UNIVERSITE PARIS VI - PIERRE ET MARIE CURIE

Master Acoustique, Traitement du signal et Informatique Appliqués à la Musique

---

Rapport de master

Discipline : Informatique

DEEP SYMBOLIC LEARNING FOR MUSICAL ORCHESTRATION ANALYSIS AND GENERATION

LÉOPOLD CRESTEL

Under supervision of Philippe Esling

Defended on September 5th 2015

Equipe représentations musicales

Institut de Recherche et Coordination Acoustique Musique (IRCAM)

Contents

Abstract

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

In this work, we try for the first time to address these questions in an automatic learning framework. We focus our effort on projective orchestration, which is the transformation from a piece for piano to an orchestral work. One the main objective of this project is to design a system of live orchestral projection. This system would take as input the sequence of chords played by a pianist and generate in real time its orchestration.

By observing a large dataset of orchestral music written by composers and their reduction for piano, we hope to be able to capture through a statistical learning method the mechanisms involved in projective orchestration. Deep neural networks seem a promising lead for its ability to model complex behaviour from a large dataset and in an unsupervised way. More importantly in our project, deep architectures organize hierarchically the extracted knowledge. If we manage to build a satisfying model of projective orchestration, it would probably highlight unknown mechanisms and increase our knowledge for orchestration.

Before tackling the vast and relatively uncovered problem of automatic projective orchestration, we tried to build a generative model for symbolic music for one instrument. Indeed, we believe that an interesting first step in order to build a model for orchestration is to be able to understand the harmonic, rhythmic and melodic structure of polyphonic music. Besides, we can rely on other work and evolve in a more delimited framework with a solid baseline to compare with our model.

As the question raised by our work span a widely different set of knowledge, we start by briefly introducing the state of the art in automatic orchestration, deep learning and its application to symbolic music models. Then we propose our own model based on Conditional Restricted Boltzmann Machines (CRBM) for symbolic music generation, compare it on a predictive task with other existing models and propose an analysis of the musical knowledge learnt by a network. We eventually propose a first step toward an automatic orchestration system with CRBM-based model. If we obtained poor results at the moment, an important step has been made by defining a precise framework made of a database a performance measure and a first baseline model.

Résumé

L'orchestration est l'art d'écrire de la musique pour un ensemble d'instruments en soulignant le développement du discours musical grâce aux nombreuses combinaisons d'instruments possibles. L'immensité du sujet a longtemps laissé songer qu'il était illusoire de vouloir construire une théorie de l'orchestration comme il en existe pour l'harmonie ou le contrepoint. L'orchestration est donc essentiellement abordée de manière empirique et enseignée à travers une collection d'exemples "canoniques".

Nous proposons d'aborder ces questions en nous appuyant sur des méthodes d'apprentissage statistiques. Nous avons concentré nos efforts suer l'orchestration *projective*, qui consiste à transformer une pièce pour piano seul en pièce pour orchestre. En effet, un des principaux objectif de ce projet est de réaliser un système de projection orchestrale temps-réel. Un tel système prendrait en entrée la séquence polyphonique de notes jouées par un pianiste et générerait en temps-réel l'orchestration de cette séquence.

Nous espérons parvenir à extraire les mécanismes mis en jeux lorsqu'un compositeur réalise une orchestration en observant une large base de données de musiques orchestrales et leurs réductions pour piano. Les réseaux de neurones profonds ont démontré leur capacité à modéliser des structures complexes pour de grands ensembles de données et apparaissent comme une piste prometteuse pour l'orchestration automatique. Plus qu'un simple modèle, les réseaux de neurones profonds offrent une véritable représentation et hiérarchisation de l'information contenue dans ces ensembles de données. Construire un modèle satisfaisant d'orchestration projective permettrait également de mettre en évidence et de comprendre certains mécanismes mis en jeux et constituer un premier pas vers une théorie de l'orchestration.

Il nous a semblé que l'élaboration d'un modèle de composition automatique de séquences polyphoniques pour un instrument était une première étape raisonnable avant d'aborder le vaste domaine de l'orchestration automatique. L'objectif était de s'assurer que les architectures que nous souhaitions utiliser était capable d'apprendre les structures harmoniques, mélodiques et rythmique sous-jacentes à toute musique polyphonique. La composition automatique offre en outre un cadre d'évaluation mieux délimité et de nombreux modèles existants auxquels se comparer.

Notre travail aborde de nombreux champs de connaissances. Un premier chapitre introduit brièvement l'état de l'art en orchestration automatique, apprentissage statistique de réseaux de neurones profonds et son application en modélisation de musique symbolique. Le chapitre suivant présente un modèle de génération musicale que nous avons développé et qui est basé sur un modèle appelé Conditional Restricted Boltzmann Machine (CRBM). Notre modèle est ensuite évalué et comparé sur une tache de prédiction avant d'effectuer une analyse du savoir musical appris par le modèle. Nous présentons finalement un modèle d'orchestration projective automatique basé sur le modèle CRBM. Bien que les résultats soient mauvais pour l'instant, une étape importante a été franchie en définissant un cadre de travail solide autout d'une base de données, une mesure de performance et un premier modèle auquel se comparer.

Part I

State of the art

Chapter 1

Automatic orchestration

Orchestration can be defined as the art of writing timbres, spanning from the musical notation to the acoustic realization of an exponential array of instrumental possibilities [TM12]. This complex discipline involves a wide set of intricate mechanisms, most of which have not yet been satisfactorily theorized. Indeed, famous composers often conjectured that orchestration would mainly remain an empirical discipline, which could only be learned through experience and never axiomatized in books. Even if several famous musicians have written orchestration treatises [Ber44, Koe41], those mostly remain recommendations and sets of existing orchestration examples from which one can draw inspiration. This scarce set of knowledge is to be compared with other traditional composition domains, like harmony, which benefits from a long history of theoretical principles and research. However, alike rhythm, harmony and intensity, timbre is "a structuring force in music" [McA13], but even for a trained composer, this projection from the symbolic domain of the score to the acoustic world of timbre can remain particularly elusive.

Hence, trying to build the scientific bases for an automatic orchestration system appears to be a daunting task. The multi-dimensional structure of timbre, its temporal aspects and the combinatorial complexity of instrumental mixtures [ECA10] make the relationships between a given orchestration and its timbre extremely difficult to predict. Few approaches have started to be investigated in the last few years, two of which will be briefly introduced in the following section, either using a set of constraints rules or optimizing micro-temporal objective functions. Those two approaches can in fact be related to the two complementary views on orchestration : *projective* or *inductive* orchestration. Projective orchestration is performed by a composer first writing a "virtual harmonic" score (for instance solely for a piano) and then trying to assign different voices in this score to the various orchestral instruments. Inductive orchestration appears when a composer targets a precise timbre and seek an optimal mixture of different instrument in the orchestra to reach this objective timbre. This inductive problem has been the main line of research over the last years, leading to systems such as Orchids (Section 1.2).
Our objective in this work was to develop the first attempt towards a projective system, by tackling the question of taking in input a piano score and finding potential projections of this score onto an orchestra. In the last section, we will briefly introduce three different methods we have thought of to address this question and provide compelling arguments on why deep learning appeared as the most promising approach amongst these.

1.1 Constraint Satisfaction Problem

One of the first attempt to produce an automatic orchestration system tried to cast the question as a Constraint Satisfaction Problem (CSP) (for a detailed presentation of CSP applied to music, see [Tru04]). CSPs have been widely used for automatic composition, and an automatic orchestration system based on constraints could be defined in the symbolic realm such as proposed in [TA11] or as part of a larger orchestration system [Car08]. Those tools are limited to symbolic constraints, typically enclosed to the set of selected instruments and the notes they are allowed to play (pitch-range or inter-instruments symbolic relationships). However, a major flaw of all these systems is that timbre is not taken into consideration, whereas it is a fundamental dimension of orchestration. This remains a critical limitation as symbolic constraints are not expressive enough to tackle the spectral complexity of orchestration.

1.2 Orchids

An inductive approach has been proposed with the *Orchidée* system [Car08] and later developed to account for temporal evolution in *Orchids* [Esl12]. The goal of these systems is to try to match a target timbre with a given set of instruments. A typical use case is when a composer wants to reach a certain target timbre (often in the form of a recorded sound) through an orchestral mixture, Orchids tries to produce an orchestration that best mimic the target while focusing on the micro-temporal structure of sounds. Several constraints can also be imposed on the number and type of instruments.
This problem can be cast into a Multi Objective Time Series (MOTS) matching problem. Different cost functions (*objectives*) are defined through the different spectro-temporal features. This defines an optimization problem in a complex multi-dimensional space. To solve it, a multiobjective genetic algorithm is used, which offers two advantages: fast computation despite the combinatorial explosion and, most importantly, the fact that it offers a range of widely different solutions, which is a great advantage in a compositional context (a variety of good solutions instead of the best one which anyways doesn't actually exists). Hence, instead of looking for a unique solution, the algorithm offers compromises between the different objectives, by producing the *Pareto front* (a set in which no solution can be considered better than the other). This occurs precisely because of the fact that the optimisation function is multi-dimensional.Furthermore, these functions account for time varying structure as they rely on the time series representations of spectral descriptors. This provides a major improvement over previous systems [Car08], which relied solely on static spectral descriptor for a given target. However, it is well-known that the perception of timbre is strongly impacted by the subtle temporal evolution of spectral features (see [PGS+11] for a detailed study of several audio descriptors).
Orchids is strongly anchored in the inductive paradigm and provides an interesting solution to the specific problem of looking for a target timbre. However, this tool remains limited to short temporal evolutions. This comes from the fact that in order to reduce the search space, an instrument can play only once without performing melodic sequences. If this was possible, the search space would be so immense that it would become impossible to densely visit its shape and hence to find a decent solution. Therefore, Orchids is particularly adapted for composers who want "to start from a timbral grain of sand" that provide them a building block, but can't perform the orchestration of a whole piece yet.

1.3 Approaching projective orchestration

In this section we introduce various ideas that could be pursued as the basis of an automatic projective orchestration system: an extension of the Orchids system to larger time-scales, a spectral constraints

based system, an automatic voice leading extraction tool and a deep learning based approach.

Extending the temporal scale of Orchids As stated earlier, one of the main drawback of Orchids is that it remains confined to rather short temporal evolutions. The original idea, from [EA10], was actually to allow composers to draw spectro-temporal maps and then to find the best orchestrations deriving from this map. Our first idea was, therefore, to extend Orchids towards larger temporalities, by orchestrating several shorter time frames and then trying to link those frames by choosing "similar" orchestration and enhancing the continuity between these. This is where this method shows its limit: it seems very difficult, if not impossible, to define a similarity measure between two unrelated orchestrations at a symbolic level. For instance, the same spectro-temporal effect can be reached with two very different orchestrations (i.e. set of instruments, and notes played by those instruments). Hence, trying to link orchestrated frames would probably results in non-continuous orchestration with absolutely no macro-temporal symbolic coherence.

Spectral constraints Another idea was to extend the CSP method to constraints specifically designed to apply on spectro-temporal descriptors. This solution has been quickly dropped as the domain of spectral variables and their cardinalities would lead to an exponentially growing search space. Therefore, almost all CSP algorithms would most probably won't be able to output a single decent solution.

Voice leading approach A radically different idea was to try to automatically extract from a symbolic musical representation (e.g. a midi file) the most *natural* voices. *Voice leading* makes reference to the way pianist or guitarist jazzmen find smooth transitions between two chords. Finding this voice leading could rely on geometric tools by trying to find the optimal geodesic in a space adapted to chord representations [Tym06]. In practice, the system would first perform a voice extraction, and then try to assign an instrument to each of these voices. We could then imagine to combine Orchids and the voice leading approach by choosing the orchestration for the first frame thanks to Orchids, and then continue the selected solution by following the various voice leadings in the symbolic score figure 1.1 on page 10. However, this approach is highly sensitive to the success of its components, and also lacks flexibility in the definition of the overall orchestral form.

Deep learning approach Automatic orchestration suffers from the fact that human knowledge struggle to model the correlations between high-level features and behaviors with the low-level features that they have access to [BCV13]. From this point of view, a promising solution would be to try to automatically infer the correlations between low-level symbolic representations and the higher-level information that is contained in a complete orchestration. Indeed, we stipulate here and will work based on the hypothesis that there exist an intuitive projection made by composers when orchestrating a piano piece.

Deep learning is a very recent research field that emerged from the field of statistical inference. These algorithms allow to automatically extract increasingly higher-level and complex patterns from a set of training data through a learning phase. Each layer is considered as a more abstract concept, as it is build around a set of correlations inside his immediately lower layer (hence a higher-level concept is a combination of lower-level knowledge). These models can be interpreted as graphical models which have connections between units reinforced during the learning phase when a strong correlation between them is observed. Furthermore, these graphical models with undirected connections provide the utmost advantage of being able to simultaneously learn (analyze) correlations, but also to be able to generate new data.

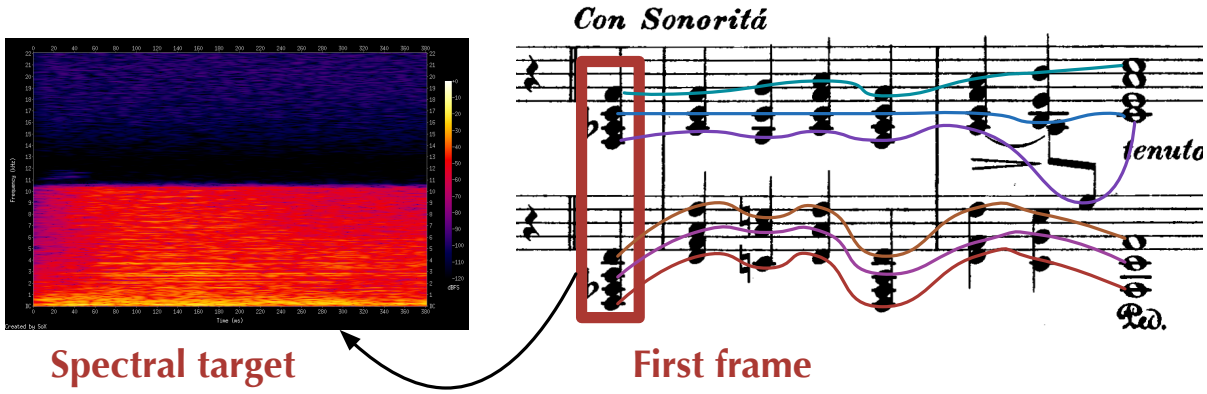


Figure 1.1: Voice leading combined with Orchids. The first frame is orchestrated thanks to Orchids by trying to match a spectral target with the symbolic score acting as a constraint on usable notes. Then, the voice leading algorithm would ensure a continuous orchestration by assigning each note to one of the instrument chosen in the first frame by Orchids.

Hence, apart from the pleasant similarity with the way a human composer learns, statistical inference methods can beneficially take advantage of the vast knowledge contained in already existing orchestral pieces, while proposing, if not totally new by its form and concept, at least unseen orchestrations. Moreover, this very flexible model can actually be adapted for a wide range of different tasks by being able to learn directly from any additional information that we would incorporate in our set of knowledge (we can for instance label orchestrated pieces by genre, and then ask the model to generate an orchestration with respect to this genre).

1.4 Conclusion

Automatic orchestration appears as a daunting task that is yet to be understood and even researched as only few scientific works tried to tackle its immensity. We distinguished two interesting methods in the literature: a CSP-based approach and the Orchids system which offers a nice solution for short temporal frames of inductive orchestration.
However, we aim in this work to address the question of projective orchestration. To propose such a macro-temporal orchestration system, we have foreseen three promising solutions. On one hand, two solutions would use Orchids as a starting point, one to orchestrate a first frame and then ensure the orchestral continuity from a voice leading point of view, a second to create separate orchestral frames that would be linked in a post-processing step. On the other hand, a radically new approach that would use statistical inference to automatically learn orchestration rules. The high potential of the deep learning methods and the fact that they have almost not been used in symbolic music analysis yet brought us to choose this solution over the Orchids-based methods.

The next chapter reviews some of the most important concepts in deep learning, focusing on generative models adapted to time series, as this will be at the core of understanding both the macro-temporal evolution of orchestration and the relationships between a piano score and its orchestral rendering.

Chapter 2

Deep learning

This chapter provides a brief introduction to deep learning, starting from its most basic concepts up to the recent models for symbolic time series analysis. Indeed, the term deep learning actually covers a wide range of models and trying to describe all of them is far beyond the scope of this report and our review will thus be limited to the concepts and models that appeared relevant for symbolic music modeling. We redirect readers interested in deep learning for Artificial Intelligence (AI) and the reason of its development [Ben09], representation learning [BCV13] or wider overviews of the variety of models [Sch15].

2.1 Introduction : Artificial Neural Networks

Neural Networks (NN) NN emerged as an AI attempt to model the way information is processed by the brain[Gar15]. A NN is composed by many simple computational units called neurons that

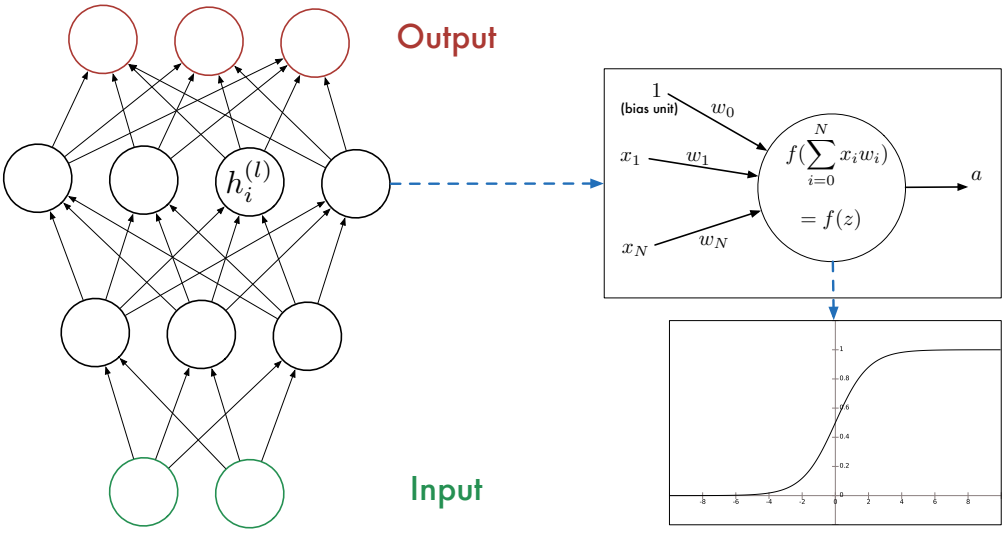


Figure 2.1: A Neural Network is a connectionist architecture where simple computational units are organized by layers. The output of a neuron is the result of its activation function applied to a weighted sum of its input. Neurons are densely connected to each other where the output of neurons in a layer become the input of neurons in the next layer.

implement an activation function. Those neurons are connected together by weighted connections and organized by layers. A neuron in a layer receives as input a weighted sum of the output of the neurons in the previous layer and output the result of its activation function applied to this input (figure 2.1 on page 11). The depth of an architecture refers to the number of layers. A *deep* architecture is simply an architecture with a large number of computing layers.
The activation is a non-linear function traditionally designed to be as close as possible to the step function while being differentiable. Good candidates for those are the sigmoid function or the hyperbolic tangent. In the standard feed-forward neural networks illustrated in figure 2.1 on page 11, activations flows from the input units through various hidden layers and up to the output layer.

Generalizing towards deep architecture In their classical instantiation, weights in a NN were simultaneously trained in all the layers thanks to a global error function. This global error function is typically defined through a classification task by comparing for a set of labelled input its original label to the one proposed by the network. The gradient of this error is back-propagated through the network to reach the weights in every layer [RHW88]. One issue with back-propagation is that the norm of the gradient of the error function decreases as it is propagated through the layers. Thus the most distant weights from the output (i.e. the first layers) of a deep architecture are changing extremely slowly. This problem, known as *gradient diffusion*, made deep architecture (actually as soon as there is more than 2 hidden layers) very difficult to train. However, inspired by the depth of the brain, and using an analogy with the representation of logic circuits [Ben09], many scientists thought that deep architectures would perform significantly better in many tasks. Hence, many efforts have been made to find an efficient learning procedure for deep architectures. Since a satisfying solution has been proposed in 2006 by G.Hinton [HOT06], many state-of-the-art methods in a wide variety of domains have been beaten by deep architectures [Sch15] and proves this approach is sounded.

The training procedure proposed by Hinton is often described as an "unsupervised greedy layer-wise" process. The idea is to successively train each pairs of unit layers formed by a deep neural network as separated small NN (figure 2.5 on page 21). Assembling those separated smaller architecture will be detailed in section 2.4, and how each level is trained in section section 2.3. The weights in each sub-models are trained by trying to minimize a reconstruction error : if the first layer is the visible layer and the second is the hidden, a known visible vector v is passed through the network to the hidden unit h (bottom-up pass) and then sent back to the same first visible unit (top-down pass), defining the visible vector v'. Learning process basically consists in trying to reduce the difference between the original vector v ans its reconstruction v'.

NN can be addressed from a probabilistic point of view, as Graphical Probabilistic Model (GPM). A major advantage is that GPM provide a powerful and sound mathematical framework from which many interesting theoretic results can be easily drawn. Among others, generative properties often naturally follows the formal definition for many architectures. Indeed, if the model allows us to get the distribution of the visible units conditionally on the hidden units, it is possible to draw sample from this distribution in order to obtain data that look like the one existing in our training dataset. After a first section introducing GPM, a second section is dedicated to the Restricted Boltzmann Machine (RBM). RBM is the model used by Hinton to explain the contrastive divergence algorithms, and are the building block of a deep architecture known as Deep Belief Network (DBN) that will be introduced in a third section.

2.2 Graphical Probabilistic Models (GPM)

Graphical models can be interpreted in a probabilistic framework, where each unit represents a random binary variable that have a certain probability to be active (equals to 1) given the states of the other units. The condition and independence relations between units can be drawn as a graph, with computation units being its nodes and edges representing conditional dependence between units. GPM can represent any probability distribution. In practice, it might require an infinite number of units to model perfectly those distributions. But a good approximation can often be reached with a reasonable number of units and an adapted architecture [FI14].

Markov Random Fields (MRF) Markov Random Field (MRF) are a certain class of GPM. A MRF is an undirected graph where the state of a unit is given only by its neighbors. More precisely, a MRF is a graph that respects the Markov property. The Markov property states that a random variable is conditionally independent of all other variables given its neighborhood (see [FI14] for details).

Hidden units To model the distribution of a set of observations of dimension m, it is often required to actually use more than m units. Hence, *hidden* (or *latent*) variables that do not correspond to direct observations, but rather correlations between visible variables, are introduced to increase the expressiveness of a model, and allow to represent complexes distributions. Hence, for a set of visible variables $\mathbf{V}$ and hidden variables $\mathbf{H}$, a model represents the joint distribution $p(\mathbf{v}, \mathbf{h})$, and the visible unit distribution (the distribution of data that we want to model) is given by the marginal probability $p(\mathbf{v}) = \sum_{\mathbf{h}} p(\mathbf{v}, \mathbf{h})$.

Energy-based model From the Markov property given above, it can be shown that the distribution $\mathbf{X} = (\mathbf{V}, \mathbf{H})$ of any MRF can be factorized as an energy-based probability distribution, also known as Gibbs distribution [FI14]:

$$p(\mathbf{x}) = \frac{1}{Z} \sum e^{-E(\mathbf{x})} \tag{2.1}$$

where $E(\mathbf{x})$ is the energy function and $Z = \sum_{\mathbf{x}} e^{-E(\mathbf{x})}$ is the partition function that ensures that the sum of probabilities of all the possible states is equal to one.

Learning in a MRF

Likelihood of the observation Learning in a MRF consists in finding the unknown underlying distribution for a set of data. A common way to find this distribution among statistical models is to rely on adjusting a set of parameters $\boldsymbol{\theta}$ of the model so that the likelihood of the training set S under the MRF distribution is maximized

$$\max_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}|S) = \max_{\boldsymbol{\theta}} \prod_{\boldsymbol{v_S} \in \boldsymbol{S}} p(\boldsymbol{v_S}|\boldsymbol{\theta}) \tag{2.2}$$

However, in order to simplify further computations, models usually rely on maximizing the logarithm of the likelihood (called the *log-likelihood* function)

$$\ln(\mathcal{L}(\boldsymbol{\theta}|S)) = \sum_{\boldsymbol{v_S} \in S} \ln(\frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\boldsymbol{v_S}, \mathbf{h})}) \tag{2.3}$$

Gradient ascent Since it is often impossible to analytically find the maximum of the likelihood function, gradient ascent on the log-likelihood is used as a search procedure to find the optimal vector parameter $\boldsymbol{\theta}_{opt}$. Gradient ascent is an iterative process which uses the derivative of the likelihood with respect to a parameter in order to update the value of this parameter. Since we want to optimize this value for the whole training dataset, the mean value of the log-likelihood gradient over the training set is used (normalized by the number of example in the training database).

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \epsilon . \frac{1}{N_S} \sum_{\boldsymbol{v_S} \in \boldsymbol{S}} \frac{\partial \ln(\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{v_S}))}{\partial \boldsymbol{\theta}} \tag{2.4}$$

where ϵ is the learning rate and N_S is the number of element in the training base.

Gradient of the log-likelihood For a given model $\boldsymbol{\theta}$ and a given training example from the training dataset $\boldsymbol{v_S} \in S$, the gradient of the log-likelihood is given by (see [FI14] for a complete derivation)

Equation 1 (Gradient of the log-likelihood in respect with the parameters)

$$\frac{\partial \ln(\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{v_S}))}{\partial \boldsymbol{\theta}} = -\sum_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{v_S}) \frac{\partial E(\boldsymbol{v_S}, \boldsymbol{h})}{\partial \boldsymbol{\theta}} + \sum_{\boldsymbol{v},\boldsymbol{h}} p(\boldsymbol{h}, \boldsymbol{v}) \frac{\partial E(\boldsymbol{v}, \boldsymbol{h})}{\partial \boldsymbol{\theta}} \tag{2.5}$$

This expression (2.5) can be decomposed as the difference between the expectation of the gradient of the energy under the model distribution and the expectation of the same variable under the distribution of the data in the training examples. It can be rewritten as

Equation 2 (Data and model driven value for the gradient of the log-likelihood)

$$\frac{\partial \ln(\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{v_S}))}{\partial \boldsymbol{\theta}} = - \underbrace{\mathbb{E}_{p(\boldsymbol{h}|\boldsymbol{v_S})} \left[\frac{\partial E(\boldsymbol{v_S}, \boldsymbol{h})}{\partial \boldsymbol{\theta}} \right]}_{\textit{Expectation under data distribution}} + \underbrace{\mathbb{E}_{p(\boldsymbol{h},\boldsymbol{v})} \left[\frac{\partial E(\boldsymbol{v}, \boldsymbol{h})}{\partial \boldsymbol{\theta}} \right]}_{\textit{Expectation under model distribution}} \tag{2.6}$$

Updates strategies : mini-batches In practice, the parameters are not updated by computing the average over *all* the example from the training set, but instead by considering iteratively subsets of the examples (called *mini-batches*) over which an average update is made (typically a hundred examples). This considerably increases the convergence of the algorithm since updates are made more recurrently. Mini-batch update also allows to reduce the noise from using single example while allowing the use of a Graphics Processing Unit (GPU) to decrease computation time.

2.3 Restricted Boltzmann Machine

The RBM can be introduced as a special case of MRF where each hidden unit is connected to every visible unit (with undirected edges), and where visible (respectively hidden) units are conditionally independent. In the first part of this section, we will detail the likelihood and derivatives for an RBM, while making the links between the GPM and NN points of view. In a second part, we explain why the model-driven part of the derivative of the log-likelihood is intractable (second term in (2.6)) and introduce the algorithm approximating this quantity known as *Contrastive Divergence (CD)*.

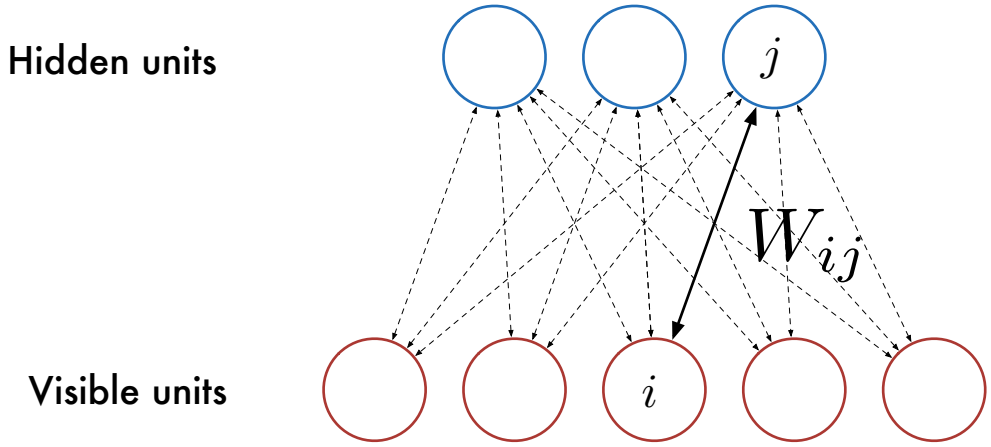


Figure 2.2: The graphical representation of a Restricted Boltzmann Machine (RBM). The weight W_{ij} represent the connection between the visible and hidden units. Visible (resp. hidden) units are conditionally independent from each other (graphically represented by the absence of connection between two visible (resp. hidden) units)

Definition of an RBM

A RBM is defined by a set of m visible units $\mathbf{V} = (V_1, ..., V_m)$ and n hidden units $\mathbf{H} = (H(1), ..., H(n))$, and the parameters of the model are defined by the weights W_{ij} between visible and hidden units, the biases over visible units a_i and the biases over the hidden units b_j (see figure 2.2 on page 15). The energy function of a RBM is given by

Equation 3 (Energy function of a *RBM*)

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\sum_{i=1}^{m} a_i v_i - \sum_{j=1}^{n} b_j h_j - \sum_{i=1}^{m}\sum_{j=1}^{n} v_i W_{ij} h_j \tag{2.7}$$

Different types of units

Depending on the underlying nature of the data that we seek to model, different types of units can be used in a RBM. Binary units have been widely used for binary state data (for instance a musical note being on or off), or discrete data such as pixel intensity by considering each visible unit as a pixel and defining its intensity as the activation probability of the corresponding unit. However, when modeling real data, Gaussian visible units [Hin10] or Rectified Linear Units [NH10] are more appropriate. Which kind of units are the most adapted for music representation will be discussed in the next chapter (chapter 3), but all the theoretical results introduced in the following sections assume that units are binaries.

Gradient ascent in a RBM

Marginal distributions From the energy function (2.7), conditional probabilities for the hidden and visible units can be derived (see section A.1). An important remark is that hidden (resp. visible) units are independent given a visible state $\mathbf{v}$ (resp. $\mathbf{h}$). Therefore, for each visible unit i and hidden unit j

Equation 4 (Conditional probabilities of the visible and hidden units in an *RBM*)

$$p(v_i = 1|\boldsymbol{h}) = sigm\left(b_i^{(v)} + \sum_j W_{ij} h_j\right) \tag{2.8}$$

$$p(h_j = 1|\boldsymbol{v}) = sigm\left(b_j^{(h)} + \sum_i W_{ij} v_i\right) \tag{2.9}$$

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

RBM in the NN paradigm The previous result shows that the RBM (built from the MRF theory), can also be interpreted as a NN, by considering it as a one-layer feed-forward neural network. Its activation function is thus defined by the conditional probability (in this case a sigmoid) and the value taken by a given hidden unit for a set of visible units in the NN is equal to the mean of this hidden unit given the visible units $\mathbb{E}\left[p(v_i|\mathbf{h})\right]$ (resp. $\mathbb{E}\left[p(h_j|\mathbf{v})\right]$).

Gradient of the log-likelihood From the derivative of the log-likelihood for a graphical model (2.6), the energy function in a RBM (2.7) and the conditional distribution of the units (2.8) we can deduce the equation of the derivative of the log-likelihood for a RBM. For a given vector of visible unit from the database, $\boldsymbol{v_S} \in S$, the left part of equation (2.6) (expectation under the data distribution) is easy to obtain (see section A.2 for detailed explanations). For the weights between visible unit i and hidden unit j, W_{ij}, the equation is

$$\sum_{\mathbf{h}} p(\mathbf{h}|\boldsymbol{v_S}) \frac{\partial E(\boldsymbol{v_S}, \mathbf{h})}{\partial W_{ij}} = sigm\left(\sum_{k=1}^{m} W_{kj} \boldsymbol{v}_{\boldsymbol{S},k} + b_j^h\right) \boldsymbol{v}_{\boldsymbol{S},i} \tag{2.10}$$

which is straightforward to compute. However the right part of equation (2.6), (expectation under the model distribution), does not factorize nicely as it is the case for the data driven term. The same calculation gives us the following result

$$\sum_{\mathbf{v},\mathbf{h}} p(\mathbf{h}, \mathbf{v}) \frac{\partial E(\mathbf{v}, \mathbf{h})}{\partial W_{ij}} = \sum_{\boldsymbol{v}} p(\boldsymbol{v}) sigm\left(\sum_{k=1}^{m} W_{kj} v_k + b_j^h\right) v_i \tag{2.11}$$

$$= \sum_{\boldsymbol{h}} p(\boldsymbol{h}) sigm\left(\sum_{k=1}^{n} W_{kj} h_k + b_i^v\right) k_j \tag{2.12}$$

As $\mathbf{h}$ and $\mathbf{v}$ are binary vectors of dimensions n and m, computing the right member of the previous equation with either the first or the second expression require a sum over 2^m or 2^n term, which quickly become intractable as the size of the network grows.

Contrastive Divergence in a RBM

To overcome this intractability, Hinton [HOT06] introduced an algorithm known as CD, which relies on two approximations of the log-likelihood gradient through Gibbs sampling [Ben09].

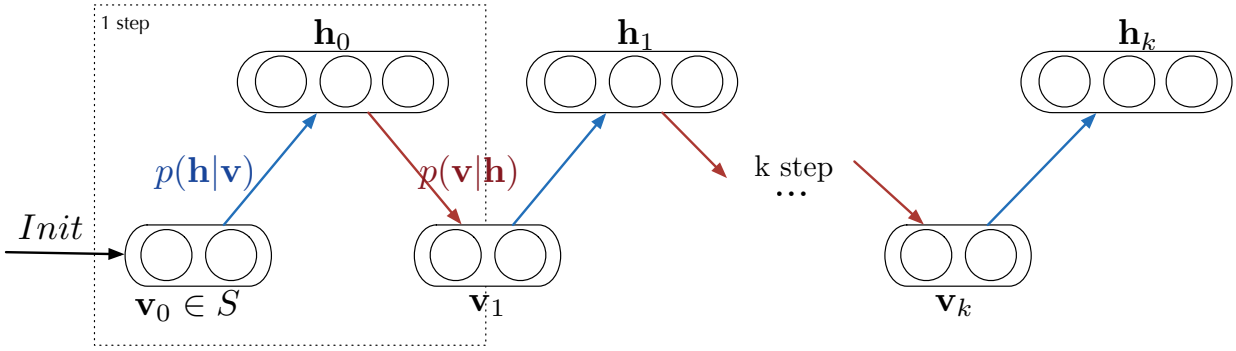


Figure 2.3: Gibbs sampling can be used to obtained a sample from a distribution close to the true distribution of the *RBM*. The independence of the hidden and visible units allows a fast implementation known as *block sampling*.

Gibbs sampling *Sampling from a model* consists in instantiating a random vector $(\mathbf{v}, \mathbf{h})$ from the model probability distribution $p(\mathbf{v}, \mathbf{h})$. Sampling directly from the model distribution is often impossible. In an RBM, it would indeed require to compute the partition function of the model (2.1) (which is intractable as seen previously). Sampling in a RBM is useful for two purposes: both the *learning procedure* and *performance evaluation* of the model. Indeed, generating sample is a way to observe the structure and regularities learnt by the model. Gibbs sampling is a type of Monte Carlo Markov Chain (MCMC) algorithm that can be used to draw sample from an unknown multivariate probability distribution using conditional distribution of each variable. An MCMC is an iterative instantiation of all the variables from previous known values that must be in theory repeated an infinite number of time to reach equilibrium. It can be proved that the chain will converge towards the underlying distribution of the model [Ben09]. MCMC is particularly adapted for an RBM since its joint probability $p(\mathbf{v}, \mathbf{h})$ is intractable but its conditional distributions have a simple factorized expression. The Gibbs sampling approximation (figure 2.3 on page 17) consists in :

- initialize the chain with a vector from the training database
- run only k iterations of the chain instead of running it until it converges

Using a starting vector drawn from the database can be justified by the fact that the model distribution get closer to the data distribution as the training goes on.

Gibbs sampling in a complex MRF can be long since units have to be sampled one by one. However, in an RBM visible and hidden units are conditionally independent. An efficient sampling known as *block sampling* can then be performed, where all units in one layer are updated in parallel given the states of the units in the other layer. Hence, learning can be performed by repeating k iterations of

- sampling a hidden vector from the visible : $\mathbf{h} \sim p(\mathbf{h}|\mathbf{v})$
- sampling a visible vector from the hidden : $\mathbf{v} \sim p(\mathbf{v}|\mathbf{h})$

A very surprising result is that the algorithm will provide very good results even for $k = 1$. More precisely it can be shown that each step of parameter update with CD-1 will increase a lower bound on the likelihood (???? REF ?? ? ?). It should be noted that using one step in the CD algorithm is sufficient for automatic feature extraction, since the algorithm will therefore be fine-tuned using a

back-propagation algorithm [BCV13]. However, when an RBM is learned for generative purposes, it is recommended to use more steps (typically $k = 10$) since a precise evaluation of the joint density is desired[Hin10]. Other forms of contrastive divergence have been investigated in the literature, among them the Persistent Contrastive Divergence (PCD) ([Tie08]). Different tuning of the CD algorithm will be described in the next chapter as it is different for each model.

Expectation approximation The second approximation consists in replacing the expectation over the full data and model distribution by a single sample in (2.6). This considerably reduces the computational load and can be justified by the fact that since the updates are done for each training sample (or mini-batch), there is a form of averaging of the approximation over many iterations. To perform this, the update process for a mini-batch of L training example is given by

- sample hidden states given the training examples $v^{(data,l)}$ of the mini-batch $\boldsymbol{h}^{(data,l)} \sim p(\boldsymbol{h}|\boldsymbol{v}^{(\boldsymbol{data,l})})$
- sample from the model distribution $(\boldsymbol{v}^{(model,l)}, \boldsymbol{h}^{(model,l)}) \sim p(\boldsymbol{v}, \boldsymbol{h})$
- $\Delta W_{ij} = \sum_{l=1}^{L} v_i^{(model),l} h_j^{(model,l)} - v_i^{(data,l)} h_j^{(data,l)}$

Finally, the update rule of the RBM parameters is given by the difference between a model driven term and a data driven term :

Equation 5 (Update rules for the parameter of a RBM)

$$\Delta W_{ij} =< v_i h_j >_{data} - < v_i h_j >_{model} \tag{2.13}$$

$$\Delta b_i^{(v)} =< v_i >_{data} - < v_i >_{model} \tag{2.14}$$

$$\Delta b_j^{(h)} =< h_j >_{data} - < h_j >_{model} \tag{2.15}$$

The computation of the data-driven values is often called positive phase and computation of the model-driven values is the negative phase. The whole CD-k algorithm can be summed up as picking a visible vector from the dataset, run a first bottom-up pass to obtain the model driven hidden units, perform $k - 1$ Gibbs sampling steps (top-down and bottom-up passes) to obtain the model driven visible and hidden units, compute the gradient with the collected statistics.

In practice, it is recommended to use the probability activation of the hidden and visible units when performing CD-k since it reduces sampling noise and correspond in fact to mean values in the case of binary units [Hin10]. However, mean values can be used only when updating the parameters, but not during alternating Gibbs sampling phase, since it would be in contradiction with the fact that visible and hidden units can exchange only a one bit of information. Indeed, this restriction acts as a strong regularizer which enhances the learning.

Regularizing the learning

The performances of a RBM can be improved by imposing some "constraints" on the way the weights of the network are learned. We can distinguish two major ways to influence the learning of a network. First, adding a supplementary term to the error function (often called a *penalty term*), can indirectly forces the behavior of network through the optimisation process (as it puts a form of learning pressure through the error value). The second method is based on directly modifying the value of units or weights in order to impose the desired behavior.

Weight-decay Weight-decay is a penalty function which aims at keeping the values of the weights W in a low range. It works by using the L_2 norm of the weights W as a penalty function: $\frac{1}{2}.\sum_{i,j} W_{ij}^2$. The derivative of this penalty function is null with respect to the hidden and visible biases, and equal to W with respect to W. Hence the update rule for biases remain unchanged and becomes for the weights W :

$$W_{ij}^{(t+1)} = W_{ij}^t + \epsilon \left[< v_i h_j >_{data} - < v_i h_j >_{model} - \nu W_{ij}^{(t)} \right] \tag{2.16}$$

Weight-decay can be interpreted as assuming a zero-mean Gaussian prior on the weights [Hin10]. One important reason to use weight-decay is that it prevents the networks from *overfitting* (with weights arbitrarily drifting to extreme values, thus preventing interesting learning). Hence, low weights helps CD to converge faster [Hin10]. Furthermore, weight-decay helps useless weights to converge faster to 0, which considerably simplify the interpretation of the learnt network and allows to easily visualize the correlation.

Sparsity Sparsity is another regularizer frequently used, whose objective is to obtain a sparse hidden representation of an input data. It means that for a given vector from the training set, we want that only a small number of hidden unit are activated by this vector. Model trained with a sparsity constraint leads to more interpretable representation [Hin10], since hidden units are forced to learn higher-level structures. This is a particularly interesting property for automatic feature extraction. The sparsity function f_{sparse} is defined as the cross-entropy between a desired mean activation p (fixed by the user) and the actual mean activation of the hidden units for a given mini-batch ($q = \mathbb{E}_{p(v)}[p(h = 1|v)] \approx \frac{1}{N_{minibatch}} \sum_{v \in \text{minibatch}} p(h = 1|v)$)

$$f_{sparse} = -p.log(q) - (1 - p).log(1 - q) \tag{2.17}$$

The gradient of the sparsity cost with respect to the different weights of the network can be easily computed, since its derivative with respect to the total input of a given unit is given by $(p - q)$. So, with the input of a given hidden unit j defined by $z = \sum_i W_{ij}.v_i + b_j$, we have

$$\begin{cases} \frac{\partial f_{sparse}}{W_{ij}} = \frac{\partial S}{z}.\frac{\partial z}{W_{ij}} = (q - p)_j.v_i \\ \frac{\partial f_{sparse}}{b_j} = \frac{\partial S}{z}.\frac{\partial z}{b_j} = (q - p)_j \end{cases}$$

Sparsity can also be imposed on the network by directly forcing some units being off (a method known as *dropout*). For each sample from the database, we randomly set to 0 an arbitrary number of hidden units among the one that are activated by this training example ([SHK+14]).

The complete CD-k algorithm that we implemented with all regularizers is provided in section A.3.

Semi-RBM

Definition RBM are a restricted version of a more general model: the Boltzmann Machine. The Semi-Restricted Boltzmann Machine (SRBM) is closer to this general model, by allowing lateral connections between visible units (figure 2.4 on page 20). The energy function of this model is defined by [OH08]:

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i=1}^{m} a_i v_i - \sum_{j=1}^{n} b_j h_j - \sum_{ij} v_i W_{ij} h_j - \sum_{i<i'} v_i v_{i'} L_{ii'} \tag{2.18}$$

The learning procedures for the visible to hidden connections and the biases remain identical, while the learning procedure is additionally applied to the lateral connections

$$\Delta L_{ii'} = < v_i v_{i'} >_{data} - < v_i v_{i'} >_{recon} \tag{2.19}$$

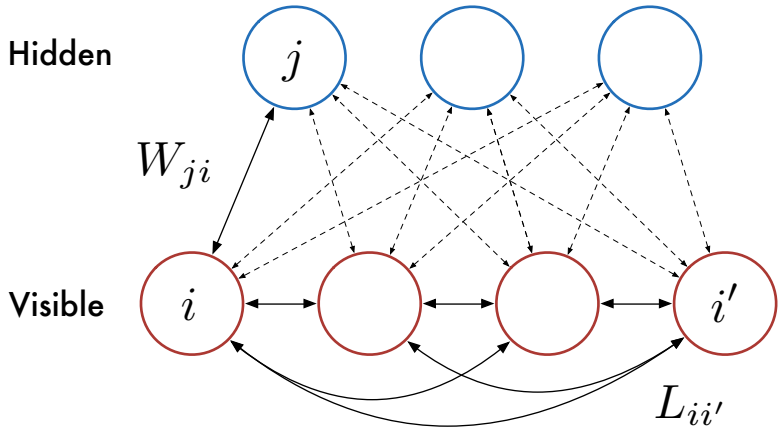


Figure 2.4: *Semi-Restricted Boltzmann Machine.* The RBM model is augmented by adding lateral connections between the visible unit

However, the fact that visible units are no longer independent from each other prevent us from performing block sampling when computing the visible units activation probability. Indeed, when training a SRBM, the positive phase (computing the hidden units activation probabilities) is unchanged, but the negative phase (computing the visible units activation probabilities) must be carefully computed. A visible unit is conditionally dependent of the hidden units, but now also to all the other visible units. To reach the equilibrium distribution given the hidden units, several steps of sequential updates of all the visible units are performed (typically around 50 steps ([OH08])). The fact that block sampling is not possible considerably slow down the training procedure.
One advantage of the SRBM is that strong pairwise correlations in the input are not masking higher structure with weaker correlations. Indeed, lateral connections can model those pairwise interactions and let the weights w take latent structure in charge. It can be seen as a kind of whitening property, where the most basic pairwise correlations are removed from the information to learn. Musically, the SRBM is an interesting starting point to infer basic harmonic rules.

2.4 Deep architectures

As the complexity of the data distribution we seek to model increases, a shallow architecture would require an increasing number of units to obtain a good model of underlying correlations. Oppositely, deep architectures can represent arbitrarily complex functions with a reasonable number of units. This observation relies on more formal results in the computational complexity of circuits. It is possible to model any boolean function with a one layer circuit of logic gates, but a more compact representation can be obtained when implementing several layers of logic gates [BCV13]. Intuitively, this result can be understood by thinking about polynomial factorisation.

Greedy layer-wise training

As explained in section 2.1, deep architectures can not be trained directly with the back-propagation algorithm. The solution proposed by Hinton in 2006 [HOT06] is to train smaller 2 layers architectures (typically a RBM but other model could be developed), composed of an input (visible) layer and a single hidden layer and then stack them to build a deeper architecture. Each group of 2 layers are trained with

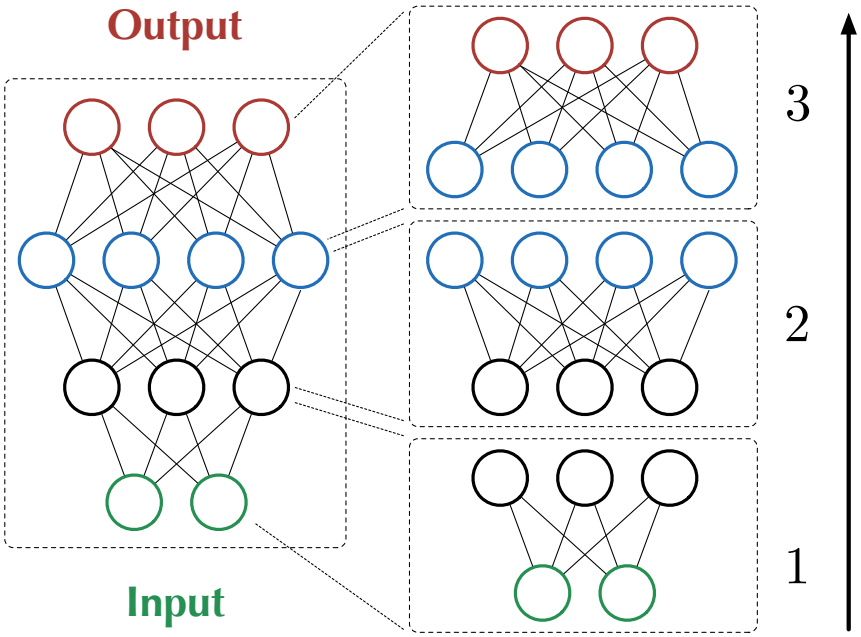


Figure 2.5: Greedy layer-wise training of a DBN

the CD algorithm described in section 2.3. Stacking the RBM is straightforward and computationally effortless. The first layer is trained using the training examples. Then, other layers are trained using as input the activations of previously trained layers given the examples (figure 2.5 on page 21).

Unsupervised training An important difference with back-propagation error is that CD is an *unsupervised* training procedure. Back-propagation requires to define an expected output of the network for each input vector. This is typically the label associated with an input when the network is trained for classification task. But the learning procedure in the CD algorithm is based solely on the capacity of the network in reconstructing the observed data through a reduced representation. This is an extremely powerful property since it allows to train a network on any data that we can find, without a painful pre-processing step consisting for example in labelling by hand an entire dataset. Furthermore, this procedure thrives on the intellectual idea that if we are able to reconstruct something using fewer parts, we are learning its inherent structure.

Different levels of representations A notable property of deep architectures that has been described by several authors is that these approaches are able to learn increasingly higher levels of abstraction in their successive layers [LEN08]. Image modeling gives a convenient framework to understand this property. This leads to the paramount question in deep learning, that is to which amount we can understand what a network has learned ? A way to understand the learning of a network is to visualize the corresponding correlations learned in each hidden unit. The normalized input (in the sense of the L_2 norm) that maximizes the activation probability of a hidden unit is named the receptive field of this hidden unit. Those receptive fields can be interpreted as what higher-level abstractions the network has learned. It can be shown that the receptive field of a hidden unit j is actually the weights W_{ij} between this hidden unit and each visible unit. In the same way, we can observe which hidden units in the first layer give the strongest activation for any unit in the second layer. Since each first layer unit

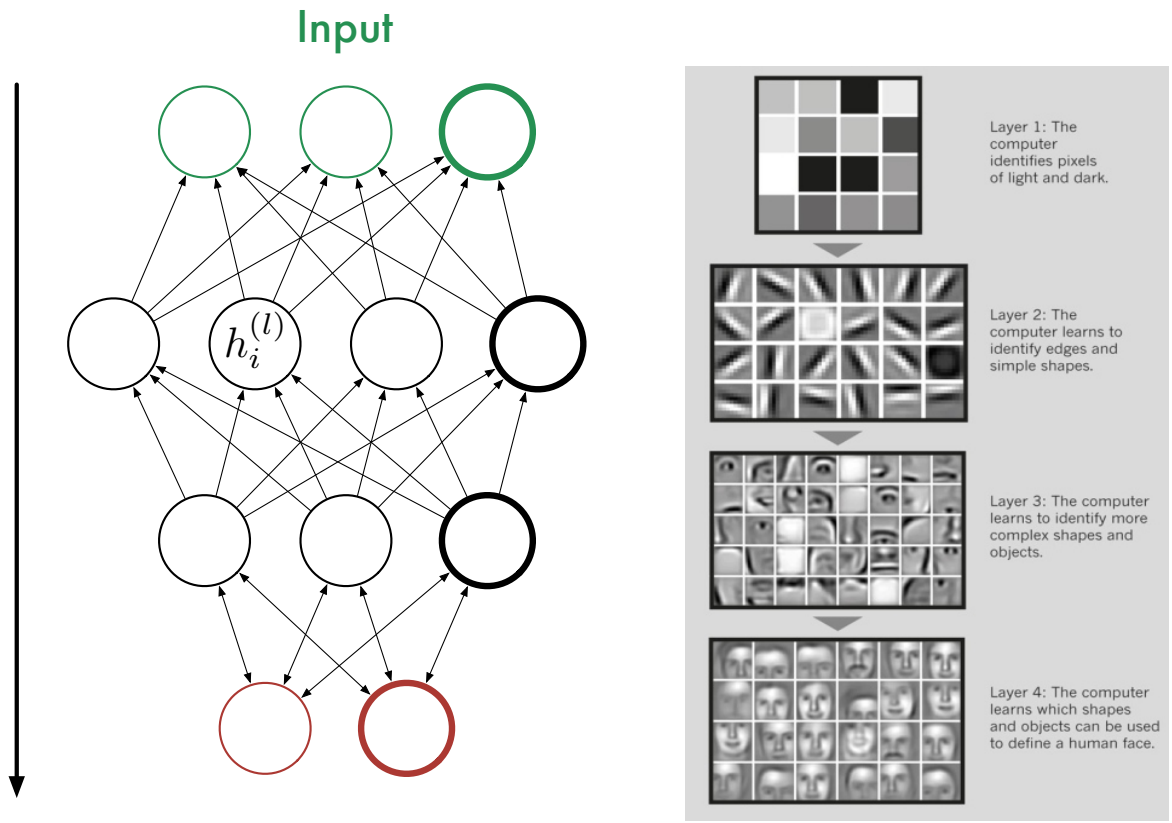


Figure 2.6: Different abstraction levels are hierarchically represented through successive layers of a DBN. Here the directed connections are those pertaining to the generative model (right side of the figure taken from [LEN08]). Be careful that the network has been represented upside-down to fit the image.

has a receptive field which is in the visible layer, what has learnt the second layer is a weighted sum of the receptive fields of the first layer units. We can then obtain the receptive field of each hidden layer by back-propagating its receptive field throughout the network. For image modeling, the hierarchical structure of the network can be easily observed through its receptive fields [LGRN09] (figure 2.6 on page 22) This notion of receptive fields and hierarchical learning is important to visualize what the network has learnt and thus have a better understanding of its behavior. Moreover, it can be interesting to observe the creative mechanisms and patterns learned by the network in order to perform further understanding of the underlying knowledge.

Deep Belief Networks

A DBN is obtained by stacking several RBM on top of each others. Propagating the data trough the already trained layers is simply a forward pass (as detailed in section **??**). When stacking the RBM, two set of biases will be available for each layer of units except for the input and output layers. Indeed, each intermediate layer is trained first as the hidden layer of an RBM and then as the visible layer of another. This leads to two sets of biases, called *generation* and *reconstruction* biases figure 2.6 on page

22. Recognition biases are learned as the hidden layer of the RBM and generation biases are learned as the visible biases of the RBM stacked over the previous one. Those two sets of weights define two different networks. One is a generative model and is used to generate data from a hidden state in the last layer by performing a top-down pass through the network. The other is a recognition model that outputs a hidden representation for an input by performing a bottom-up pass.

A generative models

GPM represent probability distributions. Therefore, it is possible to draw samples from the probability distribution defined by a deep architecture. Sampling from the distribution modeled by a network is often referred to as *sampling a model*. The data produced by a network will of course depend on the examples used during the training phase and the quality of this training. Those considerations will be discussed in the results section (chapter 5).

If we denote $\mathbf{h}^l$ the units formed by the layer l and $\mathbf{v}$ the first layer formed by the visible units, the joint distribution in a DBN is given by

Equation 6 (Sampling equation for a DBN)

$$p(\boldsymbol{v}, \boldsymbol{h}^1, ..., \boldsymbol{h}^L) = p(\boldsymbol{h}^{L-1}, h^L) \prod_{l=1}^{L-1} p(\boldsymbol{h}^{(l-1)} | \boldsymbol{h}^l) \tag{2.20}$$

where $v = h^0$. The generative model of a DBN is then given by the joint distribution of the two top layers that form a RBM and the distribution of each layer conditionally on the next layer. The conditional probabilities are easy to compute, and the joint probability formed by the top RBM can be sampled by performing alternate Gibbs sampling. Note that even though the CD-1 approximation is correct for the training procedure, this does not hold when sampling the model. Indeed, we don't simply want to reduce the reconstruction error anymore, but now really need to reach the equilibrium distribution in the RBM.

2.5 Conclusion

Deep neural networks are a branch of graphical probabilistic models that allows to model efficiently complex probability distributions. Training a neural network on a set of data means finding the weights in the networks so that the probability distribution of the model is as close as possible from the probability distribution of the training data. Training those architectures was impossible until an unsupervised greedy layer-wise training procedure has been proposed, along with what can be considered as the building block of a whole class of neural network: the RBM. By relying on reconstructing the input data, increasingly higher-level representations of the data can be automatically learned. An interesting property of these deep architectures is that they can be used as a generative model.
However, NN and deep learning have been widely investigated for machine vision tasks and engineering systems. Thus, a lot of methods are particularly relevant for image processing (e.g. Convolutional Neural Networks [LBBH98]), but not necessarily adapted for our symbolic music purposes. Proposing models specifically tailored for symbolic music processing is one of the main objectives of this work. The next section focuses on this aspect and the modelling of time series.

Chapter 3

Generative models for high-dimensional time series

Symbolic music can be represented as a high-dimensional symbolic time series and thus benefit from the works in this widely covered field.

3.1 Time modelling

We will focus in this section on statistical models for time series. Indeed, time is a fundamental dimension when trying to model a musical score. As pointed out in [Tay09] time presents a rich structure that can not be considered as a simple additional dimension of static models. In music, the multiple temporal granularities present in the structure of a piece make it particularly difficult to model. Although it is arduous to define the exact temporal granularities underlying a musical piece, they span the full range from the micro-temporal structure that defines the timbre of a sound up to the macro-temporal structure of the musical form. Between those two scales lies the note-scale temporal structure which defines the relation between one note to the next. Previous attempts to define and automatically extract those different temporal granularities have shown that this task is particularly intricate ([NG14]). Recent development in deep learning have focused on efficiently modeling time series. We redirect interested readers to [LKL14] and [Tay09] for an overview of time series modeling.

Time-series specificities

Analogously to the translation and rotational invariance propertied required for vision recognition systems, specific types of invariance might be interesting when modeling time series. Temporal translation and scaling might be two interesting invariance properties for a model of musical time series.

The scaling property is linked to the multiple granularities in musical temporalities. In practice, this multi-granularity generates two issues. The first one is to choose the extent of the smallest temporal event considered. This can be compared to the sampling rate for continuous real-valued series (such as a recorded audio signal). Since we are working with symbolic representations, it would seem logical to choose an integer division of a beat as the smallest temporal event (for instance an eight note), called here a *quantization*. Limitations of this choice arise with the representation of grace notes or rhythm faster or asymmetric to this division, which would be removed in a first approximation. The second issue involved by the multiple temporal granularities is that modeling the macro-temporal structure of a musical piece requires to embed a large historical context which will act as a kind of memory.

It is then obvious that whatever model is chosen, the number of parameters and complexity of the model increases concomitantly with the size of the memory, which renders inference of the parameter of this model (training) equally complex. Besides, for a given context size, finer quantizations will also proportionally increase the model complexity. Overall, this leads to a trade-off between the quantization and size of the memory in our model. Hence modeling long-range dependencies of musical sequences is a challenging task.

Oppositely, translation invariance might not be a desired property when trying to build a generative model for musical time series. Indeed, when musical patterns are repeated (defining the same context) or slightly variated, this repetition information is a valuable information to the overall musical form.

Traditional models

We briefly introduce in this subsection the most well-known statistical models that have been developed to capture temporal structures, as our aim is to build a relatively compact parametric model that could retain the underlying complex temporal structures in symbolic music. Parametric time series models often address the joint probability of a sequence through the conditional probability of a certain time instant given its past. This naturally induces the notion of context. An important distinction can already be made between models that use fully observable context and models with hidden states. The context in fully observable models is entirely defined by the previous point of the sequence whereas hidden states also rely on previous states not directly observed on the sequence but instead inferred from past observations of it. The use of hidden states comes from the idea that sets of points from the observable sequence is insufficient to completely describe the state of the system and that some underlying causes must be modeled to do so.

One of the first parametric model for time series is the autoregressive model, proposed by Yule in the 1920's. The output linearly depends on the N past points of the sequence plus a stochastic term. This model becomes inefficient if an even small non-linearity is introduced in the system. N^{th} order Markov model has then been introduced as a fully observable non-linear system. However fully observable systems fail to model long-range dependencies since only the last N points of the sequence can influence the next point. This is problematic since the number of parameters increase exponentially with the order N of the Markov model.

Introducing hidden states in the Markov Model led to the Hidden Markov Model (HMM). HMM allow to efficiently model long range dependencies in discrete time series with a small number of parameter since the hidden states will retain only the relevant information. Neural networks have recently shown great performances in modeling complex distribution of static data (chapter 2). The internal representation of neurons is particularly interesting since it offers a way to observe patterns and structures learnt by the network. As pointed out by Taylor [Tay09], the fact that computation is performed in parallel over the neurons seems to be in contradiction with the sequential nature of time series. A first solution would be to use these static models and apply them to buffers of sequences. However, such attempts are not really satisfactory since it dilutes the temporal nature of the data by simply considering vectors with higher dimensions. However, it seems more interesting to decouple the time from the other dimensions of the data, by representing those in a static structure, and treating time through a certain context that modulates the properties of this static network.

Before trying to produce an automatic orchestration system, a first interesting step would be to produce a generative model for symbolic music. Modeling symbolic sequences of polyphonic music defines a simpler framework to investigate learning the temporal structure of music. Above all, whereas automatic orchestration is not a widely covered problem and especially not with a neural network approach, many works on automatic composition models define a strong evaluation baseline. Besides, most of the

recent approaches in this field are based on statistical inference models and form a considerable source of information.

3.2 Automatic music generation

Automatic composition can be approached by different methods. One popular method has been to represent a composition problem under a constraint satisfaction problem ([Tru04, HLZ96]). Harmonic, rhythmic or melodic rules are used to filter the set of musical possibilities by assigning penalty to sequences that violate those rules. A drawback of those methods is that its computational load increases exponentially with the length of the generated sequence. Another limitation is that those rules are used by composers to avoid mistakes but are not self-sufficient to create interesting music. At the opposite end of the spectrum, probabilistic inference based systems try to learn those rules from existing pieces of music instead of using hand-written rules. Those models fully benefit from the vast and rich information contained in already existing music written by human composers. This review focuses on probabilistic inference methods.

Similarity with other problems

The problem of automatic composition for polyphonic music can be tackled from the more general framework of generative models of high-dimensional time series. Among this wide domain, text-generative models [SMH11] present a lot of similarity with our problem. Both processes are highly structured sequential stochastic process in a discrete space. But strong differences prevent us from simply applying the existing text modeling solutions to symbolic music generation [LP03]. Hence, while symbolic music vocabulary is relatively small, n-grams or Markov chains fail to correctly model its complex temporal structure composed of short to long-term interactions.

However, N-grams or HMM models have been used to learn the harmonic structure in order to perform harmonisation ([AW05, PBM$^+$03]), which is a very specific task where the structure of the piece is already given trough the bass line. Recent works seem to indicate that a more elaborate and general structure, Random Fields, are more adapted to represent higher level abstractions ([PBM$^+$03, BLBV12, LR14]).

Data representation

In this section, the different architectures to which we will compare our results are presented. An architecture consists in the model used, but also the data representation which is an important milestone when trying to build a musical model. We will focus on symbolic representation (MIDI representation) and will not analyse raw or signal driven data (waveform, spectrogram). Most symbolic music representations can be derived from the two-dimensional piano-roll representation (figure 3.1 on page 27), with the x-axis being the time and y-axis the pitch. Pitch is discrete, often ranging from 1 to 128 (MIDI), or 1 to 88 (piano keyboard) and units are either binaries (n(x,y) indicates if pitch y is played at time y) or continuous (intensity). Although pitch is inherently discrete in classical western music (an octave is made of 12 semi-tones), the piano-roll representation introduces a less instinctive temporal discretisation. It is often based on a symbolic subdivision of the quarter note, which provides tempo-independence but does not allow for physical time evaluation.

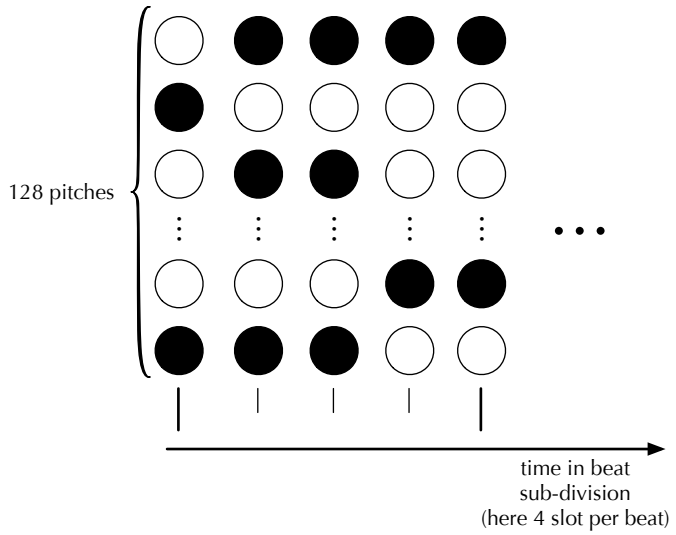


Figure 3.1: The pianoroll representation for symbolic rhythm

Previous architectures

HMM-based models have been widely used for harmonic structure analysis [AW05, PBM+03, PR01] or melodic composition [YAK10], but not for polyphonic composition. Indeed, HMM are not well-suited for multi-dimensional data and, thus, remained targeted at monophonic or chord sequences. In harmonisation models ([AW05]), an interesting "root relative" representation is used. Each note is represented by its distance to the root (given the key), while a chord label is added to indicate the harmonic context of the chord (its degree).

HMM are a particular case of MRF, and Lavrenko proposed to use a less constrained lattice of binary units directly derived from the piano-roll representation[LP03]. Here, the time structure is eluded by only considering the notes onset, while removing their duration. The efficiency of such an under-defined representation can be doubted, but it offers a first easy-to-handle representation. Intuitively, this representation might perform well on music strictly aligned on a temporal grid such as Bach's chorales. Besides, pitches are reduced to 12 through an octave-equivalent transformation. This transformation (named *pitch-class transformation* here) is interesting as it can help the system focusing on the harmonic structure of music. The field is built on *directed* temporal connection, which is a reasonable assumption that only notes played in the past influence notes in the future. A more surprising and restrictive assumption is that a certain pitch is only influenced by lower pitches at the same time. The distribution of the field is defined through a set of feature functions and parameters, representing rules learned on a training dataset.

Several works are based on RBM, such as systems that automatically creates Jazz melody over a sequence of chords [BBSK10]. Here, a data representation well fitted for their problem is to perform a separate representation of chords and melodies. However the poor temporal structure (simple concatenation of successive frames) of their system appears as an immediate limitation. Very recently, two models aimed at automatic music generation specifically targeted the temporal dimension and were able to generate interesting music sequences.

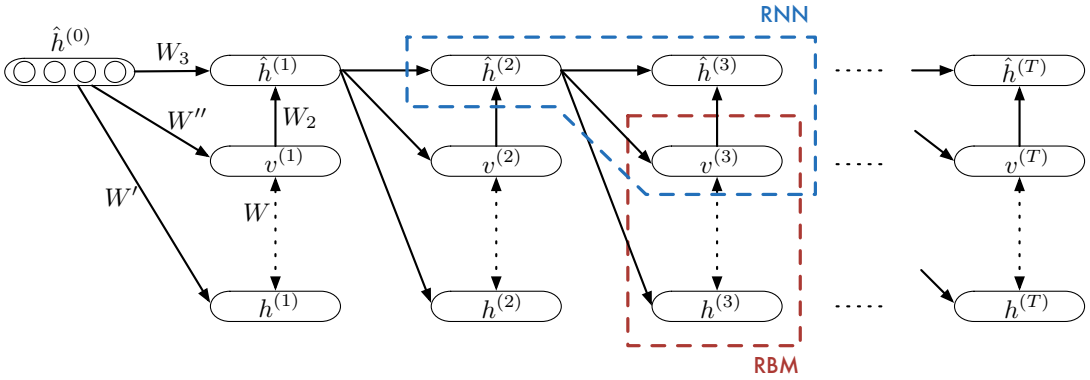


Figure 3.2: Graphical structure of the RNN-RBM. We can distinguish the two parts of the model, RNN in blue and RBM in red.

Recurrent Temporal RBM (RTRBM) A first paper [BLBV12] introduces two models with a temporal component used to produce polyphonic music sequences : the Recurrent Temporal Restricted Boltzmann Machine (RTRBM) and the Recurrent Neural Network Restricted Boltzmann Machine (RNN-RBM) (see figure 3.2 on page 28). Those two models have been inspired by the aforementioned text-generative model [SMH11]. Beside, an interesting evaluation framework is proposed in this article and will be used to evaluate the performances of our model (section 5.1). The general definition of a Temporal Restricted Boltzmann Machine (TRBM) introduced by [SHT09] is a sequence of RBM where each RBM is conditioned by the previous one. It can be seen as a HMM with an exponentially large state space (increases with N^t) but a compact parametrization (only 2 transition matrices and biases). In fact, in the simplest TRBM, only the bias of the hidden units is affected by the previous hidden unit.

The probability of a sequence of observation is given by

$$P(v_1^T, h_1^T) = P_0(v_1, h_1).\prod_{t=2}^{T} P(v_t, h_t|h_{t-1}) \tag{3.1}$$

which is the equation of an HMM, and where the conditional distribution $P(V_t, H_t|H_{t-1})$ is the distribution of an RBM whose hidden bias H_t is a function of h_{t-1} (named *dynamic bias*).

$$P(v_t, h_t|h_{t-1}) = exp(v_t^\top b_V + v_t^\top W h_t + h_t^\top (b_H + W' h_{t-1}) \tag{3.2}$$

The main difference between the RNN-RBM model and the RTRBM is that the recurrent relation is based on the mean-field values of the hidden units instead of their binary values. Hence, those mean-field values units are separated from their binary alter ego and define another set of weights that models the recurrent relations. A RNN-RBM can be trained using the Back-propagation though time (BPTT) algorithm [RHW85].

Long Short-Term Memory (LSTM) The paper [EL08] addresses blues improvisation by proposing a model based on a slightly modified Long Short-Term Memory (LSTM) network [ES02]. More precisely, their model is composed of an input layer, an output layer, and a hidden layer made of a standard feed-forward layer in parallel with several LSTM blocks, and use the same aforementioned evaluation framework [LR14]. A *LSTM* block in a network plays the same role as any other computational unit

(they compute a single output value from weighted inputs), but have a more complex structure. Since the generation process is a random selection, very unlikely musical event can still happen. Even if we want to keep a certain amount of randomness in our generative process, we still don't want those very unlikely event to happen. To do so, a threshold is applied before sampling from the output distribution in order to avoid very low probability notes to be chosen.

From this short review, we retain that a good model stands in the data representation used, the choice of the architecture and the ability to train it correctly. Whereas most of the work in automatic composition only rely on a qualitative evaluation of their result, the two last cited articles proposed a common quantitative evaluation framework. For these reasons and the fact that their results were the most interesting, we will use them as a baseline to compare our model to (see section 5.1).

3.3 Conditional models

We introduce a temporal model, the Conditional Restricted Boltzmann Machine (CRBM) proposed for modeling motion style [Tay09]. Motion data is collected by placing sensors on the different body parts of a walking person. This leads to sets of high-dimensional time series, and this model appeared promising to be applied to our problem of modeling music sequences.

Model distribution

Distribution of a CRBM The energy function of a conditional RBM (figure 3.3 on page 30), is similar to the RBM's one, but the biases of both visible and hidden units are modified by a term introducing a dependency to the past visible units. So, for binary visible and hidden units, it is defined by

$$E(v_t, h_t|v_{<t}) = -\sum_i \hat{a}_{i,t} v_{i,t} - \sum_{ij} W_{ij} v_{i,t} h_{j,t} - \sum_j \hat{b}_{j,t} h_{j,t} \tag{3.3}$$

where the biases are defined by $\hat{a}_i = a_i + \sum_k A_{ki} v_{k,<t}$ and $\hat{b}_j = b_j + \sum_k B_{kj} v_{k,<t}$. a and b are static biases, as in the RBM model, and A_{ki} and B_{kj} model the influence of the context on the current state.

We can sample this model thanks to the conditional probability of the visible units given the past visible and current hidden, and the conditional probability of the hidden units given the past and current visible units.

$$p(h_{j,t} = 1|v_t, v_{<t}) = sigm(-\hat{b}_{j,t} - \sum_i W_{ij} v_{i,t}) \tag{3.4}$$

$$p(v_{i,t} = 1|h_t, v_{<t}) = sigm(-\hat{a}_{i,t} - \sum_j W_{ij} h_{j,t}) \tag{3.5}$$

We can see in those equation that efficient block sampling is still allowed (no correlation between hidden or visible units of the same layer). Besides, if N is the temporal order of the model (which means that only the N last visible unit layers are taken into consideration), we can directly sample the $(N+1)^{th}$ visible units from the N previous visible unit from our database, without running N sampling steps which is the case in a general Temporal RBM. Hence, this considerably increase the computational efficiency at generation time. Generating data from a CRBM require to first feed him with a context. Input units are clamped while sampling the next frame $v(t)$ by alternate Gibbs sampling in the RBM part of the model. Sampling in conditionals models is described in a next section section 3.3.

Weight updates are made by contrastive divergence :

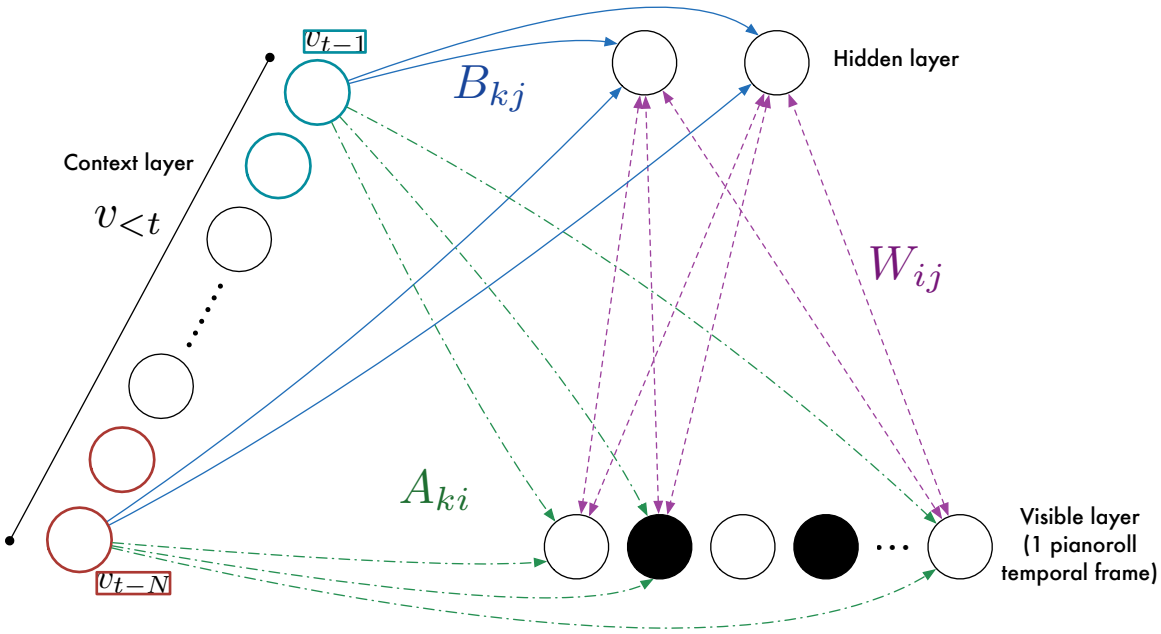


Figure 3.3: *CRBM.* The weights A_{ki} and B_{kj} model the influence of the past visible states on the biases of the current visible and hidden units.

Equation 7 (Update rules for the parameter of a RBM)

$$\Delta W_{ij} =< v_i h_j >_{data} - < v_i h_j >_{model} \tag{3.6}$$
$$\Delta b_i^{(v)} =< v_i >_{data} - < v_i >_{model} \tag{3.7}$$
$$\Delta b_j^{(h)} =< h_j >_{data} - < h_j >_{model} \tag{3.8}$$
$$\Delta A_{ik} =< v_i v_{k,<t} >_{data} - < v_i v_{k,<t} >_{model} \tag{3.9}$$
$$\Delta B_{jk} =< h_j x_k >_{data} - < h_j x_k >_{model} \tag{3.10}$$
$$\tag{3.11}$$

Conditional RBM can be stacked into a deep architecture, in the same way that RBM are stacked to build a DBN, but one has to be aware that sampling from this model require to initialize it with a context frame. Indeed, the top associative memory formed by the CRBM of the last layer has to be initialized using the recognition weights of the lower layers. Since our model is context-based, it involves to have a knowledge of the past frames. Once the context of the last layer is set, alternate Gibbs sampling can be performed in order to reach the equilibrium distribution. A top-down pass is then performed to sample the current frame of visible units given the context (see figure 3.4 on page 31). Note that the frame generated then define the context for the next frames that will be generated.

Gated connections

In the CRBM model, input (context) units directly influence the bias of both hidden and visible units. If we consider the activation function, this is equivalent to shift this function towards the positive or negative end of the abscissae. Instead of this additive interaction, the model can be influenced by the

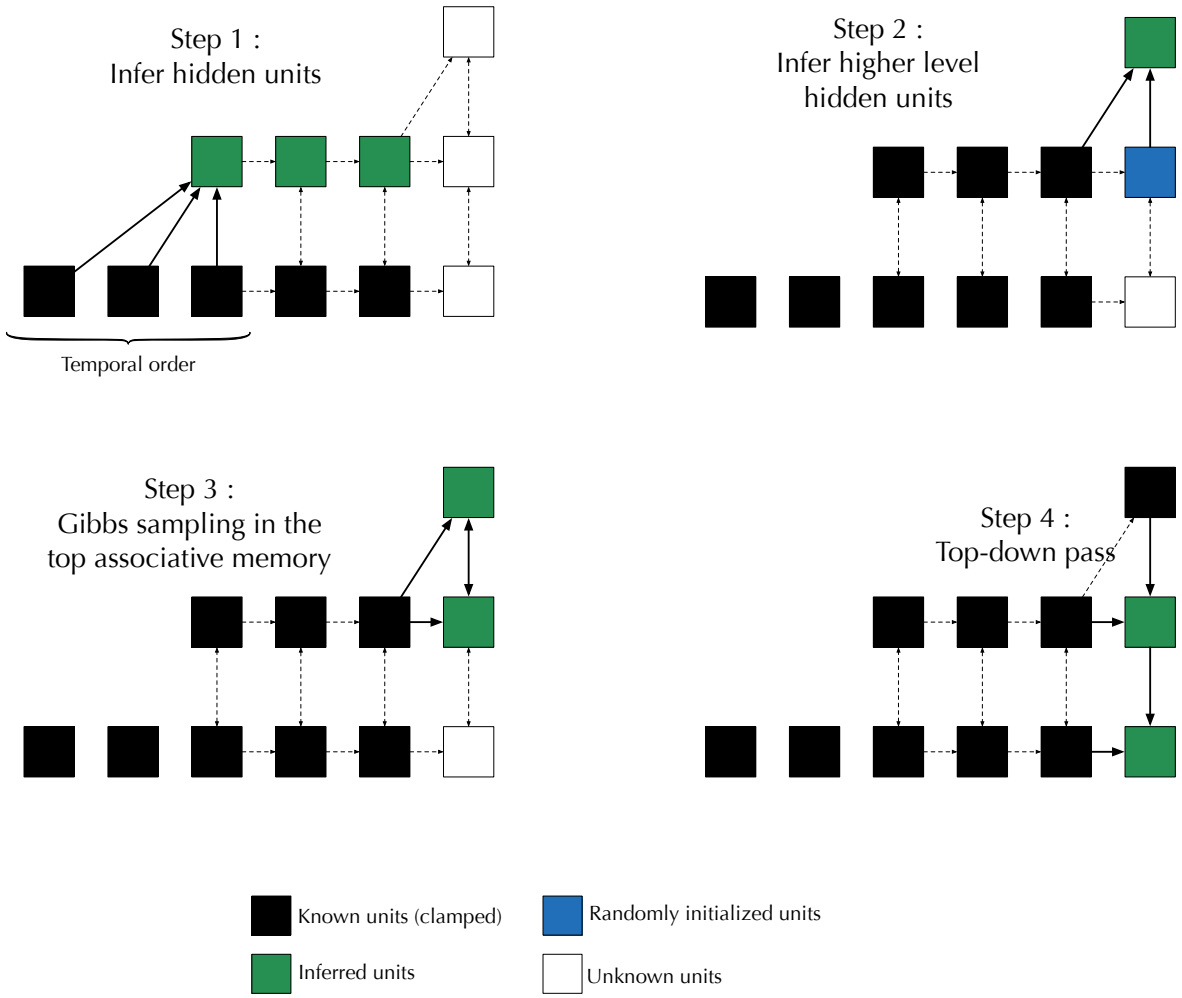


Figure 3.4: Sampling in a multi-layer CRBM

past in a multiplicative way. Hence, those multiplicative modulations directly modify the slope of the activation function, which in turn *scales* the sensibility of the activation according to the input. The general model distribution for a Gated Conditional RBM (GCRBM) can be derived from the energy function given by

$$E(v,h|x) = -\sum_{ijk} W_{ijk} v_i h_j x_k - \sum_{ij} c_{ij} v_i h_j - \sum_i a_i v_i - \sum_j b_j hj \tag{3.12}$$

W_{ijk} are the component of a three way tensor and c_{ij} are the gated biases. The computational cost implied by the tensor can be reduced by factoring it into a product of pairwise interactions : $W_{ijk} = \sum_f W^v_{if} W^h_{jf} W^x_{kf}$ where the superscript indicate which unit the weight refers to, and f indexes the factors. If $\#\{\text{input unit}\} \simeq \#\{\text{hidden unit}\} \simeq \#\{\text{input unit}\} \simeq \#\{factor\} \simeq N$, the complexity is reduced from $O(N^3)$ to $O(N^2)$ (exactly $O(3 * N^2)$). The energy function is thus modified :

$$E(v,h|x) = -\sum_f \sum_{Ijk} W^v_{if} W^h_{jf} W^x_{kf} v_i h_j x_k - \sum_{ij} c_{ij} v_i h_j - \sum_i a_i v_i - \sum_j b_j hj \tag{3.13}$$

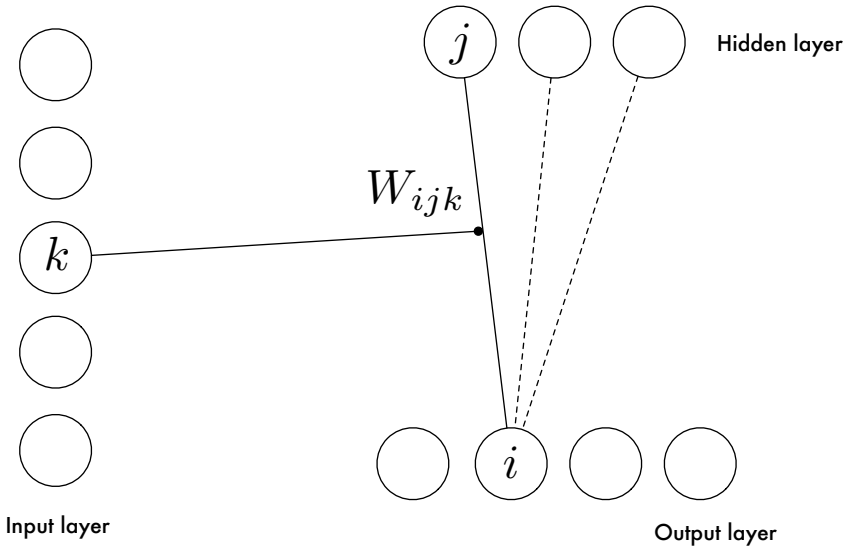


Figure 3.5: Input units in a GCRBM modulates the weights between hidden and output units

A Factored Gated RBM (FGRBM) can be trained using CD. We use the following notations: at a given time t, v_i are the visible units, h_j the hidden unit, x_k the input units which are the visible units of the $(t-N)$ previous time frames and z_l are the label units. The factor for the matrix W_{ijk} is indexed by f, for A_{ikl} by m and for B_{jkl} by n. The update rules are then given by

Equation 8 (Update rules for the parameter of a FGRBM)

$$\Delta b_i^{(v)} = < v_i >_{data} - < v_i >_{model} \tag{3.14}$$

$$\Delta b_j^{(h)} = < h_j >_{data} - < h_j >_{model} \tag{3.15}$$

$$\Delta W_{if}^{v} = < v_i \sum_j W_{jf} h_j \sum_l W_{lf} z_l >_{data} - < v_i W_{jf} h_j \sum_l W_{lf} z_l >_{model} \tag{3.16}$$

$$\Delta W_{jf}^{h} = < h_j \sum_i W_{if} v_i \sum_l W_{lf} z_l >_{data} - < h_j \sum_i W_{if} v_i \sum_l W_{lf} z_l >_{model} \tag{3.17}$$

$$\Delta W_{lf}^{z} = < z_l \sum_i W_{if} v_i \sum_j W_{jf} h_j >_{data} - < z_l \sum_i W_{if} v_i \sum_j W_{jf} h_j >_{model} \tag{3.18}$$

$$\tag{3.19}$$

A style-gated factored model For modeling different motion styles, [TH09] introduced a very interesting model where the direct influence of the past is modeled through a CRBM and stylistic feature labels are gated on either the weights W or the dynamic biases matrices A and B. Hence W, A and B are both 3 dimensional tensors. This model (illustrated in figure 3.6 on page 33), has been named Style-gated Factored Conditional RBM (Style-gated FCRBM). Its impressive performances for modeling human motion, its great modularity and the fact that it could be easily adapted to the orchestration problem proved attractive. The adaptation to the orchestration problem will be detailed in chapter 6), but it relies on using the style labels as the piano input and ensuring a continuity in the orchestration using the recent past of the orchestration to dynamically bias the visible and hidden

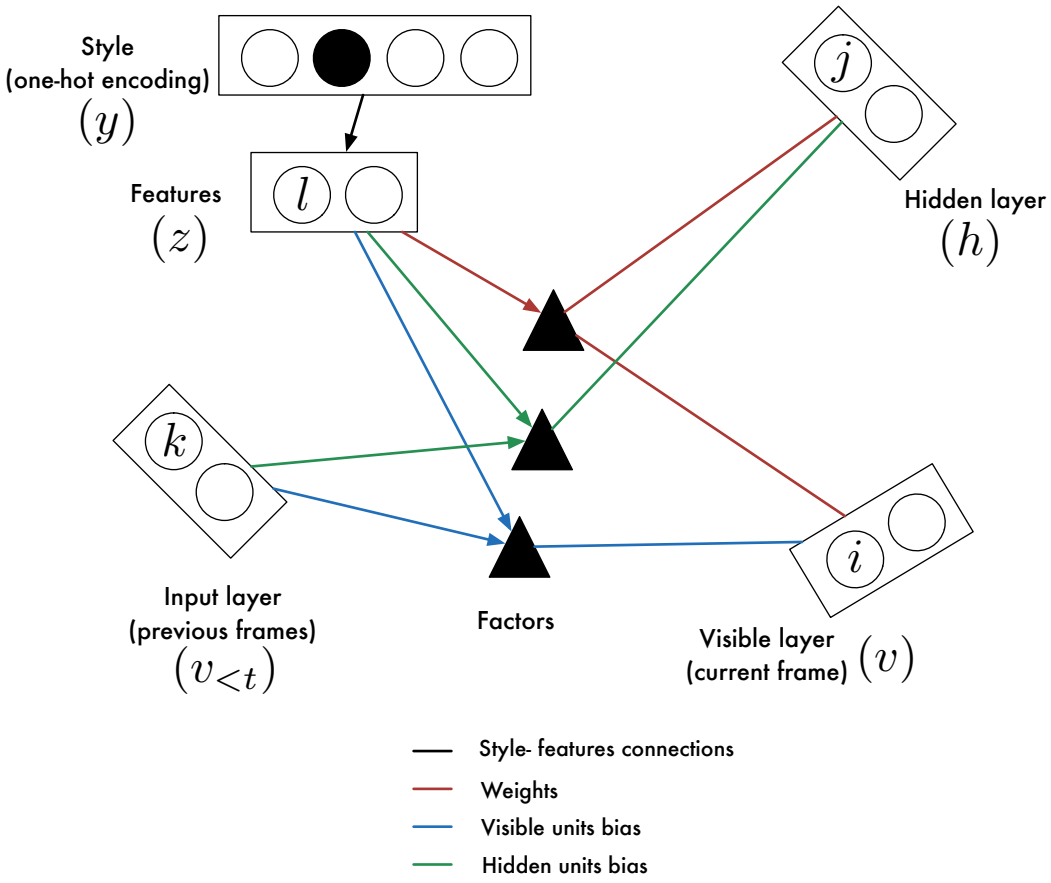


Figure 3.6: A style-gated FCRBM. The three sub-models are represented by three different colors. The style features are gated on the three interactions: weights between visible and hidden units (in red), bias on hidden units (green), bias on visible units (blue)

current unit. The energy function in a Style-gated FCRBM is given by

$$E(v_t, h_t|v_{<t}, y_t) = -\sum_f \sum_{ijl} W^v_{if} W^h_{jf} W^z_{lf} v_{i,t} h_{j,t} z_{l,t} - \sum_i \hat{a}_{i,t} v_{i,t} - \sum_j \hat{b}_{j,t} h_{j,t} \tag{3.20}$$

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

$$\hat{a}_{i,t} = a_i + \sum_m \sum_{kl} A^v_{im} A^{v<t}_{km} A^z_{lm} v_{k,<t} z_{l,t} \tag{3.21}$$

$$\hat{b}_{j,t} = b_j + \sum_n \sum_{kl} B^h_{jn} B^{v<t}_{kn} B^z_{ln} v_{k,<t} z_{l,t} \tag{3.22}$$

where m indexes the factor for the gated interactions between features, input and visible units and n the factor for the gated interactions between features, input and hidden units.

A Factored Gated Conditional RBM (FGCRBM) can be trained using CD. We use the same notations as the previous model, as the update rules for b and W remain the same. The new update rules for A and B are then given by

Equation 9 (Update rules for the parameter of a FGCRBM)

$$\Delta A^v_{im} =< v_i \sum_k A_{km} x_k \sum_l A_{lm} z_l >_{data} - < v_i \sum_k A_{km} x_k \sum_l A_{lm} z_l >_{model} \tag{3.23}$$

$$\Delta A^x_{km} =< x_k \sum_i A_{im} v_i \sum_l A_{lm} z_l >_{data} - < x_k \sum_i A_{im} v_i \sum_l A_{lm} z_l >_{model} \tag{3.24}$$

$$\Delta A^z_{lm} =< z_l \sum_i A_{im} v_i \sum_k A_{km} x_k >_{data} - < z_l \sum_i A_{im} v_i \sum_k A_{km} x_k >_{model} \tag{3.25}$$

$$\Delta B^z_{jn} =< h_j \sum_k B_{kn} x_k \sum_l B_{ln} z_l >_{data} - < h_j \sum_k B_{kn} x_k \sum_l B_{ln} z_l >_{model} \tag{3.26}$$

$$\Delta B^z_{kn} =< x_k \sum_j B_{jn} h_j \sum_l B_{ln} z_l >_{data} - < x_k \sum_j B_{jn} h_j \sum_l B_{ln} z_l >_{model} \tag{3.27}$$

$$\Delta B^z_{ln} =< z_l \sum_j B_{jn} h_j \sum_k B_{kn} x_k >_{data} - < z_l \sum_j B_{jn} h_j \sum_k B_{kn} x_k >_{model} \tag{3.28}$$

Sampling from conditional models

As aforementioned, sampling in a conditional model requires to feed the network with N already existing frames to build the input and the context is then shifted on the recently created frames to iteratively generate the successive frames. Given those N past frames, the sampling process to get the $N + 1^{th}$ visible frames is the same in every conditional model :

- Infer hidden context units if the model has several layers.
- Initialise the units of the penultimate layer (i.e. visible units in the model formed by the top two layers). If the model is not stacked, this is simply the visible layer.
- Given the inferred input units and the initialised visible units, perform several Gibbs sampling step in the top associative memory formed by the two last layers.
- If the network has more than 2 layers, perform a top-down pass through the network to the visible layer.

Hence, we have several degrees of freedom here through the choice of the temporal order N, number of Gibbs sampling steps, initialization of the visible units, and whether or not we actually sample the visible unit or keep mean-field updates. The number of Gibbs sampling steps needs to be higher than when training. Indeed, when training, one step is sufficient for approximately finding the steepest gradient direction. Here we are looking for a value as close as possible to the model distribution and must then let run the chain for considerably larger number of iterations. We typically set this value between 30 and 100. Initializing the visible units can either rely on the value of the previous frame or use randomly initialization. Although initializing at the value of the previous frame appears as a logical choice for smooth and continuous data, this might not be the case in symbolic music series since notes suddenly stop and their values drop from 1 to 0 between two frames. In practice, we observed this initialization tends to favor notes staying on forever, which is clearly not a desired behavior. Hence, we chose to initialize the visible units using a uniform random law. Two question arises: should we initialize by sampling this distribution or just using the real probability value? Should we use binary values or activation probability for the visible units at the end of the sampling process?

For the initialization, the answer is definitely to chose mean-field activation. Using binary values is more likely to place us in a poor local minimum of the energy function which would be very difficult to leave. When generating a long sequence (actually more than 1 sample), each generated frame is used to extend the context before generating the next frame. Based on the results presented in [BLBV12] we decided to use mean-field value as it allows exact inference.

Comparison with the RNN-RBM model

Main differences The conditional models (CRBM and derivatives) we evaluate are quite similar to the recurrent models (RTRBM and RNN-RBM models) presented in [BLBV12]. We have actually apprehended two main differences

- Instead of defining a simple auto-regressive relation between the past and current visible and hidden unit (as in our model), the RTRBM and RNN-RBM models induce a Markov chain architecture on the hidden units.
- In conditional models, the complete context is directly accessible at each time frames whereas in recurrent models, the context is expressed through a chain structure. This considerably modify the learning procedure as recurrent models require to back-propagate through time the gradient of different quantities which is not the case in conditional models.

Motivation for a conditional model Our choice of investigating conditional models is motivated by several reasons

- Stacking a recurrent temporal architecture is not easy. While another RBM can be added on top of the first, the whole temporal structure cannot be stacked. In a conditional model, stacking the whole architecture is possible and does not significantly increases the complexity. Hence, we address the behavior of multi-layer architectures for modeling musical sequences.
- A prime objective was to be able to understand what structure was learnt by the network. Since the transition in the conditional model occurs from the past visible units, it is easier to understand the learnt temporal relations (see section 5.3).

3.4 Conclusion

Time modeling in statistical models has been a stumbling question for many decades. The recent advent of deep architecture still falls prey to this daunting problem while offering new perspectives. Among the most promising attempts to efficiently model high-dimensional time series, two of them have been applied to musical sequences modeling. While they are both based on Recurrent Neural Network, they present some differences. The first model [LR14] adds a type of memory cell to a static structure, which modulates the signal that appears between visible and hidden layers. Its ability to automatically scale the size of its time memory according to the context and model very long time lags makes it particularly fitted for modeling the macro-temporal structure that occurs in musical series. The second model, a RNN-RBM [BLBV12], includes in a static RBM model a temporal structure similar to a Markov chain. The two cited articles are the only one, at our best knowledge, to provide a quantitative evaluation and will then be used as a baseline to evaluate our proposed model.

We propose to extend the model based on a CRBM [Tay09, TH09] where the context modifies in an additive way the biases of both the visible and hidden units and a Gated RBM (GRBM) where context

units modulate the weights W_{ij} of a standard RBM in a multiplicative 3-way interaction. Those two models can be factored to reduce the number of parameters and in the case of the 3-way interaction be able to perform efficient matrix computation. Finally, those two models can be mixed into a complex structure called GCRBM which can also be factored (FGCRBM).

Those architectures present several advantages. They are generative (a requirement in our case) and offer a great modularity. The different parts of the architectures can be connected in many different ways, by gating the bias only, the weights, both or use the difference between gated connection and dynamic biases to add different kinds of context. An unparalleled advantage is the possibility to easily observe the temporal relations learnt through the auto-regressive matrices A and B. This is particularly interesting since it is then possible to "understand" how the network generates music.

Part II

CRBM-based models for symbolic music

Chapter 4

The Harmonic Conditional RBM

In this chapter, we introduce our model that rely on the temporal structure of the CRBM model [Tay09] (section 3.3), combined with a SRBM model [OH08] (section 2.3) in order to model the harmonic structure. Generative models usually lack of quantitative evaluation of their performances. Hence, we have investigated the performances of a range of CRBM-based models (already existing models but not applied to music generation) on a prediction task, and compare them with previous music generation models introduced in the previous chapter (section 3.2).

We used the same data representation than most of the work introduced in the state of the art (section 3.2). Hence, each visible unit represent a pitch and the vector of visible units represent a temporal frame of the pianoroll. We usually set the number of visible units (possible pitches) to 88, which is the size of a piano keyboard. Several pre-processing can be applied to this data representation that can considerably modify the performances of the models. Those occur mainly at two level: on the pitches (pitch-class or root centric) or on the rhythm (quantization). There are detailed in the last section of this chapter. Since those models are sensitive to the tuning of their hyper-parameters (such as learning rate, sparsity or weight decay), we performed an automatic search based on a kernel function fitting in order to find the optimal set of parameters and will present the quantitative evaluation of all models in the next chapter.

4.1 Modelling the harmonic structure

Before trying to model a complex system with both temporal and harmonic relations, we tried to model an harmonic structure without any temporal context. Even if it is clear that the harmonic structure cannot be agnostic of time, it seems reasonable to think that a model able to understand isolated chords would be a good starting point to build a model for musical sequences. Inspired by the data representation of [LP03], we discarded the temporal dimension by considering isolated chord and trained several models with this data representation.

Polyphonic music have a strong harmonic structure, which we could model by including lateral connections between the visible units, alike the model proposed in [OH08] (section 2.3). Allowing every kind of connections between the units leads to a very general model, but would require a huge number of training example and a long training time to explore every kind of combinations. A more structured type of lateral connections can be used, that would quickly capture the underlying harmonic regularities and provide an inherent shifting and transposition invariance in the relation between notes.

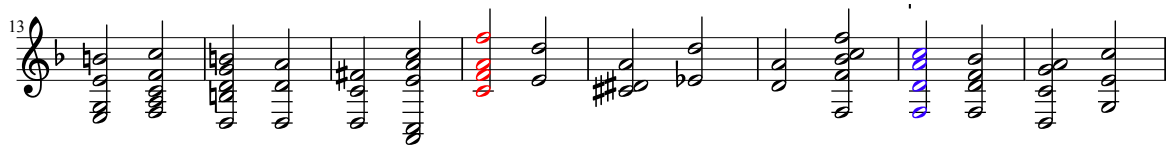


Figure 4.1: Chords learned by a Harmonic RBM trained on Bach's 4 voices Chorales. No temporal structure is modeled here, and the half note duration is arbitrarily chosen. Most of the chords are very classical (in red a major chord) while more exotic chord are still learned by the model (minor seventh).

Harmonic RBM

A first hypothesis was that the lateral weight from a visible unit i to another visible unit i' only depends on the difference between the indexes i and i', which means that the connection depends solely on the interval between two notes. For instance, we hypothesized (and will confirm), that lateral connections between units separated by a 5^{th} (interval of 7) will be stronger than lateral connections for an interval-lag of 1 (a semi-tone). A consequence is that negative and positive interval share the same weights.

$$\Delta L_\tau = \sum_i < v_i v_{i+\tau} >_{data} - < v_i v_{i+\tau} >_{model} \tag{4.1}$$

$$= \sum_i < v_i v_{i+\tau} >_{data} - < v_i v_{i+\tau-\tau} >_{model} \tag{4.2}$$

$$= \Delta L_{-\tau} \tag{4.3}$$

A 3^{rd} below has the same influence on a note that a 3^{rd} above. In this model, the lateral connections L are then represented by a vector. For a given note n, $L(i)$ represents the lateral weight from the note at pitch $n+i$ and $n-i$. The size of the vector determines the maximum range of the lateral connections. For instance, a vector L of size 12 is limited to a range of plus or minus an octave.

The energy function of this model is then given by

$$E(\mathbf{v},\mathbf{h}) = -\sum_{i\in vis} a_i v_i - \sum_{j\in hid} b_j h_j - \sum_{ij} W_{ij} v_i h_j - \sum_{\tau=1}^{\tau_{max}} \left[\sum_{\{i,i',|i-i'|=\tau\}} v_i v_{i'} L_\tau \right] \tag{4.4}$$

The update rule for the lateral connections when training the network with CD is

$$\Delta L_\tau = \frac{1}{card\,\{i,i',|i-i'|=\tau\}} \cdot \left[\sum_{\{i,i',|i-i'|=\tau\}} < v_i v_{i'} >_{data} - < v_i v_{i'} >_{model} \right] \tag{4.5}$$

After training this model on Bach chorales, we observed that (compared with a standard RBM) chords learned by our model were very clearly indicative of musical chords (figure 4.1 on page 39). This encouraging result drove us to build a temporal model based on this first building block.

4.2 The HCRBM

Training the model

Energy function Adding the temporal structure of a CRBM to the Harmonic RBM leads to an energy-based model described by figure 4.2 on page 40 and defined by the following function

$$E(\mathbf{v},\mathbf{h}) = -\sum_{i\in vis} \hat{a}_i v_i - \sum_{j\in hid} \hat{b}_j h_j - \sum_{ij} W_{ij} v_i h_j - \sum_{\tau=1}^{\tau_{max}} \left[\sum_{\{i,i',|i-i'|=\tau\}} v_i v_{i'} L_\tau \right] \tag{4.6}$$

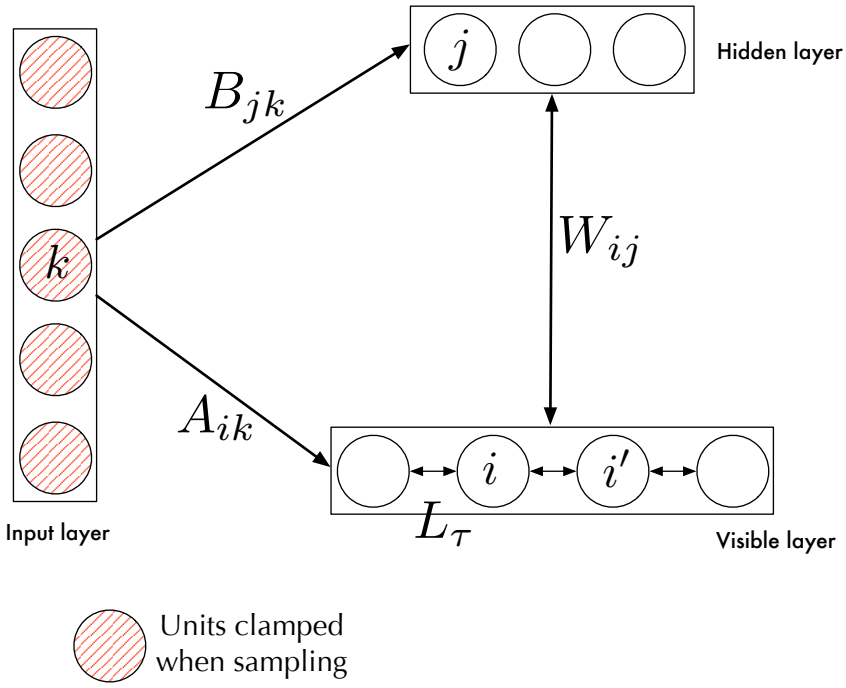


Figure 4.2: The HCRBM model. Lateral structured connections are added to the standard CRBM model. Units crossed in red are clamped to the value of past visible units during the sampling.

where $\hat{a}_i = a_i + \sum_k A_{ki} v_{k,<t}$ and $\hat{b}_j = b_j + \sum_k B_{kj} v_{k,<t}$ are the dynamic biases defined as previously.

Update rules All the weights are trained by contrastive divergence and follows exactly the same update rules than for previous model. They are reminded here for convenience :

Equation 10 (Update rules for the parameter of a Harmonic Contrastive RBM (HCRBM))

$$\Delta b_i^{(v)} = < v_i >_{data} - < v_i >_{model} \tag{4.7}$$
$$\Delta b_j^{(h)} = < h_j >_{data} - < h_j >_{model} \tag{4.8}$$
$$\Delta W_{ij} = < v_i h_j >_{data} - < v_i h_j >_{model} \tag{4.9}$$
$$\Delta A_{ik} = < v_i x_k >_{data} - < v_i x_k >_{model} \tag{4.10}$$
$$\Delta B_{jk} = < h_j x_k >_{data} - < h_j x_k >_{model} \tag{4.11}$$
$$\Delta L_\tau = \frac{1}{card\,\{i, i', |i - i'| = \tau\}} \cdot \left[\sum_{\{i,i',|i-i'|=\tau\}} < v_i v_{i'} >_{data} - < v_i v_{i'} >_{model} \right] \tag{4.12}$$

Collecting statistics One can notice that inferring the hidden units given the visible and context units is not changed in this model, as compared to a standard CRBM. Computing the visible units distribution given the hidden units remains the major difficulty of this model. As for the SRBM, block Gibbs sampling is not possible, and a long unit by unit update step has to be ran. Indeed, for each

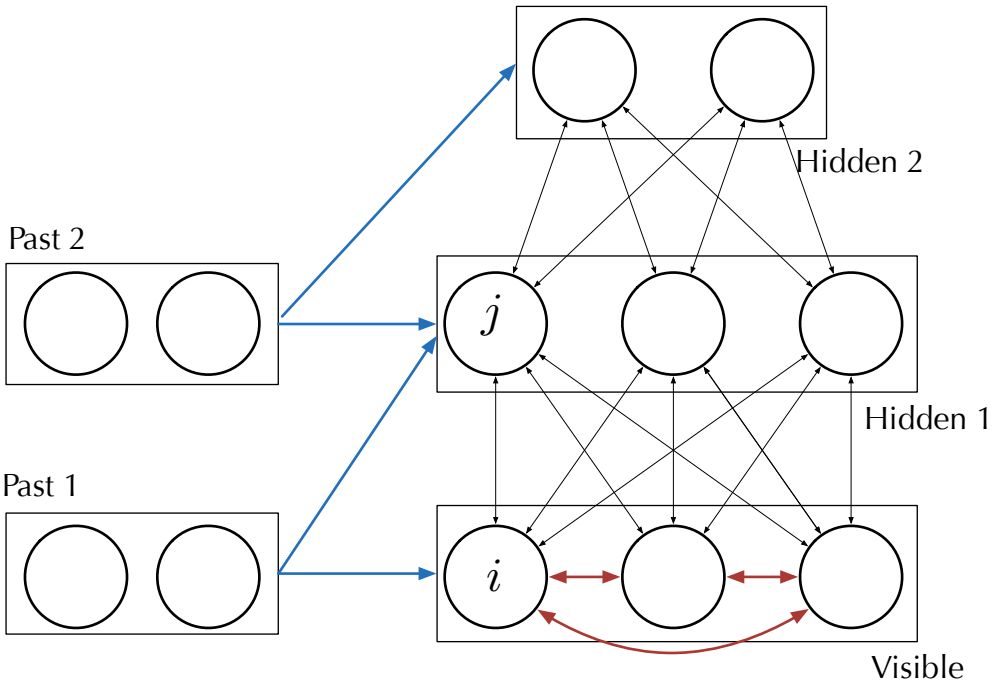


Figure 4.3: **Stacked HCRBM.** Harmonic connections are in red, and only present for the visible (first) layer. Temporal connections are in blue.

visible unit, we have easily access only to its probability distribution conditionally on the hidden units and, this is the problem, the other visible units (see section 2.3 for more details).

Stacking HCRBM The HCRBM can be stacked in a deep architecture. Indeed, it is possible to use the same model with lateral connections as a second layer. However the hidden layer *a priori* don't have a lateral structure as strong as the visible layer have. Beside, as it is explained in the next sub-section, sampling from a HCRBM is laborious. So we chose to stack over a first HCRBM a standard CRBM structure figure 4.3 on page 41 rather than stacking several HCRBM on top of each others.

Sampling from a HCRBM As for the training phase, sampling from a HCRBM is a long process since block sampling is not possible. However, we proposed in our model to stack a CRBM on top of the first layer HCRBM. Then alternate Gibbs sampling is performed in the top associative memory formed by this CRBM, where fast block sampling is possible, and a single top-down pass is performed through the HCRBM. Hence, sampling from a stacked architecture is faster in our model.

4.3 Pre-processing and regularization

Efficient modeling always include a crucial pre-processing step (whitening, PCA). This section introduces the pre-processing we thought relevant for music modeling. Rhythmic pre-processing is mainly linked to the question of quantization (number of frame per quarter note). However, several pitch pre-processing have been presented in the introduction (section 3.2). We decided to test two pre-processing :

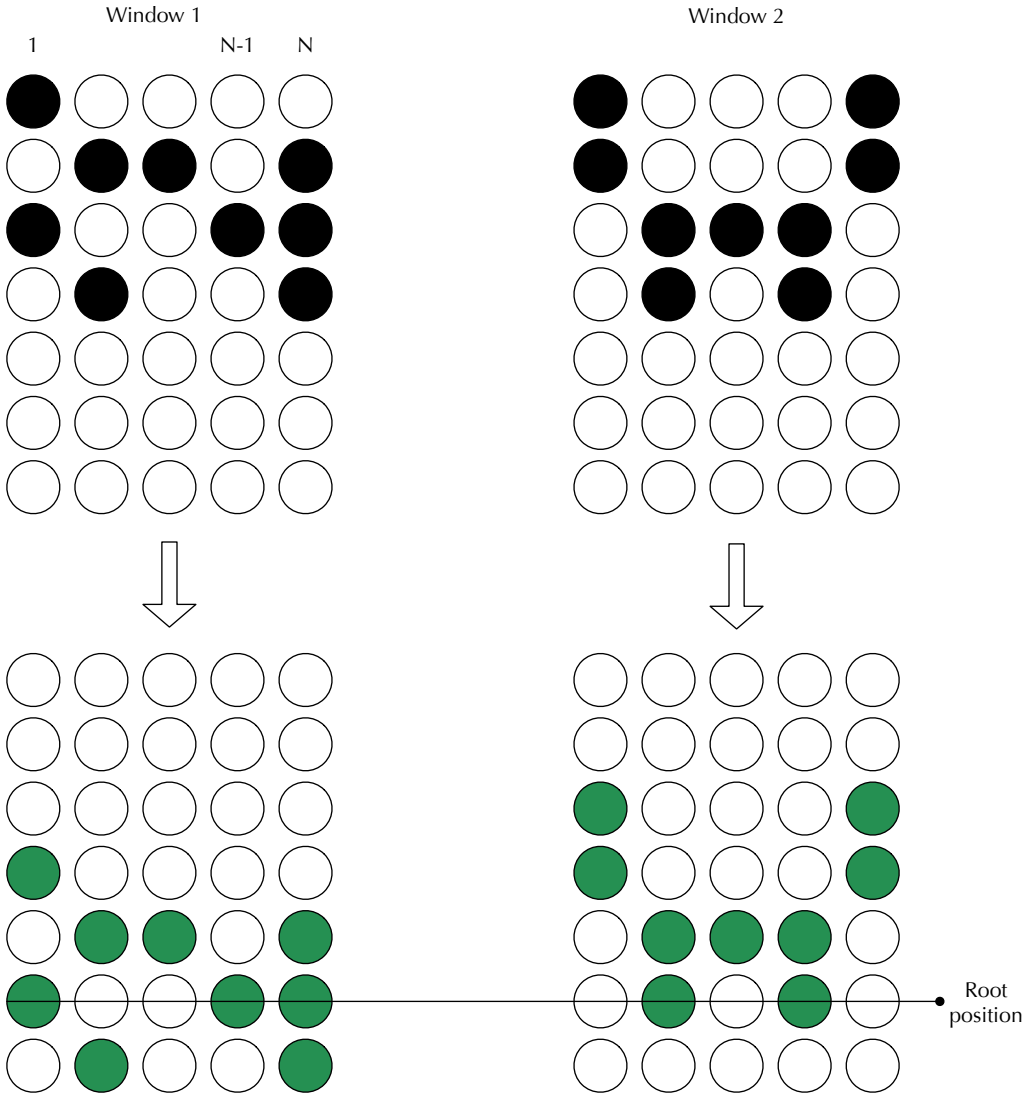


Figure 4.4: The root of the penultimate frame is used as reference. In black the original score, in green the root-centric representation.

- Pitch-class transformation: The number of pitch is reduced to twelve by computing the modulo 12 for each note. This an octave equivalent transformation. Note that if the same pitch-class is represented at several octaves, it is counted only once in the transformed version.
- Root-centric transformation: based on the representation in [AW05]. For each window of N time frames, where the N^{th} represent the current time t, all notes of this window are centered around the root of the $N-1$ frame (see figure figure 4.4 on page 42). This is supposed to emulate a kind of translation invariance property.

Chapter 5

Results

One paramount issue with unsupervised generative models lays in how to perform their evaluation. While a qualitative evaluation appears natural, the subjectivity of such an analysis is ill-suited for discriminating between different models. Quantitative evaluations of a generative model often rely on a concomitant task that naturally embeds an objective measure of accuracy. We decided to base our evaluation on a symbolic polyphonic melody prediction task.

This remainder of this chapter is organized as follows. The first section compare the results of our method and several models for the prediction task. The second section introduces a method for finding the best hyper-parameters when training our model and discusses the influence of various parameters over the results. The last section is dedicated to try to understand what the model has truly learnt. Hence, we perform a qualitative evaluation of our model by listening to generated examples, observing the learnt weights and discussing advantages and limitations through a critical judgement of those results.

5.1 Music prediction

Evaluation

For a generative model, estimating the log-likelihood of a test set under the trained model distribution might be the most reliable method for objective evaluation [BLBV12]. Indeed, this demonstrates the actual ability of the network to produce sequences that closely resembles those that we wanted to learn (and, therefore, if the model has grasped the underlying structure of the data). Expected log-likelihood [AW05, BLBV12] is given by $p(v_t|\boldsymbol{\theta}, v_{<t})$ where v_t is the current sample of the test sequence and $v_{<t}$ contains the past context of this sequence. Hence, this measure is defined as

$$\mathcal{D} = \mathbb{E}_{test}\left[-log(p(v_t|\boldsymbol{\theta}, v_{<t}))\right] \tag{5.1}$$

One of the major issue with models based on RBM is that computing the likelihood of a training set given a model require the computation of the partition function which is often intractable. Annealed Importance Sampling (AIS) [SM08] allows an estimation of this partition function, but is unsuited for conditional models. Indeed, the partition function of a RBM depend only on the weights of the model, while the partition function in a conditional model also depends on the context units ($Z(v_{<y})$). Hence, the partition function changes for every new sample and the generalization of AIS to conditional models is not straightforward.

Another way to evaluate the performances of a generative model is by solving a correlated problem (requiring to understand the same underlying structure) that naturally defines an objective error function. The most frequently used tasks for music generative models are melodic prediction [LR14, BLBV12, LP03] or polyphonic transcription [BLBV12]. In polyphonic transcription task, the model is used as a probabilistic prior to improve the performance of a purely signal-based multi-pitch estimator. On the other hand, the prediction task consists in trying to infer the next point in a sequence of polyphonic symbolic music.

The polyphonic transcription framework implies to develop a relatively accurate multi-pitch estimator and tackle a set of complex signal-based problems. For those reasons, the prediction task has been preferred in this work. Given a certain temporal granularity, the prediction task consists in finding the next frames in the chords representation given the past $N-1^{th}$ frames. Short-term prediction (predicting 1 frame) is usually preferred to long-term prediction (2 and more frames) and our model have been evaluated on short-term prediction tasks in order to compare equally to the state-of-art baselines.

Datasets

We used the four midi datasets introduced in [BLBV12] :

- *JSB Chorales*: 382 four-part harmonized chorales by J.S. Bach
- *Piano-midi.de*: 124 classical piano scores
- *Notthingham*: 1037 folk tunes scores
- *MuseData*: 883 orchestral and piano scores

Those datasets are available on this website. Note that all sets are split between training, validating and testing datasets. Since we did not use a validation step, we append this set to the training set. We reduced the number of pitches to 88 (between midi note 21 and 108) to avoid unused visible unit while having a sufficient *ambitus*.

Models

Except for repeat and random models that are purely indicative, we trained and tested five models plus our proposed model

- *Random.* The predicted frame is randomly drawn from a uniform distribution in $[0, 1]$
- *Repeat.* The predicted frame is simply the repetition of the previous frame. This is the most basic 1-order predictive model.
- *N-order RBM.* This defines a concatenation of N RBM, ie. $88 * N$ visible units. Training is made with windows of length N, and for the prediction task, the N^{th} frame is generated by sampling in the RBM while clamping the $N-1^{th}$ first units.
- *CRBM.* The input units represent the context ($N-1$ frames) and the visible units are the current frame. Sampling has been described in section 3.3.
- *HCRBM.* Our model, detailed in chapter 4.

For statistical models, the test sample is obtained by inferring the visible units given the context, as previously detailed. For each model, several architectures have been developed (i.e. number of layers, number of units in those layers), with several hyper-parameters configurations, whose influence is fully analyzed in (section 5.2).

Measure

Frame-level accuracy For evaluating accuracy, previous studies relied on the expected frame-level accuracy [BLBV12, LR14], defined by the formula

$$\text{Accuracy} = \frac{TP(t)}{TP(t) + FP(t) + FN(t)} \tag{5.2}$$

where $TP(t)$ is the number of notes correctly predicted (true positives). $FP(t)$ is the number of notes predicted which are not in the original sequence (false positive) and $FN(t)$ is the number on unreported notes (false negative).

Instead of binary values, activation probabilities are used for the predicted samples in order to reduce the sampling noise. If this probability is intractable, one should sample many predicted frames for a single time and compute the mean value of those samples.

Even though the frame-level accuracy has been repeatedly used as a baseline evaluation, we discovered a previously non-described effect to its use while performing our experiments. Hence, as the size of frames decrease, more and more are just repetitions between successive time step. We assessed this with a simple repeat model, which with a quantization of 32 (i.e. 8 frames in a quarter note), obtained an accuracy score of 95.9% , which is ridiculously high for an extremely limited model. Hence we decided to propose a novel evaluation measure based on an event-level testing.

Event-level accuracy We propose to rely on a more robust *event-level accuracy*, which measures the similarity between a true and a predicted frame each time a new event occurs (either a note on or off), instead of every new time frame. However, the prediction remains based on the $t - N$ last frames, not events. Training a model for the event-level prediction task is slightly different, since only the frames where an new event occurs are kept as training examples.

We evaluated our models on both frame-level and event-level accuracy, and discuss the pros and cons of each measure.

Results

Frame-level accuracy The predictive task defined in the state of the art is evaluated on a frame level. Hence, we compared our model with the baseline on this particular task. The results for the different models are summarized in table (figure 5.1 on page 46). For each model, we picked the best results from a grid-search performed over the architecture and the hyper-parameters. The two models that constituted our baseline have been also added, while we don't know if it was a frame-level accuracy or an event-level accuracy. [BLBV12] also gave the results for the models in [LP03] and [AW05].

We use the same quantization as the one used by [BLBV12] which is 1 frame per quarter note. Models has been trained on the *Compute Canada/Calcul Québec* cluster which offered large computing resources.

As aforementioned, frame-level accuracy can be doubted and strongly depends on the quantization. When the quantization increases, more and more frames are just repeated between two successive time step. If we use a quantization of 32 (i.e. 8 frames in a quarter note), the repeat model obtains an

Model	Frame Acc %			
	Piano-midi	Nottingham	MuseData	JSB Chorales
Random	3.25%	3.54%	2.48%	4.22%
HMM [AW05]	-	-	-	16.32%
Repeat	26.59%	55.62%	21.07%	39.60%
Random fields [LP03]	18.37%	55.34%	18.39%	22.93%
LSTM with Rprop [LR14]	-	-	-	31.91%
RNN-RBM [BLBV12]	28.92	75.40%	34.02%	33.12%
Temporal RBM	14.74%	46.29%	10.85%	30.52%
CRBM	19.14%	63.35%	18.54	31.18%
HCRBM	-	-	-	-

Figure 5.1: Frame-level expected accuracy for various musical models. The quantization is one frame per quarter note.

Model	Frame Acc % JSB Chorales
Random	4.23%
Repeat	18.6%
GCRBM	8.58%
Temporal RBM	27.77%
CRBM	26.15%
HCRBM	**28.07**%

Figure 5.2: Event-level expected accuracy for various musical models

accuracy score of 95.9% , which is ridiculously high for a very limited model. Worst, if we increase the quantization, event a model such as the RNN-RBM will learn to simply repeat the previous frame. Hence we decided to propose an alternative event-level evaluation.

Event-level accuracy The results for the event-level accuracy are displayed in figure 5.2 on page 46), with a (context) quantization of 4 frames per quarter notes (16^{th} notes). On this new task, despite the lack of comparison elements, the three models temporal-RBM, CRBM and HCRBM strongly outperformed the simple repetition model. The factored-gated model performed the worse and didn't managed to provide satisfactory results. This might come from the fact that the structure model (making the assumptions of three-way interactions) is not appropriate for this task.

Discussion : which pre-processing should we use ? The two pre-processing we have presented have not improved the results. The pitch-class evaluation is different since the prediction task is only performed for vectors of length 12. This vector can be extended in several way to an 88 vector, but it is either inefficient (defining C4 to B4 at the value of the pitch -class) or ambiguous (continuous principle, but what if two notes are equidistant from an other ?) and eventually leads to poor results. Octave is important when performing a prediction and we loose too much information with the pitch-class pre-processing. The root-centric pre-processing did not modify the performances and can be considered useless since it did not reduce the computation load neither. We believe that another pre-processing which consists in transposing all the pieces to the same tonality (C major/minor) would be the best

pre-processing [BLBV12]. However, it is a complex pre-processing which imply to be able to find the tonality (possibly several) of the piece. This should be addressed in a future work.

Discussion : which model is the best ? On the frame-level accuracy, the CRBM and temporal RBM models obtained good results, but not as good as the RNN-RBM model.

For the event-level accuracy, the HCRBM results were slightly better than the other models. This is an enthusiastic result since the lateral connections we added to the standard CRBM model seem to be an improvement.

However the differences between it and the two other temporal models (temporal RBM and the CRBM) is small enough to doubt that the model is significantly better. We lacked of time to test the model on the three other databases, but this has to be done in a future work. The doubt we have is reinforced by the fact that we expected the CRBM then HCRBM would be successively better, which is not the case.

Eventually, we will see in a next section that the CRBM actually performed better than the HCRBM model when generating long sequences (section 5.3).

Discussion : which evaluation is the best ? Both method have their drawbacks. Frame-level is quantization sensitive, which is a negative point. The *best* quantization will depend on the database. Hence this model is not generalizable to any data set. Besides, an objective for a generative model would be to be able to produce any kind of music, which would require to train the model on a very large and diverse database. In this case, which quantization choose for the whole set ?

On an other side, a model trained at an event level will be very limited when generating sequences. Indeed, since its training examples were all transitions, the model try at each frame to create a new event. Hence, the generated sequences are then very monotones (see section section 5.3).

Anyway, the good performances of a model on the predictive task does not guarantee its ability to generate interesting musical sequences. Indeed, forecasting and modelling are two different tasks. Modelling requires to really capture the long-time dependencies. We will see in the qualitative evaluation section that despite the CRBM was not performing as well as the temporal RBM on the predictive task, it is a better generative model.

However, we believe that for a given model, the tuning of the hyper-parameters that gives the best predictive score is most likely to generate interesting sequences. The next section is dedicated to this hyper-parameter analysis.

5.2 Analysis of the hyper-parameters

As the different hyper-parameters involved in the learning process can have a varying influence over the quality of the distribution learnt, being able to find the best set of hyper-parameters can condition the quality of the results for a model. After a first grid-search for the different models, we decided to run an advanced hyper-parameter analysis on the CRBM, since it was the most interesting generative model (see section 5.3).

List

We analyzed the influence of the set of hyper-parameters and range of values listed in figure 5.3 on page 48 (the role of the different hyper-parameters can be found in the description of the RBM algorithm

Architecture	Number of layer	$[1, 4]$
	Units per layer	$\{50, 64, 100, 200, 300, 400, 512, 600, 700$ $800, 900, 1024, 1100, 1200, 1300, 1400, 1500\}$
	Temporal order	$\{4, 8, 12, 16, 24, 32\}$
Training parameter	Learning rate	$[1e-4, 5e-2]$
	Momentum	$[0, 0.99]$
Regularization	Weight decay	$[0.000001, 0.01]$
	Sparsity target	$[0.001, 0.5]$
	Sparsity beta	$[0, 0.99]$
	Sparsity lambda	$[0, 0.99]$

Figure 5.3: List of the tested hyper-parameter and their range of value

insection A.3). For continuous intervals, the tested values are randomly chosen inside the corresponding range.

Method

The number of potential combinations to be evaluated for the hyper-parameters is huge. For instance, even if we discretized the continuous intervals to only 5 possible values per parameter, exhaustively browsing the whole hyper-parameter space would require to train more than 6 million architectures. Since the training time of a network is approximately an hour for 100 epochs of a hundred mini-batches, this solution is clearly infeasible.

In [BB12] it is suggested that a random search is more efficient than a grid-search for high-dimensional hyper-parameter spaces. Hence, a first step consists in evaluating on the predictive task a finite set (the number defined by the available parallelism) of randomly chosen hyper-parameters points in the 9-dimensional hyper-parameter space. Each architecture associated to a point is trained over 100 epochs. Then, a Gaussian kernel is fitted on the accuracy distribution of those randomly chosen points. By picking the maximum of this function, we should be able to find an hyper-parameter point for which the results are closer to the best possible for a given model.

Results

The search for hyper-parameters has been performed on the *Compute Canada/Calcul Québec* cluster. The complete search is not finished yet. However, a large number of architectures have already been tested, and the results can be observed in appendix (see section A.4).

Analysis

The search for the optimal hyper-parameters set several expected beahviour. The results improve with the temporal order, and with the sparsity target until the value 25 before starting decreasing. Since we use a fixed number of epochs (100) ,it is normal that the performances decreased after with a small learning rate, but does not mean that a lower learning rate is worst. It simply means that more training epoxhs would be necessary.

A more surprising result is that the result were better with a low weight decay term. This will actually be explained in detail in the next section (section 5.3). Briefly, the reason is that the weight decay forces the weight to have a Gaussian distribution centred around zero. However, to model the

strong sparsity of the data, most of the weights need to be negative, which is in contradiction with a strong weight decay coefficient.

5.3 Understanding what the model has learnt

Generated sequences

The core purpose of our model was to generate (potentially infinite) sequences of music. However, conditional models need a context whose size depend on the order of the model. Hence, all the generated sequences are initialized with the first two bars of an already existing music file from the test set of the JSB Chorales database. Therefore, the first two bars of each generated midi file have been written by Bach himself, the rest has been generated by the network.

We generated several sequences from the different models, either trained on a frame level or an event level (available at this Dropbox link). At the root of this repository, two folder contains event or frame level models. Generated sequences for each model can be found in the CRBM, HCRBM, temporal RBM and FGRBM folders.

In our opinion, the best generated samples are from the CRBM model trained on an event level, and can be found at Event_level/CRBM.

Importance of the dynamic bias over visible units Temporal RBM and FGRBM generate somehow "unstable" sequences for which the number of notes explodes over time. If we attentively compare those two models with the CRBM, it seems clear that the only real difference between them is the absence of control from the input (the past) over the bias of the visible (present) units in the first two models. Hence, a direct influence of the context over the biases of the visible unit (matrix A in conditional models, present in RNN-RBM as W'') seems to be necessary for conditional generative models.

Consistency of the prediction task The examples are sorted by models, and inside each model by their score on the prediction task. A positive result is that for a given model the best architectures for prediction are also the one providing the most interesting sequences. Hence, this shows that our assumption on the prediction task providing a certain consistency with our generative objective, by understanding the underlying structure of the musical data seems to prove correct.

CRBM : event or frame level ? The CRBM model undoubtedly outputs the most satisfactory results. We have presented both the result for a model trained at a frame level and at an event level, since they gave very different results. In both cases the model failed at learning a consistent temporal structure over long horizons.

For the frame-level learning, if a short quantization is chosen, notes are over-sustained. This can be observed on the transition matrix A. This matrix is defined for n in $[|1 : N|]$ where N is the temporal order of the model. For each n, the larger $A(i, j, n)$ is, the more likely a transition from the note j at time $t - n$ to the note i at time t is. Furthermore, the diagonal of the frame-level transition matrix A (which represent a note sustained from a frame to another) is considerably stronger than the other coefficients (which represent a note changing between two frames). Hence, when generating, a note on at a given moment tends to stay on until the end of the piece. Oppositely, for a model trained on an event-level, this effect is bypassed, but a note can start or end at each new frame, producing very

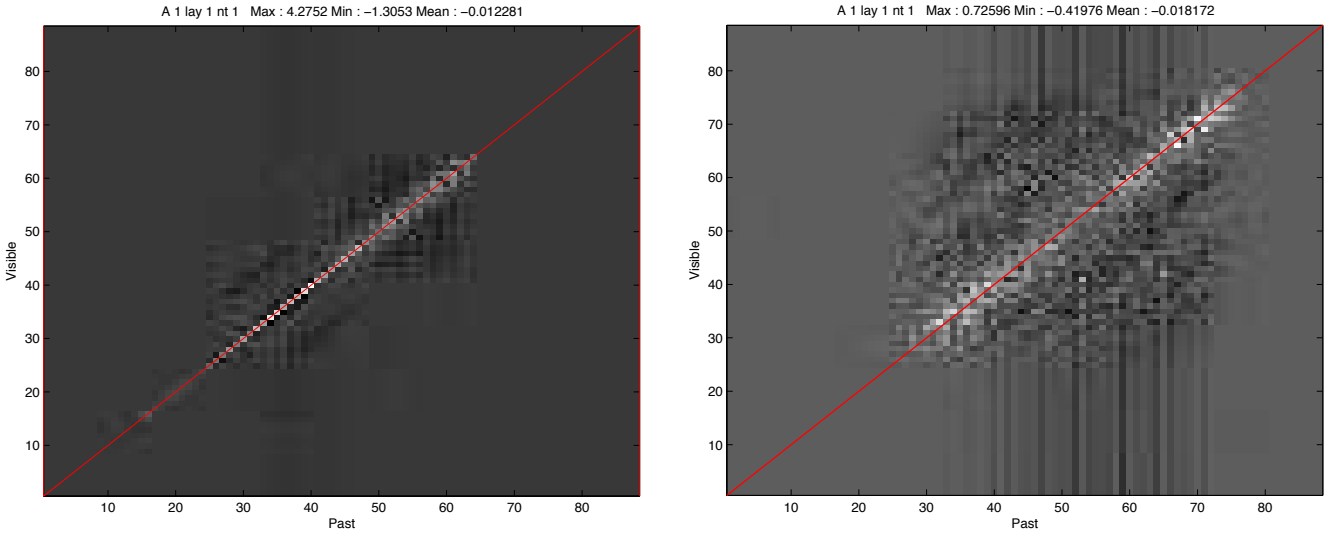


Figure 5.4: Two transition matrices A from frame $t-1$. The diagonal is in red. On the left the model has been trained on a frame level, on the right on an event level. We can observe the sustain problem appearing as the strong diagonal of the left matrix. Since the lower and higher notes never appeared in the training sequences, the transition toward them are close to -1 (complete inhibition).

"agitated" sequence. The transition matrix A of two CRBM, one trained at an event level the other at a frame levels are presented in figure 5.4 on page 50.

Number of Gibbs sampling steps Whereas using a single Gibbs sampling step when training a model with contrastive divergence is sufficient to approximately find the steepest direction of the gradient, this is clearly insufficient to reach the equilibrium distribution of the model. Hence, a larger number of alternate Gibbs sampling step is required when sampling from the model (section 2.3). Samples drawn from the model with different number of Gibbs sampling steps are presented in the results at Event_level/CRBM.

Initialization of the sampled frame As discussed before (section 3.3), the current sampling frame need to be initialized before starting Gibbs sampling. It can be set at the value of the previous frame or randomly initialized. This initialization slightly influence the result and, surprisingly, we found that the random initialization provided better results (can be observed at at Event_level/CRBM/1/Init_previous).

Harmonic modeling: HRBM

As explained in the previous chapter (section 4.1), We have first tackled the harmonic problem by testing different models with no temporal connections. Adding the lateral connections greatly improved the quality of the generated chords. After training, the lateral connections in a Harmonic RBM matched our expectations since the consonant relations (minor or major third, fifth, octave...) were favored against dissonances (second, augmented fourth). These lateral connections can be observed on figure 5.5 on page 51. A midi example for this model can be found at this (link HRBM).

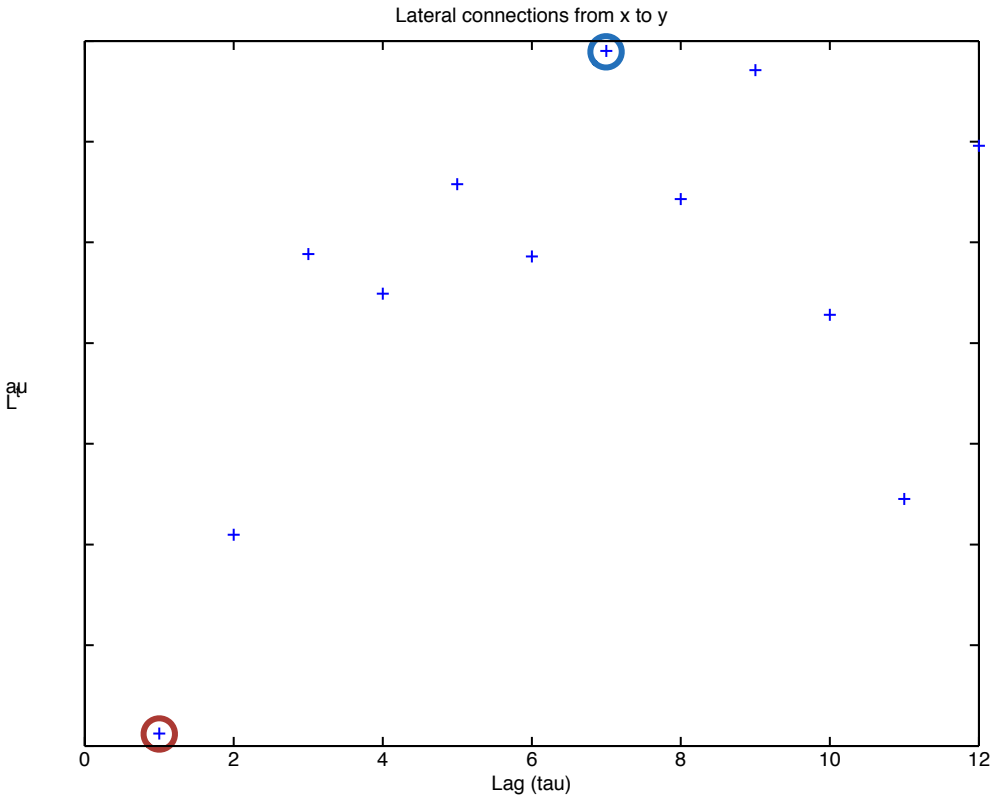


Figure 5.5: Lateral connections in a HRBM. The value for each time lag represents the influence of a visible unit over other visible units at interval τ. It can be noticed that this is mainly an inhibitive influence (weights are negatives). In blue a fifth (which is also a negative fourth) and in red second minor

Time modeling: CRBM

The temporal structure learnt by a CRBM can be observed through the matrices A and B. Since A directly explains the influence from the input units (past) on the visible units (present), observing its content gives many information on the knowledge acquired by the network. Matrices A are 3-dimensional: $Visible \times Input \times Time$. Clamping one dimension to a certain value and observing the resulting 2-dimensional matrix gives information about the influence of the two other variable over the third one.

Except when mentioned, the generated sequences drawn from the CRBM observed here are from the models which provided the best performance on the predictive test.

A_t: Observing the matrix A for a given time t gives information about the transition from the note j at time $t - n$ to the note i at time t (corresponding to the discussion between event and frame level in (figure 5.4 on page 50). The same matrix A for a time-lag of 1 and for a time-lag of 16 (the maximum for this architecture) are given in figure 5.6 on page 52. We can see that for the recent past (time-lag of 1), weights are particularly high on the diagonal and around, especially for low and high pitches. This suggest that transition from a pitch to another mainly occurs for small intervals, which is in accordance with what could be our intuitive construction of a melodic line. For larger time lags, the weights become

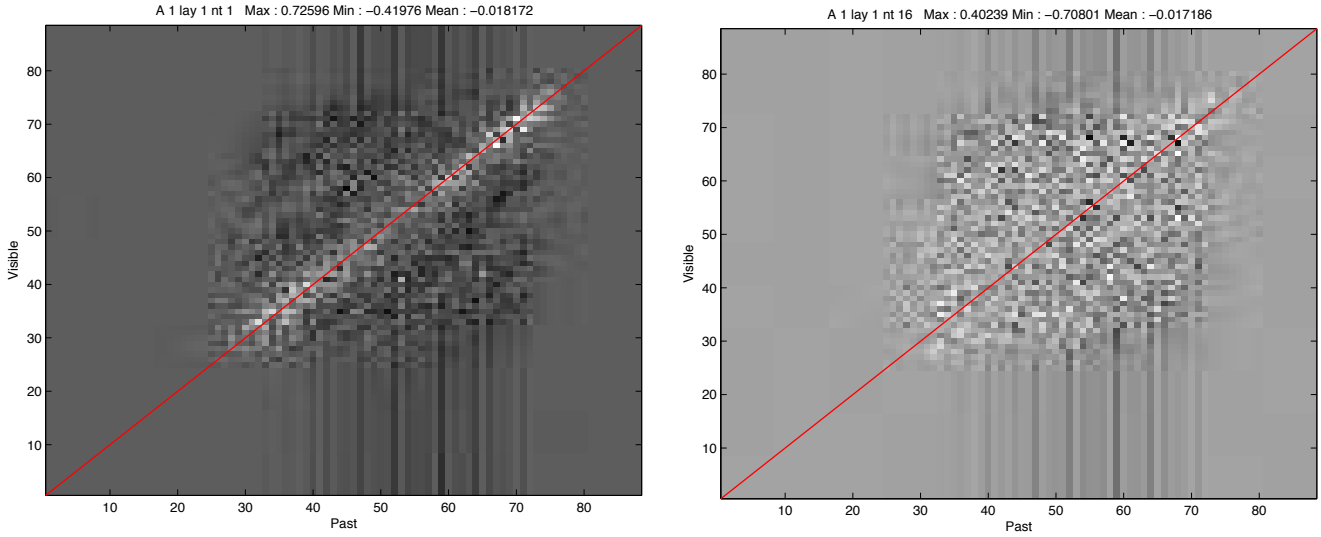


Figure 5.6: **Matrices A_1 and A_{16}** represent the influence of the frame $t-1$ and $t-16$ on the current frame. Diagonal is represented by a thin red line.

more diffuse since larger gaps become more probable. The difference between the range of values (min and max in the title of the figures) for the two time lag is also noteworthy. The long time-lag matrix $A(16)$ exhibits an inhibitive influence (negative weights) that can be understood as a "note stop" signal whereas the short time-lag matrix has an excitatory influence (positive weights, note on signal).

$\boldsymbol{A_i}$ **:** Looking at the transition matrix for a given visible unit can be understood as a *voice-leading* matrix (see figure 5.7 on page 53). A_i is a matrix whose column are time-frames and rows are pitches. It embeds how the network learned *melodic pathes* that are more likely to end on a certain pitch i.

Temporal and harmonic connections: HCRBM

The core idea behind our proposal of the HCRBM model was to impose both a strong temporal and harmonic structure. Unfortunately, the generated sequences did not showed this expected stronger understanding (link HCRBM)

A first observation is that the harmonic structure is insufficient. All the weights are negatives, which means that all the notes have an inhibitory influence over each others. Besides, the weights that have the higher value (here the closest to zero), i.e. the most stimulated intervals, are the most unexpected (augmented fourth, major second...) (see appendix figure A.8 on page 73).

We believe that this can be explained by the fact that the temporal and lateral connections have been in constant and unresolved conflict during the learning phase. Indeed, those two interactions both modify the biases of the visible units. Hence, both are using the same mechanism to model different causes. The fact that our learning data often embeds frame repeated from one to another, combined to the sparsity of the input data leads to a kind of explaining away problem between the lateral and temporal weights (see figure 5.8 on page 53). Indeed, since $v(t-1) \sim v(t)$ for most units, the temporal connection A tends to absorb the effect of the lateral connections L. This effect during the generation process can be observed on the figure in appendix (figure A.11 on page 77).

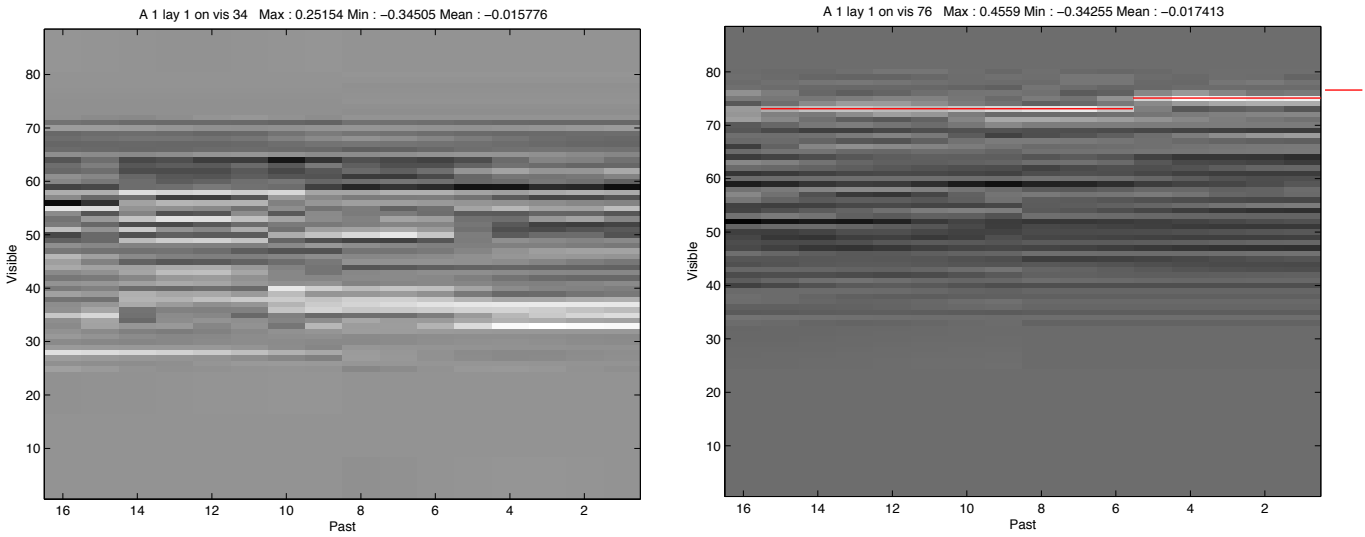


Figure 5.7: Voice leading matrices for input units 34 and 76. Be careful that time indices are for negative time lag: 1 means $t-1$. Hence, the time runs from the left to the right part of the graph. Lower pitches are at the bottom and higher at the top. The highest weights often form a diffuse cloud around the next visible note. Some matrices exhibit strong temporal structure, such as the transition matrix towards pitch 76, which shows an ascending melody

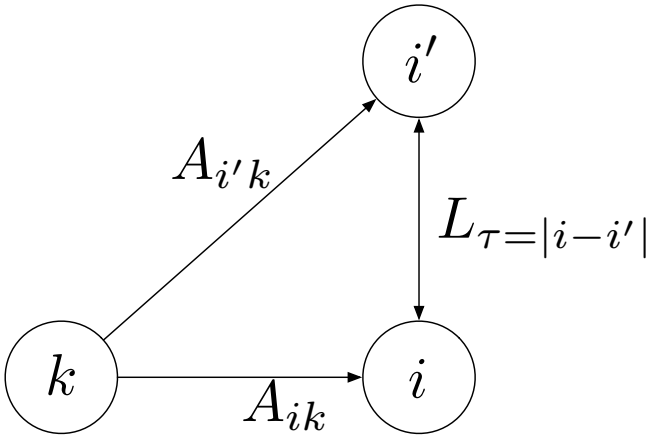


Figure 5.8: Explaining away between temporal and lateral weights. Since $\Delta L_{ii'=\tau} =< v_i v_{i'} >_{data} - < v_i v_{i'} >_{model}$, if A_{ik} and $A_{i'k}$ have already large values, $< v_i v_{i'} >_{data} << v_i v_{i'} >_{model}$ and $\Delta L_{ii'=\tau} < 0$ even if it should have been greater than 0.

Weights values

***Collapsing* sequences :** However, this does not explain the fact that the generated sequences *collapsed* after a few notes. Indeed, it has been observed that for some poorly trained CRBM and most of the HCRBM the generated sequences start with four notes chords and end after only a few notes by playing only silences. Our guess was that this*inhibitive* behaviour might be predicted by observing the mean value of input weight of visible units. The more this value is negative, the more inhibitive signals this visible units should receive. A boxplot of the mean input weights received by visible units (figure A.9 on page 74) indicates that there is no direct relation between this quantity and the behaviour of the model. Hence, we did not find any explanation or cause for this *collapsing* behaviour.

Modelling the sparsity of the data By observing the histograms of the weights of a CRBM (see appendix figure A.10 on page 76), one can notice that most of the weights are negative. Negative weights can be understood as an inhibitive influence, and the network learn those negative weights to emulate the sparse nature of the data. This is particularly obvious when looking at the visible biases.

Observing the latent units

The hidden units of a correctly trained network often represent rich, hierarchically organized along the layers and distributed representations [LEN08]. To check if latent representations learnt by the network are meaningful in the sense that they all represent different and interesting patterns, one can observe the activations of the hidden units for several visible vectors from the training dataset. This allows to see if a hidden unit is constantly on or off (which means that it is a useless unit) or if different visible vectors activate the same pattern of hidden units.

To really observe the pattern represented by the hidden units of a RBM, one possible way is to clamp the hidden unit we want to observe and perform alternate Gibbs sampling while keeping this unit clamped. This method can be used to visualize hidden units in a Temporal RBM. For context based models such as the CRBM or HCRBM, this observation will be conditioned by input units. A possible solution is to fix a context, clamp a hidden unit and observe the activated visible units. This could be seen as observing a hidden unit conditionally on a certain context (see figure 5.9 on page 55)

Observing those hidden patterns, we can see that given a certain context, the hidden units actually all play almost the same role. Indeed, the bias imposed by the context unit is so strong that the activations become almost entirely determined by the past visible units. One might be afraid that the generation process becomes entirely deterministic. This is not the case, and a sufficient amount of randomness is kept to produce various series from the same context sequence (listen to Same context).

5.4 Conclusions

In this chapter we introduced a new model, the Harmonic Conditional RBM. We evaluated its performance and the performances of several already existing models (CRBM, RBM) on a short-term predictive task thanks to the frame-level accuracy. The fact that this measure depends on the quantization is a major drawback and we proposed an other measure called event-level accuracy. We evaluated our model with this measure and used it to perform a search for the optimal set of hyper-parameters of the CRBM model. The optimal set of hyper-parameters found has been used to train an HCRBM and provided the best performances among the tested architectures. We eventually proposed a qualitative evaluation of the CRBM model by observing the weights learnt and the sequences generated after training the network on the *JSB Chorales* database.

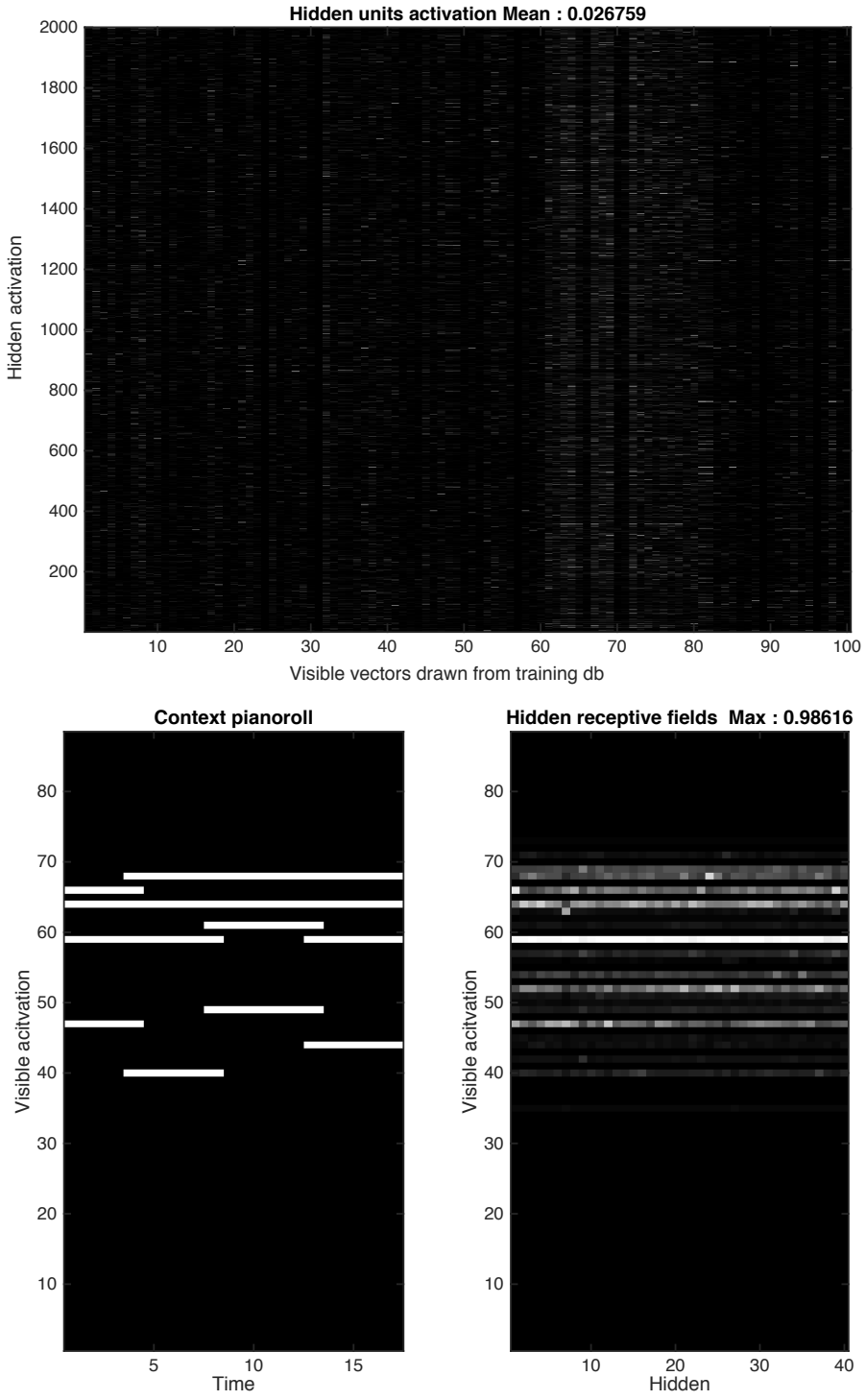


Figure 5.9: **Observing the latent units.** The figure on the left shows the activation of each hidden units (here 2000) given 100 randomly chosen vectors from the testing dataset. We can observe the effect of the sparsity constraint which impose that a hidden unit is rarely on for successive different input vectors. The figure on the right shows the context given activations of the visible units given a hidden unit. Those values are shown for the 40 maximally activated hidden units in the present context.

Even if the pre-processing we evaluated in this work have not increased the results, we still believe that this is an important step. It is more than probable that transposing all the pieces in the same tonality (for instance C major/minor) as proposed by [BLBV12] would increase the performances of our model. It has not been tested in this work since it is a complicated processing which requires to first estimate the tonality (possibly several) of each piece. This should be addressed in a future work.

We believe that log-likelihood is a more reliable evaluation framework than the predictive task when working on generative models. Investigate the neural autoregressive distribution estimator (NADE) ([LM11]) architecture, trying to include it in the CRBM model and then being able to compute the log-likelihood of a test dataset in a reasonable amount of time is an important future step.

The harmonic connections added to the CRBM model have improved the results on the predictive task, but generated sequences that appeared us less interesting that the one produced by the CRBM. The fact that no harmonic structure has been learnt can be explained by a conflict between lateral and temporal connections during the training phase. Hence, we decided to focus on the CRBM model for the qualitative evaluation and analysis of the learnt knowledge. An important distinction between frame-level and event-level training and evaluation has been made. We believe that for generative purposes, the model must be trained on a frame level to learn repeated or sustained notes. However, we obtained more interesting sequences when training models on an event-level. Hence, the qualitative evaluation of our work has been made in an event level framework.

The CRBM-based model presents the advantage to clearly showing the structures learnt by the network. However, its temporal structures seems to be more adapted to data that vary smoothly and which involve short-term dependences. The RNN-RBM embed the temporal structure in the hidden state transition long-time dependences are immediately embedded in the first layer. In the CRBM model, we hopped that those long-term dependences would be embedded when stacking a second layer on top of the first layer, but failed at training this second layer.

Future works

We think that the CRBM model should be investigated further by understanding why multi-layer architectures failed and solve this problem. The special nature of the data, sparse in high-dimension and binaries, have not been fully understood and integrated in our model, and this is probably in this direction that major improvement can be made. We should run an hyper-parameter search on the HCRBM that should increase the performance of this model on the predictive task. Because training a HCRBM is long (block sampling is impossible), we did not have the time to run it yet. We think that the conditional architecture should be investigated further, perhaps by considering an other data representation that would be more adapted to the special nature of the data. Indeed, the strong sparsity in the pianoroll representation impose constraints on the weights during the training phase that are perhaps not adapted to a conditional model.

Part III

Automatic orchestration

Chapter 6

Adaptation to the orchestration problem

This short part introduces an adaptation of the CRBM and the Style-gated FCRBM models to the automatic orchestration problem. The idea is to feed the network with the parallel information of an orchestral piece and its reduction for piano. We shortly describe the model in a first section, then present a quantitative evaluation framework for the automatic orchestration task called predictive orchestration and the result of our model in this framework. Despite the poor results we have obtained for the moment, we present some generated orchestration and how we plan to improve those first results.

6.1 An automatic orchestration model

In this section we introduces two automatic projective orchestration systems based on conditional models. There is a strong correlation between an orchestration and its reduction for piano. Harmony, rhythm and melodies are basically the same, and the orchestration is simply a projection of those structure on different instruments. Despite the apparent complexity of this problem, the state of possible combinations for an orchestration is drastically reduced when defined conditionally on a piano reduction. Hence the two models we propose here learn on a parallel database of orchestral music and its reduction for piano. There are merely an adaptation of two conditional models that have been previously introduced : the CRBM and the FGCRBM (see section 3.3).

We first thought of using lateral connections between instruments to model their co-occurrences (for instance the fact that woods often play together). As we have seen in the previous chapter that using both lateral and temporal connections could lead to a conflict when training those two different kind of weights (figure 5.8 on page 53), we have not pushed this idea any further, and instead focused on gated connections that appeared more adapted to our problem [TH09].

The visible units represent the note played by the orchestra at a given moment. Each instrument score can be represented by a pianoroll. The visible units are then simply a concatenation of the vectors formed by those pianorolls at a given moment. Note that in order to avoid useless visible units, the number of note for each instrument has been reduced to the range covered by each instrument in the database. For each time frame, the context is defined by :

- the current piano frame which defines the melodic, rhythmic and harmonic structure
- the N last orchestrated frames to ensure a continuity in the orchestration. N thus defines the order of the model.

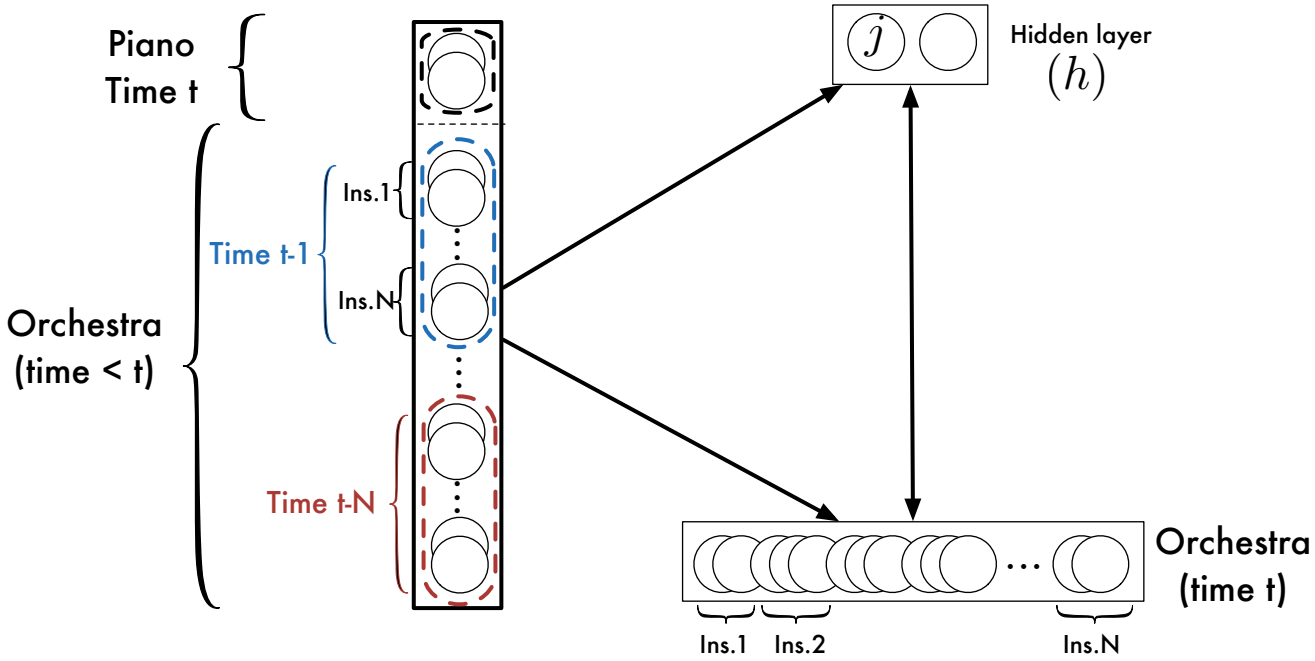


Figure 6.1: The CRBM model applied to orchestration. Context units are the concatenation of the recent past for the orchestra part and the current frame of the piano part.

Those two models are represented in figure 6.1 on page 59 and figure 6.2 on page 60. The more complex FGCRBM model has been motivated by the fact that the gated connections (from the current piano frame) are able to modify the energy landscape. The CRBM model defined by the current orchestral frame and its recent past is modulated by the piano units.

Toward an orchestral piano

The original objective was to be able to generate in real time an orchestration from a midi piano input. Before even considering the implementation of the algorithm, the real time framework imposes strong constraints over the architecture of the network. Once trained, sampling a FGCRBM is very fast since since block Gibbs sampling is possible. However, several step of Gibbs sampling are required to obtain a sample representative of the model distribution, and here a trade off must be found between speed and quality.

6.2 The orchestration prediction task

A first definition As the automatic generation problem, automatic orchestration suffers from the lack of quantitative evaluation. The different work on the domain mainly rely on qualitative evaluation [HSD12]. To our best knowledge, there has not been any attempt in the automatic orchestration field to define a task associated to performance measure. We propose here a first attempt for fill this gap by defining the orchestration prediction task.

An orchestral sequence is defined by a sequence of matrices indexed by the time t $M(t)$. The dimension of a matrix $M(t)$ is the number of instrument per the number of pitch : $N_{instrument} \times 88$.

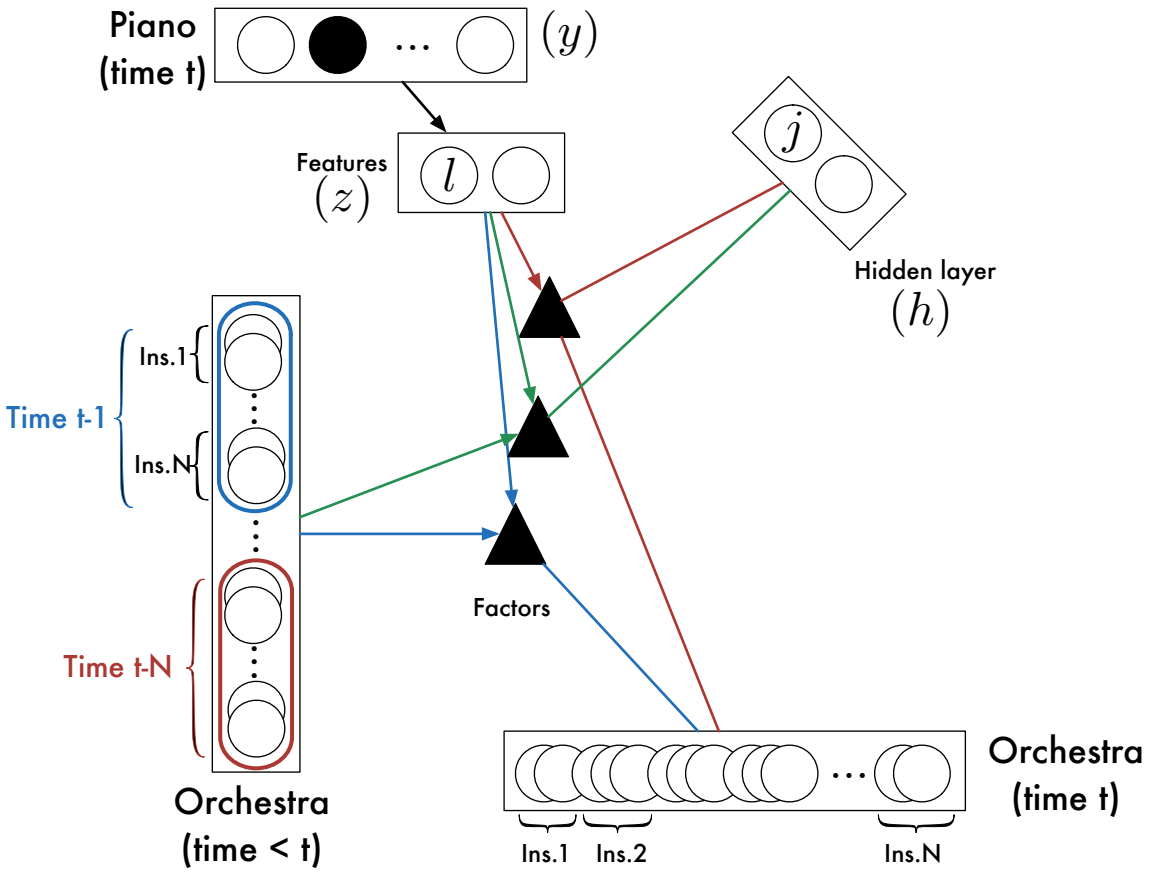


Figure 6.2: The FGCRBM model applied to orchestration. Context units are the concatentation of the recent past of of the orchestra, and style units are defined by the current frame of piano part.

We reduce the number of possible pitch to those of a grand piano (88 pitch from). One can argue that the ambitus of a piccolo for instance goes higher than the highest note of a piano in frequency. Since we work only with symbolic notations, the symbolic score of a piccolo is comprised in the symbolic score of a piano (the piccolo play the note written on the score an octave higher). In our framework we chose 14 instruments indexed by :

1. Violin
2. Viola
3. Cello
4. Double-bass
5. Harp
6. Timpani
7. Trumpet
8. Trombone
9. Tuba
10. French or Eng. horn
11. Oboe
12. Bassoon
13. Clarinet
14. Flute

For each line of the matrix, i.e. each instrument, we can compute the frame-level accuracy as presented in the previous section, and then sum the accuracy of all the instrument

$$Acc_{orchestral} = \sum_{i=1}^{N_{instrument}} Acc(M(i,t)) \tag{6.1}$$

where $M(i,t)$ is 88 size vector constituted by the pitches of the instrument i. Note that in order to compare different models, it is necessary to have the same number of instruments $N_instruments$.

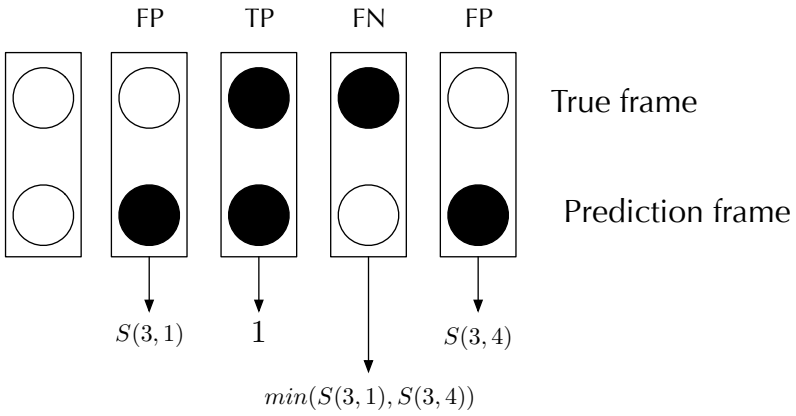


Figure 6.3: Definition of the true positive, false positive and false negative for the orchestral prediction task

Toward a better definition We propose an other definition for the orchestral predictive task that we have not had the time to test. It would include a knowledge about the similarity between instruments. Indeed, if in the original score a viola plays a certain note, it is a less important mistake if a violin playing the same note is proposed than if a trumpet is proposed.

If $M_{pred}(t)$ is the predicted orchestral matrix at time t and $M_{true}(t)$ the original matrix, we define the orchestral accuracy as following :

$$Acc_{orch}(t) = \sum_{p=1}^{88} Acc_{orch}(p,t) \tag{6.2}$$

where for all p in $[1, 88]$

$$Acc_{orch}(p,t) = \frac{TP(p,t)}{TP(p,t) + FP(p,t) + FN(p,t)} \tag{6.3}$$

False negative and false positive are defined through a similarity matrix S where $S(i,j)$ is the cost of replacing the instrument i by the instrument (see figure figure 6.3 on page 61).

- A true positive note is defined by $M_{pred}(i,p,t) = M_{true}(i,p,t) = 1$. $TP(p,t)$ is equal to the number of true positive notes.
- A false positive note is a note present in the prediction vector $M_{pred}(p,t)$ but not in the true vector $M_{true}(p,t)$. Each false positive note j is assigned a cost $S(\hat{i},j)$ where $\hat{i} = argmin_{i \in I_{on}} S(i,j)$ where I_{on} is the set of notes equal to 1 in the true vector $M_{true}(p,t)$. $FP(p,t)$ is equal to the weighted sum of the false positive notes $\sum_{j \in fp} S(\hat{i},j)$.
- A false negative note is a note present in the true vector $M_{true}(p,t)$ but not in the prediction vector $M_{pred}(p,t)$. Each false negative note i is assigned a cost $S(i,\hat{j})$ where $\hat{j} = argmin_{j \in J_{on}} S(i,j)$ where $J_{o,}$ is the set of notes equal to 1 in the prediction vector $M_{true}(p,t)$. $FP(p,t)$ is equal to the weighted sum of the false negative notes $\sum_{i \in fn} S(i,\hat{j})$.

A first remark on the matrix $S(i,j)$ is that its diagonal, while unused, should be equal to zeros, and that a low value describe two almost similar instruments. Its definition remain problematic and could

Model	Orchestral Frame Acc % DOSIM
Random	0.52%
Repeat	93.25%
CRBM	1.96%
FGCRBM	2.01%

Figure 6.4: Event-level expected accuracy for various musical models

be defined either from spectral properties or perceptual studies. The idea is to describe a similarity of timbre between two instruments.

6.3 Results

Datasets

We used a parallel database generated from a database used for a software called Digital Orchestra Simulator (DOSIM) [DOS]. The database is composed by 76 xml files. The quality of the orchestrations or the transcriptions were variable, and we kept only 39 files. Among them, some part were not orchestrated or not transcripted. Only the part were both a piano and an orchestra part were simultaneously available have been kept. Several reduction were often available for the same orchestral score. Since they were all mixed in the same *xml* file and some of them split between left and right hand, the best way we found to extract the piano part while keeping this process automatic was to simply add all the tracks marked as "reduction" to form a single track. Hence, the reduction we used were sometimes overloaded with more notes that a pianist could actually play.

We split this set of files between training and testing files. We used the second half of *Arabesque n°1* from Debussy and an orchestrated version by Denis Bouliane as a test file. The rest of the database was used to train the networks. Eventually, we obtained 33829 training frames and 1472 testing frames.

Models

We evaluated the two models presented at the beginning of this chapter (chapter 6) : the CRBM and the FGCRBM applied to projective orchestration. The quantization was in 64^th-note (16 notes in a quarter note) to capture the smallest event.

The CRBM has two layers of 500 units for the first layer and 750 for the second. A temporal order of 16 was used. The FGCRBM has one layer of 1200 hidden units. 600 factors units, 300 features units and a temporal order of 16 were used.

Both were trained on 300 epochs using 10 steps of contrastive divergence. The learning parameters were set to $1e^{-3}$ for the learning rate, $2e^{-4}$ for the weight decay, and 0.9 for the momentum. The sparsity parameters were set to 0.1 for the sparsity target, 0.5 for the sparsity coefficient (β) and 0.9 for the sparsity momentum (λ).

Results

The results for the CRBM and the FGCRBM models on the orchestral predictive task are presented in table (figure 6.4 on page 62).

The results are bad for the moment, and those two architectures failed at capturing the most basic correlations between the piano score and the orchestra score. The orchestration of the last half of Debussy's Arabesque can be heard here (FGCRBM orchestration).

A first remark is that we have not tested enough parametrization to assert that those models adapted or not. Indeed, the number of factors for instance can have a great impact over the performances of the model. An other option would be to share weights, for instance to consider that the weights between all the notes of a same instrument and its factors are the same. This would reduce greatly the number of weights and then facilitate the training procedure. Eventually, the intensity of the note has not been taken into account, but would be important in a further development.

Chapter 7

Summary and future work

7.1 Summary

Automatic orchestration is a daunting task that has been tackled by only a few works. Our initial goal was to build an automatic projective orchestration system. Deep neural networks appeared as an interesting solution to assess this vast problem. An orchestral score can be represented as a binary high-dimensional time series, and the conditional models appeared to be a promising solution in order to model the complex distribution that represent a set of orchestral scores.

Chapter 1 introduced the state of the art in automatic orchestration. We can distinguish two interesting approaches : a CSP-based approach which is limited and the Orchids system which is a nice solution for inductive orchestration, but not adapted for long temporal frames. Since we focused on projective orchestration, we proposed three new approaches : an Orchids based solution enhanced by an orchestral continuity principle, a voice leading approach, and a radically different solution based on statistical inference to automatically learn orchestration rules. The high potential of the deep learning methods and the fact that they have almost not been used in symbolic music analysis yet brought us to choose this solution over the Orchids-based methods. Chapter 2 reviews some of the most important concepts in deep learning, focusing on generative models adapted to time series. The Restricted Boltzmann Machine is introduced through the prism of Graphical Probabilistic Models which defines a powerful theoretic framework. The Semi-RBM model is then introduced as a promising model for modeling harmonic structures. Chapter 3 is dedicated to time modeling in generative statistical models. Among the most promising attempts to efficiently model high-dimensional time series, two of them applied to polyphonic musical sequences modeling can be distinguished in the literature. A first model is based on LSTM memory cells, and a second on a RNN-RBM architecture. Both models are based on Recurrent Neural Networks and have been qualitatively evaluated. Hence, they will constitute a solid baseline for the models we tried to develop.

In chapter 4, we adapted conditional models for music modeling and proposed a model for polyphonic musical sequences called Harmonic Conditional RBM. The HCRBM model is obtained by adding lateral connections between visible units in a standard CRBM model. Those architectures present the advantages of being generative and easy to train. Beside, they offer the possibility to easily observe the knowledge learnt by the network. In chapter 5, we evaluated the performance of our model and the performances of several already existing models (RBM, CRBM and GCRBM) on a short-term predictive task. We used the frame-level accuracy measure which is the canonic measure, but noticed the fact that it is sensible to the quantization. We proposed an event-level accuracy measure to solve this problem. Among the presented models, the CRBM model had the best results when using the

frame-level accuracy (31.18%), but not as good as the Recurrent Neural Networks that constituted our baseline. (33.12%). Note that the HCRBM has not been tested yet on a frame-level. The HCRBM had the best results for the event-level measure (28.07%). We then used the event-level measure to perform a search for the optimal set of hyper-parameters of the CRBM model. The optimal set of hyper-parameters found has been actually used to train an HCRBM and provided the best performances among the tested architectures. We eventually proposed a qualitative evaluation of the CRBM model by observing the weights learnt and the sequences generated after training the network on the *JSB Chorales* database. This quantitative evaluation gave us a lot of information about the ability of the proposed models for musical sequences modeling. First, the temporal relations learnt by conditional models are not able to represent long-range dependences since they only depends on the last N visible frames. However, interesting structures were learnt at a short-term level. The HCRBM model failed to learn harmonic structure because of a conflict between temporal and harmonic weights during the learning phase. Eventually, the most interesting sequences were generated by the CRBM model trained on an event-level.

An orchestration system based on conditional models is eventually introduced in chapter 6. We adapted two models, the CRBM and the FGCRBM to the projective orchestration problem. In those models, we try to predict each frame of the orchestral score thanks to the current piano frame and the recent past orchestral frames. We defined an evaluation framework which consists in a predictive task based on a frame-level accuracy measure. We also introduce a measure that would take similarities between instruments into considerations. We eventually tested our models in this framework. The results are poor for the moment, and the models obviously failed to learn any structure.

7.2 Future works

Our HCRBM model for musical sequences must be investigated further. In particular, a long but necessary hyper-parameter search must be ran. Another major improvement would be to used a validation test. Besides, we believe that our model can be greatly improved by changing the data representation and pre-processing. Indeed, we believe that the conditional models we tried to use are not adapted for the data we wanted to model. Especially the strong sparsity of the data is problematic and an other data representation could avoid this. A pre-processing step consisting in transposing all the pieces to the same tonality (C major/minor) should also greatly improve the results.

For the evaluation of the models, log-likelihood should be performed instead of a prediction task. To do so, either AIS should be performed to evaluate the partition function, which will be very-long for conditional models, either an architecture based on NADE should be developed, where the joint probability of visible and hidden units is easier to compute.

There is still a lot of work to do in order to obtain a projective orchestration system. First of all, it would surely benefit from the data representation and pre-processing improvement we could find for the aforementioned music sequences model. To improve the models themselves, we think that conditional models are a great lead, but could be simplified by sharing weights among the same instruments for instance. Sharing weights would reduce the number of weights and facilitate the training of the model. Beside, we have not tried enough configuration to draw negative conclusions, and a long yet necessary hyper-parameter search must be ran on this particular task for those two models.

A immediate improvement of those models would be to include intensity in the data representation. It would gives a precious information since a loud section would be orchestrated with a lot of instrument whereas a low intensity section would be orchestrated with fewer instruments.

A long-term improvement would be to add signal information to this model. This would be obtained by adding an information such as a spectrogram to the purely symbolic representation. Hence, a timbral target could be added. Those architecture would probably require a large number of hidden units and long training time. Hence, an auxiliary objective is to develop a complete and efficient toolbox for deep-learning applied to music, implementing most of the recent algorithm and being able to take advantage of the most recent GPU architectures to reduce the computation time to its minimum.

Appendix A

Restricted Boltzmann Machines

A.1 Marginal distribution in a RBM

In order to obtain the conditional probability of a given visible units knowing the hidden units, we first need the marginal distribution of the hidden units

$$p(\mathbf{h}) = \sum_{\mathbf{v}} p(\mathbf{v},\mathbf{h}) = \frac{1}{Z}\sum_{\mathbf{v}} e^{-E(\mathbf{v},\mathbf{h})} = \frac{1}{Z}\sum_{\mathbf{v}} e^{\sum_{i=1}^{m} a_i v_i + \sum_{j=1}^{n} b_j h_j + \sum_{i=1}^{m}\sum_{j=1}^{n} v_i W_{ij} h_j} \tag{A.1}$$

$$= \frac{1}{Z}\prod_j e^{b_j h_j} \sum_h \prod_i e^{v_i(a_i + \sum_j W_{ij} h_j)} \tag{A.2}$$

Since $\mathbf{v} = (v_1, ..., v_m)$, the sum over can be decomposed as

$$p(\mathbf{h}) = \frac{1}{Z}\prod_j e^{b_j h_j} \sum_{v_1}\sum_{v_2}...\sum_{v_m}\prod_i e^{v_i(a_i + \sum_j W_{ij} h_j)}$$

$$= \frac{1}{Z}\prod_j e^{b_j h_j} \prod_i \sum_{v_i} e^{v_i(a_i + \sum_j W_{ij} h_j)}$$

Because $\mathbf{V}$ is a binary variable, the sum over all the possible value of v_i is reduced to a sum over two terms, and

$$p(\mathbf{h}) = \frac{1}{Z}\prod_j e^{b_j h_j} \prod_i 1 + e^{a_i + \sum_j W_{ij} h_j}$$

From the marginal distribution of the hidden unit, we can get the conditional distribution of the visible units given the hidden units

$$p(\mathbf{v}|\mathbf{h}) = \frac{p(\mathbf{v},\mathbf{h})}{p(\mathbf{h})}$$

$$= \frac{\frac{1}{Z}\prod_j e^{b_j h_j}\prod_i e^{v_i(a_i + \sum_j W_{ij} h_j)}}{\frac{1}{Z}\prod_j e^{b_j h_j}\prod_i 1 + e^{a_i + \sum_j W_{ij} h_j}}$$

$$= \frac{\prod_i e^{v_i(a_i + \sum_j W_{ij} h_j)}}{\prod_i (1 + e^{a_i + \sum_j W_{ij} h_j})}$$

Hence for each visible unit i and hidden unit j

Equation 11 (Conditional probabilities of the visible and hidden units in an *RBM*)

$$p(v_i = 1|\boldsymbol{h}) = sigm\left(a_i + \sum_j W_{ij} h_j\right) \tag{A.3}$$

$$p(h_j = 1|\boldsymbol{v}) = sigm\left(b_j + \sum_i W_{ij} v_i\right) \tag{A.4}$$

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

A.2 Factorizing the data driven part of the log-likelihoof gradient in a RBM

The data driven part of the equation factorizes smoothly

$$\sum_{\mathbf{h}} p(\mathbf{h}|\boldsymbol{v_S}) \frac{\partial E(\boldsymbol{v_S}, \mathbf{h})}{\partial W_{ij}} = \sum_{\mathbf{h}} p(\mathbf{h}|\boldsymbol{v_S}) v_i^S h_j \tag{A.5}$$

The vector $\mathbf{h}$ can be decomposed in its j^{th} coordinate h_j and the other coordinates $\boldsymbol{h_{-j}}$.

$$\sum_{\mathbf{h}} p(\mathbf{h}|\boldsymbol{v_S}) \frac{\partial E(\boldsymbol{v_S}, \mathbf{h})}{\partial W_{ij}} = \sum_{\mathbf{h}} p(h_j, \boldsymbol{h_{-j}}|\boldsymbol{v_S}) h_j v_i^S \tag{A.6}$$

$$= \underbrace{\sum_{\boldsymbol{h_{-j}}} p(\boldsymbol{h_{-j}}|\mathbf{v})}_{=1} \sum_{h_j} p(h_j|\boldsymbol{v_S}) \underbrace{h_j}_{\in\{0,1\}} v_i^S \tag{A.7}$$

$$= p(h_j|\boldsymbol{v_S}) v_i^S \tag{A.8}$$

Hence,

$$\sum_{\mathbf{h}} p(\mathbf{h}|\boldsymbol{v_S}) \frac{\partial E(\boldsymbol{v_S}, \mathbf{h})}{\partial W_{ij}} = sigm\left(\sum_{k=1}^{m} W_{kj} v_k + b_j^h\right) v_i^S \tag{A.9}$$

A.3 RBM update procedure for binomial units

The CD-K algorithm for a RBM is here derived in a very practical way, in a Matlab-like syntax but with some short-cuts.

- Binary states must always be used for the hidden units when visible units are going to be reconstructed from those hidden units. In practice, it means that hidden states can take the mean-field value only on the last update (negative phase), and be sampled otherwise. Visible units can always take the mean-field value [Hin10].
- Momentum set to zero for the first epochs, since the previous values are not meaningful (think about a ball rolling down a surface).

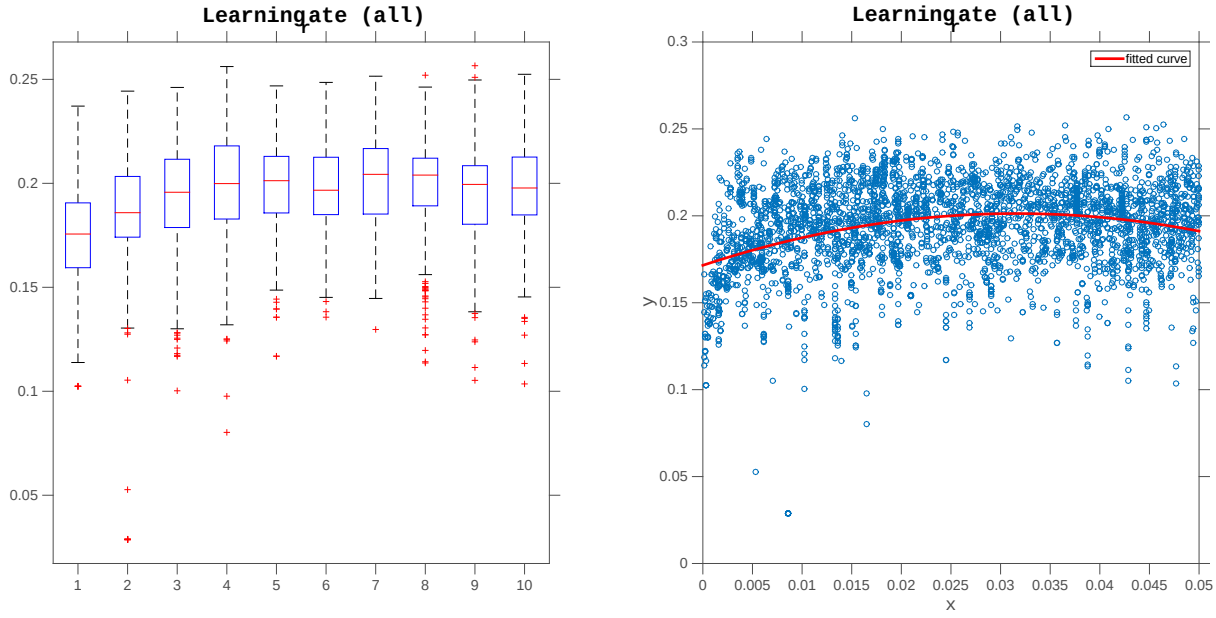


Figure A.1: Box plot and best fitting curve for the learning rate on the CRBM model.

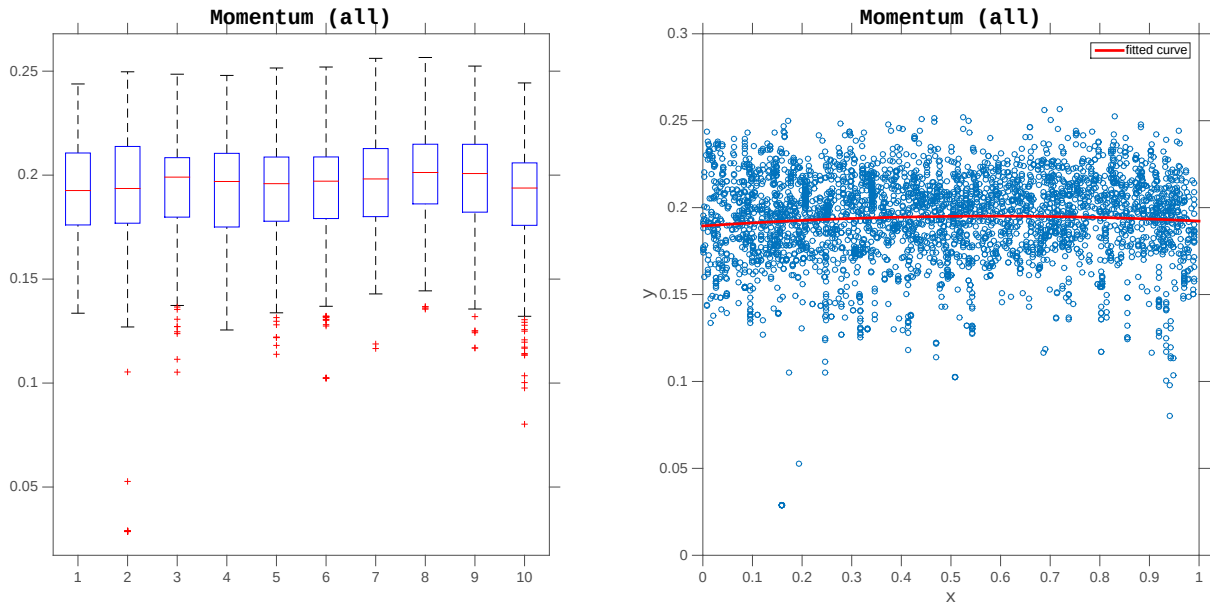


Figure A.2: Box plot and best fitting curve for the momentum on the CRBM model.

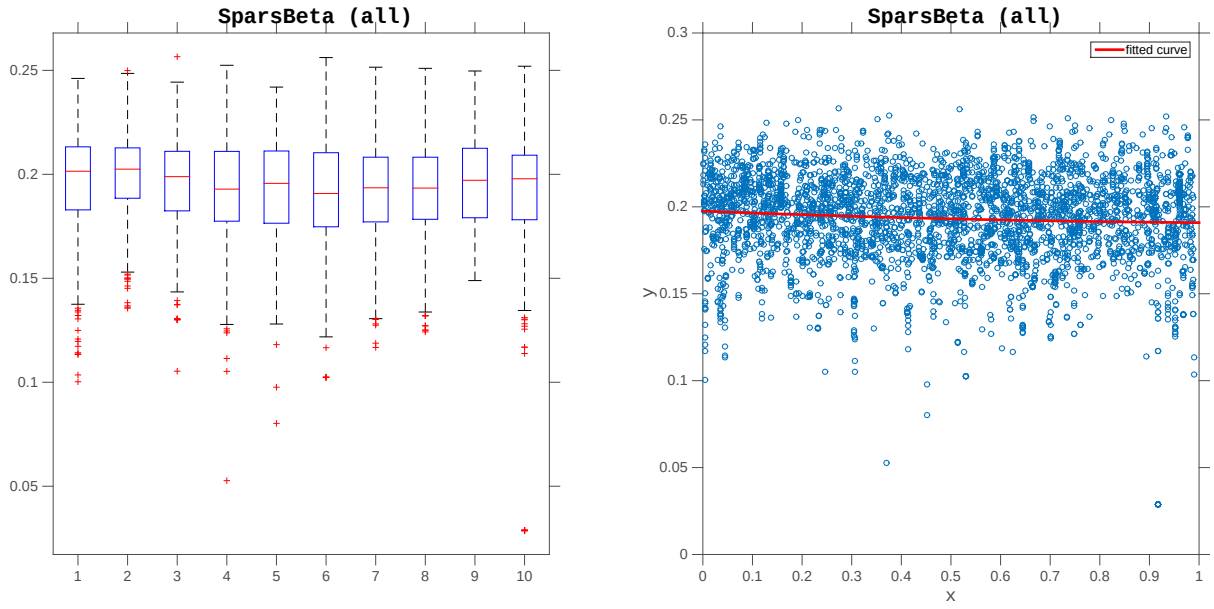


Figure A.3: Box plot and best fitting curve for the sparsity coefficient on the CRBM model.

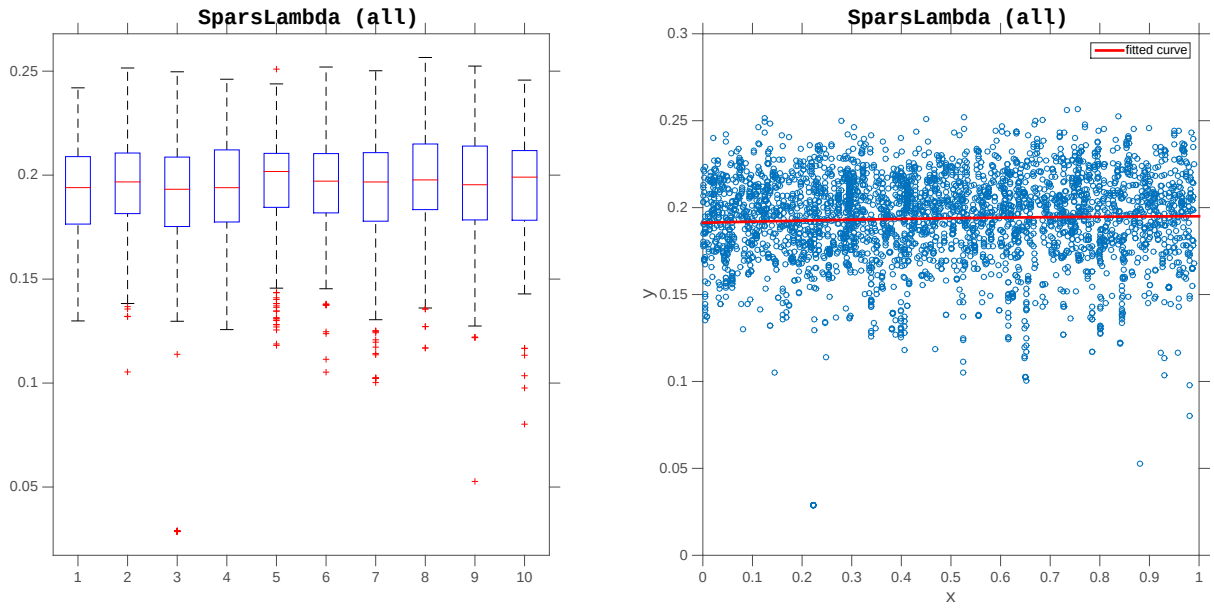


Figure A.4: Box plot and best fitting curve for the sparsity momentum on the CRBM model.

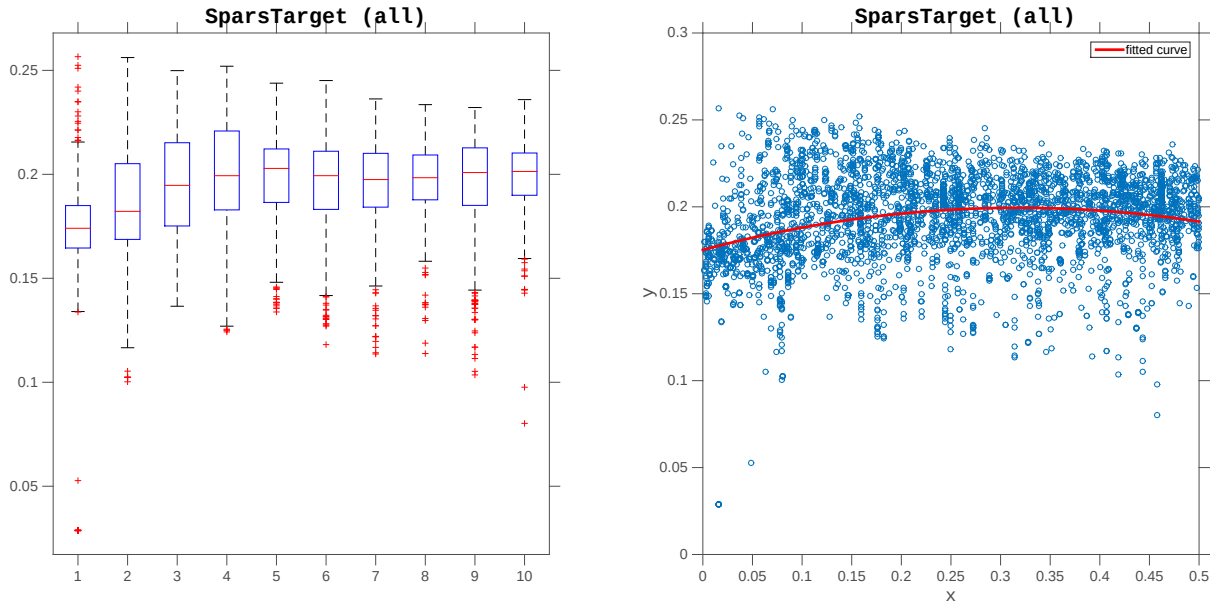


Figure A.5: Box plot and best fitting curve for the sparsity target on the CRBM model.

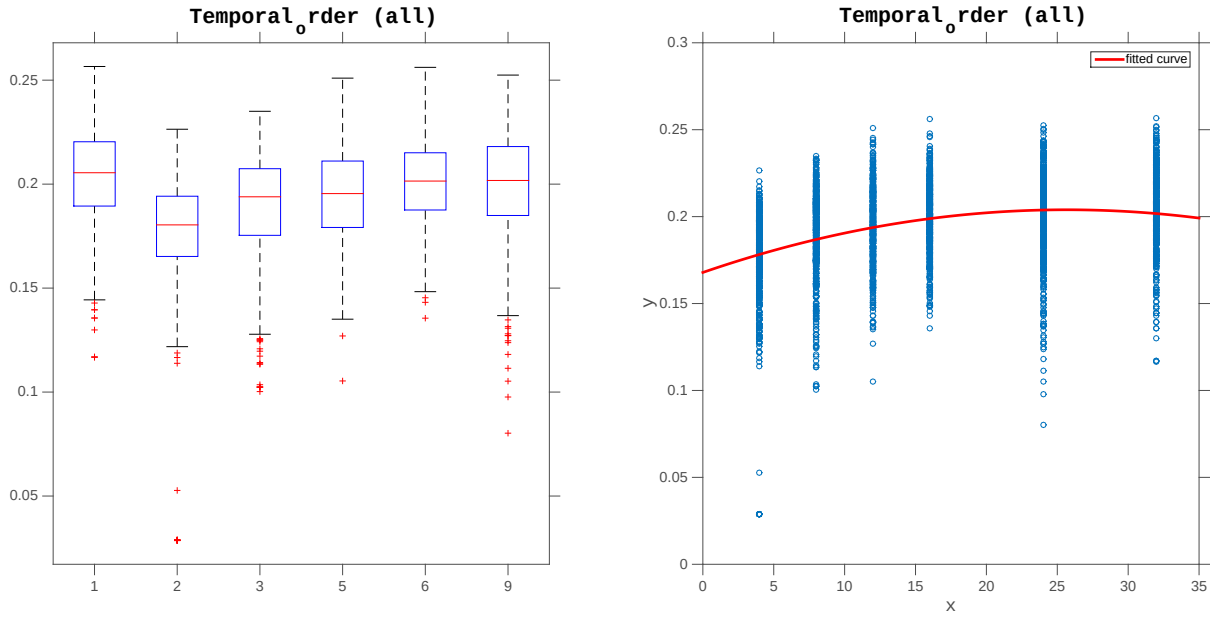


Figure A.6: Box plot and best fitting curve for the temporal order on the CRBM model.

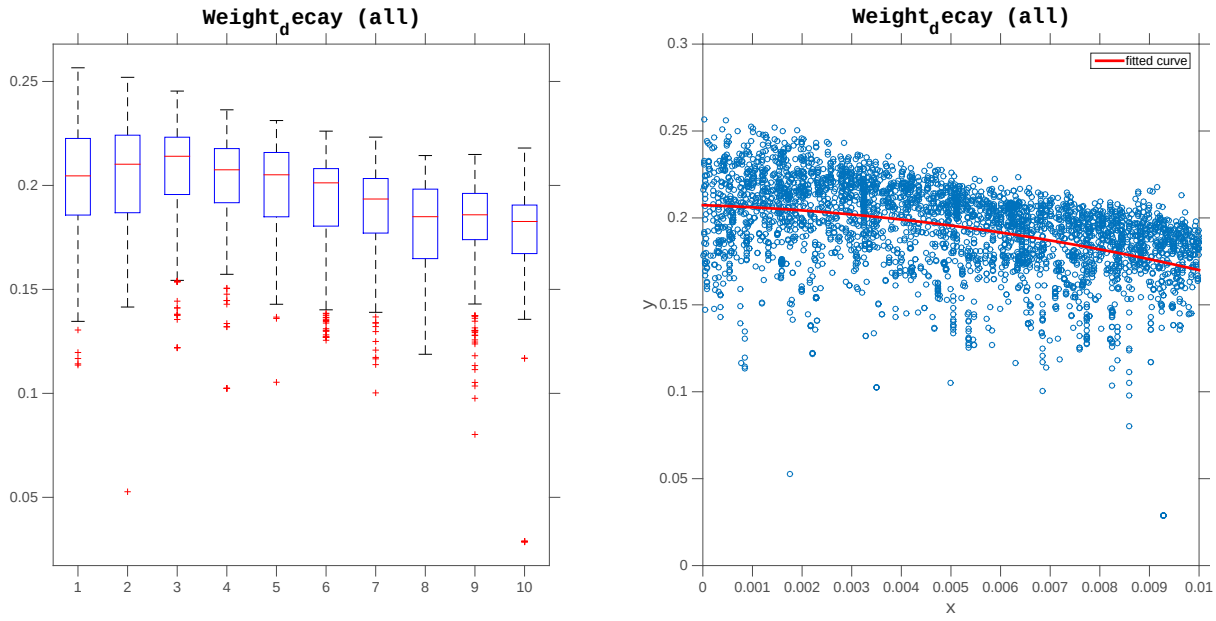


Figure A.7: Box plot and best fitting curve for the weight decay on the CRBM model.

A.4 Hyper-parameters results

A.5 Qualitative analysis

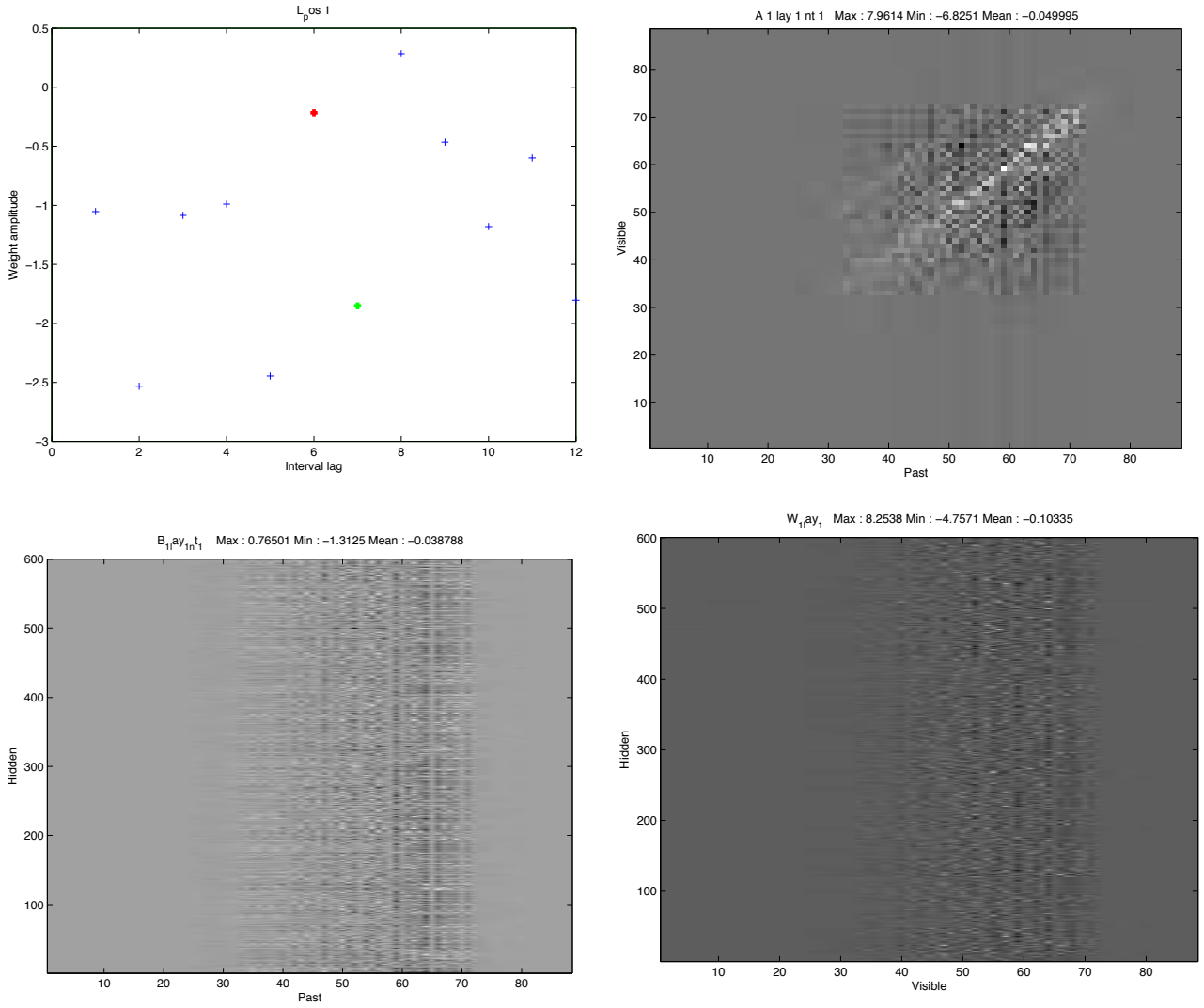


Figure A.8: Harmonic and temporal connections in a HCRBM. No classic harmonic relations are highlighted by the harmonic connections. Temporal connections looks like one of a CRBM, but with lower values.

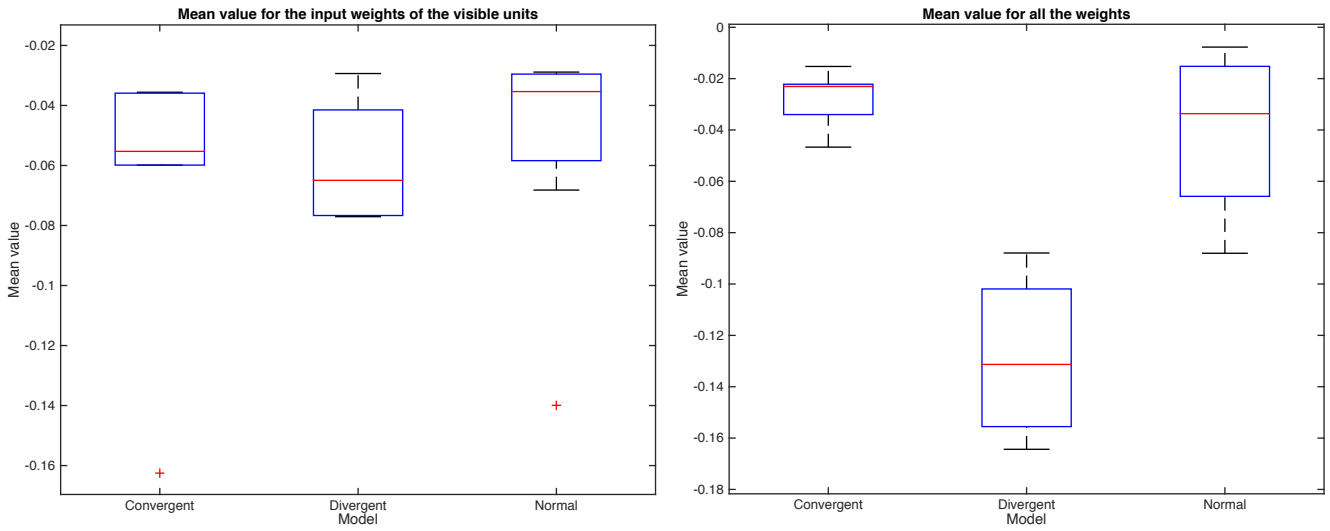


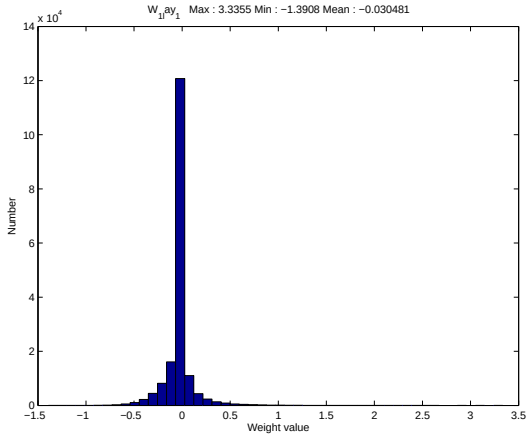
Figure A.9: Mean over the visible units of their input weight. This value as been computed on several model for the three types of observed behaviour : *collapsing* series, *diverging* (number of notes per time frame explodes) or *normal* series (number of notes stays between 3 and 5 at each time frame). Note that only the input visible units for pitch between 40 and 70 (on the 88 notes scale) have been taken into account since extreme note are often constantly off.

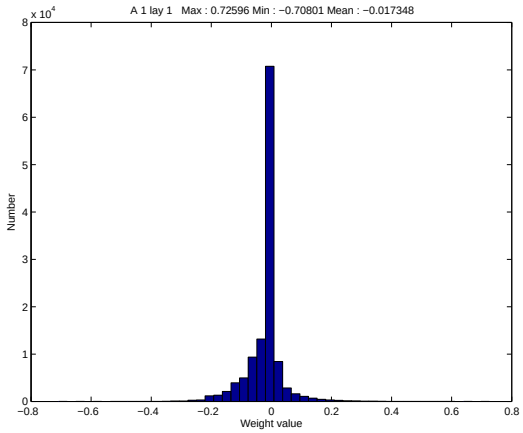
Algorithm 1 Mini-batch CD-K algorithm in a RBM

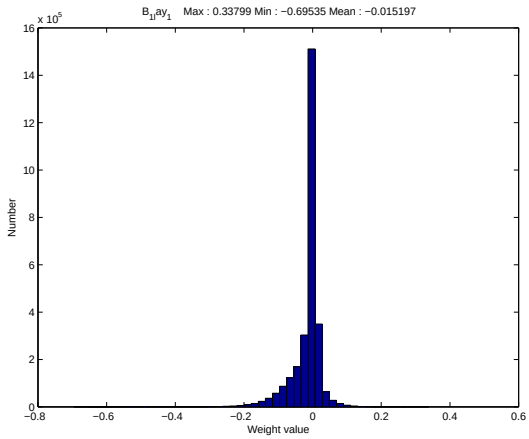
procedure CD-K(N,M)
  ϵ : learning rate
  λ : momentum
  α : weight decay
  β : sparsity rate
  for N epochs **do**
    for M mini-batch **do**
      for $\mathbf{v}_{data} \in M$ **do**
        $\mathbf{h}_{data} \sim p(\mathbf{h}|\mathbf{v}_{data})$ ▷ *Positive phase*
        $\mathbf{h}_0 = \mathbf{h}_{data}$ ▷ *Negative phase*
        for $k = 1 \vdots K-1$ **do**
          $\mathbf{v}_k \leftarrow p(\mathbf{v}|\mathbf{h}_{k-1})$
          $\mathbf{h}_k \sim p(\mathbf{h}|\mathbf{v}_k)$
        $\mathbf{v}_{model} \leftarrow p(\mathbf{v}|\mathbf{h}_K)$
        $\mathbf{h}_{model} \leftarrow p(\mathbf{h}|\mathbf{v}_{model})$
      $q = \lambda_{spars}.q + (1-\lambda_{spars}).\mathbb{E}_M\left[h^2_{model}\right]$ ▷ *Compute sparsity terms*
      $S_W = (q-p)v_i$
      $S_h = q - p$
      $\Delta W_{ij} = \lambda.\Delta W_{ij} + (1-\alpha).\epsilon.\left\{\frac{1}{|M|}.\sum_{\mathbf{v}_{data}\in M}\left[(v_i h_j)_{data} - (v_i h_j)_{model}\right] - \alpha W_{ij} - \beta.S_W\right\}$
      $\Delta b_i^{(v)} = \lambda.\Delta b_i^{(v)} + (1-\alpha).\epsilon.\left\{\frac{1}{|M|}.\sum_{\mathbf{v}_{data}\in M}\left[< v_i >_{data} - < v_i >_{model}\right]\right\}$
      $\Delta b_j^{(h)} = \lambda.\Delta b_j^{(h)} + (1-\alpha).\epsilon.\left\{\frac{1}{|M|}.\sum_{\mathbf{v}_{data}\in M}\left[< h_j >_{data} - < h_j >_{model}\right] - \beta.S_h\right\}$

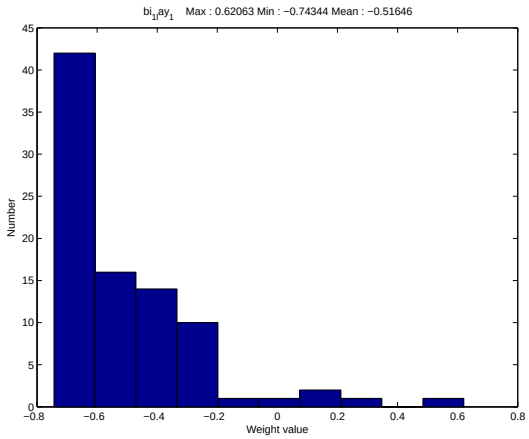
Collects statistics

Weights update


W1lay1 Max : 3.3355 Min : −1.3908 Mean : −0.030481
x 10^4
Number
Weight value


A 1 lay 1 Max : 0.72596 Min : −0.70801 Mean : −0.017348
x 10^4
Number
Weight value


B1lay1 Max : 0.33799 Min : −0.69535 Mean : −0.015197
x 10^5
Number
Weight value


bi1lay1 Max : 0.62063 Min : −0.74344 Mean : −0.51646
Number
Weight value

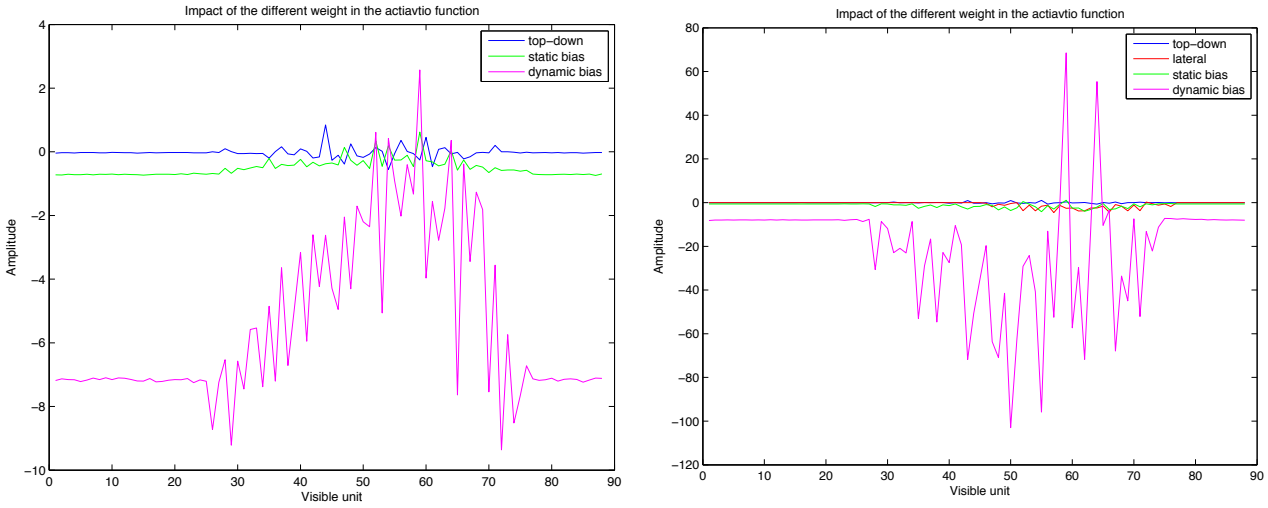


Figure A.11: **Influence of the different weights over the activation of the visible units.** On the left for the CRBM, on the right for the HCRBM. We can observe that whereas the different influences in the CRBM are equally distributed for the middle pitches, the temporal connections in the HCRBM totally overweights the other contributions.

Glossary

AI Artificial Intelligence.

AIS Annealed Importance Sampling.

BPTT Back-propagation though time.

CD Contrastive Divergence.

CRBM Conditional Restricted Boltzmann Machine.

CSP Constraint Satisfaction Problem.

DBN Deep Belief Network.

DOSIM Digital Orchestra Simulator.

FGCRBM Factored Gated Conditional RBM.

FGRBM Factored Gated RBM.

GCRBM Gated Conditional RBM.

GPM Graphical Probabilistic Model.

GPU Graphics Processing Unit.

GRBM Gated RBM.

HCRBM Harmonic Contrastive RBM.

HMM Hidden Markov Model.

MCMC Monte Carlo Markov Chain.

MOTS Multi Objective Time Series.

MRF Markov Random Field.

NN Neural Networks.

PCD Persistent Contrastive Divergence.

RBM Restricted Boltzmann Machine.

RNN-RBM Recurrent Neural Network Restricted Boltzmann Machine.

RTRBM Recurrent Temporal Restricted Boltzmann Machine.

SRBM Semi-Restricted Boltzmann Machine.

Style-gated FCRBM Style-gated Factored Conditional RBM.

TRBM Temporal Restricted Boltzmann Machine.

List of Figures

Bibliography

[AW05] Moray Allan and Christopher KI Williams. Harmonising chorales by probabilistic inference. Advances in neural information processing systems, 17:25–32, 2005.

[BB12] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. The Journal of Machine Learning Research, 13(1):281–305, 2012.

[BBSK10] Greg Bickerman, Sam Bosley, Peter Swire, and Robert M Keller. Learning to create jazz melodies using deep belief nets. 2010.

[BCV13] Yoshua Bengio, Aaron Courville, and Pierre Vincent. Representation learning: A review and new perspectives. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 35(8):1798–1828, 2013.

[Ben09] Yoshua Bengio. Learning deep architectures for ai. Foundations and trends in Machine Learning, 2(1):1–127, 2009.

[Ber44] Hector Berlioz. Grand traité d'instrumentation et d'orchestration modernes. Schonenberger, 1844.

[BLBV12] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. arXiv preprint arXiv:1206.6392, 2012.

[Car08] Grégoire Carpentier. Approche computationnelle de l'orchestration musicale : Optimisation multicritère sous contraintes de combinaisons instrumentales dans de grandes banques de sons. PhD thesis, IRCAM, 2008.

[DOS] *DOSIM :* digital orchestration simulator. `http://www.dosimmusic.com/index.html`. Accessed: 2015-01-27.

[EA10] Philippe Esling and Carlos Agon. Composition of sound mixtures with spectral maquettes. In International Computer Music Conference, 2010.

[ECA10] Philippe Esling, Grégoire Carpentier, and Carlos Agon. Dynamic musical orchestration using genetic algorithms and a spectro-temporal description of musical instruments. Applications of Evolutionary Computation, pages 371–380, 2010.

[EL08] Douglas Eck and Jasmin Lapalme. Learning musical structure directly from sequences of music. University of Montreal, Department of Computer Science, CP, 6128, 2008.

[ES02] Douglas Eck and Juergen Schmidhuber. Finding temporal structure in music: Blues improvisation with lstm recurrent networks. In Neural Networks for Signal Processing, 2002. Proceedings of the 2002 12th IEEE Workshop on, pages 747–756. IEEE, 2002.

[Esl12] Philippe Esling. Multiobjective time series matching and classification. PhD thesis, IRCAM, 2012.

[FI14] Asja Fischer and Christian Igel. Training restricted boltzmann machines: An introduction. Pattern Recognition, 47(1):25–39, 2014.

[Gar15] James Garson. Connectionism. In Edward N. Zalta, editor, The Stanford Encyclopedia of Philosophy. Spring 2015 edition, 2015.

[Hin10] Geoffrey Hinton. A practical guide to training restricted boltzmann machines. Momentum, 9(1):926, 2010.

[HLZ96] Martin Henz, Stefan Lauer, and Detlev Zimmermann. Compoze-intention-based music composition through constraint programming. In Tools with Artificial Intelligence, 1996., Proceedings Eighth IEEE International Conference on, pages 118–121. IEEE, 1996.

[HOT06] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. Neural Comput., 18(7):1527–1554, July 2006.

[HSD12] Eliot Handelman, Andie Sigler, and David Donna. Automatic orchestration for automatic composition. In Proc. of MUME, pages 43–48, 2012.

[Koe41] Charles Koechlin. Traité de l'orchestration. Éditions Max Eschig, 1941.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.

[LEN08] Honglak Lee, Chaitanya Ekanadham, and Andrew Y Ng. Sparse deep belief net model for visual area v2. In Advances in neural information processing systems, pages 873–880, 2008.

[LGRN09] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In Proceedings of the 26th Annual International Conference on Machine Learning, pages 609–616. ACM, 2009.

[LKL14] Martin Längkvist, Lars Karlsson, and Amy Loutfi. A review of unsupervised feature learning and deep learning for time-series modeling. Pattern Recognition Letters, 42:11–24, 2014.

[LM11] Hugo Larochelle and Iain Murray. The neural autoregressive distribution estimator. In International Conference on Artificial Intelligence and Statistics, pages 29–37, 2011.

[LP03] Victor Lavrenko and Jeremy Pickens. Polyphonic music modeling with random fields. In Proceedings of the eleventh ACM international conference on Multimedia, pages 120–129. ACM, 2003.

[LR14] I.-Ting Liu and Bhiksha Ramakrishnan. Bach in 2014: Music composition with recurrent neural network. CoRR, abs/1412.3191, 2014.

[McA13] Stephen McAdams. Timbre as a structuring force in music. In Proceedings of Meetings on Acoustics, volume 19, page 035050. Acoustical Society of America, 2013.

[NG14] Henri NG. Inférence de connaissances signal symboliques à partir de séries temporelles multivariées pour l'orchestration musicale. Master's thesis, IRCAM, 2014.

[NH10] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In Proceedings of the 27th International Conference on Machine Learning (ICML-10), pages 807–814, 2010.

[OH08] Simon Osindero and Geoffrey E Hinton. Modeling image patches with a directed hierarchy of markov random fields. In Advances in neural information processing systems, pages 1121–1128, 2008.

[PBM+03] Jeremy Pickens, Juan Pablo Bello, Giuliano Monti, Mark Sandler, Tim Crawford, Matthew Dovey, and Don Byrd. Polyphonic score retrieval using polyphonic audio queries: A harmonic modeling approach. Journal of New Music Research, 32(2):223–236, 2003.

[PGS+11] Geoffroy Peeters, Bruno L Giordano, Patrick Susini, Nicolas Misdariis, and Stephen McAdams. The timbre toolbox: Extracting audio descriptors from musical signals. The Journal of the Acoustical Society of America, 130(5):2902–2916, 2011.

[PR01] François Pachet and Pierre Roy. Musical harmonization with constraints: A survey. Constraints, 6(1):7–19, 2001.

[RHW85] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

[RHW88] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. Cognitive modeling, 5:3, 1988.

[Sch15] Jürgen Schmidhuber. Deep learning in neural networks: An overview. Neural Networks, 61:85–117, 2015.

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 15(1):1929–1958, 2014.

[SHT09] Ilya Sutskever, Geoffrey E Hinton, and Graham W Taylor. The recurrent temporal restricted boltzmann machine. In Advances in Neural Information Processing Systems, pages 1601–1608, 2009.

[SM08] Ruslan Salakhutdinov and Iain Murray. On the quantitative analysis of deep belief networks. In Proceedings of the 25th international conference on Machine learning, pages 872–879. ACM, 2008.

[SMH11] Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In Proceedings of the 28th International Conference on Machine Learning (ICML-11), pages 1017–1024, 2011.

[TA11] Charlotte Truchet and Gerard Assayag. Constraint Programming in Music. Wiley, May 2011.

[Tay09] Graham William Taylor. Composable, distributed-state models for high-dimensional time series. PhD thesis, University of Toronto, 2009.

[TH09] Graham W Taylor and Geoffrey E Hinton. Factored conditional restricted boltzmann machines for modeling motion style. In Proceedings of the 26th annual international conference on machine learning, pages 1025–1032. ACM, 2009.

[Tie08] Tijmen Tieleman. Training restricted boltzmann machines using approximations to the likelihood gradient. In Proceedings of the 25th international conference on Machine learning, pages 1064–1071. ACM, 2008.

[TM12] Damien Tardieu and Stephen McAdams. Perception of dyads of impulsive and sustained instrument sounds. Music Perception, 30(2):117–128, 2012.

[Tru04] Charlotte Truchet. Contraintes, recherche locale et composition assistée par ordinateur. Unpublished doctoral dissertation, Université Paris, 2004.

[Tym06] Dmitri Tymoczko. The geometry of musical chords. Science, 313(5783):72–74, 2006.

[YAK10] Ryosuke Yamanishi, Keisuke Akita, and Shohei Kato. Automated composing system for submelody using hmm: A support system for composing music. In HyunSeung Yang, Rainer Malaka, Junichi Hoshino, and JungHyun Han, editors, Entertainment Computing - ICEC 2010, volume 6243 of Lecture Notes in Computer Science, pages 425–427. Springer Berlin Heidelberg, 2010.