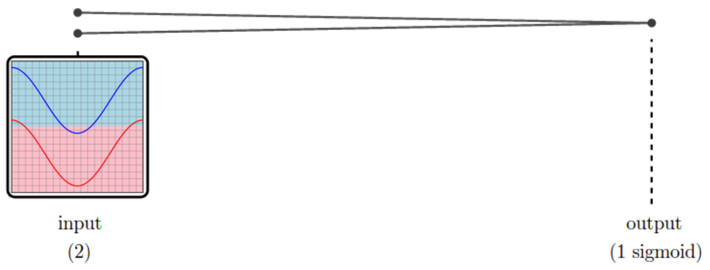


Figure 10: Space separation with one linear output layer. Image from [3].

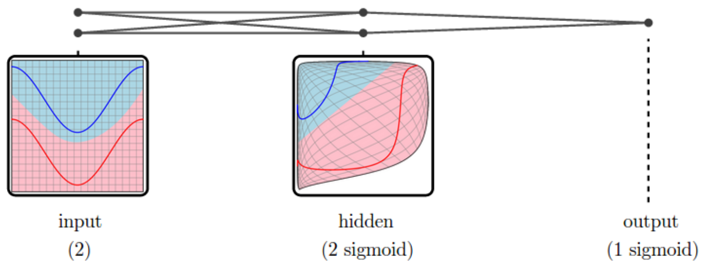


Figure 11: Space transform operated by successive hidden layer. Image from [3].

2.2.2 *Multi-Layer Perceptron*

The *Multi-Layer Perceptron* (MLP) is a feed-forward model organized as a succession of layers of neurons defined previously. 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 $h^l \in \mathbb{R}^{M_l}$, where M_l is the number of neurons contained in this layer (see Figure 12). For $1 \leqslant l \leqslant L$, the parameters of a layer are defined by a weight matrix $W^l \in \mathbb{R}^{M_l \times M_{l-1}}$ and a bias vector $b^l \in \mathbb{R}^{M_l}$. Therefore, the activation of neuron m in layer l is computed with the following equation

$$h^l_m = f\left(\sum_{j=1}^{M_{l-1}} (W^l_{m,j} . h^{l-1}_j) + b^l_m\right) \tag{5}$$

Let us rewrite the equation 5 such as $b_m^l = W_{m,0}^l.h_0^{l-1}$.

$$h_m^l = f(\sum_{j=0}^{M_{l-1}} (W_{m,j}^l.h_j^{l-1})) = f(y_m^l) \tag{6}$$

Thus, y_m^l is the net input for the neuron m in the layer l.

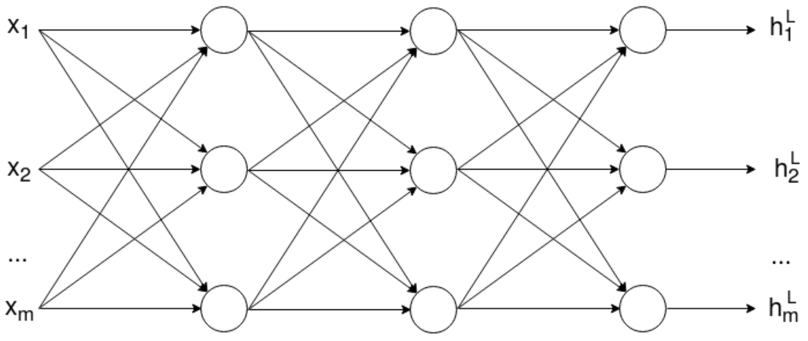


Figure 12: Multi-Layer Perceptron.

2.2.2.1 *Back propagation*

In order to train the MLP, it would seem that we should define a procedure that takes all layers into account, leading to a very complicated formulation. However, the introduction of the *back-propagation* method by Rumelhart and Hinton [56] allows to decompose this into a set of simple operations. The main idea is to see that adding layers amounts to add a function to the previous output. Therefore, in the *forward pass*, this output is updated by relying on the weight and bias of the neurons in this layer. Therefore, 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 neurons in the network have been updated.

After the last layer L, we compute the loss of our network (see Section 2.1.2), that we take here to be the squared error between the desired output δ_i^L and the output of the network h_i^L :

$$\mathcal{L}(\delta_i^L, h_i^L) = \frac{1}{2}||\delta_i^L - h_i^L||^2 \tag{7}$$

The derivative of this loss is then given by

$$\mathcal{L}(\delta_i^L, h_i^L)' = e_i^L = [\delta_i^L - h_i^L].f'(y_i^L) \tag{8}$$

Using the chain rule, the error is then back propagated by computing error for preceding layers :

$$e_i^l = f'(y_i^l) \sum_{j=0}^{M_{l+1}} W_{ij}^{l+1}.e_j^{l+1} \quad ; \quad l \in [1...L-1] \tag{9}$$

Finally, the weights are updated in the corresponding layer :

$$W_{ij}^{l} = W_{ij}^{l} + \lambda . e_{i}^{l} h_{j}^{l-1} \tag{10}$$

Where λ is the learning rate.

2.2.2.2 *Limitations and deep learning solutions*

As seen in the previous part, a MLP is composed of layers of artificial neurons as a simple model that tries to mimic biological neurons. Hence, simple neural networks are composed by a few layer of neurons to reduce their computational costs. However, an architecture with such insufficient depth would require an infinite number of computational elements [5] to solve complex tasks. Conversely, deep architectures can exploit non-linear hierarchies to solve this issue. Thus, we use deep learning when we need higher level of abstractions for our problem. This approach allows to construct optimal abstractions by discriminating the important variations at each layer by focusing on information at different scales. Nevertheless, adding several layers leads to a gradient diffusion problem during the back-propagation. It can be solved by a layer pre-training [4].

Deep learning usually relies on an encoder/decoder (unsupervised) model that aims to deconstruct an object and reconstruct it from fewer parts. In other words we are learning its structure. This notion of encoder/decoder seems fit to our musical structure problem. Indeed, each layer can be seen as a different abstraction (notes, chords, keys).

2.2.3 *Convolutional 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 [13, 28]. The architecture of CNNs were inspired by the organization of the animal visual cortex [30]. The main idea of CNN is to introduce invariance properties (such as translation, rotation, perspective distortion) into the transform computed by neural networks. Compared to the MLP, 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.

2.2.3.1 *Convolutional Layers*

A convolutional layer is defined by a set of convolution kernels that are applied in parallel to the inputs and produce a set of output *feature maps*. These kernels are learned during the training and hopefully each of these should describe a recurrent characteristic of the training data. After applying each kernel to the input, we obtain a set of feature maps, that is defined by a three-dimensional tensor $h \in \mathbb{R}^{M \times I \times J}$ where M is the number of kernel, I is the height and J the width of each kernel. During the feed-forward pass we apply a convolution by sliding the kernel over every local area of the input.

For instance, as seen in Figure 13, we start from the top left region and move the kernel until it reaches the bottom border of the input matrix. If we define the input as the matrix X, then the output feature maps are defined by $Y = X * h_m$ for every kernels. Here, the operator $*$ is a 2D discrete convolution

$$(A * B)_{i,j} = \sum_{r=1}^{T} \sum_{s=1}^{F} A_{r,s} B_{r+i-1,s+j-1} \tag{11}$$

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

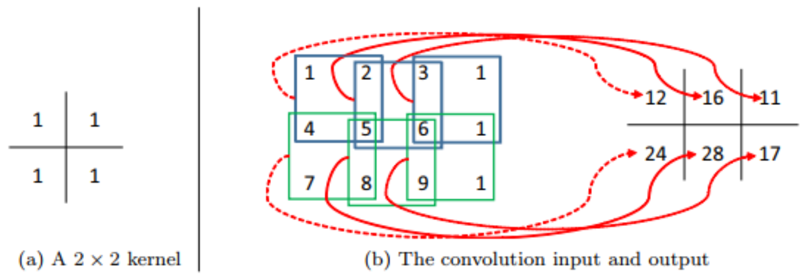


Figure 13: Illustration of the convolution operation, image taken from [57].

Here, we note that depending on the size of the kernel, the dimension of the output will be smaller than that of the input. Therefore, to keep the dimensions constant, we can use a zero-padding operation which consists to add zeros to the border of the original input. Furthermore, the norm is to apply an activation function after each convolutional layer.

2.2.3.2 *Pooling Layer*

Each convolutional layer significantly increases the dimensionality of the input 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, L2-norm pooling or max pooling. For instance, the max pooling (depicted in Figure 15) operation only keeps the maximum value in each regions of a partition of the input. The pooling size refers to the factor to which the map is reduced.

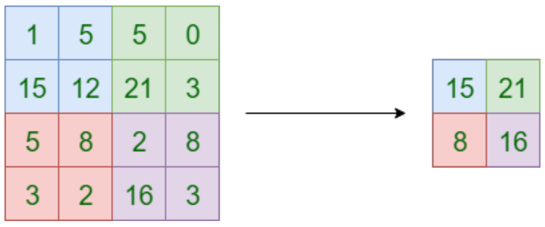


Figure 14: Example of a 2X2 max pooling.

2.2.3.3 *Fully-connected layer*

The overall structure of a convolutional network usually consists of a succession of convolution, activation and pooling layers. However, in order to then perform correlation between the different feature maps, this architecture is typically followed by one or many fully-connected layers (see Figure 15). This fully-connected layer is a standard MLP, where each neuron in layer l is connected to all neurons in layer $l+1$.

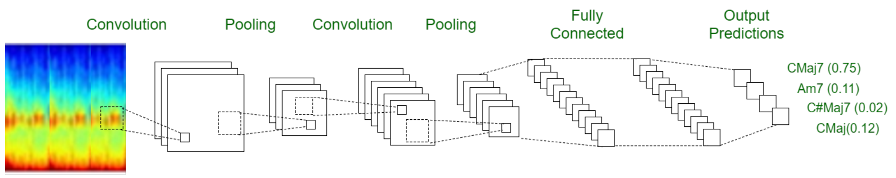


Figure 15: Example of a convolutional neural network for a task of ACE.

2.2.4 *Ladder Networks*

Recently, the *ladder network* (LN) has been proposed to allow a combination of supervised and unsupervised learning [47]. In this approach, we use an auxiliary unsupervised learning task (here denoising) to support supervised learning. This model has proved to give very satisfying results for the hand-written digit classification task [48]. However, as far as we know, the ladder network has not yet been applied to audio classification tasks. In this section we present the ladder network architecture.

2.2.4.1 *Lateral connections and noise injection*

One of the particularity of the LNs comes from the lateral connections of the encoder/decoder. As seen in Figure 16, each layer of the encoder is laterally connected to the decoder. These connections aim at relieving the pressure of higher layers. Thus, each layer concentrates more efficiently on their degree of abstraction.

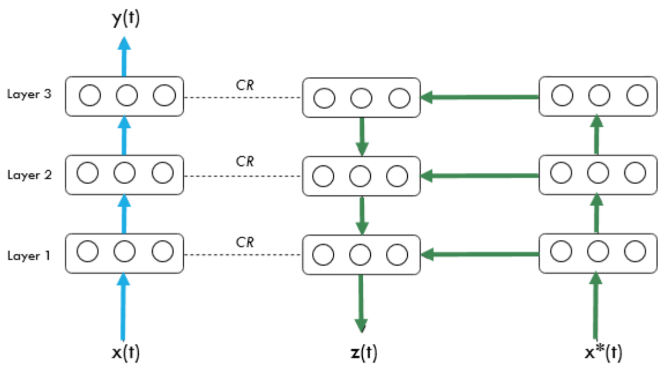


Figure 16: Semi-supervised learning on ACE task, the supervised classification (in blue) is reinforced by the unsupervised denoising autoencoder task (in green)

The lateral connection (from the layer l of the encoder to the layer l of the decoder) and the vertical connection (from the layer $l+1$ of the decoder to the layer l of the decoder) are combined by a *combination function* before to enter in the layer l of the decoder. This combination function has a non-negligible impact on the overall performance of the LN. In [46], Pezeshki et al. improve gradually the combination function from a simple linear combination to a function enhanced by multiplicative terms and sigmoïd non-linearity functions.

However, the choice of the combinator function seems less crucial than the type of noise introduced in the denoising autoencoder. Indeeed, the introduction of noise into each layer of the encoder has a strong regularization effect that helps for the generalization. Nevertheless, the most important aspect of the LN lies in its semi-supervised training.

2.2.5 *Reconstruction cost and loss function*

The LN can be trained to minimize simultaneously the sum of supervised (Cross Entropy) and unsupervised (L_2 reconstruction) cost functions. We consider the dataset with N labeled examples $\{x(t), y(t)\}_{1 \leqslant t \leqslant N}$ and M unlabeled examples $\{x(t)\}_{N+1 \leqslant t \leqslant M}$. The output of $x(t)$ performed by the supervised part of the network is written $y(t)$. The cost function is then defined by

$$\text{Cost} = -\sum_{t=1}^{N} \log P(y(t) = a(t) | x(t)) + \sum_{t=N+1}^{M} \sum_{l=1}^{L} \lambda_l \text{ReconsCost}(z^{(l)}(t), \hat{z}^{(l)}(t)) \tag{12}$$

Where L is the total number of layers, $\hat{z}^{(l)}$ and $z^{(l)}$ are respectively the outputs of the supervised and unsupervised path at the layer l . λ_l define the contribution of each layer by their reconstruction cost. Moreover, these reconstructions costs are the most important properties of the LD and provide the desired regularization from unlabeled data.

Finally, each layer of the LN, initially classic MLPs, can be replaced by more specific ones such as CNNs.

2.3 TEMPORAL STRUCTURE AND MEMORY

2.3.1 *Recurrent Neural Networks*

One of the largest issue in NN is its incapacity to store temporal information. However, in our setup, we need to discover the underlying structure of a music signal. Therefore, we need to design our model with an understanding of time. Recurrent Neural Networks (RNN) show promising results in many learning fields [11] including musical structure generation [16]. To define recurrent networks, we usually rely on loops inside the network, where the output of a neuron is directly linked to itself for the next sequence element. The RNN can also be interpreted as a neural networks with lateral connections as seen in Figure 17

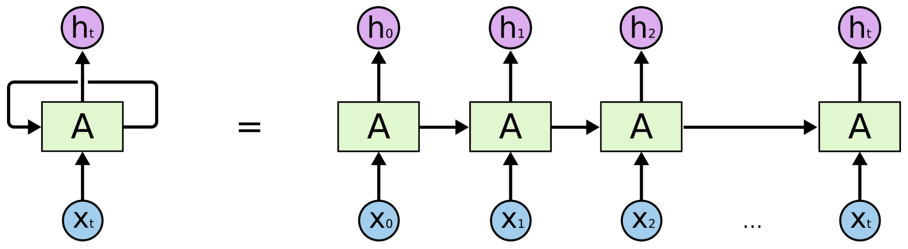


Figure 17: Loops of a RNN unfolded through time, image taken from [54].

The inputs x_i are processed by the RNN and it outputs a value h_i. The transformation is operated by a module $A(x_i, h_{i-1})$ that takes in arguments the input x_i and the output of the previous step h_{i-1}.

When have then a mathematical formalism for the RNN :

$$h_i = A(x_i, h_{i-1}) \tag{13}$$

In a standard RNN, the function A has a very simple structure such as a linear layer followed by a tanh layer.

The training of a RNN is realized by the 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.

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 end. The two most used gated RNN structures are the Long Short-Term Memory (LSTM) and the Gated Recurrent Unit (GRU).

2.3.2 *LSTM*

Generic RNN are limited in there capacity to learn long-term dependencies and are hard to train [42]. An elegant variant, called LSTM, has been introduced by Hochreiter and Schmidhuber [21]. It provides good results in many use case [53].

Similarly to classic RNN, LTSM network performs a transformation of an input x_i by taking into account the transformations of the previous time steps. The difference between the two cells lies in the structure of the module A. In classic recurrent neuron, the output of the previous step h_{i-1} is send through a tanh layer (see Figure 18). The reminiscence of information between distant time steps is then relatively poor.

In LSTMs, an additional internal connection is present (see Figure 19). This connection carries information through time steps with only some minor linear interactions. Thus, the information between distant time steps could be unchanged.

In addition to this memory cell, the LSTM is composed by three gates which control how to update and used the memory cell. The gates are principally the combination of a sigmoïd layer followed by a point-wise multiplication layer.

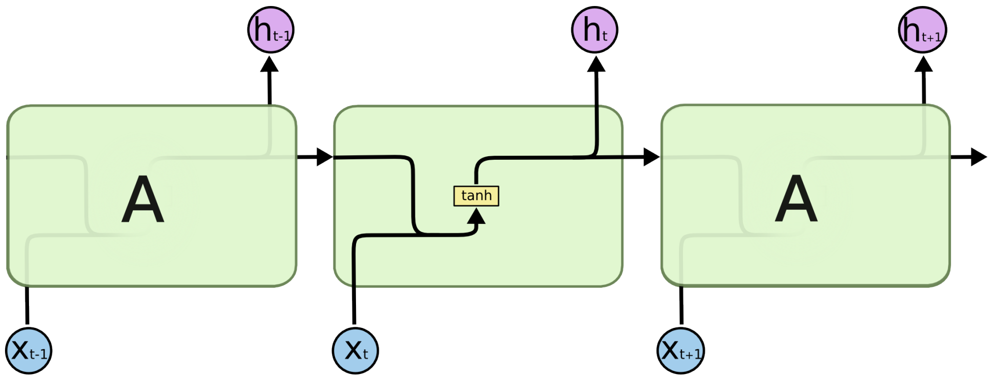


Figure 18: Internal structure of a classical RNN cell, image taken from [54].

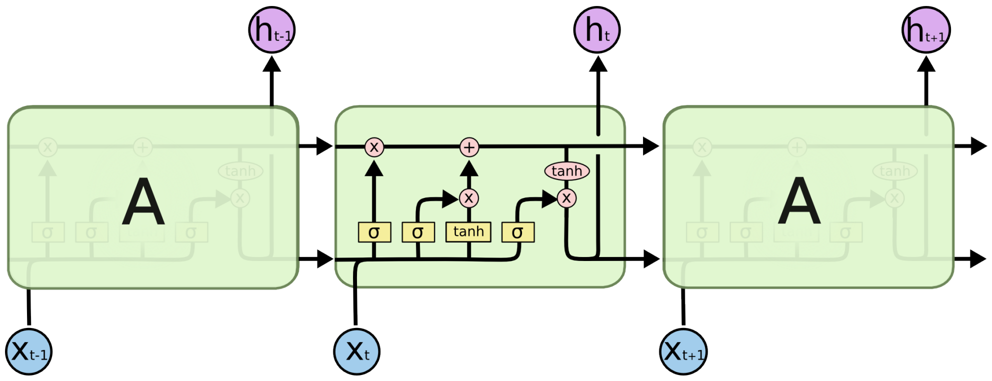


Figure 19: Internal structure of a LSTM cell, image taken from [54].

1. The forget gate controls which part of the memory cell to forget. It looks at h_{i-1} and x_i, 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 memory dimension must be output to the next time step.

Furthermore, a LTSM network is composed by layer of LSTM units. This construction allows to detect patterns at different abstraction levels. This hierarchy vision is reminiscent of the structure hierarchy in an audio track. Thus, LSTM network could operate at different time scales.

3 EXTRACTING CHORD LABELS FROM AUDIO SIGNAL

3.1 CHORD PROGRESSION EXTRACTION

3.1.1 *Definition of the workflow*

Across the several tasks in the field of Music Information Retrieval (MIR), the Automatic Chord Estimation (ACE) from audio signal is a topic that has been widely studied over the past years [33]. Throughout the literature, chord recognition systems are usually designed by following the same workflow.

1. First of all, the audio signal is transformed into a time-frequency representation. In the chord classification task, the Short-Term Fourier Transform (STFT) or a logarithmic representation such as the Constant-Q Transform (CQT) are often used.
2. Secondly, features are extracted and the signal is represented by a *chromagram* (a matrix with 12 rows representing each pitch class, and N columns corresponding to the number of time frames of the audio track).
3. At the end, each time frame (or related chromagram frame) is assigned by the system to a chord label. A post-filtering step, which is also called *chord sequence decoding* is often performed to smooth out the sequence of outputs.

3.1.2 *Related works in Automatic Chord Extraction*

In the seminal paper by Wakefield in 1999 [55], the first mathematical grounds for the definition of *chromagram* opened up its use for the ACE field. Consequently, Hidden Markov Models (HMM) have been amongst the first models investigated for chord extraction [51] as post-filtering methods. Following these works, multiple variants of this workflow were explored, either where other features have been exploited such as the tonal centroid [18], or replacing the classification methods with probabilistic N-Grams [50] or Dynamic Bayesian Networks (DBN) [32].

In the recent years, several works relied on neural networks for ACE. For instance, Humphrey and Belo [23] applied CNN to classify major and minor chords. This new approach has raised a different perspective on the problem, where a context (which takes several seconds of pitch spectra) is classified directly by a CNN, without relying on the explicit definition and computation of sound features beforehand.

Since then, the use of deep learning in the field of chord extraction has been explored in different parts of the traditional workflow. Boulanger-Lewandowsky et al. [8] trained a RNN as a post filtering method, over a feature extraction pipeline learned by a Restricted Boltzmann Machine (RBM), making this whole approach entirely based on learning methods.

The previously cited approaches were trained to distinguish between major and minor chords only. However, based on the excellent results provided by learning methods, following works considered other chord types (seventh, augmented, or suspended) mapped to major/minor as a novel classification problem. Hence, the Deep Chroma Extractor system was proposed [27] to target this problem, which aims to learn how to extract a chroma representation from the audio signal with deep networks. Following this work, the same authors proposed a method combining a CNN for feature extraction with a Conditional Random Field (CRF) for chord sequence decoding [26]. The CNN is trained to predict chord labels for each audio frame, which provides hidden representations that are used as features for the subsequent pattern matching and chord sequence decoding stage performed by the CRF.

3.1.2.1 *MIREX*

In order to perform a systematic and objective evaluation of the robustness of the proposed approaches, ACE systems are compared in an annual evaluation called the Music Information Retrieval Evaluation eXchange (MIREX). For the ACE task, the MIREX chord label dictionary is composed of five different alphabets.

1. Chord root note only;
2. Major and minor N (no chord), maj, min;
3. Seventh chords N, maj, min, maj7, min7, 7;
4. Major and minor with inversions N, maj, min, maj/3, min/b3, maj/5, min/5;
5. Seventh chords with inversions N, maj, min, maj7, min7, 7, maj/3, min/b3, maj7/3, min7/b3, 7/3, maj/5, min/5, maj7/5, min7/5, 7/5, maj7/7, min7/b7, 7/b7.

In order to perform these comparisons, MIREX evaluates the quality of an automatic extraction by comparing the results to a ground truth created by one or more human annotators. The Chord Symbol Recall (CSR) measure is typically used to estimate how well the predicted chords match with the ground-truth.

$$\mathrm{CSR} = \frac{total\ duration\ of\ segments\ where\ (annotation\ =\ estimation)}{total\ duration\ of\ annotated\ segments} \tag{14}$$

We present in Table 1 the methods proposed for the ACE task in MIREX 2016 as we will be using their results (detailed in Table 2) on the same datasets as a baseline in order to evaluate the success of our proposed methods (see Chapter 4).

Algorithm	Contributors
CM1 (Chordino)	Chris Cannam, Matthias Mauch
DK1-DK4	Junqi Deng, Yu-Kwong Kwok
FK2, FK4	Filip Korzeniowski
KO1 (shineChords)	Maksim Khadkevich, Maurizio Omologo

Table 1: Contributors for MIREX 2016 in ACE, for more information see MIREX website [31]

Algorithm	1	2	3	4	5
CM1	78.56	75.41	72.48	54.67	52.26
DK1	79.21	76.19	74.00	66.02	64.15
DK2	77.84	74.49	71.93	61.61	59.47
DK3	80.03	77.55	74.79	68.40	65.88
DK4	76.05	72.96	71.41	62.77	61.44
FK2	86.09	85.53	82.24	74.42	71.54
FK4	82.28	80.93	78.03	70.91	68.26
KO1	82.93	82.19	79.61	76.04	73.43

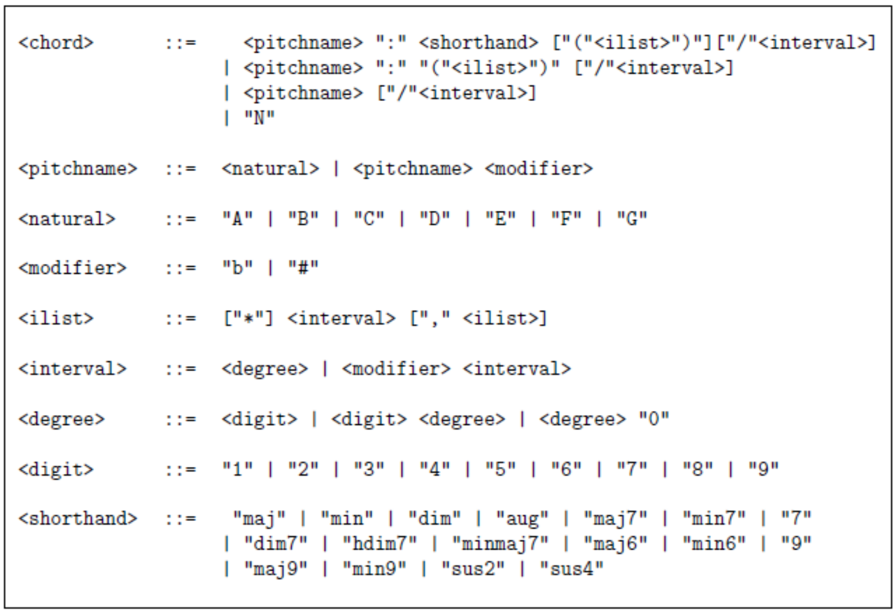
Table 2: ACE on Isophonics 2009 for MIREX 2016

3.2 DATASETS AND AUDIO PRE-PROCESSING

We introduce in this section the datasets and the audio pre-processing that will be used in the evaluation, based on the MIREX evaluation guidelines.

3.2.1 *Isophonics datasets*

The Isophonics dataset is the major dataset for ACE over the past years. It has been collected by the Centre for Digital Music at Queen Mary of the University of London and is composed by 5 subsets. The subsets are from The Beatles, Carole King, Queen, Michael Jackson and Zweieck. The sum of all the subsets represents 300 MP3 files at a sampling rate of 44100Hz with annotations on chords, keys, beats and structural segmentations. Nevertheless, the different subsets that composed Isophonics have not the same level of confidence. Besides all of them the Beatles's dataset (180 songs) created by Christopher Harte [17] is the one with the highest level of confidence. The annotations of the chords follows the syntax described on the Figure 20



```
<chord>      ::=   <pitchname> ":" <shorthand> ["("<ilist>")"]["/"<interval>]
                 | <pitchname> ":" "("<ilist>")" ["/"<interval>]
                 | <pitchname> ["/"<interval>]
                 | "N"

<pitchname>  ::=  <natural> | <pitchname> <modifier>

<natural>    ::=  "A" | "B" | "C" | "D" | "E" | "F" | "G"

<modifier>   ::=  "b" | "#"

<ilist>      ::=  ["*"] <interval> ["," <ilist>]

<interval>   ::=  <degree> | <modifier> <interval>

<degree>     ::=  <digit> | <digit> <degree> | <degree> "0"

<digit>      ::=  "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

<shorthand>  ::=   "maj" | "min" | "dim" | "aug" | "maj7" | "min7" | "7"
                 | "dim7" | "hdim7" | "minmaj7" | "maj6" | "min6" | "9"
                 | "maj9" | "min9" | "sus2" | "sus4"
```

Figure 20: Syntax of Chord Notation in Isophonics, image taken from [17].

3.2.2 *Audio transformation*

In order to obtain an efficient representation of input signals, we need to compute a time-frequency transform of the musical tracks. The analyzed data is then a two-dimensional matrix $X[m, k]$ where k is a bin centered around a frequency and m is the temporal index of the frame. However, this representation might remain computationally intensive for learning. Therefore, a dimensionality reduction can be performed by reducing the frequency resolution or temporal quantification.

In this section, we discuss our choices regarding spectral transforms and also introduce the audio data transposition which allows us to augment the training dataset with transposed chords.

3.2.2.1 *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. 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.

$$X_{STFT}[m, k] = \sum_{n=0}^{N-1} x[n].w[n-m].e^{-i\frac{2\pi k}{N}n} \tag{15}$$

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.

3.2.2.2 *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. 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 :

$$Q = \frac{f_k}{\delta f_k} \tag{16}$$

The windows length N becomes a function of the bin number $N[k]$ and the windowing function becomes a function of the window length :

$$X_{CQT}[m,k] = \frac{1}{N[k]} \sum_{n=0}^{N[k]-1} x[n].W[k,n-m].e^{-i\frac{2\pi Q}{N[k]}n} \tag{17}$$

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 will be very helpful to simplify our implementation of the audio transposition for augmenting the training dataset.

3.2.2.3 *Audio transposition*

In the CQT we define a number of bin per octave. Hence, we can artificially transpose our signal with a translation on our CQT representation. We also transpose the metadataset by remplacing the $< natural >$ and the $< modifier >$ accordingly (see Figure 20).

For our trainings, we choose to work with transpositions between -6 and +6 demi-tone.

3.2.3 *Chord alphabet reduction*

All ACE algorithms do not generate the same alphabets of chords and this choice influences the performance evaluation [44]. As mentioned in our motivation, we are interested in the functional aspect of the chord labels. Hence, we built our own alphabets that slightly differ from these presented in the subsubsection 3.1.2.1.

We use three different alphabets for our classification task :

1. Major and minor: N (which means no chord), maj, min;
2. Major and minor, seventh, diminished chords: N, maj, min, maj7, min7, 7, dim;

3. Major and minor, seventh, diminished, augmented chords: N, maj, min, maj7, min7, 7, dim, aug;

The most studied alphabet between them is the n°2. It represents the harmonization of the major scale (Figure 3). With this alphabet we can describe nearly all the jazz, rock and pop songs.

I	II	III	IV	V	VI	VII
C Major	D Minor	E Minor	F Major	G Major	A Minor	B Diminished

Table 3: Harmonization of the C Major scale.

Isophonics'dataset is annotated much richer than our desired alphabet, we then applied a chord alphabet reduction. This mapping is a surjection $M : C_{M0} \to C_{M2}$ where C_{M0} is the original domain with the Harte's notation [17] and $C_{M2} = \{N, maj, min, maj7, min7, 7, dim\}$ is the domain of the alphabet n°2. Our mapping associated all existing chords in C_{M0} with a chord in C_{M2}.

3.2.4 *Transforming the chords into pitch vectors*

In a basic classification, a model does not take into account the distance between the output classes. Nonetheless, we want to include in our model some information from music theory. For instance, a $CMaj7$ is nearer to a $Am7$ than a $C\#Maj7$. Our conception is based on harmonic network such as Tonnetz, that has been already exploited in many works on music information extraction [10, 24]. Thus, every chord of our alphabet is transformed into a 12-D vector where each dimension represents a pitch class.

3.3 OUR MODELS

In this section, we present the results obtained for our different models. We trained two families of models, namely CNN and LN. Each of these families are composed by different variants of these models. For the CNN, we have the *Vanilla* one which has been introduced earlier in this report (see Section 2.2.3). Here, we also tested the new proposals called Residual CNN and the Dense CNN. Both of these models are CNN with additional *skip-connections* between layers. These connections carry the untouched information from one layer to the next, in parallel to the transformation path. For more details, the interested reader can refer to the literature [19, 22]. We also implemented the LN, while performing the same types of improvements. We replaced the classic MLP layers with CNN layers and Residual CNN layers. For all the models we are working on the Beatles dataset introduced previously. We transform the input pressure signal through a CQT with 24 bins per octave and a frequency range of 27 Hz to 3.5kHz. Therefore, each frame contains 169 frequency bins, and we set the temporal context to 10 steps. Each temporal step is equivalent to 0.2s from an MP3 file, This implies that our overall context windows represent 2s of sound. These parameter choices are based on previous works [23, 26].

To evaluate our results in an objective way, we provide in Table 4 the results of a CNN for Maj/Min chord extraction without any post-filtering method. These results have been obtained on the Isophonic dataset with multiple transpositions.

Train	Valid	Test
82.81	77.8	77.48

Table 4: 5-Fold recognition accuracy for ACE on Maj/Min classification task on Isophonics dataset with a CNN, results taken from [23]

A fold is a repartition of our dataset between train, valid and tests subsets. On the table 4 the authors present results that are the mean of 5 different folds. Nevertheless, our models were only tested on one fold.

We trained our models with different configurations : with and without transposition, with and without the chord to vector transformation and on the three different alphabets of the section 3.2.3.

3.3.1 *Convolutional networks*

The CNNs that we trained are composed by 3 layers, each layer has 500 neurons. The two first convolutional layers have 32 kernels (11 bin heights by 3 bin width), the last convolutional layer has also 32 kernels (5 bin heights by 2 bin width).

3.3.1.1 *Vanilla*

The table 5 shows the results of the ACE task using a vanilla CNN without transposition on the dataset and without the chord to pitch vector space transformation.

	Train	Valid	Test
Alphabet n°1	85.51	73.16	73.34
Alphabet n°2	80.93	60.56	62.39
Alphabet n°3	81.71	55.27	59.80

Table 5: 1-Fold recognition accuracy for ACE on three different alphabet with a classic CNN on the Isophonics dataset.

The other models are still under calculation, this report will be completed in an updated version.

4 INTRODUCING TEMPORAL CONTEXT AND MEMORY

The interest of adding a temporal context in our model is twofold. On one hand, this is a post-filtering method that completes our ACE models. On the other hand, the memory of the musical structures can be used for predictive models.

4.1 POST-FILTERING METHODS FOR ACE

The models presented on the previous part do not represent the whole chain of ACE described on section 3.1.1. Indeed, the post filtering step is missing. This step consists of smoothing the chord sequence obtained after the feature extractions. Among other methods, we decided to train LSTM layers to achieve this task.

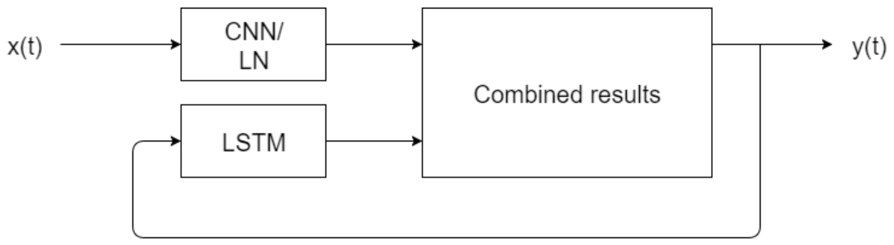


Figure 21: Complete ACE system using CNN/LN as feature extractor and LSTM as post-filtering method.

On the Figure 21 the CNN (or the LN) extracts the features from the signal $x(t)$. These feature are combined to the prediction of the LSTM realized on the last extracted features $y(t-1), y(t-2), ..., y(0)$.

4.1.1 *Real Book*

In order to augment the amount of data for the training of the LSTM, we used the dataset [12] composed of 2847 taken from the real book. The real book is a compilation of jazz classics that has been firstly collected by the students of the Berklee College of Music during the seventies. As of today we count a lot of existing books (e.g Realbook (1, 2, 3), New Realbook (1, 2, 3), the Fakebook, the real Latin book). The annotations of this dataset follow the ones of the Isophonics dataset.

4.2 GENERATING NEW DATA FROM MEMORY

In this part we present the generative aspect of the LSTM. We use the same training than for the ACE task. Nevertheless, instead of doing a prediction at only one step, we use the LSTM to generate chord sequences.

4.2.1 *Generation of a chord sequence*

As shown in Figure 22, we have as input a chord sequence $\{Gm, D7, Gm, D7, B\flat, Gm\}$ where each element is contained in an alphabet C_M. The output of the LSTM is a vector with a dimension equals to the number of elements in C_M. This vector contains the prediction probabilities for the next chord. If we select the chord with the highest probability, we can add it to our initial sequence. Thus, at time $t+1$ we use this new sequence as input of the LSTM. A $t+n$ we obtain a scenario of length n elements from the alphabet C_M.

Moreover, if we don't choose the chord with the highest probability but the second one, we obtain a slight variant in the predicted chord. Thus, we can generate multiple scenarios from the LSTM.

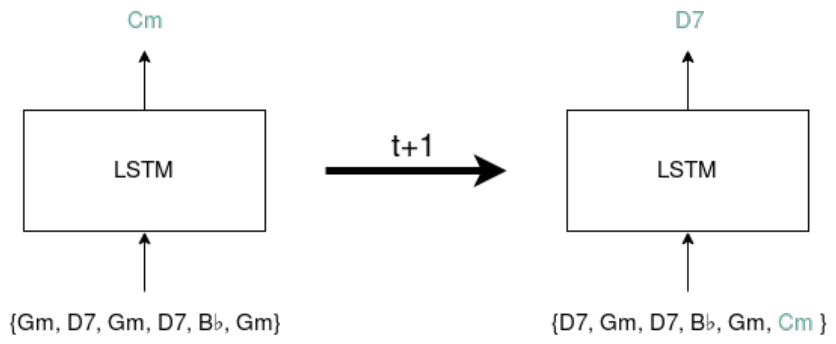


Figure 22: The chord sequence prediction is realized with adding element to the chord sequence at each time step.

4.3 OUR MODELS

Our LSTM model contains three layers. The number of LTSM units by layer not exceed 500. During the training we use a maximum of 8 time-steps for the back-propagation. The time-step is relative to each song and corresponds to its beat. The table 6 shows the results on the tree different alphabets (see section 3.2.3.)

	Train	Valid	Test
Alphabet n°1	38.55	30.05	31.44
Alphabet n°2	26.04	27.05	24.45
Alphabet n°3	14.62	15.88	16.62

Table 6: 1-Fold chord prediction task with a LSTM trained on 8 time-step sequence from the realbook dataset.

At first view these results seem perfectible. Nevertheless, deeper studies must be realized in order to analyze the output of the LSTM for a given chord sequence. Indeed, even if the predicted chord is not formally exact it could have a musical interest.

4.4 TOWARDS REAL-TIME PREDICTION

In our approach, we use machine learning methods for both the classification and the prediction task. We have then divided our system in two part : a detection system which aims to discover the chords playing and a predictive system.

The chord sequence sent to the LSTM is the one discovered by the ACE system in the musician's live record. Our overall software architecture is depicted on Figure 23.

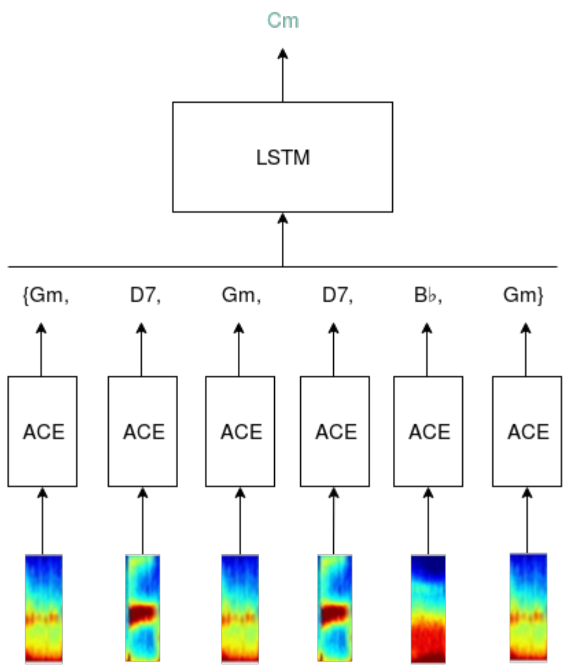


Figure 23: Our overall system architecture. Each frame of the audio flux is processed by the ACE system. Then the chord sequence is sent to a LSTM in order to predict the next chord.

We plan to use this software for real-time application. Indeed, the musician signal would be process by MAX/MSP in order to extract CQT frames. Then, the overall system should generate music chord sequences based on this audio stream. Finally, the predicted chord sequence would be sent at each time step to a scenario-based music generation software (e.g. ImproteK).

5 CONCLUSION AND PERSPECTIVES

The goal of this internship was to enhance computer co-improvisation processes by introducing inference of short-term scenario. Thus, we separated this task between the ability to extract structures in an audio stream on one hand, and to generate symbolic music based on this information on the other hand. Our application case was the design of an ACE system and a subsequent chord sequence generator.

For the ACE task, we focused on the feature extraction part of the workflow. We pre-processed the data from Isophonics with CQT and STFT transforms. Then, we augmented the data by using sets of transpositions from the spectral transforms. We implemented various architectures to compare the results between CNN, Residual CNN, Dense CNN, LN, Convolutional LN and Residual LN models. We assessed different models as well as their parameters and their training and testing procedures. The CNN models are trained with a supervised learning algorithm whereas the LN models are trained in a semi-supervised way. Furtehrmore, we proposed another training process that takes as output a transformation from the chord label annotation to a pitch vector representation. This allowed us to define a more accurate distance criterion for chords. We obtained results comparable to the state of the art even without taking into account all our enhancements. Indeed, the global ACE system that includes LSTM post-filtering has not been tested yet. However, given its preliminary results, we can expect it to improve our results even further.

Secondly, we designed a LSTM network for chord sequence generation. We trained this LSTM network with the realbook dataset, which allows to cast our problem into a purely symbolic inference task. This approach allows to generate a scenario which is the continuation of a given chord sequence. Moreover, the scenarios are not pre-calculated but created from a probabilistic study on a large corpus. We showed the applicability of this system and its encouraging results.

Finally, we designed the overall architecture of the system and proposed an overarching workflow of our application that could be applicable to real-time setups. Nevertheless, we did not yet realize application case use with musicians.

FUTURE WORK

Given the large amount of models and enhancements proposed in this work, we are still currently waiting for the complete results for all the models. As soon as all models are assessed, we plan to first investigate the reasons behind their relative successes and failures and to perform hyper-optimization of the parameters for the best models. Then, this model will be trained on different musical annotations (chroma, chord progressions), in order to develop a multi-scale system able to scrutinize temporal relationships at different time scales.

Furthermore, we strongly believe that the post-filtering method would also largely improve our results. On the other hand, the LSTM outputs could also be interpreted in a musical way. We intend to define this evaluation procedure in order to gain some insights on the inference and generation process.

Regarding the overall system, a real-case use of our modules including real-time human improvisation will be performed to confirm the qualities and drawbacks of our proposals. Therefore, we plan to test the complete system in real case application in order to have feedback from musicians.

Finally, a study on multivariate signals that could allow structure inference and generation for multiple musicians could be performed by relying on our overall framework.

BIBLIOGRAPHY

[1] Gérard Assayag and Georges Bloch. “Navigating the oracle: A heuristic approach.” In: *International Computer Music Conference'07*. 2007, pp. 405–412.

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

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

[4] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. “Greedy layer-wise training of deep networks.” In: *Advances in neural information processing systems*. 2007, pp. 153–160.

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

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

[7] Léon Bottou. “Large-scale machine learning with stochastic gradient descent.” In: *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[8] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. “Audio Chord Recognition with Recurrent Neural Networks.” In: *ISMIR*. Citeseer. 2013, pp. 335–340.

[9] Joel Chadabe. “Some reflections on the nature of the landscape within which computer music systems are designed.” In: *Computer Music Journal* (1977), pp. 5–11.

[10] Elaine Chew. “Towards a mathematical model of tonality.” PhD thesis. Massachusetts Institute of Technology, 2000.

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

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

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

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

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

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

[17] Christopher Harte. "Towards automatic extraction of harmony information from music signals." PhD thesis. 2010.

[18] Christopher Harte, Mark Sandler, and Martin Gasser. "Detecting harmonic change in musical audio." In: *Proceedings of the 1st ACM workshop on Audio and music computing multimedia*. ACM. 2006, pp. 21–26.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

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

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

[22] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. "Densely connected convolutional networks." In: *arXiv preprint arXiv:1608.06993* (2016).

[23] Eric J Humphrey and Juan Pablo Bello. "Rethinking automatic chord recognition with convolutional neural networks." In: *Machine Learning and Applications (ICMLA), 2012 11th International Conference on*. Vol. 2. IEEE. 2012, pp. 357–362.

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

[25] David Brian Huron. *Sweet anticipation: Music and the psychology of expectation*. MIT press, 2006.

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

[27] Filip Korzeniowski and Gerhard Widmer. "Feature learning for chord recognition: the deep chroma extractor." In: *arXiv preprint arXiv:1612.05065* (2016).

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

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

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

[31] *MIREX ACE 2016*. http://www.music-ir.org/mirex/wiki/2016:Audio_Chord_Estimation_Results. Accessed: 2017-09-20.

[32] Matthias Mauch. "Automatic chord transcription from audio using computational models of musical context." In: (2010).

[33] Matt McVicar, Raúl Santos-Rodríguez, Yizhao Ni, and Tijl De Bie. "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 (2014), pp. 556–575.

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

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

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

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

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

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

[40] *Overfitting*. https://en.wikipedia.org/wiki/Overfitting. Accessed: 2017-09-20.

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

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

[43] Jouni Paulus, Meinard Müller, and Anssi Klapuri. “State of the Art Report: Audio-Based Music Structure Analysis.” In: *ISMIR*. 2010, pp. 625–636.

[44] Johan Pauwels and Geoffroy Peeters. “Evaluating automatically estimated chord sequences.” In: *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE. 2013, pp. 749–753.

[45] Geoffroy Peeters. “Sequence Representation of Music Structure Using Higher-Order Similarity Matrix and Maximum-Likelihood Approach.” In: *ISMIR*. 2007, pp. 35–40.

[46] Mohammad Pezeshki, Linxi Fan, Philemon Brakel, Aaron Courville, and Yoshua Bengio. “Deconstructing the ladder network architecture.” In: *arXiv preprint arXiv:1506.02351* (2016).

[47] Antti Rasmus, Harri Valpola, and Tapani Raiko. “Lateral connections in denoising autoencoders support supervised learning.” In: *arXiv preprint arXiv:1504.08215* (2015).

[48] Antti Rasmus, Mathias Berglund, Mikko Honkala, Harri Valpola, and Tapani Raiko. “Semi-supervised learning with ladder networks.” In: *Advances in Neural Information Processing Systems*. 2015, pp. 3546–3554.

[49] Arnold Schoenberg and Leonard Stein. *Structural functions of harmony*. 478. WW Norton & Company, 1969.

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

[51] Alexander Sheh and Daniel PW Ellis. “Chord segmentation and recognition using EM-trained hidden Markov models.” In: (2003).

[52] Aarti Singh, Robert Nowak, and Xiaojin Zhu. “Unlabeled data: Now it helps, now it doesn’t.” In: *Advances in neural information processing systems*. 2009, pp. 1513–1520.

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

[54] *Understanding LSTM Networks*. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`. Accessed: 2017-09-20.

[55] Gregory H Wakefield. “Mathematical representation of joint time-chroma distributions.” In: *International Symposium on Optical Science, Engineering, and Instrumentation, SPIE*. Vol. 99. 1999, pp. 18–23.

[56] Paul John Werbos. *The roots of backpropagation: from ordered derivatives to neural networks and political forecasting*. Vol. 1. John Wiley & Sons, 1994.

[57] Jianxin Wu. "Introduction to Convolutional Neural Networks." In: 2017.

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

COLOPHON

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

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

Final Version as of March 9, 2018 (`classicthesis` version 4.2).