Université de Montréal

Espaces de timbre générés par des réseaux profonds convolutionnels

par
Simon Lemieux

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des arts et des sciences
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en informatique

août, 2011

Université de Montréal
Faculté des arts et des sciences

Ce mémoire intitulé:

Espaces de timbre générés par des réseaux profonds convolutionnels

présenté par:

Simon Lemieux

a été évalué par un jury composé des personnes suivantes:

Yoshua Bengio,	président-rapporteur
Douglas Eck,	directeur de recherche
Gena Hahn,	membre du jury

RÉSUMÉ

Il est avant-tout question, dans ce mémoire, de la modélisation du timbre grâce à des algorithmes d'apprentissage machine. Plus précisément, nous avons essayé de construire un espace de timbre en extrayant des caractéristiques du son à l'aide de machines de Boltzmann convolutionnelles profondes.

Nous présentons d'abord un survol de l'apprentissage machine, avec emphase sur les machines de Boltzmann convolutionelles ainsi que les modèles dont elles sont dérivées. Nous présentons aussi un aperçu de la littérature concernant les espaces de timbre, et mettons en évidence quelque-unes de leurs limitations, dont le nombre limité de sons utilisés pour les construire. Pour pallier à ce problème, nous avons mis en place un outil nous permettant de générer des sons à volonté. Le système utilise à sa base des *plug-ins* qu'on peut combiner et dont on peut changer les paramètres pour créer une gamme virtuellement infinie de sons. Nous l'utilisons pour créer une gigantesque base de donnée de timbres générés aléatoirement constituée de vrais instruments et d'instruments synthétiques. Nous entrainons ensuite les machines de Boltzmann convolutionnelles profondes de façon non-supervisée sur ces timbres, et utilisons l'espace des caractéristiques produites comme espace de timbre.

L'espace de timbre ainsi obtenu est meilleur qu'un espace semblable construit à l'aide de MFCC. Il est meilleur dans le sens où la distance entre deux timbres dans cet espace est plus semblable à celle perçue par un humain. Cependant, nous sommes encore loin d'atteindre les mêmes capacités qu'un humain. Nous proposons d'ailleurs quelques pistes d'amélioration pour s'en approcher.

Mots clés: espace de timbre, machine de Boltzmann convolutionnelle, apprentissage machine, génération automatique de timbres, architectures profondes, extraction de caractéristiques.

ABSTRACT

This thesis presents a novel way of modelling timbre using machine learning algorithms. More precisely, we have attempted to build a timbre space by extracting audio features using deep-convolutional Boltzmann machines.

We first present an overview of machine learning with an emphasis on convolutional Boltzmann machines as well as models from which they are derived. We also present a summary of the literature relevant to timbre spaces and highlight their limitations, such as the small number of timbres used to build them. To address this problem, we have developed a sound generation tool that can generate as many sounds as we wish. At the system's core are plug-ins that are parameterizable and that we can combine to create a virtually infinite range of sounds. We use it to build a massive randomly generated timbre dataset that is made up of real and synthesized instruments. We then train deep-convolutional Boltzmann machines on those timbres in an unsupervised way and use the produced feature space as a timbre space.

The timbre space we obtain is a better space than a similar space built using MFCCs. We consider it as better in the sense that the distance between two timbres in that space is more similar to the one perceived by a human listener. However, we are far from reaching the performance of a human. We finish by proposing possible improvements that could be tried to close our performance gap.

Keywords : space timbre, convolutional boltzmann machines, machine learning, automatic timbre generation, deep architectures, feature extraction.

TABLE DES MATIÈRES

RÉSUMÉ . . . . . **v**

ABSTRACT . . . . . **vii**

TABLE DES MATIÈRES . . . . . **ix**

LISTE DES TABLEAUX . . . . . **xiii**

LISTE DES FIGURES . . . . . **xv**

LISTE DES SIGLES . . . . . **xvii**

NOTATION . . . . . **xix**

REMERCIEMENTS . . . . . **xxi**

CHAPITRE 1 : INTRODUCTION **1**

CHAPITRE 2 : APPRENTISSAGE MACHINE **3**

2.1 Concepts généraux 3

2.2 Types d'apprentissages 3

2.2.1 Apprentissage supervisé 3

2.2.1.1 Régression 4

2.2.1.2 Classification 4

2.2.2 Apprentissage non-supervisé 4

2.2.2.1 Estimation de densité 4

2.2.2.2 Groupage 5

2.2.2.3 Extraction de caractéristiques et réduction de dimensionnalité 5

2.2.3 Apprentissage semi-supervisé 7

2.2.4 Modèles paramétriques et non-paramétriques 7

2.3 Apprentissage par minimisation du risque empirique 7

2.3.0.1 Descente de gradient 9

2.4 Généralisation 10
2.5 Approfondissement de quelques modèles 13
2.5.1 Machines de Boltzmann 13
2.5.2 Machines de Boltzmann avec unités cachées 15
2.5.3 Machines de Boltzmann restreintes 17
2.5.4 Machines de Boltzmann restreintes convolutionnelles 19

CHAPITRE 3 : LE TIMBRE 23
3.1 Définition 23
3.2 Espaces de timbre 23
3.2.1 MFCC 25

CHAPITRE 4 : BASE DE DONNÉES 27
4.1 Motivation 27
4.2 Moteur de génération de timbres 27
4.2.1 MIDI 27
4.2.2 Audio units 28
4.2.2.1 Synthétiseurs 28
4.2.2.2 Effets 29
4.2.2.3 Instrument 29
4.2.2.4 Hôte 29
4.3 Génération aléatoire de timbres 30
4.4 Résumé 31

CHAPITRE 5 : EXPÉRIENCES 33
5.1 Base de données 33
5.1.1 Ensemble de validation 34
5.1.2 Ensembles de test 34
5.2 Prétraitement des données 34
5.3 Apprentissage 36
5.3.1 Évaluation du modèle 37
5.4 Résultats et analyse 38

CHAPITRE 6 : CONCLUSION **45**

BIBLIOGRAPHIE **47**

LISTE DES TABLEAUX

5.I Moyennes sur les sujets des τ de Kendall des différents modèles . . . 41

LISTE DES FIGURES

2.1 Algorithme des *k*-moyennes. 6
2.2 Effet du taux d'apprentissage. 11
2.3 Régression polynomiale avec différents degrés de polynômes. 12
2.4 Une couche de CRBM. 20
2.5 *Probabilistic max-pooling* pour $c = 2$. 22

4.1 Génération de timbres aléatoires. 31

5.1 Sorties d'un DCRBM. 39
5.2 τ de Kendall entre différents sujets et modèles. Pâle = meilleur. . . . 40
5.3 τ de Kendall entre les sujets et les modèles. Pâle = meilleur. 42

LISTE DES SIGLES

API	Application programming interface
CRBM	Machine de Boltzmann restreinte convolutionnelle
MDS	Multidimensionalscaling
MFCC	Mel-frequency cepstral coefficients
MIDI	Musical instrument digital interface
PCA	Analyse en composantes principales (principal component analysis)
PMSC	Principal Mel-Spectrum Components
RBM	Machine de Boltzmann restreinte (restricted Boltzman nmachine)

NOTATION

Soient $a = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}$ un vecteur, $w = \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,m} \\ w_{2,1} & w_{2,2} & \dots & w_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n,1} & w_{n,2} & \dots & w_{n,m} \end{bmatrix}$ et w' des matrices, t un tenseur à 3 dimensions (qu'on interprétera comme une liste de matrices) u un tenseur à 4 dimensions (qu'on interprétera comme une liste de tenseurs à 3 dimensions)

a_i	i-ième élément de a
$w_{i,j}$	j-ième élément de la i-ième ligne de w
t_k	k-ième matrice de t
$t_{k,i,j}$	j-ième élément de la i-ième ligne de la k-ième matrice de t
u_l	l-ième tenseur de dimension 3 de u
$u_{l,k}$	k-ième matrice du l-ième tenseur de dimension 3 de u
$u_{l,k,i,j}$	j-ième élément de la i-ième ligne de la k-ième matrice du l-ième tenseur de dimension 3 de u
$\tilde{w}$	$\begin{bmatrix} w_{n,m} & \dots & w_{n,2} & w_{n,1} \\ \vdots & \ddots & \vdots & \vdots \\ w_{2,m} & \dots & w_{2,2} & w_{2,1} \\ w_{1,m} & \dots & w_{1,2} & w_{1,1} \end{bmatrix}$ (matrice basculée dans les deux sens)
$w \bullet w'$	$\sum_{i,j} w_{i,j} w'_{i,j}$ (produit élément par élément suivi de la somme)
$w * w'$	convolution entre les deux matrices. Le type de convolution dépend du contexte (valide ou pleine).

$\mathscr{D}$	ensemble d'entrainement
$\mathscr{D}_{test}$	ensemble de test
$\mathscr{D}_{valid}$	ensemble de validation
$x^{(i)}$	*i*-ième entrée d'un ensemble d'apprentissage
$y^{(i)}$	*i*-ième sortie d'un ensemble d'apprentissage
$\mathscr{P}$	loi de probabilité de laquelle sont tirées les données
$\mathbb{E}_{\mathscr{X}}$	espérance sur l'ensemble $\mathscr{X}$ ou sur la distribution de probabilités $\mathscr{X}$, selon le cas

REMERCIEMENTS

Je remercie évidemment Douglas Eck pour m'avoir donné l'opportunité de rédiger le présent mémoire.

Je tiens aussi à remercier mes amis et ma famille pour leur appui et leurs encouragements.

Finalement je remercie particulièrement François Maillet, Philippe Hamel, Stanislas Lauly, Sean Wood, Émilie Gosselin, ainsi que tous les membres du Gamme, du Lisa et du Brams pour leur aide et leur compagnie.

CHAPITRE 1

INTRODUCTION

La musique occupe une place dans la vie de chacun d'entre nous. Parfois de façon consciente, et parfois inconsciemment, nous réussissons à en analyser plusieurs aspects très facilement, tâche pourtant souvent difficile, voir impossible, pour un ordinateur. Par exemple, en écoutant une pièce de musique, on peut reconnaitre son style, y distinguer des instruments, battre la mesure, etc. Ces tâches, pourtant triviales pour un être humain, sont extrêmement complexes pour une machine. Cependant, des solutions de plus en plus performantes sont peu à peu découvertes, beaucoup utilisant l'apprentissage machine, un outil très efficace pour ce genre de problème. Dans le présent mémoire, nous nous attarderons à un problème en particulier : celui de l'espace de timbre.

Pour résumer simplement, un espace de timbre est un espace dans lequel chaque point représente un timbre donné (pensons un instrument de musique donné), et où la distance entre deux timbres correspond à leur dissimilarité comme perçue par un être humain. Deux timbres "semblables" (par exemple issus de deux cuivres) seront donc proches dans cet espace ; deux timbres "différents" (par exemple les timbres issus d'un piano et d'une trompette) seront loin. Pouvoir représenter les timbres dans un tel espace a plusieurs intérêts. Avant tout, cela permet d'avoir une base solide pour organiser différents instruments de musique de façon cohérente et utile. Typiquement, dans une banque d'instruments, ceux-ci sont étiquetés à la main avec plus ou moins de détail ; il peut alors être fastidieux de trouver un instrument avec une sonorité précise. Avec un espace de timbre, on peut par exemple retrouver extrêmement facilement tous les instruments dont la sonorité est proche d'un instrument donné. Ceci est d'autant plus pertinent depuis que nous avons des instruments électroniques. Nous ne sommes plus limités dans le nombre de timbres possibles comme autrefois.

Beaucoup de recherche a déjà été effectuée dans le domaine. Cependant, la plupart des expériences ont à notre avis quelques lacunes. Par exemple, la plupart n'utilisent qu'un nombre très limité de sons, ou alors n'utilisent que des sons artificiels (car plus facile à générer en grand nombre). C'est pour pallier à ces limitations que nous avons conçu un outil qui nous permet de générer des banques de sons énormes pouvant conte-

nir potentiellement n'importe quel son. Nous avons ensuite décidé d'appliquer comme modèle des machines de Boltzmann restreintes convolutionnelles, agencées de manière profonde, pour construire un espace de timbre de façon non-supervisée. Chacun des aspects est décrit en détail dans les prochains chapitres.

Premièrement nous faisons un survol de l'apprentissage machine au chapitre 2, et décrivons avec plus de détails le modèle que nous avons implémenté et utilisé pour nos expériences à la section 2.5. Puis, au chapitre 3, nous donnons un aperçu de ce qui a déjà été fait dans le domaine des espaces de timbres. Ensuite, nous décrivons également les outils que nous avons implémentés pour générer notre base de données au chapitre 4. Finalement, nous présentons nos expériences au chapitre 5 et analysons leurs résultats dans la section 5.4, tout en proposant des pistes d'amélioration.

CHAPITRE 2

APPRENTISSAGE MACHINE

Cette section se veut un survol de l'apprentissage machine, avec emphase sur les éléments relatifs aux modèles qui nous ont intéressés ici. Nous recommandons vivement au lecteur curieux sur l'apprentissage machine *Pattern recognition and machine learning[7]*, ouvrage qui nous a été utile pour la rédaction du présent chapitre, mais surtout comme référence tout au long de nos études en apprentissage machine.

2.1 Concepts généraux

L'apprentissage machine, c'est d'abord trouver des régularités dans un ensemble de données $\mathscr{D}$, qu'on appelle ensemble d'apprentissage. On fait habituellement l'hypothèse que les données dans $\mathscr{D}$ sont des échantillons indépendants provenant d'une certaine loi de probabilités $\mathscr{P}$. Ensuite, on veut pouvoir utiliser les connaissances "apprises" sur de nouveaux exemples issus de la même distribution, mais qui n'étaient pas présent dans l'ensemble $\mathscr{D}$. Cette habileté de pouvoir généraliser à de nouvelles donnés constitue le principal défi de l'apprentissage machine. Nous élaborerons sur ce dernier aspect à la section 2.4.

2.2 Types d'apprentissages

À des fins de simplicité, nous ne présenterons ici que deux types principaux d'apprentissage machine, à savoir supervisé et non-supervisé (ainsi que le mélange des deux qu'est l'apprentissage semi-supervisé). Finalement nous verrons la différence entre modèle paramétrique et non-paramétrique.

2.2.1 Apprentissage supervisé

Dans un contexte d'apprentissage supervisé, l'ensemble d'apprentissage est constitué de couples, i.e. $\mathscr{D} = \left\{(x^{(i)}, y^{(i)}) \mid i = 1 \dots N\right\}$, où $x^{(i)}$ est une entrée et $y^{(i)}$ sa sortie désirée correspondante. Typiquement, ce qu'on voudra apprendre dans ce genre de situa-

tion, c'est une fonction f telle que $f(x)$ est "proche" de y, pour $(x,y) \in \mathscr{D}$.

2.2.1.1 Régression

En régression, $\mathscr{D}$ a la particularité que les $y^{(i)}$ sont des nombres réels. Par exemple, on pourrait vouloir apprendre une fonction qui prédit, en fonction de divers paramètres sur un individu contenus dans $x^{(i)}$, le salaire horaire du même individu $y^{(i)}$.

2.2.1.2 Classification

Pour la classification, on a que les $y^{(i)}$ sont différentes classes, auxquelles on veut associer les $x^{(i)}$ correspondant. Un exemple classique est celui l'où on veut classifier des chiffres écrits à la main, encodés dans une image de petite taille en noir et blanc. Dans ce cas, les $x^{(i)}$ seront les valeurs des pixels d'une image, et $y^{(i)}$ le chiffre que représente l'image (0,1,2... ou 9).

2.2.2 Apprentissage non-supervisé

Dans un cas où nous n'avons que des entrées, et pas de sorties ($\mathscr{D} = \{x^{(i)} | i = 1...N\}$), il y a tout de même quelque chose à apprendre. En fait, toutes sortes de régularités dans les données peuvent être apprises. En voici quelques exemples.

2.2.2.1 Estimation de densité

Une première tâche qui peut être accomplie sur ce genre de données est d'estimer la loi de probabilité de laquelle sont échantillonnés les exemples dans $\mathscr{D}$. Si l'estimation est bonne, elle assignera à des exemples provenant de la même distribution une forte probabilité, et une faible probabilité à des exemples issus d'une autre distribution. Par exemple, on pourrait donner à un modèle seulement des caractères 3 écrits à la main. Si le modèle apprend bien, il assignera une haute probabilité à un nouveau 3, et une faible probabilité à un 4 par exemple.

Certains modèles vont plus loin que simplement apprendre la distribution, ils peuvent aussi générer de nouveaux exemples de la distribution apprise. C'est ce qu'on appelle les modèles génératifs. Dans l'exemple précédent, un bon modèle génératif pourrait générer un nouveau 3 n'étant pas dans $\mathscr{D}$.

2.2.2.2 Groupage

Le groupage (ou *clustering*) consiste à essayer de séparer en groupes les données. Supposons que nous ayons, comme dans l'exemple de la section 2.2.1.2, des images représentant des caractères mais que nous n'ayons pas les $y^{(i)}$, *i.e.* nous ne savons pas à priori quel caractère est représenté par une image donnée. On peut alors imaginer qu'un modèle de groupage associerait tous les 0 au même groupe, les 1 à un autre groupe, etc.

Un exemple d'algorithme simple de groupage est celui des k-moyennes. Supposons qu'on a $x^{(i)} \in \mathbb{R}^n$, on initialise d'abord aléatoirement k points dans $\mathbb{R}^n$ que nous appellerons *moyennes*. On associe ensuite chaque $x^{(i)}$ à la moyenne la plus proche. On change ensuite nos k moyennes pour le centroïde de chaque groupe. On recommence ce simple procédé jusqu'à stabilité, pour obtenir une division en k groupes. L'algorithme est illustré à la figure 2.1.

2.2.2.3 Extraction de caractéristiques et réduction de dimensionnalité

Parfois, lorsque nos entrées sont de trop haute dimensionnalité, il peut arriver qu'on veuille simplement réduire leur taille. Un bon exemple est lorsqu'on veut visualiser ces données ; on les projette alors dans $\mathbb{R}^2$ ou $\mathbb{R}^3$. On peut aussi vouloir réduire la dimensions de données pour faciliter la tâche à un autre algorithme d'apprentissage. Un algorithme classique est l'*analyse en composantes principale*s (PCA) . Cet algorithme projette les données de façon linéaire dans un espace de plus faible dimension, en mettant les axes dans le sens où les données varient le plus. Il existe aussi des modèles plus complexes, comme les auto-encodeurs. Dans un auto-encodeur, les entrées sont "encodées" dans une version plus compressée, puis décodées. On apprend les transformations d'encodage et de décodage de façon à ce que les données décodées ressemblent aux entrées. Les versions encodées des entrées seront ensuite utilisées comme caractéristiques. Parfois, et c'est d'ailleurs le cas dans les expériences que nous avons effectuées, les caractéristiques extraites par un tel algorithme prennent un nouveau sens, souvent plus utile que les données originales.

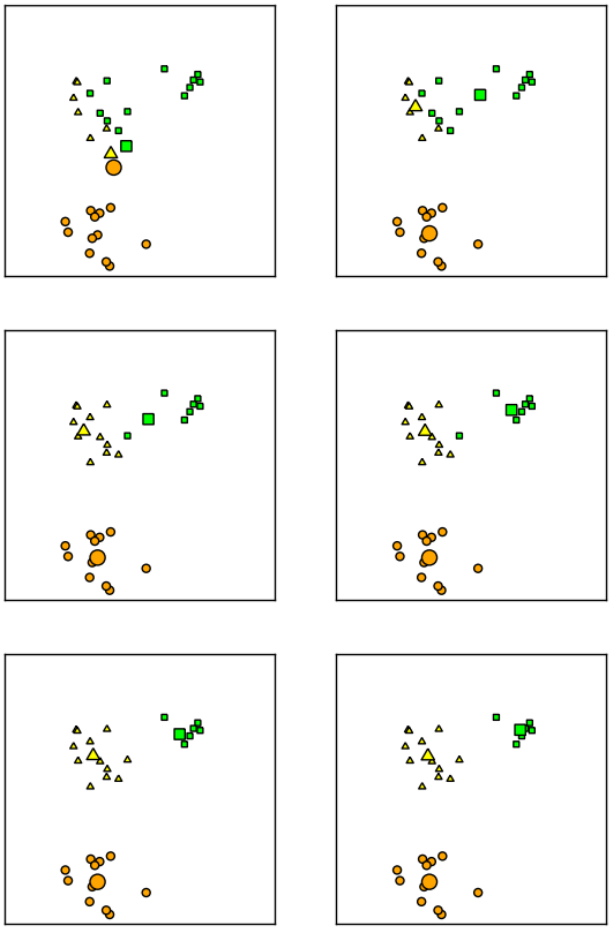

FIGURE 2.1 – Algorithme des *k*-moyennes.
Se lit de gauche à droite et de bas en haut. Colonne de gauche : on associe chaque point à la moyenne la plus proche. Colonne de droite : on ajuste les moyennes des nouvelles classes. (Les moyennes sont représentées par les plus gros points)

2.2.3 Apprentissage semi-supervisé

À mi-chemin entre les apprentissages supervisé et non-supervisé se situe l'apprentissage semi-supervisé. Ce type d'apprentissage est utilisé lorsque seulement certaines entrées $x^{(i)}$ ont des sorties $y^{(i)}$ correspondantes, les autres n'ayant pas de sorties. Un exemple simpliste pourrait être nos caractères écrits à la main, mais où la plupart des exemples ne seraient pas étiquetés (pas de sortie désirée associée). On peut imaginer qu'un algorithme de groupage (non-supervisé) pourrait séparer les entrées en (10) groupes, pour ensuite associer à chaque élément d'un groupe le caractère qui est le plus présent parmi les éléments du même groupe ayant une sortie.

2.2.4 Modèles paramétriques et non-paramétriques

Avant de poursuivre avec la façon dont sont entrainés certains modèles, nous devons distinguer les modèles paramétriques des modèles non-paramétriques, deux classes de modèles d'apprentissage. D'abord, les modèles paramétriques possèdent, comme leur nom l'indique, un nombre de paramètres fixe, qui seront ajustés selon les exemples d'apprentissage. L'important à noter ici, c'est que le nombre de paramètre ne change pas selon le nombre d'exemples. De l'autre côté, il y a les modèles dit non-paramétriques, dont le nombre de paramètres va grandir au fur et à mesure que N, le nombre d'exemples d'apprentissage, augmente. Un exemple typique de modèle non-paramétrique est celui des *k plus proches voisins*. Ce modèle de type supervisé garde en mémoire tous les exemples d'apprentissage avec leur classe, et associe à un nouvel exemple la classe la plus fréquente parmi ses k plus proches voisins (selon une métrique donnée). Pour la suite, nous nous concentrerons uniquement sur des modèles paramétriques.

2.3 Apprentissage par minimisation du risque empirique

Traditionnellement, pour apprendre de façon concrète, on définira un coût C, qui nous indiquera à quel point notre modèle f est "bon" pour un exemple donné. Être "bon" sera défini au cas par cas selon le problème à résoudre et selon les données. Par exemple, en régression, on pourrait penser au coût des moindre carrés

$$C(f,(x,y)) = (y - f(x))^2$$

où (x, y) est un exemple avec sa cible. Ce coût sera petit si notre modèle envoie x près de y et grand sinon. On définit ensuite le *risque réel*, qui est l'espérance de ce coût sur des exemples (avec leur cible) tirés d'une distribution $\mathscr{P}$, i.e.

$$\mathop{\mathbb{E}}_{(x,y)\sim\mathscr{P}}[C(f,(x,y))]$$

C'est ce risque qu'on voudra minimiser. En pratique, nous n'avons généralement accès qu'à un nombre fini d'exemples $\mathscr{D}$, sur lequel on évaluera le *risque empirique*

$$R(f,\mathscr{D}) = \mathop{\mathbb{E}}_{(x,y)\in\mathscr{D}}[C(f,(x,y))]$$

Supposons que nous sommes dans un cas de régression, et que nous voulons utiliser le coût des moindres carrés. Supposons aussi que notre modèle f_θ a des paramètres $\theta = \begin{bmatrix} \theta_1 & \theta_2 & \cdots & \theta_p \end{bmatrix}^T$. On cherchera alors la combinaison de paramètres $\hat{\theta}$ qui minimise cette erreur sur nos données, *i.e.*

$$\hat{\theta} = \arg\min_\theta (R(f_\theta,\mathscr{D})) = \arg\min_\theta \frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)} - f_\theta(x^{(i)})\right)^2$$

Pour une tâche de classification, on prendra plutôt comme erreur

$$C(f,(x,y)) = 1_{y\neq f(x)}$$

i.e. 1 si notre fonction classifie mal l'exemple x et 0 sinon. Au final, on voudra donc optimiser le ratio d'exemples mal classifiés

$$\frac{1}{N}\sum_{i=1}^{N} C\left(f,(x^{(i)},y^{(i)}\right) = \frac{1}{N}\sum_{i=1}^{N} 1_{y^{(i)}\neq f(x^{(i)})}$$

Pour la classification, il s'agit de l'erreur la plus intuitive. Seulement, il n'est pas nécessairement évident d'en dériver un algorithme d'apprentissage. Par exemple, le risque empirique n'est pas différentiable par rapport aux poids du modèle, et donc la méthode décrite dans la section suivante 2.3.0.1 sera inapplicable. Une solution est d'utiliser une autre erreur, qui, bien que moins directe, serve le même but, tout en ayant certaines propriétés plus intéressantes. Par exemple, on utilise souvent, pour la classification, l'erreur

d'entropie croisée, qui a l'avantage d'être différentiable.

Dépendamment du coût, différentes techniques d'optimisation pourront être utilisées. Parfois, on aura directement une solution analytique au problème d'optimisation, par exemple dans le cas d'une régression linéaire avec l'erreur des moindre carrés. Mais pour des modèles complexes, il faudra une technique d'optimisation adaptée au modèle. Dans la section suivante 2.3.0.1, nous présentons un algorithme général d'optimisation utilisé pour beaucoup de modèles.

Nous n'avons donné ici que des exemples d'apprentissage supervisé mais le même traitement s'applique à l'apprentissage non-supervisé.

2.3.0.1 Descente de gradient

Toujours dans un contexte de modèle paramétrique, on veut trouver les paramètres $\theta = \begin{bmatrix} \theta_1 & \theta_2 & \cdots & \theta_p \end{bmatrix}^T$ d'une fonction f_θ qui optimisent un coût C, que l'on supposera différentiable par rapport à tous les paramètres. Comme dans la section précédente, on cherche $\hat{\theta} = \arg\min_\theta (R(f, \mathscr{D}))$. Voici en quoi consiste la technique de descente de gradient. D'abord on initialise les paramètres (θ) au "hasard" (selon le modèle, cette initialisation peut avoir un gros impact sur l'apprentissage, il faut donc la faire judicieusement). Ensuite, on dérive le coût par rapport à chacun des paramètres. Finalement, on modifie les paramètres en fonction des dérivées obtenues. Formellement, soit

$$\nabla R = \begin{bmatrix} \frac{\partial R(f_\theta, \mathscr{D})}{\partial \theta_1} & \frac{\partial R(f_\theta, \mathscr{D})}{\partial \theta_2} & \cdots & \frac{\partial R(f_\theta, \mathscr{D})}{\partial \theta_p} \end{bmatrix}^T$$

le gradient de R par rapport aux paramètres θ. On ajustera les paramètres comme suit

$$\theta \leftarrow \theta - \varepsilon \nabla R$$

où ε est le taux d'apprentissage.

Géométriquement, ce gradient pointe dans le sens le plus "à pic" de la fonction de risque empirique dans l'espace des paramètres. Se déplacer dans le sens contraire du gradient nous permet donc de diminuer cette erreur.

Il ne faut pas choisir le taux d'apprentissage ε trop petit car les paramètres bougeront trop lentement et l'apprentissage prendra trop de temps, mais il ne faut pas non plus le

prend trop gros, dans quel cas rien ne sera appris. L'effet du taux d'apprentissage est illustré à la figure 2.2.

Finalement, une dernière variante de cette technique est la descente de gradient avec mini-batch. Au lieu d'utiliser le gradient du risque empirique $R(\mathscr{D})$, on utilisera $R(\mathscr{D}')$, le gradient du risque évalué sur un sous-ensemble de l'ensemble d'entrainement $\mathscr{D}' \subset \mathscr{D}$. À chaque nouvelle étape, un nouveau sous-ensemble de même taille sera choisi. Il est à noter que cela ajoute une composante aléatoire dans l'apprentissage, puisqu'il dépendra de l'ordre dans lequel les données seront présentées au modèle. D'ailleurs, dans le cas où $|\mathscr{D}'| = 1$, on appelle typiquement la technique, *descente de gradient stochastique*. Dans le cas où $\mathscr{D} = \mathscr{D}'$ (le premier cas discuté), on l'appellera *descente de gradient batch.*

2.4 Généralisation

Le véritable test d'un modèle n'est pas son risque empirique, mais bien le risque évalué sur un ensemble différent $\mathscr{D}_{test}$, dont les exemples sont issus de la même distribution que les exemples d'apprentissage, sans toutefois avoir été utilisés lors de l'entrainement. En effet, apprendre par coeur est aisé pour plusieurs modèles (pensons aux modèles non-paramétriques). On appelle ce phénomène le *surapprentissage*.

Pour éviter le surapprentissage, on peut limiter la *capacité* du modèle. La capacité d'un modèle, c'est à quel point celui-ci peut apprendre des variations complexes dans les données. Si la capacité est trop grande (par rapport au nombre d'exemples d'apprentissage), le modèle pourrait avoir tendance à utiliser toute sa capacité à représenter parfaitement les données d'entrainement, et généralisera moins bien. En revanche, un modèle de trop petite capacité ne pourra pas apprendre assez sur des données complexes. Un compromis doit être effectué lors du choix de la capacité d'un modèle. À la figure 2.3, on présente une régression polynomiale, *i.e.* $f_\theta = \theta_0 + \theta_1 x + \theta_2 x^2 + ... + \theta_m x^m$ avec différents m. Plus le degré du polynôme utilisé pour faire la régression est élevé, plus il est à même de représenter des fonctions complexes, et donc plus sa capacité est élevée.

Dans l'exemple précédent, nous avons vu que modifier le nombre de paramètres d'un modèle peut modifier sa capacité. Il y a d'autres façons de limiter la capacité, qu'on regroupe sous le terme *régularisation*. Plutôt que de réduire le nombre de paramètres,

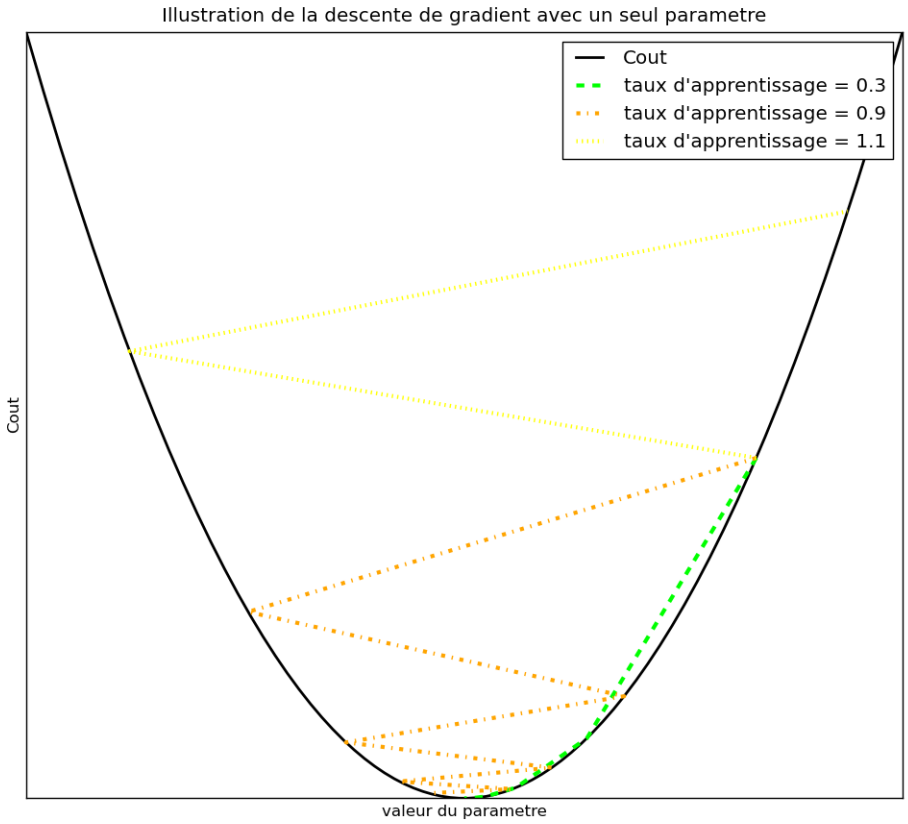

FIGURE 2.2 – Effet du taux d'apprentissage.
L'unique poids à apprendre a été initialisé à 2. L'algorithme de descente de gradient est ensuite appliqué avec différents taux. On constate que le plus grand taux empire le coût. Les deux autres taux l'améliorent à différentes vitesses.

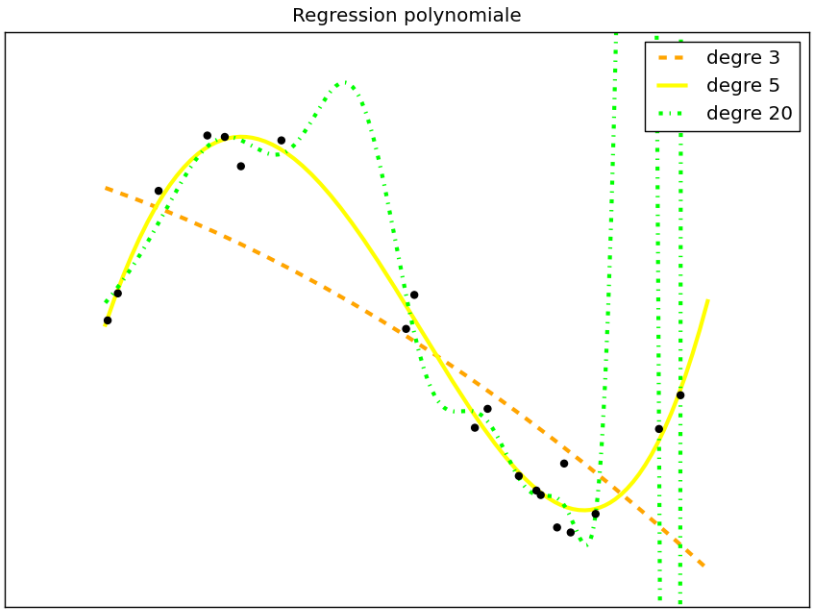

FIGURE 2.3 – Régression polynomiale avec différents degrés de polynômes.
Un modèle avec une trop petite capacité ne représente pas bien les données ; un modèle ayant une trop grande capacité pour le nombre d'exemples tend à apprendre par coeur.

une forme typique de régularisation est de limiter leur taille. On ajoutera par exemple au risque qu'on tente d'optimiser une pénalité pour les poids trop élevés. On pourrait donc avoir comme nouveau risque

$$R(f_\theta, \mathscr{D}) = \mathop{\mathbb{E}}_{(x,y)\in\mathscr{D}}[C(f_\theta,(x,y))] + \lambda \|\theta\|_2^2$$

où $\|\theta\|_2^2 = \theta_0^2 + \theta_1^2 + ... + \theta_m^2$ et λ est un nouvel hyper-paramètre au modèle.

Nous n'avons pas encore parlé d'hyper-paramètres. Les hyper-paramètres sont des paramètres qu'on choisit au début et qui restent fixes durant l'apprentissage (contrairement aux paramètres ajustables). Le degré du polynôme (m) dans l'exemple précédent est donc un hyper-paramètre. Habituellement, on entrainera plusieurs modèles avec différentes combinaisons d'hyper-paramètres et on testera le résultat sur un ensemble de validation $\mathscr{D}_{valid}$ (ensemble distinct des ensembles d'entrainement et de test). La combinaison d'hyper-paramètres qui donnera la plus petite erreur sur l'ensemble de validation sera retenue. Ensuite seulement peut-on tester le meilleur modèle sur l'ensemble de test $\mathscr{D}_{test}$.

2.5 Approfondissement de quelques modèles

Nous aborderons dans cette section les différents modèles desquels est dérivé le modèle que nous avons choisi pour nos expériences. Il sera question de modèles basés sur des machines de Boltzmann. L'article de Scholarpedia sur les machines de Boltzmann, dont G. E. Hinton est responsable[24], a été particulièrement utile pour la rédactions des premières sous-sections qui suivent.

2.5.1 Machines de Boltzmann

Une machine de Boltzmann est composée d'unités $\mu_1, \mu_2, ...\mu_n$ qui peuvent prendre les valeurs 0 ou 1. Chaque paire d'unités μ_i et μ_j est "reliée" par une connexion symétrique $w_{ij} = w_{ji} \in \mathbb{R}$. De plus, à chaque unité μ_i possède un biais b_i associé. On choisira ensuite les valeurs des μ_i les unes après les autres, d'après la formule

$$Prob(\mu_i = 1) = \sigma(b_i + \sum_{j\neq i} w_{ij}\mu_j)$$

jusqu'à ce que les μ_i atteignent un équilibre. Une fois à l'équilibre, on a que la probabilité d'avoir $\mu_i = x_i$ pour $i = 1 \dots n$ (ou $\mu = x$ si $\mu = \begin{bmatrix} \mu_1 & \mu_2 & \dots & \mu_n \end{bmatrix}^T$ et $x = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix}^T$) est égale à

$$P(x) = Prob(\mu = x) = \frac{e^{-E(x)}}{\sum_{x'} e^{-E(x')}}$$

où la sommation du bas est effectuée sur toutes les possibilités de valeurs de μ (donc 2^n termes dans la sommation), et où

$$E(x) = -\sum_i x_i b_i - \sum_{i<j} x_i x_j w_{ij}$$

est appelé l'énergie correspondant à la configuration x.

Ce qu'on voudra faire, avec ce modèle, c'est apprendre des poids (w et b) qui assignent une grande probabilité à des vecteurs x d'un ensemble d'entrainement $\mathscr{D}$. On voudra donc maximiser l'espérance de la probabilité des exemples d'entrainement,

$$\mathop{\mathbb{E}}_{x \in \mathscr{D}} [P(x)]$$

De façon équivalente, on peut minimiser le coût suivant

$$C = -\mathop{\mathbb{E}}_{x \in \mathscr{D}} [\log(P(x))]$$

Pour pouvoir utiliser l'algorithme de descente de gradient, dérivons le coût par rapport aux poids

$$\begin{aligned}
\frac{\partial C}{\partial w_{ij}} &= -\mathop{\mathbb{E}}_{x \in \mathscr{D}} \left[\frac{1}{P(x)} \frac{\partial P(x)}{\partial w_{ij}} \right] \\
&= -\mathop{\mathbb{E}}_{x \in \mathscr{D}} \left[\frac{1}{P(x)} P(x) \left(\sum_{x'} P(x') \frac{\partial E(x')}{\partial w_{ij}} - \frac{\partial E(x)}{\partial w_{ij}} \right) \right] \\
&= \mathop{\mathbb{E}}_{x \in \mathscr{D}} \left[\frac{\partial E(x)}{\partial w_{ij}} \right] - \sum_{x'} P(x') \frac{\partial E(x')}{\partial w_{ij}} \\
&= \mathop{\mathbb{E}}_{x \in \mathscr{D}} [x_i x_j] - \mathop{\mathbb{E}}_{x \in \{\mu \, possibles\}} [x_i x_j]
\end{aligned}$$

où la deuxième espérance est calculée sur toutes configurations possibles des unités de la machine de Boltzmann, lorsque celle-ci est à l'équilibre.

De façon similaire on peut obtenir

$$\frac{\partial C}{\partial b_i} = \mathop{\mathbb{E}}_{x \in \mathscr{D}}[x_i] - \mathop{\mathbb{E}}_{x \in \{\mu \text{ possibles}\}}[x_i]$$

On utilisera ensuite ces dérivées pour faire de la descente de gradient et ainsi minimiser notre coût.

2.5.2 Machines de Boltzmann avec unités cachées

Cela devient beaucoup plus intéressant lorsqu'on ajoute, en plus des unités représentant des données d'entrainement (qu'on appellera dorénavant unités visibles), des unités cachées η_j, qui ne sont pas liées à des données observées. Le modèle pourra alors représenter des distributions plus complexes, qui seraient impossibles à représenter avec seulement des connections directement entre les unités visibles. Modifions notre notation pour s'adapter à cette situation. Les unités μ seront divisé en deux groupes, les unités visibles $\nu_1 = \mu_1, \nu_2 = \mu_2, \ldots, \nu_l = \mu_l$ et les unités cachées $\eta_1 = \mu_{l+1}, \eta_2 = \mu_{l+2}, \ldots, \eta_m = \mu_{l+m}$. Nous avons donc l unités visibles et m unités cachées pour un total de $n = l + m$ unités. Si on écrit aussi $\nu = \begin{bmatrix} \nu_1 & \nu_2 & \ldots & \nu_l \end{bmatrix}^T$ et $\eta = \begin{bmatrix} \eta_1 & \eta_2 & \ldots & \eta_m \end{bmatrix}^T$, on peut alors voir le vecteur μ comme la concaténation des vecteur ν et η. Dans les équations qui suivent, de façon analogue, nous aurons que u est la concaténation de x et h. Enfin, on gardera $b_1 b_2, ..., b_l$ pour les biais des unités visibles, mais renommerons $c_1 c_2, ..., c_m$ les biais des unités cachées.

Adaptons à cette nouvelle notation les probabilités et fonctions d'énergie.

$$P(x,h) = Prob(\nu = x, \eta = h) = \frac{e^{-E(x,h)}}{\sum_{x',h,} e^{-E(x',h')}}$$

où

$$E(x,h) = -\sum_i x_i b_i - \sum_j h_j c_j - \sum_{i<j} u_i u_j w_{ij}$$

ainsi que la probabilité marginale sur ν

$$P_v(x) = Prob(v = x) = \sum_h P(x,h)$$

Dans ce contexte, on voudra minimiser

$$C = - \mathop{\mathbb{E}}_{x \in \mathscr{D}} [\log (P_v(x))]$$

Dérivons C par rapport aux poids

$$\begin{aligned}
\frac{\partial C}{\partial w_{ij}} &= - \mathop{\mathbb{E}}_{x \in \mathscr{D}} \left[\frac{1}{P_v(x)} \frac{\partial P_v(x)}{\partial w_{ij}} \right] \\
&= - \mathop{\mathbb{E}}_{x \in \mathscr{D}} \left[\frac{1}{P_v(x)} \frac{\partial \sum_h P(x,h)}{\partial w_{ij}} \right] \\
&= - \mathop{\mathbb{E}}_{x \in \mathscr{D}} \left[\frac{1}{P_v(x)} \sum_h \frac{\partial P(x,h)}{\partial w_{ij}} \right] \\
&= - \mathop{\mathbb{E}}_{x \in \mathscr{D}} \left[\frac{1}{P_v(x)} \sum_h \left[P(x,h) \left(\sum_{x',h'} P(x',h') \frac{\partial E(x',h')}{\partial w_{ij}} - \frac{\partial E(x,h)}{\partial w_{ij}} \right) \right] \right] \\
&= - \mathop{\mathbb{E}}_{x \in \mathscr{D}} \left[\frac{1}{P_v(x)} \sum_h \left[P(x,h) \left(\mathop{\mathbb{E}}_{x',h'} \left[\frac{\partial E(x',h')}{\partial w_{ij}} \right] - \frac{\partial E(x,h)}{\partial w_{ij}} \right) \right] \right] \\
&= - \mathop{\mathbb{E}}_{x \in \mathscr{D}} \left[\frac{1}{P_v(x)} \sum_h P(x,h) \mathop{\mathbb{E}}_{x',h'} \left[\frac{\partial E(x',h')}{\partial w_{ij}} \right] \right] + \mathop{\mathbb{E}}_{x \in \mathscr{D}} \left[\frac{1}{P_v(x)} \sum_y P(x,h) \frac{\partial E(x,h)}{\partial w_{ij}} \right] \\
&= - \mathop{\mathbb{E}}_{x \in \mathscr{D}} \left[\frac{1}{P_v(x)} P_v(x) \right] \mathop{\mathbb{E}}_{x',h'} \left[\frac{\partial E(x',h')}{\partial w_{ij}} \right] + \mathop{\mathbb{E}}_{x \in \mathscr{D}} \left[\frac{1}{P_v(x)} \sum_h P(x,h) \frac{\partial E(x,h)}{\partial w_{ij}} \right] \\
&= - \mathop{\mathbb{E}}_{x',h'} \left[\frac{\partial E(x',h')}{\partial w_{ij}} \right] + \sum_{x \in \mathscr{D}} \left[P_v(x) \frac{1}{P_v(x)} \sum_h P(x,h) \frac{\partial E(x,h)}{\partial w_{ij}} \right] \\
&= \mathop{\mathbb{E}}_{\substack{x \in \mathscr{D} \\ h \in \{\eta \ poss.\}}} [u_i u_j] - \mathop{\mathbb{E}}_{u \in \{\mu \ poss.\}} [u_i u_j]
\end{aligned}$$

où la première espérance est calculée avec les exemples d'apprentissage comme unités visibles et toutes les possibilités d'unités cachées, lorsque le système est à l'équilibre. La seconde espérance est calculée sur toutes les configurations d'unités possibles, toujours à l'équilibre.

Similairement, on peut trouver

$$\frac{\partial C}{\partial b_i} = \mathop{\mathbb{E}}_{x \in \mathscr{D}}[x_i] - \mathop{\mathbb{E}}_{x \in \{\nu \, poss.\}}[x_i]$$

et

$$\frac{\partial C}{\partial c_i} = \mathop{\mathbb{E}}_{\substack{x \in \mathscr{D} \\ h \in \{\eta \, poss.\}}}[h_i] - \mathop{\mathbb{E}}_{h \in \{\eta \, poss.\}}[h_i]$$

On remarque que ces dérivées sont extrêmement semblables aux dérivées dans le cas où toutes les unités sont visibles. Cela nous permet de faire la descente de gradient à peu près de la même manière.

2.5.3 Machines de Boltzmann restreintes

Apprendre avec des machines de Boltzmann est cependant très lent. On peut accélérer l'apprentissage en restreignant les connections entre les unités. Dans une machine de Boltzmann restreinte (RBM) [25], on empêche les connections entre unités visibles ainsi que les connections entre les unités cachées - autrement dit les connections vont toutes d'une unité visible à un unité caché (ou vice versa). On pourra alors mettre à jour les unités visibles ou cachées en parallèle en utilisant

$$Prob(\eta_j = 1 | \nu = x) = \sigma(c_j + \sum_i x_i w_{ij})$$

et

$$Prob(\nu_i = 1 | \eta = h) = \sigma(b_i + \sum_j w_{ij} h_j)$$

Cela nous permet d'ailleurs de simplifier quelques formules, notamment

$$\frac{\partial C}{\partial w_{ij}} = \mathop{\mathbb{E}}_{\substack{x \in \mathscr{D} \\ h \in \{\eta \, poss.\}}}[x_i h_j] - \mathop{\mathbb{E}}_{\substack{x \in \{\nu \, poss.\} \\ h \in \{\eta \, poss.\}}}[x_i h_j]$$

De plus, on approximera les espérances par des moyennes sur un petit nombre d'exemples. Finalement, pour calculer l'espérance sur toutes les possibilités d'unités μ, nous ferons de l'échantillonnage de Gibbs à partir d'un exemple d'apprentissage, *i.e.* on choisit comme unités visibles ν un exemple x, duquel on échantillonne des unités

cachées η ; à partir de ces unités cachées on peut échantillonner des unités visibles et on continue aussi longtemps qu'on veut. Il a été montré qu'en pratique, échantillonner une seule fois donne de bons résultats [14]. Cela donne l'algorithme de la *contrastive divergence*, dont le pseudo-code est présenté à l'algorithme 1.

Algorithm 1: Constrastive-divergence (CD-1)

Entrée: Une mini-batch d'exemples $x^{(1)}, x^{(2)}, ..., x^{(m)}$
Taux d'apprentissage ε

for $l = 1$ *à* m **do**
  $h^{(l)} \leftarrow \sigma(w^T x^{(l)} + c)$;
  $h_s^{(l)} \sim B(1, h^{(l)})$;
  $\hat{x}^{(l)} \leftarrow \sigma(w h_s^{(l)} + b)$;
  $\hat{x}_s^{(1)} \sim B(1, \hat{x}^{(l)})$;
  $\hat{h}^{(l)} \leftarrow \sigma(w^T \hat{x}_s^{(l)} + c)$;
  $\hat{h}_s^{(l)} \sim B(1, \hat{h}^{(l)})$;
end

foreach i **do**
  $b_i \leftarrow b_i - \varepsilon \frac{1}{m} \sum_l \left(x_i^{(l)} - \hat{x}_i^{(l)} \right)$;
end

foreach j **do**
  $c_j \leftarrow c_j - \varepsilon \frac{1}{m} \sum_l \left(h_j^{(l)} - \hat{h}_j^{(l)} \right)$;
end

foreach i, j **do**
  $w_{i,j} \leftarrow w_{i,j} - \varepsilon \frac{1}{m} \sum_l \left(x_i^{(l)} h_j^{(l)} - \hat{x}_i^{(l)} \hat{h}_j^{(l)} \right)$;
end

Finalement, on peut utiliser la couche cachée générée par un RBM comme entrée pour un deuxième RBM. En répétant ce procédé plusieurs fois, on obtient un réseau profond de RBM. Ces réseaux profonds se sont avérés très utiles pour apprendre des bonnes caractéristiques sur des données, ou comme initialisation des poids pour des tâches de classification par descente de gradient. [15]

Pour l'implémentation concrète de ce modèle, nous recommandons le tutoriel sur l'apprentissage profond de Theano [6] ainsi que le guide de Hinton [13].

2.5.4 Machines de Boltzmann restreintes convolutionnelles

Récemment, Lee et al. [19] (parallèlement à Desjardins et al. [9]) ont introduit la machine de Botlzmann restreinte convolutionnelle (CRBM) . On peut voir un CRBM comme un RBM où plusieurs poids ont été enlevés, et d'autres sont partagés (le même poids relie différentes paires d'unités). Cependant, il est plus simple de définir un CRBM comme suit.

Bien que nous les utiliserons sur des données audio, il est plus simple de conceptualiser un CRBM qui prend en entrée des données visuelles. On supposera donc que notre CRBM prend en entrée une image à plusieurs dimensions, qu'on représentera par un tenseur v dont les dimensions sont (dimension de l'image, hauteur de l'image, largeur de l'image) qu'on notera (d_v, n_v, n_v) (on suppose ici que les images sont carrés pour simplifier la notation, mais ce n'est absolument pas nécessaire). On aurait par exemple qu'une image en couleur RGB de 28×28 pixels sera inscrite dans un tenseur de dimensions $(d_v, n_v, n_v) = (3, 28, 28)$. Ensuite, on regroupe les poids w en différents "filtres", représentés par un tenseur de dimension 4, dont les dimensions vont comme suit : (nombre de filtres, dimension des images, hauteur du filtre, largeur du filtre), qu'on notera (d_η, d_v, n_w, n_w) (encore une fois les deux derniers termes sont identiques seulement à des fins de simplification). Le nombre de filtres ainsi que leur hauteur et largeur (d_η et n_w) sont des hyper-paramètres. Ensuite, le biais b des unités visibles est un vecteur de longueur d_v. Donc, dans notre exemple d'image RGB, il y aurait un biais par couleur. Finalement, la couche cachée aura les dimensions (d_η, n_η, n_η) où $n_\eta = n_v - n_w + 1$, et son biais sera un vecteur de longueur d_η, de façon analogue aux biais des unités visibles. Les unités visibles et cachées ainsi que les poids correspondants à 4 filtres sont illustrés à la figure 2.4.

Les liens entre les différentes couches et poids sont donnés par la fonction d'énergie suivante

$$E(v, \eta) = -\sum_{l=1}^{d_\eta} \left(\sum_{k=1}^{d_v} v_k * \tilde{w}_{l,k} \right) \bullet \eta_l - \sum_{k,i,j} b_k v_{k,i,j} - \sum_{l,i,j} c_l \eta_{l,i,j}$$

Les probabilités d'avoir 0 ou 1 pour les différents unités deviennent

$$Prob(\eta_{l,i,j} = 1) = \sigma\left(p_{l,i,j}\right)$$

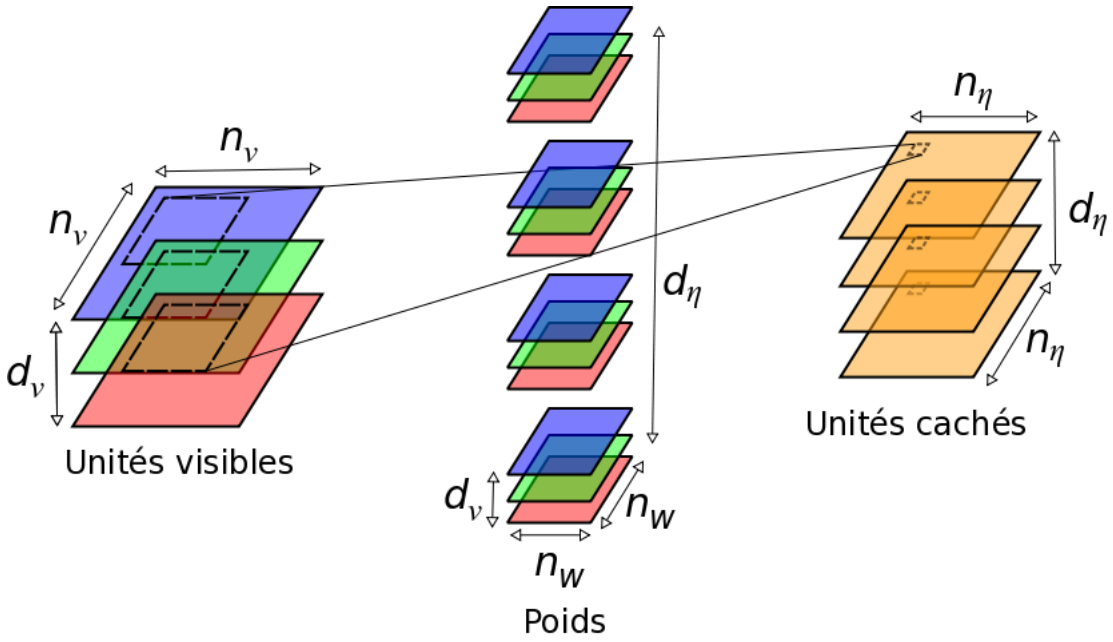

FIGURE 2.4 – Une couche de CRBM.

$$Prob(v_{k,i,j} = 1) = \sigma\left(p_{k,i,j}\right)$$

où

$$p_{l,i,j} = \left(\sum_k v_k * \tilde{w}_{l,k}\right)_{i,j} + c_l$$

$$p_{k,i,j} = \left(\sum_l \eta_l * w_{l,k}\right)_{i,j} + b_k$$

Il est à noter que les différentes convolutions ne sont pas nécessairement les mêmes. En effet, la convolution d'une matrice de dimensions (a,b) avec une matrice de dimensions (c,d) peut donner ou bien une matrice de dimensions $(a+c-1, b+d-1)$, ou bien une matrice de dimensions $(a-c+1, b-d+1)$. Il faut choisir le type de convolution selon les dimensions des différents éléments du modèle que nous avons énoncés.

On peut ensuite empiler plusieurs CRBM afin d'obtenir une architecture profonde. Pour ce faire, Lee utilise un étage de *pooling* propre aux CRBM, qu'il appelle le *probabilistic max-pooling*. L'idée de base est, comme dans d'autres réseaux à base de convolution,

de contracter les représentations apprises à la couche cachée. Les pixels voisins ayant souvent de l'information redondante, cela permet de simplifier le modèle. Cela a aussi pour effet de rendre le modèle invariant à de petites translations des images d'entrée. Avec le *probabilistic max-pooling*, on veut d'abord diviser chaque image l de la couche cachée en voisinages de dimensions $c \times c$ (typiquement c sera un petit entier). Ensuite, au lieu d'échantillonner la couche cachée en utilisant des lois binomiales, on restreint les valeurs possibles, pour un voisinage donné $V = \{(i,j)|i,j \text{ font parti du voisinage }\}$, à au plus un seul 1. On utilise, pour ces $c+1$ possibilités de résultats une multinomiale avec les probabilités suivantes.

$$Prob(\eta_{l,i,j} = 1) = \frac{\exp(p_{l,i,j})}{1 + \sum_{i,j \in V} \exp(p_{l,i,j})}$$

$$Prob(\eta_{l,i,j} = 0, \forall (i,j) \in V) = \frac{1}{1 + \sum_{i,j \in V} \exp(p_{l,i,j})}$$

Finalement, chaque voisinage est contracté en une seule unité à l'étage de *pooling*. L'unité sera 1 si une des unités du voisinage était 1 et 0 sinon. Les différents cas possibles sont illustrés à la figure 2.5.

Le CRBM profond est le modèle que nous avons implémenté et utilisé pour nos expériences sur le timbre. Lee et al. l'ont d'ailleurs également utilisé sur des données audio avec un certain succès [20], ce qui nous a motivé à les essayer.

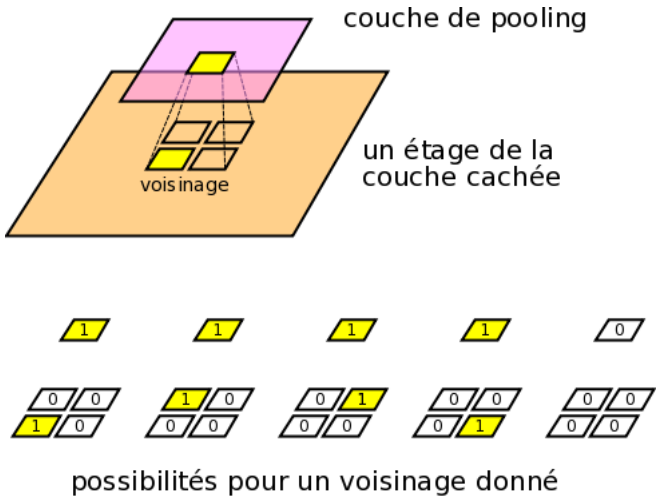

FIGURE 2.5 – *Probabilistic max-pooling* pour $c = 2$.

CHAPITRE 3

LE TIMBRE

3.1 Définition

Le timbre est un attribut du son difficile à définir. Pour un musicien, le timbre est plutôt lié à l'instrument qui produit le son. Par exemple une trompette et un trombone (ou un autre cuivre) produiront des timbres similaires, mais ces timbres seront différents de celui produit par un instrument à corde, par exemple une guitare. On utilisera aussi des termes plus ou moins précis comme "sombre", "rond", "brillant", etc. pour décrire le timbre.

Lorsqu'on désire une définition plus formelle, on cite généralement la définition de l'American Standards Association, à savoir que le timbre est l'attribut (multidimensionnel) du son qui nous permet de distinguer deux sons de même hauteur, volume et durée.

3.2 Espaces de timbre

La tâche qui nous intéresse ici est de pouvoir représenter différents timbres ou instruments dans un espace métrique, où la distance entre deux timbres correspond sensiblement à la distance perçue par un être humain.

Beaucoup de recherche a déjà été effectuée dans cette optique. Les premières expériences dans le domaine ont toutes la même base. On présente à des sujets des paires de timbres issues d'une petite base de donnée, et on leur demande de quantifier subjectivement la dissimilarité entre les différentes paires de timbres. Ensuite, on envoie ces données à un algorithme de type *multidimensional scaling* (MDS) , qui projette les différents timbres dans un espace en basse dimension (typiquement 2 ou 3) où les distances mesurées par les sujets sont préservées le plus possible. Finalement les différentes dimensions du nouvel espace créé peuvent être associées à différents attributs du sons. Ces attributs varient généralement en fonction des types de sons utilisés pour les expériences. Sur des expériences faites sur des sons synthétiques, on a trouvé que des attributs reliés au spectre du son étaient plus importants que les attributs temporels pour discriminer différents timbres [22]. D'autres études, cette fois-ci portant sur des

sons d'instruments réels (ou simulations d'instruments réels), ont eu des résultats sensiblement différents [10, 11, 17, 18, 27]. Dans ces études, des caractéristiques ayant trait à l'enveloppe du son ou le temps d'attaque ont été associés aux axes principaux des espace de timbre générés. On peut tout de même conclure de ces expériences que les caractéristiques spectrales et temporelles sont importantes dans la perception du timbre. Pour un traitement plus exhaustif de ces expériences, nous vous référons à McAdams et al. [21].

Le problème avec ce genre d'études est qu'elles sont habituellement faites sur un petit nombre de sons, puisqu'il est long de faire écouter des paires de sons à différents sujets. Les attributs associés aux différentes dimensions de l'espace de timbre généré auront de la difficulté à généraliser à de nouveaux timbres. Pour palier à ce problème, il faut une technique qui peut se passer de sujets, comme les suivantes.

Hourdin et al.[16] ont proposé un espace de timbre légèrement différent. Au lieu d'être un point dans l'espace, un timbre donné est représenté par une suite de points, qui une fois reliés forment un chemin. Chacun des points représente le contenu spectral du son à un instant donné (toutes les 4 millisecondes). Les différents points d'un timbre sont dans un espace à 80 dimensions où la moitié des dimensions représentent la fréquence des harmoniques, et l'autre moitié leur amplitudes respectives, obtenus avec un algorithme de filtrage hétérodyne (*heterodyne filtering*). Finalement, un algorithme de réduction de dimensionnalité est employé. Ainsi, on peut ramener l'espace de 80 dimensions à 8 en gardant 90% de l'information. Hourdin et al. obtiennent ainsi, de façon automatique, un espace de timbre qui s'avère avoir plusieurs points en commun avec les espaces de timbre construits à partir de mesures de dissimilarité entre timbres données par des sujets. Notons finalement que 40 instruments ont été utilisés pour ces expériences.

Quelques auteurs, dont De Poli et Prandoni[8], ainsi que Terasawa et al.[26], ont utilisé comme base pour générer un espace de timbre, les *Mel-frequency cepstral coefficients* (MFCC) . (Nous présentons une description détaillée des MFCC à la sous-section 3.2.1). Dans leur travaux, Terawasa et al. génèrent des sons artificiels et statiques (qui n'évoluent pas dans le temps) à partir de MFCC donnés. Les sons sont ensuite écoutés par des sujets qui tentent de mesurer la dissimilarité entre les sons. Leurs résultats semblent suggérer qu'en effet, les MFCC sont de bons descripteurs du timbre. Il faut cependant

noter que les expériences ont été effectuées ici aussi sur un nombre limité de sons. Le fait de n'utiliser que des sons statiques limite aussi la portée de l'étude.

De Poli et al., quand à eux, ont utilisé des outils de type "apprentissage statistique" sur les MFCC afin de construire un espace de timbre. L'espace de timbre ainsi conçu correspond assez aux espaces de timbres construits à partir de dissimilarités entre timbres demandées à des sujets. Encore une fois, peu d'instruments ont été employés.

3.2.1 MFCC

Les MFCC ont été très utilisés (avec un grand succès) dans le domaine de la reconnaissance de la parole. Ils sont sensés bien représenter le timbre de la voix humaine, et sont par exemples utiles pour différencier différents phonèmes. Il serait donc plausible qu'ils soient aussi de bons descripteurs du timbre pour des instruments de musique. Voici comment les MFCC d'un extrait sonore sont calculés.

1. On découpe l'extrait en courtes fenêtres en appliquant une fonction de fenêtrage (par exemple une fenêtre de Hamming).
2. On calcule la transformée de Fourier (discrète) pour chacune des fenêtres et on garde l'amplitude.
3. On passe ensuite chacune des fenêtres résultantes dans une banque de filtres triangulaires qui se chevauchent, espacées selon l'échelle Mel, où

$$\text{mel}(f) = \begin{cases} f & \text{si } f < 1\text{kHz} \\ 2595 \log_{10}\left(1 + \frac{f}{700}\right) & \text{si } f > 1\text{kHz} \end{cases}$$

4. On prend le logarithme des données résultantes.
5. Finalement, on applique la transformée en cosinus discrète, de laquelle on ne garde typiquement que quelques premières composantes.

Une façon courante de décrire le système de production de la parole est de le résumer à un système source-filtre. La source est la combinaison de l'air qui est expulsé des poumons et des cordes vocales, alors que le filtre est la forme du canal vocal. La source produit le son et le filtre en modifie les composantes spectrales. Or, ce sont ces composantes spectrales qui sont importantes lorsqu'on distingue par exemple un *a* d'un *o* (timbre

différent). La source influencera plutôt la hauteur du son (voix aigüe, voix grave). En reconnaissance de la parole, les MFCC sont en fait une façon d'ignorer la source et de ne garder que le filtre, à partir d'un son produit.

Habituellement, on utilise, en plus des MFCC, leurs dérivées dans le temps pour décrire avec plus de précision des sons.

CHAPITRE 4

BASE DE DONNÉES

4.1 Motivation

Comme en témoignent les études sur le timbre que nous avons recensées, les bases de données utilisées sont souvent limitées. Généralement, les différents instruments utilisés sont joués et enregistrés par des professionnels. Chaque note/accord doit être joué séparément pour chaque instrument. Cette procédure est laborieuse et c'est pourquoi on a en bout de ligne un nombre limité d'instruments. Pour pouvoir avoir plus d'instruments, on peut avoir recours à la génération de sons artificiels. Mais alors, dépendamment de la méthode avec laquelle les sons sont générés, ils risquent d'être limités en timbre. Notre moteur de génération de timbres se veut pallier à ces inconvénients.

4.2 Moteur de génération de timbres

Nous avons donc conçu, pour les besoins de cette recherche, un moteur capable de générer potentiellement n'importe quelle note (ou accord... ou partition !) pour n'importe quel instrument. Le moteur est à la base en C et est interfacé en Python[5]. Le python étant un langage interprété dont la syntaxe est extrêmement simple, cela nous donne à la fois beaucoup de puissance et de flexibilité pour la génération des timbres. Les différentes composantes du moteur sont détaillées dans les sous-sections qui suivent.

4.2.1 MIDI

Le standard MIDI (*Musical Instrument Digital Interface*) tient une place importante dans la génération de nos timbres. Il s'agit d'abord et avant tout d'un protocole qui permet à différents instruments de musique électroniques de communiquer (par exemple un clavier et un ordinateur). Les données transmises sont symboliques, c'est-à-dire que l'audio lui-même n'est pas transmis. Seulement des informations du genre "jouer un fa# avec un volume de 100 et une durée de 1 seconde" sont envoyées. Ces informations peuvent être aussi écrites dans un fichier, et c'est ce que nous utilisons principalement

ici. On peut voir un fichier MIDI comme une partition, décrivant quand doivent être jouées quelle notes et de quelle façon. Cette "partition" sera ensuite jouée par un *Audio Unit*, que nous décrivons dans la section suivante.

4.2.2 Audio units

Les éléments de base de notre moteur de génération de timbre sont les *audio units*. Il s'agit simplement d'un *plug-in* qui reçoit en entrée des données MIDI et/ou un signal audio, et retourne en sortie des données MIDI et/ou audio. Seuls deux types d'*audio units* nous intéressent. D'abord, ceux qui prennent en entrée un signal midi, et retournent un signal audio (signal qui correspondra aux données musicales MIDI, rendu avec un timbre donné). On appellera ces *audio units* des *synthétiseurs*. Finalement, on s'intéressera aussi aux *plug-ins* qui reçoivent et renvoient du signal audio, qu'on nommera *effets*. Ceux-ci servent à modifier un signal audio.

Les *audio units* ont également la particularité de posséder des paramètres ajustables, qui vont modifier leur effet sur les données qu'ils traitent. Une fois les paramètres choisis, ils peuvent être sauvegardés dans un fichier pour être réutilisés. On appelle ce fichier un *preset*.

Les *audio units* sont partie intégrante de l'engin audio sur Mac OS X[2]. De ce fait, notre générateur de timbre ne fonctionne que sous Mac (du moins pour l'instant). Le choix de cette plateforme a été principalement motivé par le fait que l'auteur avait accès à des ordinateurs Mac. Windows et Linux ayant aussi des équivalents aux *audio units*, notre moteur pourrait être adapté pour ces plateformes.

4.2.2.1 Synthétiseurs

Les synthétiseurs, comme leur nom l'indique, servent à synthétiser du son. On leur donne des notes MIDI et elles les "jouent" avec un certain instrument. Les différents paramètres du synthétiseur servent d'abord à ajuster le timbre du son produit. Par exemple, on peut imaginer un *audio unit* voulant reproduire le son d'un piano, dans lequel on retrouvera des paramètres pour modifier la brillance ou la réverbération du piano, mais ce sera toujours un piano. À l'opposé, on a le synthétiseur classique, où on additionne une série de signaux périodiques de différentes formes pour produire un

son. Les paramètres sont alors le type d'onde (sinusoïdale, en dents de scie, carrée, etc.) à additionner, leurs amplitudes, etc. La fréquence des signaux additionnés est déterminée en fonction des notes MIDI passées au *plug-in*. Cela nous donne un instrument synthétique, mais dont la gamme de timbres possible est beaucoup plus vaste.

Pour nos expériences, nous avons utilisé ces deux types de synthétiseurs, *i.e.* des synthétiseurs synthétiques et d'autres qui ne font que reproduire des notes enregistrées sur de vrais instruments physiques.

4.2.2.2 Effets

On utilise ensuite les effets pour modifier le signal créé par un synthétiseur. Par exemple, pour nos expériences, on a utilisé des filtres, distorsions, chorus, échos, et *reverb*s. Les paramètres servent alors à ajuster comment et dans quelle mesure l'effet modifie le signal audio. Nous les utilisons pour diversifier encore plus les différents timbres produits par les synthétiseurs.

4.2.2.3 Instrument

Prenons un synthétiseur, auquel on ajoute une série d'effets, et, en ajustant leur différents paramètres, on obtient une quantité virtuellement infinie de timbres. C'est sur ce principe qu'est basé notre générateur de timbre. Par *instrument*, on entend une telle combinaison d'*audio units* (synthétiseur plus effets).

Vu le nombre d'*audio units* disponibles, les possibilités sont infinies. À partir des effets simples disponibles gratuitement sur internet jusqu'aux synthétiseurs professionnels d'instruments d'orchestres de haute qualité, tout est possible.

4.2.2.4 Hôte

Finalement, il ne nous reste qu'à charger différents instruments dans un *hôte audio unit* pour pouvoir les utiliser à notre guise. C'est cette partie qui est le coeur du moteur. C'est là que sont gérées les interactions entre les différents *audio units*, les données MIDI et le système d'exploitation. La base est programmée en C en utilisant essentiellement des API fournies par *Apple[2]* à cet effet. Nous avons ensuite porté l'hôte en python, donnant une immense flexibilité au moteur. Il est alors beaucoup plus facile de spécialiser le

moteur pour des besoins plus précis. Par exemple, la génération de timbres aléatoires, expliquée à la section suivante, a été programmée en python.

Il existe de nombreux *hôte audio unit* permettant de charger des *audio units* et de les utiliser. La particularité du nôtre est qu'il est programmable, et, par conséquence, on peut générer beaucoup de sons en peu de temps. Avec une liste de *presets* pour des *audio units* donnés, ainsi qu'une liste de fichiers MIDI, on peut générer l'audio correspondant à toutes les combinaisons de MIDI et de *presets* avec seulement quelques lignes de code (voir l'exemple de l'annexe I). C'est un outil qui, à notre connaissance, n'existait pas avant.

4.3 Génération aléatoire de timbres

Naïvement, on pourrait croire que générer un timbre aléatoire est trivial : on prend un instrument, quelques effets, et on leur choisit des paramètres aléatoires. Malheureusement, cette approche ne marche que rarement en pratique. La plupart du temps, des paramètres entièrement aléatoires donneront un "timbre" qui n'a rien de musical, et donc inutilisable. En fait, la plupart du temps, aucun son ne sera émis, plusieurs paramètres ayant souvent la possibilité de bloquer le son complètement. La solution que nous avons trouvée est d'aider le programme à choisir les paramètres aléatoires. Pour chaque paramètre d'un *audio unit* donné, on échantillonne sa valeur selon une distribution expressément choisie, au besoin en prenant en compte les valeurs des paramètres déjà choisis. De cette manière, on s'assure que le timbre généré a du sens. En général, même en choisissant à la main des distributions pour tous les paramètres, le volume du timbre final reste difficile à prévoir. On ajuste donc le paramètre "volume" du *plug-in* (la grande majorité en ont un) de façon à obtenir un volume donné. Cela nous permet de générer plusieurs timbres différents ayant sensiblement le même volume. Le processus pour générer plusieurs timbres aléatoires à partir d'un fichier MIDI, d'un synthétiseur et de trois effets est illustré à la figure 4.1.

Donc, pour pouvoir utiliser un *audio unit* donné pour générer des timbres aléatoires, il suffit d'aider le programme a choisir des paramètres qui ont du sens. La complexité de cette tâche varie selon le *plug-in*, mais n'a pas vraiment posé de réel problème en ce qui nous concerne.

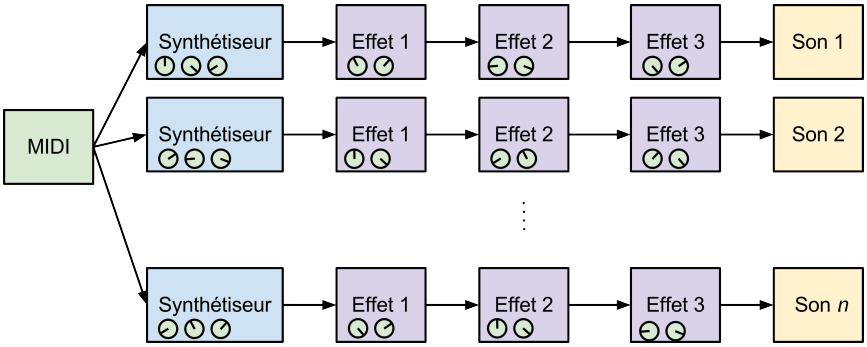

FIGURE 4.1 – Génération de timbres aléatoires.

Autre point positif dans le choix des *audio units* : les timbres générés aléatoirement peuvent ensuite être utilisé directement dans n'importe quel séquenceur de musique supportant les *audio units*.

4.4 Résumé

En résumé, notre moteur peut générer n'importes quelles notes sur n'importe quel instrument (en autant qu'il soit disponible sous la forme d'un *audio unit*). De plus, pour un instrument donné, on a la capacité de générer un timbre aléatoire, en modifiant les paramètres du *audio unit*. Finalement, puisque interfacé en python, le tout peut être automatisé efficacement. On peut voir comment le moteur est utilisé de façon concrète à l'annexe I.

CHAPITRE 5

EXPÉRIENCES

Nous allons ici tester l'utilisation de CRBM profonds sur une base de donnée de timbres. Nous espérons que les caractéristiques apprises par ce modèle non-supervisé donneront un bon espace de timbre. Nous allons ensuite comparer les distances entre timbres dans cet espace avec les distances comme perçues par des humains. Nous vérifierons également comment ce modèle se compare à d'autres modèles plus simples.

5.1 Base de données

Nous avons utilisé notre moteur pour générer assez de timbres afin que, lors de l'apprentissage, jamais un timbre ne soit vu deux fois. La première moitié des sons utilisent à la base des sons de vrais instruments échantillonnés professionnellement (l'*audio unit* utilisé étant Kontakt 3 [3]) et l'autre moitié utilise des sons électroniques, générés à partir d'un *audio unit* appelé Automat1 [1], disponible gratuitement en ligne. Entre zéro et cinq effets, avec des paramètres aléatoires, sont ensuite ajoutés à chaque instrument. Les effets sont des effets standards, soient : filtres, distorsions, chorus, échos, et *reverb*s. Nous obtenons ainsi une base de donnée énorme et diversifiée.

Nous avons ensuite utilisé, pour générer l'audio, un fichier MIDI contenant une seule note, un Si-3 d'une durée de 1.5 seconde. Il est à noter que le son généré n'est pas nécessairement de 1,5 secondes. En effet un *audio unit* peut interpréter le signal MIDI à sa façon et par exemple ne faire qu'une note staccato, dans quel cas le son durera beaucoup moins que 1,5 secondes.

Le fait d'utiliser seulement une note simplifie grandement le problème. En effet, le timbre d'un instrument peut changer avec la hauteur de la note jouée ; on s'évite cette considération en prenant la même note pour comparer les instruments.

Des exemples de sons issus de notre base de données sont disponibles sur internet [1].

1. http://www-etud.iro.umontreal.ca/~lemiesim/echantillons_sons/

5.1.1 Ensemble de validation

Afin de valider les différents modèles obtenus par différentes combinaisons d'hyper-paramètres, nous avons fabriqué un ensemble de validation. L'ensemble contient 985 timbres (différents de l'ensemble d'entrainement mais fabriqués selon le même procédé), que nous avons mis en ordre de longueur. L'auteur a ensuite écouté chaque paire de sons consécutifs et annoté à quel point les timbres étaient semblables, sur une échelle de 1 à 6. Les ordonner par ordre de longueur fait en sorte que les sons comparés ont à peu près la même longueur, ce qui permet de simplifier la comparaison (voir la formule 5.1 de la section 5.3.1). En effet, comme on peut le voir à la figure 5.1, les caractéristiques apprises pour un son sont représentés par une matrice où un axe représente le temps, et l'autre la dimension des caractéristiques à chaque instant. Lorsqu'on compare deux sons de la même longueur, le tout est bien aligné. Si, par contre, on comparait un son court avec un son long, une partie du son long serait comparée avec du vide ajouté à la fin du son court.

5.1.2 Ensembles de test

L'ensemble de test est comme l'ensemble de validation, sauf qu'il est plus petit (99 timbres, bien-sûr différents des timbres des ensembles d'entrainement et de validation), et qu'il a été annoté par 10 personnes.

5.2 Prétraitement des données

Notre générateur de timbre produit des données audio que nous convertissons en *mono* et que nous rééchantillonnons à 22050 Hz. Cette compression ne perd presque pas de qualité et est assez standard. Les sons produits ne sont pas tous de la même longueur, et donc, des "0" ont été ajoutés à la fin des sons les plus courts pour qu'ils aient tous la même longueur : cela simplifie l'algorithme. Ensuite, chaque son a été découpé en fenêtres de 1024 échantillons, avec un chevauchement de 3/4 de fenêtre. On applique ensuite les étapes 1 à 4 du calcul des MFCC, comme expliqué à la sous-section 3.2.1.

Ensuite, au lieu d'appliquer la transformée en cosinus discrète (étape 5), on utilise la PCA pour transformer les données de manière à ce qu'elles aient une moyenne de 0 et la matrice identité comme covariance (cette transformation est connue sous le nom

de *PCA whitening*). Plus précisément, on calcule sur l'ensemble d'entrainement (ou un sous-ensemble), la moyenne

$$\bar{x} = \mathop{\mathbb{E}}_{x \in \mathscr{D}}[x]$$

ainsi que la matrice de covariance

$$S = \mathop{\mathbb{E}}_{x \in \mathscr{D}}\left[(x-\bar{x})(x-\bar{x})^T\right]$$

Comme S est symétrique, on peut la réécrire comme

$$S = UDU^T$$

où U est une matrice orthonormale ($UU^T = I$) des vecteurs propres de S et où D est la matrice diagonale des valeurs propres correspondantes. On modifie ensuite chacun des éléments de l'ensemble d'entrainement comme suit :

$$\tilde{x} = D^{-1/2}U^T(x-\bar{x})$$

On peut facilement vérifier que la moyenne de l'ensemble d'entrainement modifié devient 0 et que sa matrice de covariance devient la matrice identité :

$$\begin{aligned}
\mathop{\mathbb{E}}_{x \in \mathscr{D}}[\tilde{x}] &= \mathop{\mathbb{E}}_{x \in \mathscr{D}}\left[D^{-1/2}U^T(x-\bar{x})\right] \\
&= D^{-1/2}U^T \mathop{\mathbb{E}}_{x \in \mathscr{D}}[x-\bar{x}] \\
&= D^{-1/2}U^T(\bar{x}-\bar{x}) \\
&= 0
\end{aligned}$$

$$\begin{aligned}
\mathop{\mathbb{E}}_{x\in\mathscr{D}}\left[\tilde{x}\tilde{x}^T\right] &= \mathop{\mathbb{E}}_{x\in\mathscr{D}}\left[D^{-1/2}U^T(x-\bar{x})\left(D^{-1/2}U^T(x-\bar{x})\right)^T\right] \\
&= \mathop{\mathbb{E}}_{x\in\mathscr{D}}\left[D^{-1/2}U^TUD^{-1/2}\right] \\
&= D^{-1/2}U^T\mathop{\mathbb{E}}_{x\in\mathscr{D}}\left[(x-\bar{x})(x-\bar{x})^T\right]UD^{-1/2} \\
&= D^{-1/2}U^TSUD^{-1/2} \\
&= D^{-1/2}DD^{-1/2} \\
&= I
\end{aligned}$$

Les caractéristiques ainsi obtenues sont extrêmement similaires aux *Principal Mel-Spectrum Components* (PMSC) utilisées par Hamel et al. [12]. Nous appellerons d'ailleurs ces caractéristiques par le même nom, soit PMSC.

5.3 Apprentissage

Les données sont alors prêtes à être envoyées au CRBM profond, décrit à la section 2.5.4, où nous ferons la convolution dans le temps seulement. Nous avons utilisé un CRBM profond à deux couches (notons qu'utiliser une ou trois couches donnait de moins bons résultats), auquel nous avons ajouté une pénalité pour les poids plus gros, *i.e.* nous avons ajouté $\lambda \|w\|_2^2$ au coût. Nous avons aussi soustrait aux biais des unités cachées une quantité fixe ε à chaque batch, pour renforcer la sparsité de la représentation. Bien que cette astuce soit différente de celle utilisée par Lee et al. [19], nous l'avons trouvée efficace et plus simple à utiliser.

Nous avons testé deux approches pour entrainer le modèle. Premièrement, vu l'énorme quantité d'hyper-paramètres à optimiser, nous avons d'abord essayé différentes combinaisons d'hyper-paramètres, mais en conservant les paramètres fixes (paramètres qui sont normalement appris). Notons que cette approche a récemment été justifiée par Saxe et al. [23] Puis, nous avons calculé l'erreur sur l'ensemble de validation, et gardé le meilleur modèle. Deuxièmement, nous avons fait la même chose, c'est-à-dire essayé différents hyper-paramètres, mais cette fois-ci nous avons en plus appris les poids. Cette deuxième méthode pourrait sembler à priori la meilleure, mais il faut tenir compte

qu'entrainer le modèle prend beaucoup de temps, et donc les combinaisons d'hyper-paramètres que l'on peut tester ainsi, en un temps raisonnable, sont moindres. De plus, cela nous permet de voir ce que l'architecture à elle seule du modèle permet d'accomplir, et de vérifier à quel point l'apprentissage aide.

Nous avons comparés ces modèles avec quelques modèles plus simplistes. D'abord, nous avons utilisé directement comme caractéristiques les PMSC (voir section 5.2), afin de vérifier à quel point les CRBM profonds améliorent les résultats. Ensuite, nous avons évidemment utilisé les MFCC (avec leurs dérivées), caractéristiques que nous espérons battre. Les MFCC ont été calculés avec 40 fenêtres triangulaires, et nous avons retenus les 13 premières composantes de la DCT, ainsi que les premières et deuxièmes dérivées correspondantes, ce qui nous donne un total de 39 composantes. Nous avons aussi agrégé ces 39 composantes dans le temps en faisant la moyenne sur des fenêtres dont la longueur a été choisie en testant sur l'ensemble de validation.

5.3.1 Évaluation du modèle

Afin d'évaluer les performances de notre modèle, il nous faut une façon de comparer deux espaces de timbre. Plus simplement, nous allons comparer les distances entre les timbres selon un modèle avec les distances entre les timbres selon des humains. Si une paire de sons est proche pour un humain, elle devra être proche pour le modèle, et vice versa.

Nous avons décidé de comparer les mesures des humains et des modèles en utilisant le τ de Kendall. Soient deux paires i et j de sons données parmi n paires, a_i et a_j leurs distances selon un humain et b_i et b_j les distances selon un modèle, le τ de Kendall est défini comme suit

$$\tau = \frac{\sum_{i<j} \text{signe}(a_i - a_j) \cdot \text{signe}(b_i - b_j)}{\binom{n}{2}}$$

Lorsque deux paires de sons ont des distances consistantes ($a_i \neq a_j$, $b_i \neq b_j$ et $a_i < a_j \iff b_i < b_j$), l'intérieur de la sommation sera 1, et sinon il sera -1 (et 0 si $a_i = a_j$ ou $b_i = b_j$, dans quel cas on ne peut pas vraiment comparer). Le dénominateur ne sert qu'à normaliser par le nombre de paires considérées. Plus τ est grand, plus les distances de l'humain et du modèle sont consistantes. Évidemment, il est aussi possible de comparer

deux humains avec le τ de Kendall.

Avec cette mesure, un modèle qui donne des distances au hasard aura un τ de 0, lorsque comparé à n'importe quel modèle. Un modèle "parfait" (toujours en accord avec le modèle avec lequel on le compare) aura un τ de 1. Un τ de -1 correspondrait à un modèle qui dit toujours le contraire.

Nous avons les distances entre timbres comme perçues par des humains pour les ensembles de validation et de test, et les distances d'un modèle seront simplement les distances entre les caractéristiques apprises pour deux timbres. Les distances pour un modèle sont calculées de la manière suivante. Soit f le modèle qui prend en entrée un son x et produit un vecteur de caractéristiques associé, la distance entre deux sons x et x' sera définie comme

$$\left\| f(x) - f(x') \right\|_1 = \sum_i \left| \left[f(x) - f(x') \right]_i \right| \tag{5.1}$$

À la figure 5.1, on peut voir les sorties pour trois sons, d'un DCRBM dont la dernière couche est formée à partir de 25 filtres. Les matrices sont transformées en vecteurs avant d'être envoyées à la formule précédente. Les 2 premiers sons sont semblables et le troisième est différent. Les trois sons sont disponibles sur internet [1].

5.4 Résultats et analyse

Nous présentons ici les résultats des différents modèles. Chaque modèle prend en entrée un timbre et produit un vecteur de caractéristiques, que nous comparons pour obtenir une distance entre deux timbres. Pour les figures et tableaux, nous avons donnée aux modèles les surnoms suivants : *mfcc*, *pmsc*, *dcrbm_no_train* (CRBM profond sans entrainement), *dcrbm_train* (CRBM profond avec entrainement). Aussi, nous avons aussi ajouté un modèle, *random*, qui donne des distances aléatoires entre deux timbres.

On peut voir à la figure 5.2 le τ de Kendall entre chaque paire d'humains et de modèles. On remarque d'abord que, la plupart du temps, les humains sont très consistants entre eux. En effet, la moyenne des τ de toutes les paires d'humains est de 0.43. Les modèles, quand à eux, sont malheureusement très peu performants, quand on les compare avec les humains. Ils font cependant tous clairement mieux que le hasard.

1. http://www-etud.iro.umontreal.ca/~lemiesim/echantillons_sons/

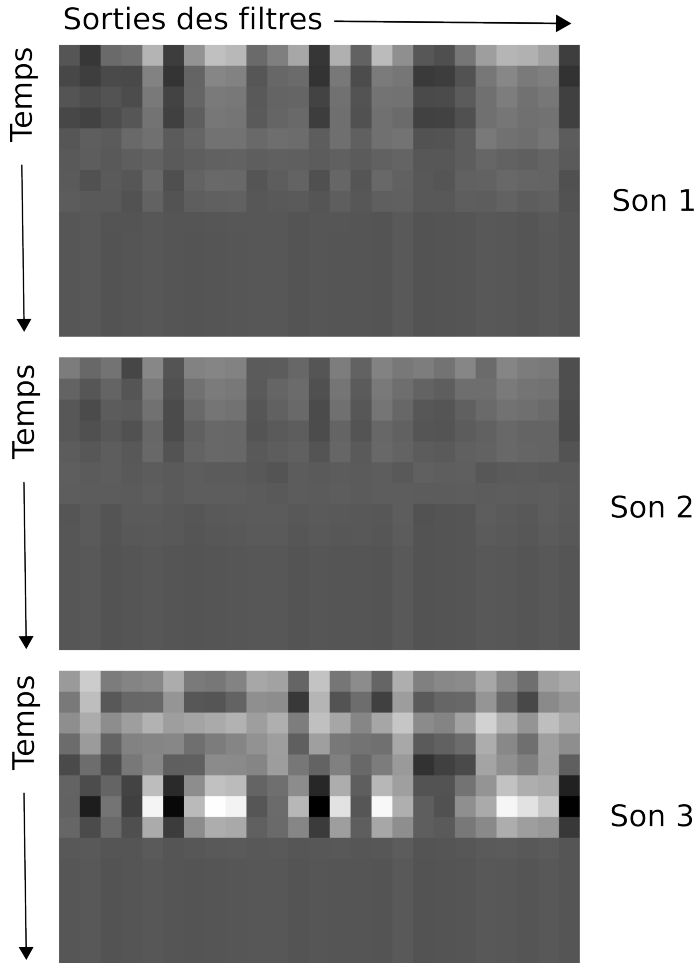

FIGURE 5.1 – Sorties d'un DCRBM.

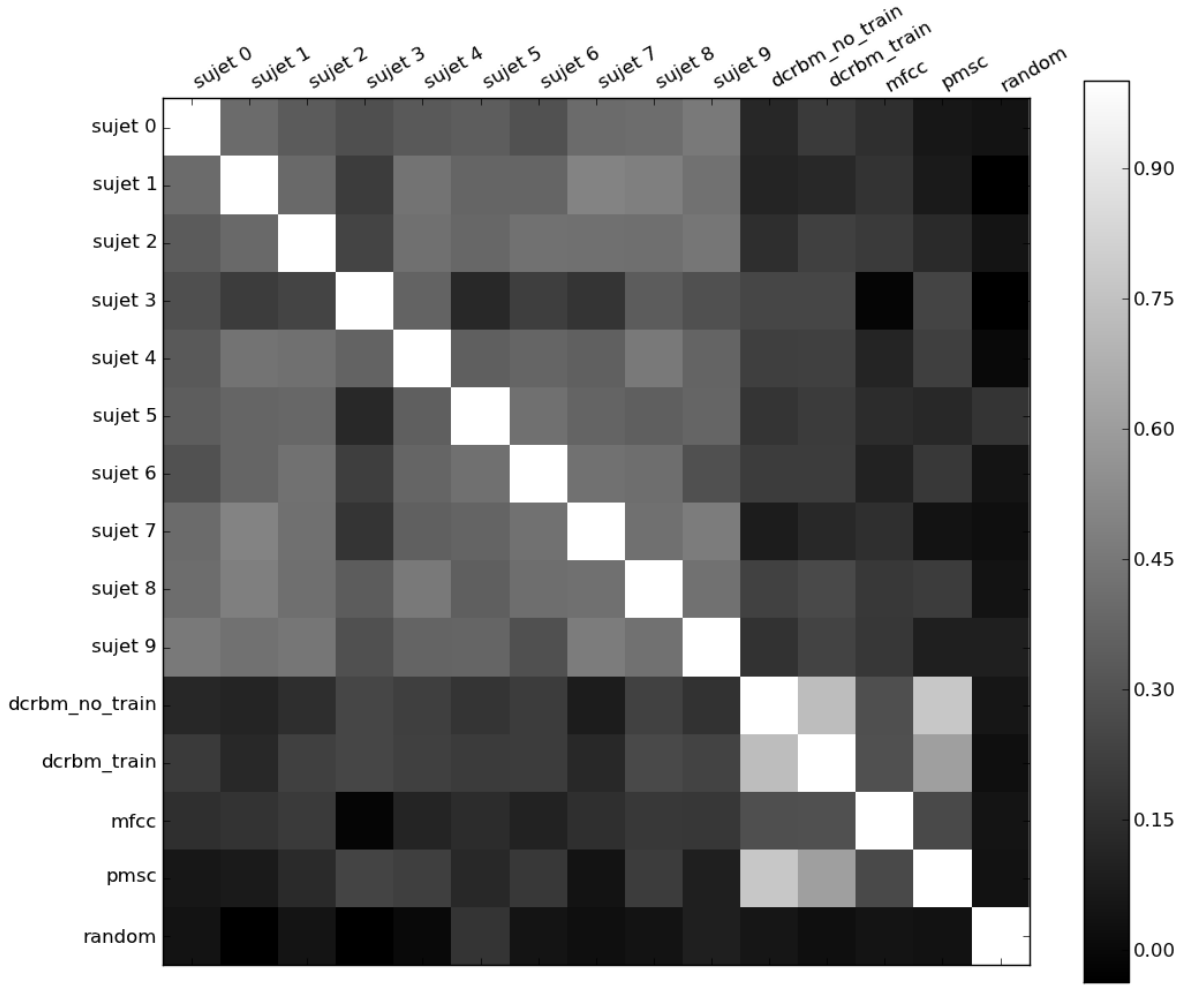

FIGURE 5.2 – τ de Kendall entre différents sujets et modèles. Pâle = meilleur.

On peut mieux comparer les modèles en ignorant les comparaisons entre les différents sujets humains, comme présenté à la figure 5.3. Pour comparer les modèles entre eux de façon plus précise, nous avons calculé, pour chacun, la moyenne des τ lorsque comparés aux humains, qu'on peut voir à la table 5.I. Les PMSC semblent performer aussi bien que les MFCC. Le CRBM profond sans entrainement semble les battre de peu, signe que l'architecture du réseau aide. L'entrainement des poids, heureusement, améliore le tout de façon significative, par rapport aux MFCC et aux PMSC.

La meilleure architecture, lorsque nous n'avons pas entrainé les poids du CRBM profond était comme suit : deux couches avec 75 filtres chacune, dont les tailles étaient de 17 et 4, avec des fenêtres de *pooling* de taille 4 et 10, respectivement. L'architecture du meilleur CRBM profond après entrainement était la suivante : deux couches avec également 75 filtres chacune, de tailles 18 et 6, ainsi que des fenêtre de *pooling* de taille 3 et 6, respectivement.

Cela dit, même si les MFCC sont battus, on est loin de pouvoir comparer ce modèle à un humain. Il est probable qu'entrainer de façon non-supervisée ne soit tout simplement pas suffisant. Peut-être, en effet, faudrait-il ajouter une composante supervisée pour pouvoir apprendre une meilleure représentation du timbre. Aussi, le fait de limiter nos données à une seule note par timbre peut être un facteur. En effet, dans la vraie vie, lorsque nous entendons des sons, un timbre donné est souvent entendu sur plusieurs notes. Par exemple si l'on frappe sur une table à différents endroits, dépendamment de l'endroit où l'on frappe, le timbre sera similaire, alors que la note peut changer. Peut-être que cette association (différents notes correspondantes au même timbre) serait assez pour apprendre un meilleur espace de timbre.

La façon dont nous comparons les caractéristiques associées aux sons pourrait aussi être à améliorer. En effet, la métrique expliquée à la fin de la section 5.3.1 est très simpliste. Cette métrique était efficace pour comparer différents modèles entre eux, puisqu'ils utilisent tous la même, et donc la comparaison est plutôt faite sur les caractéristiques apprises. Cependant, il apparait clair que les humains utilisent une métrique

Modèle	dcrbm_no_train	dcrbm_train	mfcc	pmsc	random
τ moyen	0.171	0.206	.140	.139	.04

TABLE 5.I – Moyennes sur les sujets des τ de Kendall des différents modèles

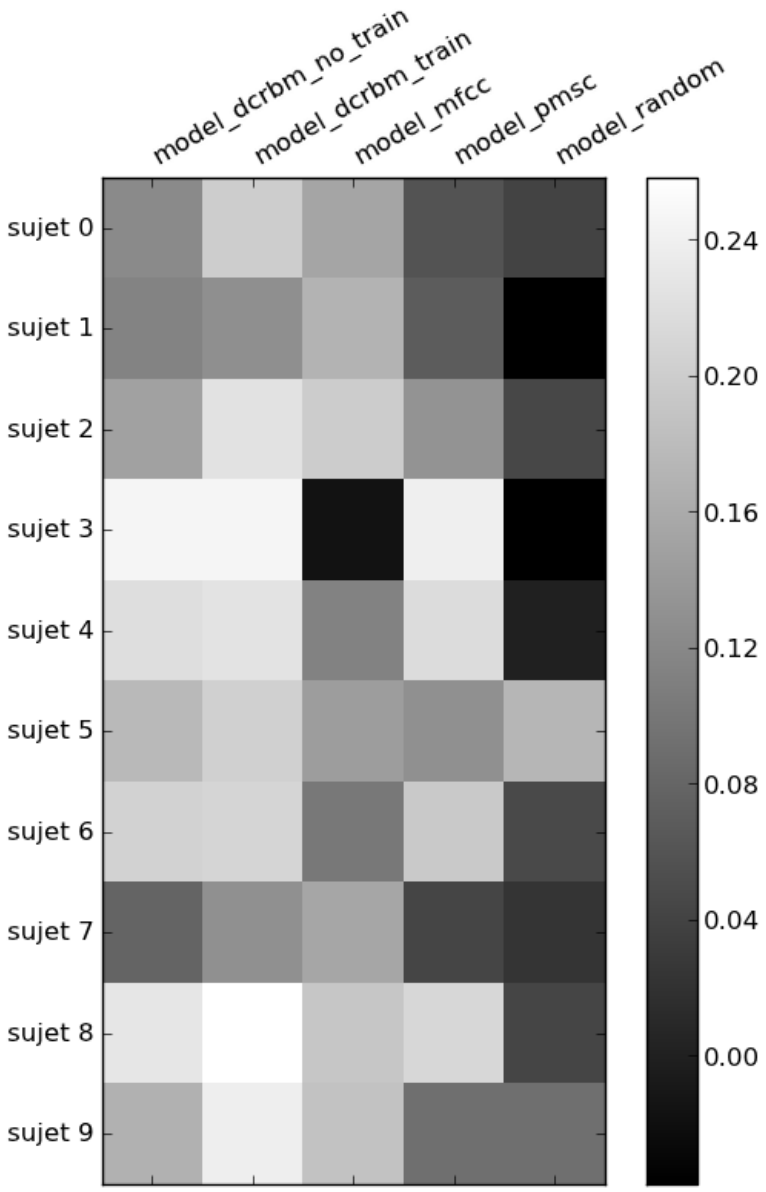

FIGURE 5.3 – τ de Kendall entre les sujets et les modèles. Pâle = meilleur.

beaucoup plus subtile pour comparer deux sons. La façon de calculer la distance entre les caractéristiques de sons devra être améliorer si l'on veut qu'un modèle se compare à un humain aussi bien que les humains se comparent entre eux.

Finalement, nous avons découvert que les CRBM profonds sont extrêmement difficiles à entrainer. En particulier, ils sont composés de tellement d'hyper paramètres, et de tellement de façons possibles d'ajuster l'entrainement, qu'il se peut bien que nous soyons passé à côté d'un détail qui aurait pu donner de meilleurs résultats.

CHAPITRE 6

CONCLUSION

Au cours de la recherche effectuée pour ce mémoire, nous avons grandement parfait notre connaissance de l'apprentissage machine. Nous avons notamment implémenté et expérimenté avec les machines de Boltzmann restreintes convolutionnelles profondes. De façon plus concrète, nous les avons utilisé afin de générer un espace de timbre, et comparé avec l'espace des MFCC, caractéristiques du sons reconnues pour bien représenter le timbre. Ce qui distingue cette recherche de ce qui s'est déjà fait précédemment dans le domaine de l'espace de timbre, c'est que nous avons entrainé notre modèle de façon non-supervisée sur une énorme quantité de sons d'instruments (acoustiques et électroniques). Bien qu'il manque encore beaucoup de travail pour réussir à créer un espace de timbre capable de rivaliser avec un humain, nous avons tout de même réussi à battre significativement les MFCC. Plusieurs avenues sont encore à explorer. Ajouter une composante supervisée à l'apprentissage pourrait par exemple aider à fabriquer un meilleur espace de timbre. Utiliser plusieurs notes par timbres devrait aussi être envisagé, puisque c'est le genre de stimuli que nous humains avons dans la vie de tous les jours. Ensuite, évidemment, nous pourrions essayer toutes sortes de modèles différents d'apprentissage machine.

Une des plus grandes contributions de ce mémoire est selon nous le moteur de génération de timbres, qui est très général et pourrait générer toutes une gamme de bases de données audio. Nous avons conçu cet outil pour pallier au fait que beaucoup d'études sur le timbre sont effectuées avec un nombre limité de sons. Nous avons personnellement utilisé notre moteur pour générer des timbres aléatoires, mais toutes sortes d'autres utilisations pourraient être envisagées. Ce moteur permet d'amener à un autre niveau les expériences sur le son où une grande quantité d'exemples est nécessaire. Nous espérons d'ailleurs qu'il pourra être utile à d'autres chercheurs.

Beaucoup d'aspects du timbre restent encore à élucider. C'est un sujet aussi intéressant que complexe, et dans lequel l'apprentissage machine peut clairement s'avérer un outil de choix. Nous espérons que le présent mémoire saura encourager d'autres chercheurs à essayer de mieux comprendre le timbre à l'aide de l'apprentissage machine.

BIBLIOGRAPHIE

[1] Alphakanal. `http://blog.alphakanal.de/category/automat`.

[2] Apple. `http://www.apple.com`.

[3] Native instruments. `http://www.native-instruments.com`.

[4] pyau. `http://code.google.com/p/pyau/`.

[5] Python. `http://www.python.org`.

[6] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano : a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, jun 2010. Oral.

[7] C.M. Bishop et al. *Pattern recognition and machine learning*. Springer New York :, 2006.

[8] G. De Poli and P. Prandoni. Sonological models for timbre characterization. *Journal of New Music Research*, 26(2) :170–197, 1997.

[9] G. Desjardins and Y. Bengio. Empirical evaluation of convolutional RBMs for vision. Technical report, Tech Report, 2008.

[10] J.M. Grey. Multidimensional perceptual scaling of musical timbres. *Journal of the Acoustical Society of America*, 61(5) :1270–1277, 1977.

[11] J.M. Grey and J.W. Gordon. Perceptual effects of spectral modifications on musical timbres. *The Journal of the Acoustical Society of America*, 63 :1493, 1978.

[12] Philippe Hamel, Simon Lemieux, Yoshua Bengio, and Douglas Eck. Temporal pooling and multiscale learning for automatic annotation and ranking of music audio. ISMIR, 2011.

[13] G. Hinton. A practical guide to training restricted boltzmann machines. *Momentum*, 9 :1, 2010.

[14] G.E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8) :1771–1800, 2002.

[15] G.E. Hinton and R.R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786) :504, 2006.

[16] C. Hourdin, G. Charbonneau, and T. Moussa. A multidimensional scaling analysis of musical instruments' time-varying spectra. *Computer Music Journal*, 21(2) :40–55, 1997.

[17] P. Iverson and C.L. Krumhansl. Isolating the dynamic attributes of musical timbre. *The Journal of the Acoustical Society of America*, 94 :2595, 1993.

[18] C.L. Krumhansl. Why is musical timbre so hard to understand. *Structure and perception of electroacoustic sound and music*, pages 43–53, 1989.

[19] H. Lee, R. Grosse, R. Ranganath, and A.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.

[20] H. Lee, Y. Largman, P. Pham, and A. Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. *Advances in neural information processing systems*, 22 :1096–1104, 2010.

[21] Stephen McAdams, Suzanne Winsberg, Sophie Donnadieu, Geert De Soete, and Jochen Krimphoff. Perceptual scaling of synthesized musical timbres : Common dimensions, specificities, and latent subject classes. *Psychological Research*, 58(3) :177–192, December 1995.

[22] J.R. Miller and E.C. Carterette. Perceptual space for musical structures. *The Journal of the Acoustical Society of America*, 58 :711, 1975.

[23] A. Saxe, P.W. Koh, Z. Chen, M. Bhand, B. Suresh, and A. Ng. On random weights and unsupervised feature learning. In *Workshop : Deep Learning and Unsupervised Feature Learning (NIPS)*, 2010.

[24] Scholarpedia. Boltzmann machine. `http://www.scholarpedia.org/article/Boltzmann_machine`.

[25] P. Smolensky. Information processing in dynamical systems : Foundations of harmony theory. *Parallel distributed processing : Explorations in the microstructure of cognition*, 1 :194–281, 1986.

[26] H. Terasawa, M. Slaney, and J. Berger. Determining the euclidean distance between two steady state sounds. In *Proceedings of the 9th International Conference on Music Perception and Cognition (ICMPC9)*, 2006.

[27] D.L. Wessel. Timbre space as a musical control structure. *Computer music journal*, 3(2) :45–52, 1979.

Annexe I

Utilisation du moteur de génération de timbres

Pour installer et utiliser le moteur, il suffit de suivre les instructions sur la page web du projet[4]. Voici un bref aperçu de la manière dont il est utilisé à l'intérieur d'un interpréteur Python. Ce code devrait marcher directement sur n'importe quel ordinateur Mac assez récent ; il utilise des *audio units* fournis avec le système d'exploitation.

```
# chargement de la librairie
import pyau
# initialisation de l'hote
h = pyau.Host()
# ajout d'un instrument avec comme synthetiseur "DLSMusicDevice"
t = h.add_track('DLSMusicDevice')
# ajout d'un deuxieme instrument avec le meme synthetiseur
t2 = h.add_track('DLSMusicDevice')
# si un clavier MIDI est branche, la commande suivante
# enverra ses signaux midis au cet instrument
t.arm()
# on peut ajouter un effet
t.add_effect('AUDistortion')
# on peut afficher a tout instant un l'hote ou un instrument
print h
# >> Tracks :
# >> 0: [ DLSMusicDevice ] => [ AUDistortion ] -- ARMED
# >> 1: [ DLSMusicDevice ]
print t
# >> [ DLSMusicDevice ] => [ AUDistortion ] -- ARMED
#
# on peut aussi afficher les parametres d'un audio unit
for p in t.effects[0].get_parameters():
    print p
# >> delay = 4.6 msecs
# >> decay = 9.73
# >> delay mix = 3.4 %
# >> ...
# >> wet/dry mix = 44.6 %
#
# enlevons le dernier instrument
t.remove_last_instrument()
# maintenant, en supposant qu'on a dans le dossier courant
# note0.mid, note1.mid, ..., note150.mid
# on peut "faire jouer" tous ces fichiers midis
# par notre instrument et ecrire les donnees audio
# dans des fichiers wav correspondants
```

```
for i in range(150):
    h.midifile = 'note%i.mid' % i
    h.bounce('note%i.wav' % i)
# on aurait pu egalement retourner sous forme d'un array numpy
```

Ceci n'est qu'un minuscule aperçu des possibilités de *pyau*. Les possibilités sont en effet énormes. D'ailleurs, son potentiel est loin d'être limité à la génération de timbres automatique. Déjà, des collègues de laboratoire l'utilisent à des fins très diverses, par exemple pour la création de stimuli pour des expériences en neuro-psychologie.