The Live Orchestral Piano : a conditional model for musical orchestration

LEOPOLD CRESTEL, Institut de Recherche et Coordination Acoustique/Musique
PHILIPPE ESLING, Institut de Recherche et Coordination Acoustique/Musique

Additional Key Words and Phrases: Automatic orchestration, Conditional Restricted Boltzmann Machine, deep learning

ACM Reference Format:
Léopold Crestel, Philippe Esling, 2010. The Live Orchestral Piano : a conditional model for musical orchestration *ACM Trans. Embedd. Comput. Syst.* V, N, Article A (January YYYY), 13 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Musical orchestration is the subtle art of writing musical pieces for the orchestra, by combining the spectral properties specific to each instrument in order to achieve a particular sonic goal. This apparently simple definition evokes the tremendous complexity that lies in the transformation from a symbolic representation (the score) to an acoustic rendering (the performance of the piece by an orchestra). As pointed out by Walter Piston [**?**], the gap between the two is huge, and the musical notation fails at describing most of the important dimensions of an orchestral work (timbre, couleurs, REFERENCE BLENDING et effets orchestraux). An immediate observation is the versatility of the different interpretations of the same written piece. A more embarrassing consequence is the fact that orchestration remains a mainly empirical discipline, learned through the observation of orchestrations from the elder [Berlioz 1844; Koechlin 1941], but not axiomatized in books. Behind the lack of theoretic construction shines the huge complexity of the discipline. Indeed, if we solely consider the range of notes, dynamics, chords and playing modes provided by a single musical instrument, we can foresee the infinite number of combinations provided by the many instrument that form an orchestra. This huge combinatorial problem is complicated by the non-linear behaviour of instrument mixtures' spectral properties. Computer-aided orchestration aims at designing tools to help composers exploring the massive sets of instrumental possibilities, discover new kind of solutions and formalize unexplained mechanisms in order to build a theory of orchestration.

The wide definition we gave of orchestration actually covers many different subproblems from which arise two different paradigms for writing orchestral music. Among those, we can distinguish *injective* and *projective* orchestration [**?**]. The *injective* paradigm was born from the problem of finding the best instrument in order to reconstruct a particular sound. A typical question would be which instrument and how should the play in order to recreate the sound of a crying baby. A more abstract

Author's addresses: L. Crestel and P.Esling, Représentation Musicales, Institut de Recherche et Coordination Acoustique/Musique;

DOI: http://dx.doi.org/10.1145/0000000.0000000

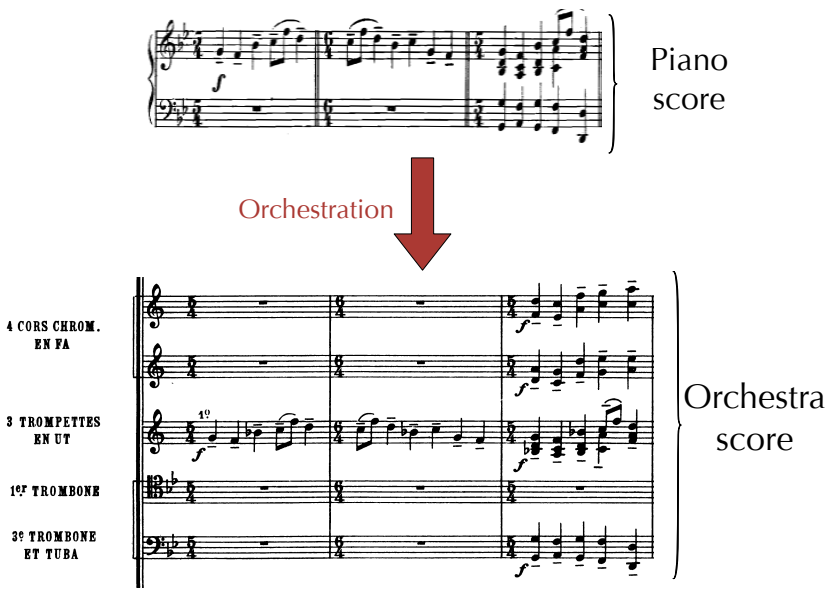

Fig. 1. *Projective orchestration*. A piano score is extended (projected) on an orchestra. For one piano score, many acceptable orchestration exist. Our hypothesis is that a piano score is strongly correlated to any of the orchestration that could be produced from this piece.

and general point of view is to define a timbre target that should be matched with an instrumental mixture. This timbral target is defined by a set of spectro-temporal descriptors. Hence, rhythmic, harmonic and melodic structures emerge as the background of the timbral landscape depicted.

In the projective paradigm, orchestration reinforce the elements of an already existing harmonic, rhythmic and melodic structure. Typically, projective orchestration can be seen as the projection of a piano score on an orchestra (see figure 1) and the classic and romantic repertoires contain a huge amount of example (e.g. the piano reduction by Liszt of the symphonies of Beethoven, *les tableaux d'une exposition*, a piano piece from Modest Moussorgsky orchestrated by Ravel).

Computer-based tool for orchestration have already been developed in the injective framework. Orchids [**?**] is a system based on a genetic algorithm which try to find a mixture of instrument as close as possible of a given timbre target. Several measures based on spectro-temporal descriptors of the instruments are used to build a multi-objective criterion.

The objective of our work is to produce a system able to automatically perform the projective orchestration of a piano score. Practically, for a given piano score as an input it produces an orchestra score. To our best knowledge, this problem has never been tackled before. Our system will rely on statistical inference and more precisely on the most recent advent in machine learning. We will focus on a set of model called conditional Restricted Boltzmann Machine (*cRBM*).

Statistical inference refers to methods which suppose that a set of data has been drawn from a probability distribution. The objective is to deduce properties of this underlying distribution, by observing a subset of those data. In our case, the objective is to infer rules of orchestration by observing pieces written by famous composers (experts). Inferring properties about this underlying probability distribution is called the *learning* phase. Our motivations for using statistical inference are the following. First, it

seems to be an elegant solution to overcome our difficulty in grasping the many facets of musical orchestration. Those methods present a pleasant analogy with empirical learning in the sense that they will automatically learn by observing the vast amount of already existing orchestrations. Those methods are tailored to take advantage of the huge amount of information contained in the already existing orchestration performed by famous composers. The training procedure of our system can then be seen as trying to find the correlations between a piano score and its projective orchestration by a composer.

In the light of what has been said about the multi-modal aspect of orchestration, it can seems absurd to try to build a system based on the mere symbolic information. Indeed, acoustic properties are not described by the limited musical notation used in scores. However, if the acoustic properties are not explicitly described in the score, a composer will write with a deep awareness of them. Hence, if we consider orchestration made by famous composers, their knowledge of acoustic and spectral properties of instrument mixtures is embodied in the purely symbolic transformation between the piano and orchestra score. By learning on such a set of orchestration, we believe that a purely symbolic system will be able to infer this embodied knowledge.

Statistical inference actually covers a wide range of domains. Among them, deep learning recently appeared as a most promising field in intelligence artificial and representation learning. Its impressive results when dealing with huge amount of data, and high dimensional problems let us think that it might be a particularly fitted method for projective orchestration of symbolic sequences. In particular, promising results in text modelling (ref text modeling sutskever...) and symbolic music generation (ref boulanger), which both present many similarities with our problem, are encouraging. The first step before building a model is to choose a good data representation. To represent the information written on the scores we used the *pianoroll* representation. It is inspired by works in automatic music generation and leads to sparse binary vectors which is also the case of naives word vectors (ref Text modeling). The *pianoroll* representation of the scores are then feed to a specifically tailored model based on the Restricted Boltzmann Machine (*RBM*). More precisely, we investigated two models called conditional RBM (*cRBM*) and Factored Gated *cRBM* (*FGcRBM*) [Taylor 2009]. While being able to model complex distributions through latent units, those models implement a notion of context which allow us to model the influence of the past over the present and of the piano over the orchestra. Those model also meet our expectation by being able to generate automatically orchestration after the learning phase. Mathematically, it means that we are able to sample from the distribution learned by the model. If correctly trained, the sampled data will boast structuring elements present in the training set of data. Those models are then called *generatives*. Evaluate generatives models is complicated because of the lack of objective criterion. As pointed out before, for a given piano score, many different orchestration can be proposed and none of them are better than the other.

The test dataset must present the same characteristics of the training dataset while being different. Usually, the set of data we want to model is partitioned into three subsets called training, validating and testing datasets. The validating set is used to define a stopping criterion during the training phase, while the test dataset is used for evaluation purposes. The likelihood of a training test under the modelled distribution is probably the best evaluation measure for generative models. It is the probability that those "ideal" set of data have been generated by the trained system. Unfortunately, the likelihood cannot be easily computed for the proposed models. To evaluate their performance, we define for the first time a quantitative evaluation framework for projective orchestration systems. We took inspiration from the many previous works in automatic music composition (REFS BOULanger ET BAHCLSTM). Most of them rely on a

frame-level predictive task based on an accuracy measure. This evaluation framework originally defined for music prediction is then extended to the projective orchestration task. We highlighted a major flaw in this evaluation framework and proposed a modified version based on an event-level instead of a frame-level measurement. Thanks to this objective criterion, we could explore (through a grid-search, this is the ugly truth...) the hyper-parameter space, select a configuration and compare the different models between them : *cRBM*, *FGcRBM*, and baseline models (random generation and repeat of the previous frame).

An interesting property of the *cRBM* and *FGcRBM* models is that their generative step is sufficiently fast for real-time application. Hence, we propose a system which orchestrate in real-time the performance of a pianist. Our implementation consists in a *Max-MSP* patch that get the midi information from the piano, send it to the trained network which generates the orchestration through a Matlab script and send it back to the Max patch. The communication between the server (Matlab) and client (MAX) is done through the *OSC* protocol. We called this system the Live Orchestral Piano (*LOP*).

Rather than an objective, building an automatic system for orchestration is a starting point in the knowledge quarry task we want to accomplish. By trying to mimic famous composer, we might discover the higher level knowledge we are trying to discover. The long-term goal is to be able to extract from the system we have built a better understanding of orchestration toward its theorization. One advantage of the *cRBM* model is that we can easily observe the correlations and structures learnt by the system and then have a insight into the knowledge extracted by the system.

This paper is organized as follows. In sections 2 we introduce the state of the art in conditional models through three well known models: the RBM, the CRBM and the *FGCRBM*. The orchestration projection task is presented in the section 3 along with an evaluation framework based on a frame-level accuracy measure. The previously introduced models are then evaluated in this framework and the results displayed. The section 4 introduces a real-time *projective* orchestration system using the presented architectures.

2. STATE OF THE ART

We used three models in our work : the *RBM*, the *cRBM* and the *FGcRBM*. They are presented in the following section by increasing level of complexity, each model adding a new *degree of freedom* to the previous one.

2.1. Restricted-Boltzmann Machine

An *RBM* [Hinton et al. 2006] models the probability distribution of a set of random variables graphically represented as *units*. Those units are divided between visible units, denoted by the vector $\boldsymbol{v} = (v_1, ..., v_m)$, and hidden units, $\boldsymbol{h} = (h_1, ..., h_n)$. In the simple model we present here, the visible units models the observed data and hence have the same dimensions, and hidden units some explanatory unobserved factors. The visible and hidden units are connected by weights W_{ij}. The joint probability of the visible and hidden units is given by $p_{model}(\boldsymbol{v}, \boldsymbol{h}) = \frac{\exp^{-E(\boldsymbol{v},\boldsymbol{h})}}{Z}$ where

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

is the energy function associated to the model and $Z = \sum_{v,h} \exp^{-E(v,h)}$ is the (usually intractable) partition function. $\boldsymbol{\theta} = \{\boldsymbol{W}, \boldsymbol{a}, \boldsymbol{b}\}$ is the set of parameters of the network.

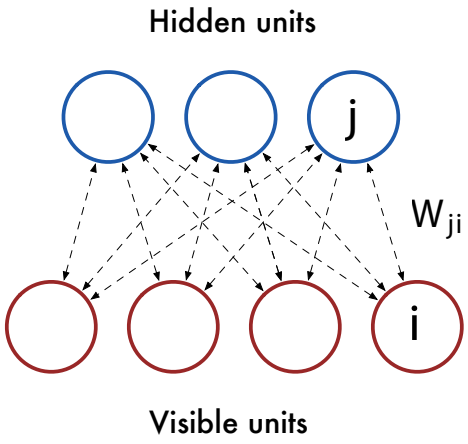

Fig. 2. The *Restricted Boltzmann Machine* (RBM) is defined by its units (i or j) and weights (W_{ij}). Units are divided between visible units (4 units at the bottom) and hidden units (3 top-most). Weights and units define an energy function. Training an RBM consists in lowering the energy function around the example from a training set. Inference in this model is easy to perform since the hidden (resp. visible) units are independent from each others.

In this context, training a model on a set of observed data means that we want the distribution of the model (p_{model}) to be as close as possible to the hypothetical real distribution of those data (p_{data}). A commonly used criterion is to maximize the likelihood of the training set, which can be defined as the probability those data have been sampled from the distribution of the model. The vector from the training set $\mathcal{D}$ are designated by $\boldsymbol{v}^{(l)}$. Minimizing the negative log-likelihood is often preferred

$$\mathcal{L}(\boldsymbol{\theta}|\mathcal{D}) = \frac{1}{N_{\mathcal{D}}} \sum_{\boldsymbol{v}^{(l)} \in \mathcal{D}} - \ln \left[p(\boldsymbol{v}^{(l)}|\boldsymbol{\theta}) \right] \tag{2}$$

where $N_{\mathcal{D}}$ is the size of the dataset.

The search for the minimum of a non-linear function can be tackled by using gradient descent (REFERENCE). The gradient of the negative log-likelihood of a vector from the training database $\boldsymbol{v}^{(l)}$ is given by

$$-\frac{\partial \ln \left[p(\boldsymbol{v}^{(l)}|\boldsymbol{\theta}) \right]}{\partial \boldsymbol{\theta}} \approx \mathbb{E}_{p(\boldsymbol{h}|\boldsymbol{v}^{(l)})} \left[\frac{\partial E(\boldsymbol{v}^{(l)}, \boldsymbol{h})}{\partial \boldsymbol{\theta}} \right] - \mathbb{E}_{p(\boldsymbol{v},\boldsymbol{h})} \left[\frac{\partial E(\boldsymbol{v}, \boldsymbol{h})}{\partial \boldsymbol{\theta}} \right] \tag{3}$$

It is noteworthy to mention that it is formed by the difference between two expectation of the same quantity. The expectation on the left is often referred to as the *data driven* term since it is an expectation over the distribution of the hidden units conditionally on a visible sample from the data distribution. The expectation on the right (minus sign) is referred to as the *model driven* term since it is an expectation over the joint distribution of the model. Unfortunately this quantity intractable because of the model driven term which involves a sum over all the possible configurations of the hidden (alternatively visible) units in order to compute the partition function (REF).

A training algorithm called Contrastive divergence (CD) [Hinton 2002] rely on an approximation of the model driven term by running a k-step Gibbs chain to obtain a sample $\boldsymbol{v}^{(l,k)}$

$$-\frac{\partial \ln \left[p(\boldsymbol{v}^{(l)}|\boldsymbol{\theta}) \right]}{\partial \boldsymbol{\theta}} \approx \mathbb{E}_{p(\boldsymbol{h}|\boldsymbol{v}^{(l)})} \left[\frac{\partial E(\boldsymbol{v}^{(l)}, \boldsymbol{h})}{\partial \boldsymbol{\theta}} \right] - \mathbb{E}_{p(\boldsymbol{h}|\boldsymbol{v}^{(l,k)})} \left[\frac{\partial E(\boldsymbol{v}^{(l,k)}, \boldsymbol{h})}{\partial \boldsymbol{\theta}} \right] \tag{4}$$

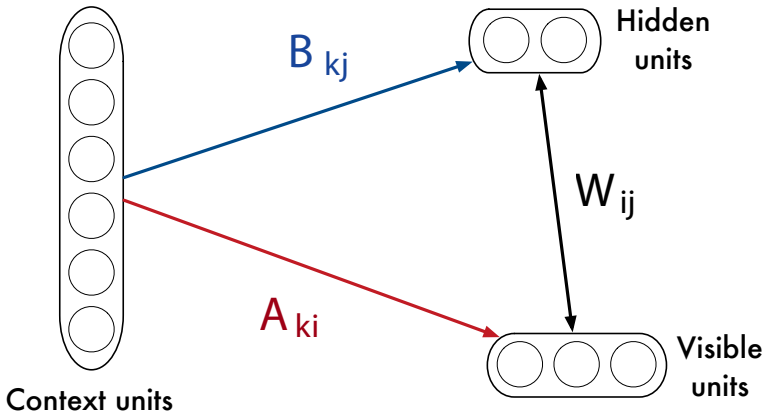

Fig. 3. *Conditional RBM* adds a layer of context units to the standard RBM architectures. Those context units linearly modify the bias of both visible and hidden units.

Running a Gibbs sampling chain consists in alternatively sampling the hidden units knowing the visible units and the visible knowing the hidden (5).

$$p(v_i = 1|\boldsymbol{h}) = \sigma\left(a_i + \sum_j W_{ij} h_j\right) \tag{5}$$

$$p(h_j = 1|\boldsymbol{v}) = \sigma\left(b_j + \sum_i W_{ij} v_i\right) \tag{6}$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the *sigmoid* function. Note that sampling from the marginal distribution is easy since visible units (respectively hidden units) are independent from each others. Hence, knowing the hidden units, all the visible units can be sampled in one step. This allows for a fast implementation of the samplings through matrix calculus, known as *block sampling*. It has been proved [Bengio 2009] that the samples we obtain after an infinite number of iteration will be drawn from the joint distribution of the visible and hidden units of our model. Another approximation consists in starting the Gibbs chain from the sample $\boldsymbol{v}^{(l)}$, which increases the convergence of the chain, and limiting the number of sampling steps to a fixed number K. After evaluating the statistics for the distribution ($\boldsymbol{h} \sim p(\boldsymbol{h}|\boldsymbol{v}^{(\boldsymbol{l})})$ and $\boldsymbol{v}^{(\boldsymbol{l},\boldsymbol{k})}$ and $\boldsymbol{h} \sim p(\boldsymbol{h}|\boldsymbol{v}^{(\boldsymbol{l})})$ from the Gibbs sampling chain), the parameters can be updated. The whole algorithm is called Contrastive Divergence-K (CD-K). In a *RBM* the update rules are given by

$$\Delta W_{ij} = < v_i h_j >_{data} - < v_i h_j >_{model} \tag{7}$$

$$\Delta a_i = < v_i >_{data} - < v_i >_{model} \tag{8}$$

$$\Delta b_j = < h_j >_{data} - < h_j >_{model} \tag{9}$$

2.2. Conditional RBM

The Conditional *RBM* model [Taylor 2009] is an extension of the Restricted-Boltzmann Machine (RBM) in which dynamic biases are added to the static biases of the visible and hidden units (respectively $\boldsymbol{a}$ and $\boldsymbol{b}$). This dynamic bias linearly depends on a set of units called context units ($\boldsymbol{x}$). Those context units can be used to represent the influence on the visible units of any other factor. Typically, when trying to model time

series, those context units can be used to model the influence of the recent past frames over the current frame. Hence, if we now consider that the visible units are given for a certain time frame $\boldsymbol{v}(t)$, context units can be defined as the concatenation of the N last time frames $\boldsymbol{x}(t) = \left(v_1^{(t)}, ..., v_m^{(t)}, ..., v_1^{(t-N)} ..., v_m^{(t-N)}\right)$, where N denotes the temporal order of the model. The energy function of the Conditional RBM is given by

$$E(\boldsymbol{v}(t), \boldsymbol{h}(t)|\boldsymbol{x}(t)) = -\sum_i \hat{a}_i(t) v_i(t) - \sum_{ij} W_{ij} v_i(t) h_j(t) - \sum_j \hat{b}_j(t) h_j(t) \tag{10}$$

where the biases are defined by

$$\hat{a}_i(t) = a_i + \sum_k A_{ki} x_k(t)$$

$$\hat{b}_j(t) = b_j + \sum_k B_{kj} x_k(t)$$

We will refer to the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ as *auto-regressive matrices*.

This model can be trained using CD, since the marginal probabilities of visible and hidden units are the same as the *RBM*, simply replacing the static biases by dynamics ones. The following update rules are unchanged for $\boldsymbol{W}$, $\boldsymbol{a}$ and $\boldsymbol{b}$, and are the following for the auto-regressive matrices

$$\Delta A_{ik} = < v_i x_k >_{data} - < v_i x_k >_{model} \tag{11}$$

$$\Delta B_{jk} = < h_j x_k >_{data} - < h_j x_k >_{model} \tag{12}$$

$$\tag{13}$$

2.3. Generative models

Once trained on a dataset using CD, those models represent a probability distribution p. If correctly trained, this distribution is supposed to be close (in the sense of the Kullback-Leibler divergence [Hinton 2002]) to the underlying probability distribution of the data. Then, by sampling from p, we are able to reproduce data *similar* to the one observed in the training dataset. This sampling process is called the generative step.

Sampling from *RBM*-based models is not straightforward.

Since the partition function is intractable, the marginal probability of the hidden and visible units cannot be sampled. The generation process can be described as follow. After randomly setting the visible units vector (for each index i, $\boldsymbol{v}_i \sim \mathcal{U}(0,1)$), alternate Gibbs sampling is performed in order to approximate the joint distribution of the hidden and visible units conditionally on the context units. Given the context units, one step of alternate Gibbs sampling consists in sampling the hidden units knowing the visible, then sampling the visible units knowing the hidden. In theory the visible sample obtained is from the model distribution after an infinite number of steps. If theoretically an infinite number of steps is necessary, 20 to 100 steps are typically used in practice.

2.4. Approximations

In practice, several approximations are made both during the training and generative phases. The CD-K algorithm and its approximation, which consist in performing only a limited number of sampling steps and for the training phase starting the Gibbs chain with a sample from the training set, has already been discussed.

2.4.1. Mean-field values. When evaluating the data and model-driven terms in equation 4, the two terms are the expectation of the same quantity under different distribution. For the data-driven term, this expectation is analytically derivable through the

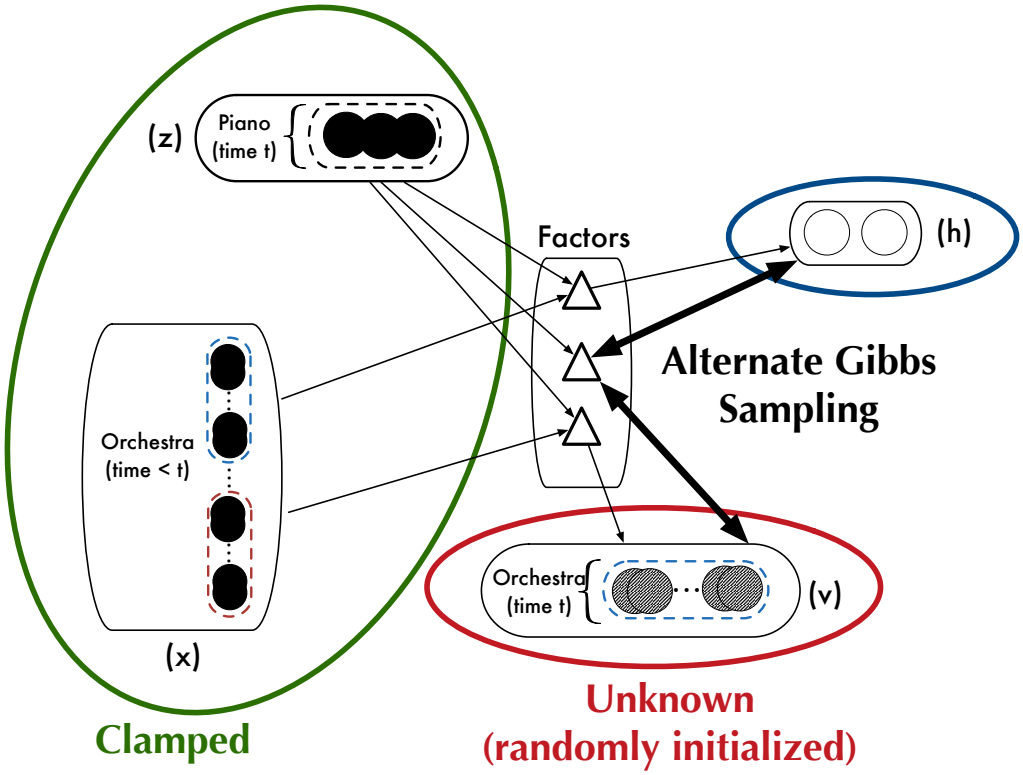

Fig. 4. *Sampling in a FGCRBM*. Context and Features units are respectively clamped to the last ($t-1$ to $t-N$) orchestral frames and the current (t) piano frame. Visible units are randomly initialized. Then, several Gibbs sampling step are performed, in our case 40.

marginal probability of the hidden units, and ($\mathbb{E}[\boldsymbol{h}] = \sigma(\frac{1}{\boldsymbol{b}+\sum_i W_{ij}v_i})$). For the model-driven term, expectation cannot be easily calculated since we do not have access to the joint probability. Approximating the expectation by running N Gibbs chains and computing an estimator would be too time consuming. Hence, a first idea is to "approximate" the expectation over $p(h|v^{(l,k)})$ by the value of the hidden vector obtained in the last step of the Gibbs chain ($\boldsymbol{h} \sim p(\boldsymbol{v^{(l,k)}})$).

Following the mathematical definition of an *RBM*, the visible and hidden units must be sampled (thus equal to binaries values) from the marginal probabilities at each step of the Gibbs chain. However, it is possible to speed up the convergence of the algorithm by replacing the sampling value by the probability itself when updating the hidden vector at the last step k of the Gibbs chain : $\boldsymbol{h} = p(\boldsymbol{v^{(l,k)}})$. Doing this, we get rid of the sampling noise since the mean value of the marginal distribution is picked [Hinton 2010]. Note that it is also possible to do so with the visible units at each time step, but it has been shown that it leads to a worse density estimation, which is not a problem when the algorithm is used only as a pre-training step before

2.4.2. Number of sampling steps. It is interesting to note that $K = 1$ in the CD-K algorithm during the training phase. This is because the objective is to modify the energy

of our model in order to increase the likelihood of the training set under the model distribution. It has been shown that even a small number of sampling steps guarantees to increase a lower bound over the likelihood [Bengio 2009]. When generating data, we seek to obtain a sample from the distribution of the model. Therefore, a larger number of Gibbs sampling steps need to be performed in order to effectively obtain a good approximation.

3. PROJECTIVE ORCHESTRATION

In this section, we present how to perform automatic projective orchestration using a cRBM. In particular, we detail which kind of data representation we used and the modelling function of the different units. A objective criterion of the performances of a model is necessary. Not only to compare the different systems, but also to find the best set of hyper-parameter for a given model. To our best knowledge, there is no quantitative evaluation framework for automatic projective orchestration. We propose here a first attempt in order to fill this gap by defining a projective orchestration inference task. Given a test database composed by piano scores and orchestrations proposed by experts (famous composer), it consists at each time frame t to generate an orchestral vector $\hat{Orch}(t)$ knowing the piano frame $Piano(t)$ and the recent past of sequence of orchestral vectors $Orch(t-1), ...Orch(t-N)$. The predicted vector $\hat{Orch}(t)$ is then compared to the ground-truth $Orch(t)$ via an accuracy measure. The database used and results obtained with the ***cRBM*** are then presented in the two last sections.

3.1. Formalization

Conditional models allow to generate sequences of data under a certain context. Projective orchestration can be seen as producing an orchestral score, conditionally on a piano score. In order to guarantee a form of temporal continuity in the orchestration, the recent past of the orchestral sequence is added in the contextual information.

3.1.1. Data representation. Before being able to train the previously introduced models, the first step is to choose an adapted representation for the piano and orchestra information. The *pianoroll* representation is often used to model sequences of symbolic music. this is a binary matrix $Pianoroll(p,t)$ of dimension $N_T \times N_P$. For a single instrument N_T is the temporal length of the musical sequence, and N_P the number of pitches potentially played by the instrument. Hence, for a certain time quantisation $Pianoroll(p,t)$ specifies if a pitch p is played at time frame t. The dynamics are ignored and each time frame can be represented by a binary vector $Pianoroll(t) \in \{01\}^{N_P}$. As depicted in figure 5, this representation can be easily extended to an orchestra composed by N_I instruments by simply concatenating the *pianorolls* of each instrument over the pitch dimension.

In the proposed system, two different *pianorolls* are used to represent the piano and the orchestra score. For the piano score, $N_P = 88$ since this is the number of keys for a standard grand piano. The orchestra representation require more attention. Each instrument has a different playing range. Hence, the size of their pitch dimension is different for each. In practice, we limited it to the pitch observed in the training dataset. The observed range is denoted $R(i)$ for each instrument i in an set of instrument $I = \text{violin}, \text{viola}, ...$ If the training base is sufficiently large, every note that can be played by each instrument will be seen at least once. This restriction acts as an important constraint over the playing range of each instrument by preventing the system to assign absurd notes to the different instrument (for instance a low register note to a piccolo).

Note that we follow the usual orchestral simplifications used when writing orchestral scores by grouping together all the instruments of a same section. For instance,

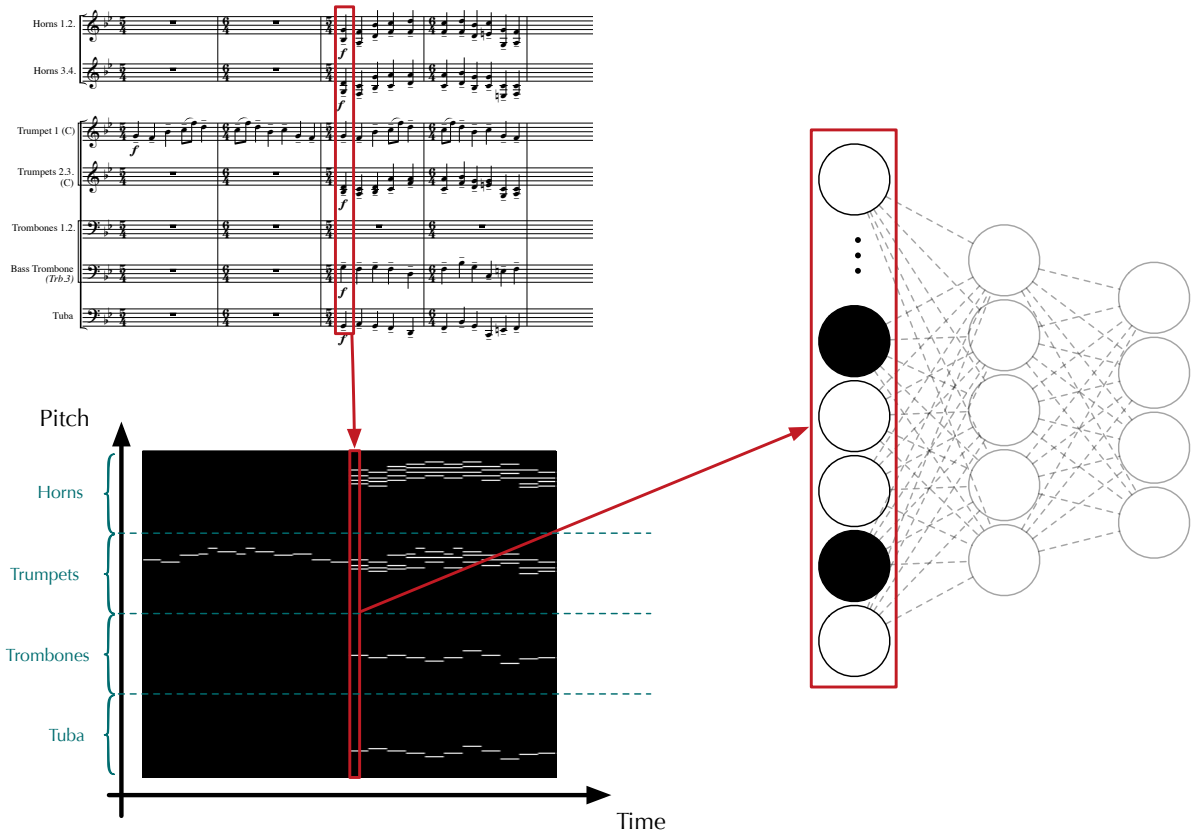

Fig. 5. The successive visible units of a neural network could be the successive temporal frames of a *pianoroll*. A *pianoroll* is a representation of musical events, discrete on both the frequency (pitch) and the time (frames) scales. For a single instrument, a pitch p at time t can be either played or not, which is represented by a one or zero in the *pianoroll*. This definition is extended to an orchestra by simply concatenating the *pianorolls* of each instruments along the pitch dimension. One can see on the figure that the same instruments are grouped together event if they don't play the same thing. For instance, trumpets 1, 2, 3 and 4 are grouped under one trumpet part, which then contains 4 voices chords.

Index	Instrument
1	Violin
2	Viola
3	Cello
4	Double-bass
5	Harp
6	Timpani
7	Trumpet

Index	Instrument
8	Trombone
9	Tuba
10	French horn
11	Oboe
12	Bassoon
13	Clarinet
14	Flute

Fig. 6. Indexation used for the orchestra instruments

the section *violin 1*, composed by many instrumentalists (10 or more), is written as a single part. Eventually, following the notations we just introduced, the dimension of an orchestra *pianoroll* composed by the instrument set I are given by $N_T \times \sum_{i \in I} R(i)$.

In our framework we chose 14 instruments indexed by

3.1.2. Modeling orchestral sequences. For a piece of music, we define two sequences of vectors $Orch(t)$ and $Piano(t)$ with t in $[|1, N_T|]$ where N_T is the lenght of the piece. Those are respectively defined by the sequence of column vector from the *pianoroll* representation of the orchestra part and of the piano part.

At each time frame t, the visible units of the cRBM represent the current orchestral vector ($Orch(t)$), conditional units are used to model the influence of the past orchestral vectors $Orch(t-1), ..., Orch(t-N)$ and the influence of the current piano frame ($Piano(t)$) over the visible units. The context units are then defined by the concatenation of the past orchestral frames and the current piano frame $Context(t) = [Piano(t), Orch(t-1), ..., Orch(t-N)]$.

Note that we do not restrain the possible pitches of each instrument to the pitches seen in the piano score at each specific time frame. Indeed, orchestrating a piano score cannot be reduced to the simple repartition between the different instruments of the same notes already written in the piano score. Instead, the harmonic structure is often enriched by extra notes, ranging from simple octaviations to harmonic expansions (more complex chords with a specific colour). The melody might be doubled by several instruments at the unison, octave or even double octave. + Something about the phrasing ? A wind instru can't play the same phrasing as a pianist. Reciprocally, a pianist can't tremolo

3.2. Evaluation

Building a quantitative evaluation framework for generative models is rarely straightforward, especially since computing the likelihood of a test sample is intractable in the models we used. A common practice is to define an auxiliary task close to the generative objective but easier to evaluate. Typically, in automatic music generation, the models are evaluated through a predictive task based on a frame-level accuracy measure [Liu and Ramakrishnan 2014; Boulanger-Lewandowski et al. 2012; Lavrenko and Pickens 2003]. We introduce an extension of this evaluation framework to an orchestral context through a task called *frame-level orchestral inference*. This framework heavily relies on the previous works in automatic music generation and we discovered a bias in the measurement due to the temporal scale used (frame-level). We then propose a new evaluation framework and present the results of our system in a newly introduced *event-level* framework.

3.2.1. Frame-level accuracy. The frame-level accuracy of a model is defined as the mean value of the accuracies measured for each time frame of a testing set. For each time frame, we try to predict the orchestral frame $\hat{Orch}(t)$ knowing the recent past $Orch(t-1), ..., Orch(t-N)$ and the piano frame $Piano(t)$ and compare it to the original frame $Orch(t)$. The accuracy measure the difference between the predicted and original frames [Boulanger-Lewandowski et al. 2012; Liu and Ramakrishnan 2014]

$$\text{Accuracy} = \frac{TP(t)}{TP(t) + FP(t) + FN(t)} \tag{14}$$

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. In the case this probability is intractable, one should sample many predicted frames for each time frame and compute the mean value of those samples.

Model	Orchestral Frame-level (%)
Random	0.51
Repeat	93.25
CRBM	45.07
FGCRBM	2.05

Fig. 7. Frame-level accuracy

Model	Orchestral Event-level (%)
Random	0.50
Repeat	93.25
CRBM	27.78
FGCRBM	

Fig. 8. Event-level accuracy

3.2.2. Event-level accuracy. A major flaw in the frame-level accuracy measure is its dependence to the rhythmic quantization chosen. Indeed, when the quantization become too small, it becomes highly probable that a frame is simply repeated at the next time frame. Hence, the best predictive model is simply a model which predict that the next time frame is the same as the frame $t-1$. This is not a desirable behaviour, and we propose an evaluation based on an event-level to address that issue.

A musical event is defined as a change in the orchestral score, either a note being switched on or off. The predictive event-level accuracy measure relies on the same accuracy measure previously defined (14). The only difference is that it occurs at each new event instead of each new time frame. Since the task is slightly different, the training phase has been consequently changed so that a model is trained only on new event frames.

3.3. Database

We used a parallel database of piano scores and their orchestration by famous composers. The database consists of 76 *XML* files. Given the complexity of the distribution we wanted to model and the reduced size of the database we have accessed to, we decided to keep as a test dataset only the last half of one track from our database. Hence 75 and a half files were used to train our model. We chose to do so in order to have the best generation ability. The contrastive-divergence algorithm has been applied on mini-batches of size 100 during the training phase. Thus we obtained 335 mini-batches for the frame-level measure, and 89 for the event-level measure. For each instrument, the pitch range is reduced to the *tessitura* observed in the training dataset (). We used a rhythmic quantization of 8 frames per beat.

3.4. Results

We evaluated the *CRBM* and *FGCRBM* previously presented Section 2 in both the frame-level and event-level frameworks. Those two models are compared to a random prediction of each frame and to a simple model that outputs the previous frame as the current frame prediction (i.e. a untrained 1 order linear predictor). The results are presented in the following tables Figure 8 on page 12 and Figure **??** on page **??**.

4. LIVE ORCHESTRAL PIANO

5. CONCLUSION AND FUTURE WORKS

ACKNOWLEDGMENTS

The authors would like to thank

REFERENCES

Yoshua Bengio. 2009. Learning deep architectures for AI. *Foundations and trends in Machine Learning* 2, 1 (2009), 1–127.

Hector Berlioz. 1844. *Grand traité d'instrumentation et d'orchestration modernes*. Schonenberger.

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

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

Asja Fischer and Christian Igel. 2014. Training restricted Boltzmann machines: An introduction. *Pattern Recognition* 47, 1 (2014), 25–39.

Geoffrey Hinton. 2010. A practical guide to training restricted Boltzmann machines. *Momentum* 9, 1 (2010), 926.

Geoffrey E Hinton. 2002. Training products of experts by minimizing contrastive divergence. *Neural computation* 14, 8 (2002), 1771–1800.

Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. 2006. A Fast Learning Algorithm for Deep Belief Nets. *Neural Comput.* 18, 7 (July 2006), 1527–1554. DOI:http://dx.doi.org/10.1162/neco.2006.18.7.1527

Charles Koechlin. 1941. *Traité de l'orchestration*. Éditions Max Eschig.

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

I.-Ting Liu and Bhiksha Ramakrishnan. 2014. Bach in 2014: Music Composition with Recurrent Neural Network. *CoRR* abs/1412.3191 (2014). http://arxiv.org/abs/1412.3191

Stephen McAdams. 2013. Timbre as a structuring force in music. In *Proceedings of Meetings on Acoustics*, Vol. 19. Acoustical Society of America, 035050.

Graham William Taylor. 2009. *Composable, distributed-state models for high-dimensional time series*. Ph.D. Dissertation. University of Toronto.

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

Charlotte Truchet and Gerard Assayag. 2011. *Constraint Programming in Music*. Wiley.