Rapport de stage

stage ATIAM

IRCAM, équipe Analyse/Synthèse

Apprentissage de descripteurs audio par Deep learning, application pour la classification en genre musical

Auteur :
Quentin Fresnel

Encadrant :
Geoffroy Peeters

Du 16 février au 31 juillet 2015

Résumé

Ce rapport de fin de stage vise à explorer l'utilisation de réseaux de neurones profonds à la classification en genre musical. Les approches classiques d'indexation audio se déroulent en deux étapes : la première étant l'extraction de descripteurs audios, c'est-à-dire le calcul de nouvelles représentations plus compactes de l'information essentielle, et la deuxième la modélisation des corrélations entre ces descripteurs selon leur catégorie. Ces descripteurs audios sont conçus manuellement par des experts comme par exemple les MFCC, les Chromas ou le contraste spectrale. Avec l'arrivée récente des approches par réseaux de neurones profonds et leur capacité à fournir des représentations de haut niveau, des travaux ont été menés pour tenter d'utiliser ces représentations comme des descripteurs audios. Les résultats obtenus pour des tâches de classification sont parmi les meilleurs aujourd'hui. Notre contribution principale est d'une part l'application de ces méthodes à des descripteurs synchronisées sur les temps et d'autre part de proposer une nouvelle architecture dans laquelle on entraîne un réseau par classe.

Abstract

This master's thesis aims at exploring the use of deep belief networks for musical genre audio classification. Standard methods for audio indexation takes place in two stages : first, new smaller representations of essential information, called audio features, are extracted ; second, a model of correlations between those features according to their class is built. Those audio features are designed manually by experts like e.g. MFCCs, chromas or spectral contrast. With recent advent of deep neural networks and their ability to extract high-level representations, several works tried to use these representations as audio features. The results for classification tasks are among the best today. Our main contribution is in one hand the use of beat-synchrone descriptors and in the other hand to propose a new architecture in which we train one different network for each class.

Remerciements

Je tiens à remercier tout particulièrement mon encadrant de stage, Geoffroy Peeters, qui a su être attentif, disponible et a été d'une aide précieuse tout au long de ce stage. Il m'a fait confiance et m'a épaulé dans l'exploration d'un domaine nouveau pour moi et pour l'équipe.

Je remercie également Frédéric Cornu pour le temps passé à m'expliquer le fonctionnement des serveurs de l'équipe et pour les discussions passionnantes de fin de journée.

Merci à David Doukhan pour les projets musicaux et à Ugo pour son accueil chaleureux, et ses conseils.

Merci également à Philippe Esling pour ses idées et sa vision très claire.

Merci à Léopold, Axel, Hugo, Adrien, Stéphane, Damien, Céline et Stacy, stagiaires de l'Ircam, pour leurs échanges des plus variés.

Pour la relecture attentive de ce manuscrit, je remercie vivement Julien et Victoire.

Je remercie enfin toute l'équipe analyse-synthèse, Kévin et Rémi, mes collègues de bureau ainsi qu'Enrico, Luc, nicolas et Axel pour leur accueil, leurs réponses à mes questions techniques ou pratiques et leur bonne humeur qui ont constitué un cadre idéal pour ce stage.

Table des matières

Abbréviations

CD divergence contrastive (Contrastive Divergence) 17

CQT transformée à Q constant (constant Q transform) 34, 35, 38

DBN Deep Belief Network 9, 33, 39

DNN réseau de neurones profond (Deep Neural Network) 7, 32

GBRBM machine de Boltzmann restreinte gaussienne-binomiale (Gaussian Binomial Restricted Boltzmann Machine) 16

GTZAN Genres Tzanetakis, ensemble de fichiers audios étiquetés par classes et rassemblés par Tzanetakis. 6, 9

MLP perceptron multicouche (Multi Layer Perceptron) 30–33

MRF champs de Markov aléatoire (Markov Random Field) 10

RBM machine de Boltzmann restreinte (Restricted Boltzmann Machine) 10, 12, 16, 17, 33

SVM machine à vecteurs de support (Support Vector Machine) 7

Notations

E énergie du réseau, fonction de $\mathbf{v}$ et $\mathbf{h}$. Elle est définie dans les équations 5 et 13. 11

$\mathbf{W}$ matrice $(N \times K)$ des poids synaptiques d'un RBM. 11

θ vecteur de paramètres à optimiser ($\mathbf{b}$, $\mathbf{c}$ et $\mathbf{W}$ dans le cas du RBM). 13, 14

$\mathbf{b}$ vecteur de taille N de biais visible. 11

$\mathbf{c}$ vecteur de taille K de biais caché. 11

$\mathbf{h}$ vecteur des K unités cachées : $\mathbf{h} = (h_1, \ldots, h_K)$ 11

$\mathbf{v}$ vecteur des N unités visibles : $\mathbf{v} = (v_1, \ldots, v_N)$ 11, 14

$\mathbf{x}$ les lettres minuscules en gras désignent des vecteurs, les lettres majuscules en gras désignent des matrices. 10

$\mathcal{F}$ énergie libre, définie par l'équation 4 11, 14

$\mathcal{N}(\nu, \sigma)$ loi normale de moyenne ν et de variance σ. 22

$\otimes$ multiplication de vecteurs terme à terme : $u \otimes v = (u_i v_i)$	22
$\sum_{\mathbf{v},\mathbf{h}}$ somme sur toutes les valeurs possibles de $\mathbf{v}$ et $\mathbf{h}$ (idem pour $\sum_{\mathbf{v}}$ et $\sum_{\mathbf{h}}$).	11

1 Introduction

1.1 Concepts

1.1.1 Le problème de classification en genre

Pour parler de classification en genre musical, il faut d'abord parler de classification, et de genre musical.

Le genre musical est un concept que l'on comprend facilement mais qui n'est pas vraiment définissable. C'est une notion culturelle et qui n'est pas perçue de la même manière par tout le monde. Les dénominations sont d'origines variées : on reconnaît du rap à sa structure répétitive et à la voix parlée, de la salsa par son rythme, de la musique classique par sa formation, du jazz à la présence de passages improvisés, du rock indépendant par la qualité de l'enregistrement, de l'électro par le le type de source sonore, etc... Autant d'informations de haut niveau que l'on ne peut pas toujours extraire automatiquement du signal sonore brut. François Pachet et Daniel Cazaly en donnent un exemple dans leur étude [36] dans laquelle ils analysent les systèmes de tri par genre de trois sites internet : AllMusicGuide, Amazon et Mp3 afin de définir une nouvelle taxonomie des genres musicaux. Leur constat est que sur environ 1700 genres au total, seuls 70 sont communs aux trois sites. Ils remarquent également que selon les sites, certains meta-genres (genres larges contenant des sous-genres, par exemple "Rock") sont biens plus détaillé que sur d'autres. Enfin des définitions de genres que l'on pourrait qualifier de naturels, par exemple "Rock" ou "Pop" diffèrent. Leur étude s'étend aussi aux systèmes mis en place par les distributeurs de musique pour guider physiquement les clients au sein des rayons ainsi qu'aux techniques utilisées par les entreprises de protection des droits (SACEM par exemple), qui sont également très différents.

Le genre est donc une notion mal définie au départ puisqu'elle n'est pas unique, dépend du contexte d'utilisation et du publique concerné. Bref la culture n'est pas un système linéaire et heureusement.

Un système de classification automatique est un système capable de trier des données d'entrée en leur assignant une ou plusieurs classes. Dans le cas de la classification en genre musical, l'idée est de trier automatiquement des signaux audio en plusieurs classes : les genres musicaux. Ainsi, un système est dépendant des choix faits pour définir les genres possibles. Dans une étude comparant les performances obtenues par des humains et les performances obtenues par des algorithmes de classification [31] automatique, on constate que la précision obtenue par des sujets sur différents ensembles de musique sont entre 75% et 90 % en moyenne. Cela confirme le caractère subjectif du genre musical évoqué dans le paragraphe précédent.

Les premiers travaux de reconnaissance de genre datent de 1995 ([32]), et, comme le souligne Bob L. Sturm dans [44], les nombreux articles dans le domaine se placent en général, de même que le présent travail, dans le cadre de la classification, basée sur une vérité terrain qui est un ensemble de musique annoté en genre. De cette manière, on cherche à résoudre un problème bien définit et qui attribue un score de performance. Mais il faut garder à l'esprit que ce cadre est restrictif et que le fonctionnement d'une solution

sur un ensemble donné ne peut que présager un fonctionnement sur d'autres fichiers audio. La majorité de ces travaux se déroule en deux étapes : le calcul de features adaptées et la classification de ces features, souvent en utilisant des techniques d'apprentissage machine.

1.1.2 L'apprentissage machine, l'intelligence artificielle

L'apprentissage machine est l'art d'apprendre une tâche donnée à une machine en lui fournissant des exemples de données sur lesquelles il s'entraîne. On peut distinguer deux grands types d'apprentissage : l'apprentissage supervisé (méthodes bayésiennes, SVM, k-NN...) et l'apprentissage non supervisé (PCA, RBM, algorithme EM...). Le premier implique d'avoir une base annotée, c'est-à-dire un ensemble de vecteurs d'entraînement dont on connaît la sortie désirée, par exemple une base de musique annotée en genre ; le deuxième, parfois appelé *clustering*, ne nécessite que des données d'entraînement et cherche à représenter les vecteurs d'entrée en estimant leur probabilité jointe. Le *machine learning* est une discipline incontournable qui s'utilise dans des domaines de recherche de plus en plus variés. Le traitement du signal audio en fait naturellement partie.

Le concept d'intelligence artificielle n'est pas nouveau puisque Alan Turing proposait son test d'intelligence dans les années 1950. Sans rentrer dans le détail de l'évolution de l'intelligence artificielle, les techniques récentes de réseaux de neurones profonds ont considérablement relancé la tendance avec une avancée significative : la capacité à apprendre automatiquement des concepts de haut niveau (voir partie 1.3). L'idée d'utiliser ce genre de réseau à des fins de classification musicale est donc assez naturelle puisque comme on l'a vu précédemment, la notion de genre est liée à des éléments d'assez haut niveau. On peut donc espérer que ces approches permettront d'apprendre des descripteurs pertinents, pour la classification.

1.2 Motivations

À l'heure du big data et de l'avènement des services de streaming, la classification automatique, pas seulement en genre musical d'ailleurs, est une nécessité. Les collections numérique de musique, des labels aux plateformes de streaming et de téléchargement en ligne, sont très conséquentes et pour la plupart ont des méta données (c'est-à-dire les informations annexes comme le titre, l'artiste, le style...) hétérogènes voire fausses. Les systèmes capables de les détecter automatiquement sont donc très prisés.

1.3 État de l'art

1.3.1 La classification audio

Comme nous l'avons vu précédemment, la classification en genre musical est devenu un axe de recherche important, avec environ 400 publications depuis 1995 ([44]). Un ensemble d'entraînement de référence utilisé dans ces travaux est la base GTZAN [48] (voir 4.1.1). Très largement utilisée, elle fourni un cadre simple : des extraits sonores de 30 secondes sont répartie de manière équilibrée sur 10 genres musicaux. La plupart

des techniques utilisées se déroulent en deux étapes : l'extraction de descripteurs audio puis la classification de ces descripteurs. On appelle descripteurs audio des nouvelles représentations plus compactes de l'information essentielle. L'étape de classification est souvent effectuée avec une machine à vecteur de support SVM, technique d'apprentissage machine supervisée.

Parmi les meilleurs résultats aujourd'hui [1], on peut citer le travail de Kaichun K. Chang, Jyh-Shing Roger Jang et Costas S. Iliopoulos, qui atteignent une précision de 92,7 % sur la base GTZAN dans [9]. De nombreux descripteurs sont utilisés : MFCC, centroïde spectral, contraste spectral, flux spectral etc... Mais la nouveauté de cette approche réside dans l'utilisation d'une étape de classification basée sur un échantillonnage compressif (compressive sampling).

Une autre technique basée elle aussi sur l'utilisation de descripteurs, a fourni de bons résultats ici au sein de l'équipe analyse-synthèse de l'Ircam : Geoffroy Peeters et Enrico Marchetto [39] ont développé un système basé sur la fusion tardive de quatre systèmes de classifications différents. Ils fonctionnent tous les quatre selon le principe extraction/classification de descripteurs, trois de ces systèmes de classification sont des machines à vecteurs de support (Support Vector Machines) (SVMs) et le dernier est un 5-plus proches voisins. La précision est de 87,4% sur GTZAN.

Une autre approche, celle qui obtient les meilleurs résultats à ce jour [2] est celle de Yannis Panagakis, Constantine Kotropoulos et Gonzalo R. Arce dans [38] et [37]. qui obtiennent une précision de 92,4 % et 93,7%. Ces deux articles proposent deux méthodes très proches qui ne rentrent pas vraiment dans le cadre classique d'extraction/classification de descripteurs. L'idée est ici de construire un dictionnaire redondant à partir d'exemples audio étiquetés et d'un modèle de représentation de signaux audio comparable au système auditif humain. Ces représentations ayant une dimension trop élevée (tenseurs d'ordre 4), une réduction de dimension est nécessaire : c'est là le cœur du travail proposé. Ainsi, les atomes du dictionnaire sont les représentations, après réduction de dimensions, des exemples d'entrainement. La classification se fait ensuite en projetant les exemples de test sur ce dictionnaire et en gardant la décomposition la plus parcimonieuse. Les labels des atomes sur lesquels cette décomposition est faite indiquent la classe du vecteur de test.

1.3.2 Les réseaux de neurones profonds

Les réseaux de neurones sont réapparus en 2006 avec des travaux de Geoffrey Hinton [22], [18] sous un nouveau paradigme, les réseaux de neurones profonds (Deep Neural Networks) (DNNs). Les différences avec les réseaux de neurones utilisés auparavant sont essentiellement que ces modèles sont génératifs et capables d'apprendre de manière non supervisée. La nouveauté est que l'information apprise par les couches les plus profondes est assimilable à des abstractions haut niveau. Ce concept est présent dès les premiers travaux [22] dans lequel les auteurs remarquent que la sortie de la première couche

1. Ces résultats sont remis en doutes par Bob Sturm dans [45]
2. Résultats également remis en question par Bob L. Sturm dans [45]

fourni une représentation bas niveau et que les couches supérieures, formant la *mémoire associative* du réseau, apprend une représentation de plus haut niveau. Cette idée se confirme sur des applications aux images comme par exemple [29]. Les représentations des poids montrent que les premières couches se comportent comme des détecteurs de bords orientés et autres formes géométriques élémentaires quand les couches de plus haut niveau apprennent à reconnaître des éléments, par exemple les yeux ou le nez si on les entraine sur des visages, puis pour la dernière couche des visages presque complets. Plusieurs travaux (comme [28] par exemple) montrent que cette notion de niveaux d'abstractions organisés par couches présente des similarités avec le système cognitif humain, un pas en avant pour l'intelligence artificielle. Elles ne tardent d'ailleurs pas puisqu'en 2012, Alex Krizhevsky, Ilya Sutskever et Geoffrey E. Hinton arrivent en tête du concours ImageNet, un concours de classification d'images naturelles, avec 11% de taux d'erreur d'avance sur la seconde équipe, une avancée significative, grâce à des méthodes de deep learning[27].

Les applications sont nombreuses, des domaines ciblés comme le *Natural langage processing* [10] ou la reconnaissance de parole [30] aux problèmes plus généraux, la réduction de dimension [18] la régression [23] ou la classification [40], le *deep learning* fait écrire.

Ces méthodes proposent donc des résultats compétitifs, une approche inspirée de la biologie, un nouveau paradigme de modèle génératif, de nombreuses applications, il ne manquait plus qu'un article de review complet, que Yoshua Bengio a publié en 2009 : [2]. Les réseaux de neurones profonds sont ainsi devenus en quelques années un axe de recherche prisé et incontournable, utilisé dans tous les domaines.

1.3.3 La classification audio avec des réseaux de neurones profonds

Les applications à la classification audio ne se sont pas faites attendre longtemps : dès 2009, une première tentative est faite avec des réseaux de neurones profonds convolutifs [30]. L'entraînement est fait sur la base ISMIR 2004 (une autre base de référence en reconnaissance de genre musical) de manière non supervisée puis la représentation obtenue est utilisée en entrée d'un système de classification de type SVM. Les résultats obtenus avec les représentations fournies par le réseau sont meilleurs que ceux obtenus à partir des descripteurs classiques (spectre, MFCC). Un autre élément intéressant de ce travail, qui compare l'efficacité du réseau de neurone profond convolutif sur différentes tâches de classification audio, est que dans le cas de la classification de la parole, la première couche agit comme un détecteur de phonèmes. Les réseaux convolutifs permettent de visualiser les descripteurs appris et ici, différents phonèmes sont clairement reconnaissables.

Un autre travail intéressant, celui de Philippe Hamel et Douglas Eck, propose d'utiliser des Deep Belief Networks (DBNs) (réseau utilisé dans ce stage) pour la classification audio en genre. Les vecteurs d'entrées sont des vecteurs de spectre (transformée de Fourier discrète) calculés sur des trames de signal audio. La classification se fait avec un SVM entrainé sur la sortie d'une couche du réseau. La classe est ensuite obtenue en regardant les prédictions de tous les vecteurs d'un titre et en gardant la plus fréquente. Les auteurs obtiennent de meilleures performances en agrégeant les vecteurs sur des périodes assez longue (de l'ordre de 5 secondes). Le résultat obtenu en agrégeant les vecteurs d'entrée

et en entrainant le SVM sur une concaténation des trois couches cachées est de 84,3% de précision.

Deux dernières approches que l'on peut citer sont celles de Sander Dieleman et al. [11] et d'Erik M. Schmidt et Youngmoo E. Kim [42] qui utilisent respectivement un réseau convolutif et un DBN sur un extrait de la base million song database. Les deux approches ont en commun que le réseau de neurones est appliqué sur une double représentation du signal, un vecteur représentant les caractéristiques spectrales du signal et un vecteur représentant les caractéristiques temporelles ; et que les représentations du signal d'entrée sont synchronisés sur le temps (*beat-synchrone*). L'information de placement des temps nécessaire à cette synchronisation sont obtenus automatiquement, par l'algorithme de détection d'*Echonest* dans [11] et par une détection automatique conçue par les auteurs dans [42].

1.4 Plan et contributions

1.4.1 Plan

Nous nous intéressons, dans la partie 2, à la présentation des machines de Boltzmann restreintes et à l'explication de l'algorithme de divergence contrastive. Les formules sont interprétées et plusieurs calculs sont menés en détail afin de réellement comprendre les procédés et leur justification.

La partie 3 présente rapidement les origines des réseaux de neurones profonds et explique ensuite le fonctionnement des Deep Belief Network (DBN), architecture utilisée lors de ce stage.

Enfin la partie 4 décrit les tests effectués, les données utilisées, et les prétraitements appliqués. La transformée à Q constant ainsi que la transformée de Box-Cox sont présentées.

La conclusion 5 est une synthèse rapide de ce rapport et une occasion de proposer des améliorations du système et des pistes qui n'ont pas pu être explorées pendant le stage.

1.4.2 Contributions

Nos contributions sont principalement les annotations des temps sur une de la base GTZAN, l'application d'un DBN à des données Beat-synchrone et enfin l'application de la transformée de Box-Cox au données avant de les entrer dans le réseau afin de respecter au mieux l'hypothèse gaussienne. Malgré les difficultés pour obtenir des résultats au niveau de l'état de l'art, nous proposons des résultats relatif afin d'évaluer l'efficacité des procédés proposés.

2 Machines de Boltzmann restreintes

Cette partie assez conséquente a pour but de présenter précisément les machines de Boltzmann restreintes. Les calculs sont menés en détail car dans beaucoup d'articles, on trouve les formules principales des modèles à énergie ainsi que les formules de mise à jour mais peu d'explications supplémentaires.

2.1 Modèle

2.1.1 Modèles graphiques

Les machines de Boltzmann restreintes (Restricted Boltzmann Machines) (RBMs) sont des cas particuliers des modèles graphiques, et plus précisément des champs de Markov. Cette partie est inspirée de [15], qui constitue une bonne introduction (voir [6], [26] pour plus d'informations).
Les modèles graphiques sont des modèles probabilistes dans lesquels les relations de dépendance conditionnelle et d'indépendance des variables aléatoires sont modélisées par un graphe. Chacun des nœuds n_i de ce graphe représente une variable aléatoire x_i. Parmi les modèles graphiques (factor graphs, Bayesian networks...), nous nous intéressons ici aux champs de Markov aléatoires (MRFs) autrement appelés réseaux de Markov ou modèles graphiques non dirigés. La propriété de Markov, qui définit ces modèles, stipule que chaque nœud est indépendant de chacun des autres nœud conditionnellement à son voisinage directe (voir figure 1).

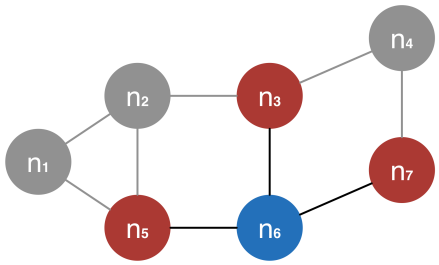


FIGURE 1 – Exemple de modèle graphique, le nœud n_6 est indépendant des nœuds n_1 n_2 et n_4.

Dans ces modèles, et Le théorème d'Hammersley-Clifford nous permet d'établir qu'il existe une fonction d'énergie E telle que la probabilité d'un vecteur $\mathbf{x}= (x_1, \ldots, x_n)$ peut s'exprimer comme :

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

avec Z une constante de normalisation appelée fonction de partition. Cette distribution est la *distribution de Gibbs*.

Une machine de Boltzmann est un modèle graphique dans lequel les nœuds sont divisés en 2 groupes : les unités visibles et les unités cachées, états propres au réseau sensés modéliser des corrélations existant entre les unités visibles.

2.1.2 Modèles à énergie

Les modèles à énergie sont une famille de modèles probabilistes dont la probabilité peut s'exprimer comme une distribution de Gibbs. Introduisons le concept d'unités visibles et d'unités cachées [3], on note $\mathbf{v}$ le vecteur des N unités visibles, $\mathbf{h}$ le vecteur des K unités cachées et E l'énergie du réseau.

$$p(\mathbf{v}, \mathbf{h}) = \frac{e^{-E(\mathbf{v},\mathbf{h})}}{Z} \tag{1}$$

avec Z la fonction de partition :

$$Z = \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \tag{2}$$

Où $\sum_{\mathbf{v},\mathbf{h}}$ signifie somme sur tous les $\mathbf{v}$ et $\mathbf{h}$ possibles. Cette notation est un abus car les unités visibles peuvent être continues. Dans ce cas, la somme représente en fait l'intégrale $\int_{\mathbf{v}} d\mathbf{v}$ sur toutes les valeurs possibles de $\mathbf{v}$. On définit l'énergie libre $\mathcal{F}$ de manière à avoir :

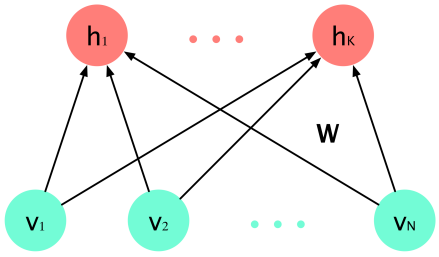
$$p(\mathbf{v}) = \frac{e^{-\mathcal{F}(\mathbf{v})}}{Z} \tag{3}$$

Donc

$$\mathcal{F}(\mathbf{v}) = -\log \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \tag{4}$$

On remarque que la fonction de partition Z est souvent impossible à calculer car elle implique toutes les configurations possibles de $\mathbf{v}$ et $\mathbf{h}$, cela signifie une somme à 2^{N+K} termes, donc une complexité exponentielle, mais elle est constante par rapport à $\mathbf{v}$ et $\mathbf{h}$.

Dans le cas des machines de Boltzmann restreintes, il n'y a aucune connexion entre les unités d'une même couche, comme représenté ci-après



Notons $\mathbf{b}$ le vecteur $(b_1, b_2, \ldots, b_J)$ de biais des unités visibles, $\mathbf{c}$ le vecteur $(c_1, c_2, \ldots, c_J)$ de biais des unités cachées et $\mathbf{W}$ la matrice $W_{(i,k)}$ de poids. L'énergie E vaut alors

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{b}'\mathbf{v} - \mathbf{c}'\mathbf{h} - \mathbf{v}'\mathbf{W}\mathbf{h} \tag{5}$$

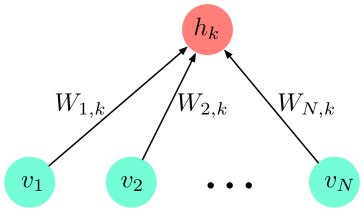
3. On appellera indifféremment neurone ou unité les nœuds du réseau de neurones.

On peut interpréter cette formulation comme un produit d'expert, notion introduite par Geoffrey Hinton dans [19]. On parle de produit d'expert lorsque l'énergie peut se mettre sous la forme

$$E(\mathbf{x}) = \sum_i f_i(\mathbf{x})$$

où chaque expert f_i agit sur $\mathbf{x}$ comme un détecteur de configuration.

Ici, la contribution de chaque neurone caché à l'énergie totale est une combinaison linéaire de ses entrées plus un biais.



$$E_k(\mathbf{v}) = -\sum_{i=1}^{N} W_{i,k} v_i - c_k = -(\mathbf{v}'\mathbf{W})_k - c_k$$

Alors on peut écrire que

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{b}'\mathbf{v} + \sum_{k=1}^{K} h_k E_k(\mathbf{v})$$

On est donc dans un cas de produit d'experts : les neurones cachés peuvent être vus comme des détecteurs de configuration de $\mathbf{v}$. Pour des unités cachées binaires, la valeur 1 signifie que l'expert détecte que le vecteur $\mathbf{x}$ est dans une certaine configuration.

2.2 Optimisation

2.2.1 Descente de gradient

La minimisation de l'énergie libre d'un RBM est un problème complexe qui n'a pas de solution analytique. L'entraînement se fait donc avec une descente de gradient sur une fonction de coût. Les paramètres sont initialisés *aléatoirement* (voir 2.4), ce qui constitue un point de départ dans l'espace des paramètres, puis ils sont mis à jour de manière à minimiser cette fonction. Cela signifie que le minimum obtenu en cas de convergence est local et non optimal a priori.

Il existe beaucoup de techniques différentes pour effectuer une descente de gradient mais l'une des premières questions à se poser est la fréquence des mises à jour. On peut par exemple, mettre à jour les paramètres à chaque nouvel exemple d'entraînement (descente de gradient *on-line*). On peut sinon créer des ensembles d'exemples, appelé batchs, calculer les mises à jour sur chacun des exemples d'un batch puis appliquer une

mise à jour qui est la moyenne de toutes ces mises à jour temporaires. Autrement dit on calcule les directions qui devraient être prises pour chacun des exemples d'entraînement du batch puis on prend une direction moyenne. On appelle traditionnellement *batch gradient* une descente de gradient dont la mise à jour est calculée sur l'ensemble des exemples d'entraînement. Dans notre cas, on créé des sous-ensembles de quelques exemples (de l'ordre de la centaine). On appelle parfois ces ensembles des mini-batchs, par opposition au *batch gradient*.

L'intérêt des minibatchs est que cela régularise un peu l'apprentissage : on fait moins de mises à jour mais elles sont supposées plus pertinentes. Mais la vrai différence vient du fait que l'on peut paralléliser les calculs des mises à jour sur tous les exemples du minibatch. La différence de temps de calcul est significative.

2.2.2 Dérivation de la fonction de coût

La fonction de coût est basée sur la log-vraisemblance. Soit $S = \{\mathbf{v}_1, \ldots, \mathbf{v}_l\}$ un ensemble de vecteurs d'entraînement et $\theta \in \Theta$ un vecteur de paramètres du réseau dans l'espace de paramètres. La vraisemblance $\mathcal{L}(\theta \mid S)$ vaut :

$$\mathcal{L}(\theta|S) = p(S|\theta) = \prod_{1 \leq s \leq l} p(\mathbf{v}_s|\theta)$$

Donc la log-vraisemblance vaut (le signe moins permet d'avoir une fonction à minimiser) :

$$-\ln \mathcal{L}(\theta|S) = -\sum_{s=1}^{l} \ln p(\mathbf{v}_s|\theta) = \sum_{s=1}^{l} \mathcal{L}(\theta|\mathbf{v}_s)$$

Exprimons le terme de log-vraisemblance d'un vecteur d'entraînement $\mathbf{v}$:

$$\begin{aligned} -\ln \mathcal{L}(\theta|\mathbf{v}) &= \ln p(\mathbf{v}|\theta) \\ &= \ln \frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \\ &= \ln \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} - \ln \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \end{aligned}$$

Pour connaître le pas d'une étape de descente de gradient, on dérive la fonction de coût par rapport à θ (qui est continue et dérivable) :

$$\begin{aligned} -\frac{\partial \ln \mathcal{L}(\theta|\mathbf{v})}{\partial \theta} &= -\frac{\partial}{\partial \theta} \ln \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} + \frac{\partial}{\partial \theta} \ln \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \\ &= -\frac{1}{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}} \sum_{\mathbf{h}} \frac{\partial e^{-E(\mathbf{v},\mathbf{h})}}{\partial \theta} + \frac{1}{\sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}} \sum_{\mathbf{v},\mathbf{h}} \frac{\partial e^{-E(\mathbf{v},\mathbf{h})}}{\partial \theta} \\ &= \frac{1}{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}} \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial \theta} - \frac{1}{\sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}} \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial \theta} \end{aligned} \tag{6}$$

L'énergie libre d'un vecteur d'entrée $\mathbf{v}$ est une grandeur caractéristique. Elle est proportionnelle à $p(\mathbf{v})$ et est simple à calculer. Dérivons maintenant l'énergie libre par rapport à θ afin d'exprimer la dérivée de la log-vraisemblance $\mathcal{L}(\theta|\mathbf{v})$ par rapport à $\mathcal{F}(\mathbf{v})$. D'après l'équation 4 :

$$\frac{\partial \mathcal{F}(\mathbf{v})}{\partial \theta} = \frac{1}{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}} \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial \theta} \tag{7}$$

On remarque que la dérivée de l'énergie libre est égale au premier terme du membre de droite de l'équation 6.

De plus, en reprenant les équations 1 et 7, on a :

$$p(\mathbf{v}) = \sum_{\mathbf{h}} p(\mathbf{v},\mathbf{h}) = \frac{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}}{\sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}} \tag{8}$$

Donc d'après les expressions 7 et 8,

$$E_p\left[\frac{\partial \mathcal{F}(\mathbf{v})}{\partial \theta}\right] = \sum_{\mathbf{v}} p(\mathbf{v}) \frac{\partial \mathcal{F}(\mathbf{v})}{\partial \theta} = \frac{1}{\sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}} \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial \theta} \tag{9}$$

On remarque que cette expression est égale au deuxième terme du membre de gauche de l'équation 6.

Donc d'après les équations 7 et 9 on peut réécrire l'expression 6 :

$$-\frac{\partial \ln \mathcal{L}(\theta|\mathbf{v})}{\partial \theta} = \underbrace{\frac{\partial \mathcal{F}(\mathbf{v})}{\partial \theta}}_{\text{phase positive}} \underbrace{-E_p\left[\frac{\partial \mathcal{F}(\mathbf{v})}{\partial \theta}\right]}_{\text{phase négative}} \tag{10}$$

L'équation 10 comporte deux termes : la phase positive et la phase négative.

La phase positive s'exprime comme la dérivée de l'énergie libre $\mathcal{F}$ en fonction des paramètres θ appliquée au vecteur d'entraînement $\mathbf{v}$. Elle facilement calculable avec l'expression de l'énergie libre en fonction des paramètres (équation 12 dans le cas binaire ou 14 dans le cas gaussien).

La phase négative, elle, implique une espérance E_p selon la loi de probabilité p du modèle, c'est-à-dire la probabilité $p(\mathbf{v})$ que le réseau assigne au vecteur $\mathbf{v}$. Pour être calculée de manière exacte, elle implique le calcul d'une somme (une intégrale à autant de dimensions que d'unités visibles dans le cas gaussien) sur toutes les configurations de $\mathbf{v}$ possibles. Ce calcul étant trop lourd, il nécessite des approximations (voir chapitre contrastive divergence 2.3.1)

On remarque que cette expression contient des dérivées par rapport à θ. On peut assez facilement caluler cette dérivée pour $\theta = W$, b ou c [4]. Je ne le fais pas ici car grâce à la librairie theano [5] qui permet d'effectuer du calcul symbolique, l'équation est

4. Voir deeplearning.net/tutorial/rbm.html
5. Voir deeplearning.net/software/theano/

directement exploitable. En effet la fonction `theano.tensor.grad` calcule le gradient d'une expression symbolique en fonction de la variable demandée.

L'objectif de l'apprentissage est de faire croître la log-vraisemblance des exemples d'entraînements, donc trouver un jeu de paramètres que la dérivée de la fonction de coût soit négative. L'idée est ainsi d'avoir une phase positive faible, donc de diminuer l'énergie libre de l'exemple d'entraînement $\mathcal{F}(\mathbf{v})$ et donc d'augmenter la probabilité $p(\mathbf{v})$; et une phase négative élevée, donc de faire croître l'énergie libre des vecteurs générés par le réseau, donc de diminuer leur probabilité. En d'autres termes, mettre à jour le réseau revient à lui montrer de bons exemples : les données d'entraînement, et de mauvais exemples : les vecteurs qu'il génère lui-même.

2.2.3 Factorisation de l'énergie libre

L'intérêt de cette formulation réside dans la factorisation de l'énergie libre.

D'après les équations 4 et 5, on a :

$$\mathcal{F}(\mathbf{v}) = -\log \frac{1}{Z} \sum_{\mathbf{h}} e^{(\mathbf{b}'\mathbf{v}+\mathbf{c}'\mathbf{h}+\mathbf{v}'\mathbf{W}\mathbf{h})}$$

On peut distinguer trois termes :

$$\mathcal{F}(\mathbf{v}) = \underbrace{-\mathbf{b}'\mathbf{v}}_{B\text{ : influence du biais}} \quad \underbrace{-\ln \sum_{\mathbf{h}} e^{(\mathbf{c}'+\mathbf{v}'\mathbf{W})\mathbf{h}}}_{H\text{ : influence des unités cachées}} + \underbrace{\ln Z}_{C\text{ : terme constant}} \tag{11}$$

Développons un peu le terme H :

$$\begin{aligned} H &= -\ln \sum_{\mathbf{h}} e^{(\mathbf{c}+\mathbf{v}'\mathbf{W})\mathbf{h}} \\ &= -\ln \sum_{\mathbf{h}} e^{\sum_k (c_k+(\mathbf{v}'\mathbf{W})_k)h_k} \\ &= -\ln \sum_{\mathbf{h}} \prod_k e^{(c_k+(\mathbf{v}'\mathbf{W})_k)h_k} \end{aligned}$$

La somme sur tous les $\mathbf{h}$ possibles se décompose comme suit, avec $\sum_{h_k}$ la somme sur toutes les valeurs de h_k possibles :

$$\begin{aligned} e^{-H} &= \sum_{h_1}\sum_{h_2}\cdots\sum_{h_K}\prod_k e^{(c_k+(\mathbf{v}'\mathbf{W})_k)h_k} \\ &= \left(\sum_{h_1} e^{(c_1+(\mathbf{v}'\mathbf{W})_1)h_1}\right)\left(\sum_{h_2} e^{(c_2+(\mathbf{v}'\mathbf{W})_2)h_2}\cdots\right)\left(\sum_{h_K} e^{(c_K+(\mathbf{v}'\mathbf{W})_K)h_K}\right) \\ &= \prod_k \sum_{h_k} e^{(c_k+(\mathbf{v}'\mathbf{W})_k)h_k} \end{aligned}$$

C'est cette factorisation qui fait l'intérêt du modèle, qui peut être interprété comme un produit d'experts. Dans le cas binomial, $h_i \in \{0, 1\}$. Donc $\sum_{h_i} f(h_i) = f(0) + f(1)$, donc

$$H = -\ln \prod_k 1 + e^{(c_k + (\mathbf{v}'\mathbf{W})_k) h_k} = -\sum_k \ln \left(1 + e^{(c_k + (\mathbf{v}'\mathbf{W})_k) h_k}\right)$$

Ainsi,

$$\mathcal{F}(\mathbf{v}) = \underbrace{-\mathbf{b}'\mathbf{v}}_{B} \underbrace{-\sum_k \ln \left(1 + e^{(c_k + (\mathbf{v}'\mathbf{W})_k) h_k}\right)}_{H} + \underbrace{\ln Z}_{C} \tag{12}$$

2.2.4 Cas des unités visibles gaussiennes

Les raisonnements précédents impliquent des unités binaire mais dans notre cas, les vecteurs d'entraînement prennent des valeurs continues. Geoffrey Hinton dans [18] et Yoshua Bengio dans [3], [2] ont proposé d'utiliser des unités visible gaussiennes, c'est à dire des unités visibles pouvant prendre des valeurs continues et modélisées par une gaussienne.

On appellera machines de Boltzmann restreintes gaussiennes-binomiales (Gaussian Binomial Restricted Boltzmann Machines) (GBRBMs) pour préciser lorsque la couche visible est composée d'unités gaussiennes. Dieleman, dans [11], utilise des unités visibles gaussiennes pour son RBM sur des descripteurs de timbre. Il utilise des unités binaires sur les chromas car même si les chomas sont compris entre 0 et 1, l'interprétation en probabilité n'est pas absurde. En effet, l'auteur explique que l'apprentissage est bien plus stable avec des unités binaires. Par ailleurs, il respecte la recommandation de Geoffrey Hinton concernant le pas d'apprentissage plus faible pour les unités réelles. Lee et Largman dans [30] posent les définitions avec les valeurs réelles et avec les valeurs binaires sans en dire plus. Il semble qu'ils utilisent les entrées à valeurs réelles. Erik M. Schmidt et Youngmoo E. Kim dans [42] utilisent aussi des entrées gaussiennes.

Dans le cas où les unités visibles sont gaussiennes, l'expression de l'énergie (5) est un peu modifiée :

$$E(\mathbf{v}, \mathbf{h}) = \sum_i \frac{(v_i - b_i)^2}{2\sigma_i} - \mathbf{c}'\mathbf{h} - \sum_{k,i} h_k W_{k,i} \frac{v_i}{\sigma_i}$$

En considérant que les gaussiennes sont centrée et de variance 1, on a :

$$E(\mathbf{v}, \mathbf{h}) = \sum_i \frac{(v_i - b_i)^2}{2} - \mathbf{c}'\mathbf{h} - \mathbf{h}'\mathbf{W}\mathbf{v} \tag{13}$$

Dans ce cas, seul le terme B dans l'expression de l'énergie libre (11) est modifié, ce qui donne :

$$\mathcal{F}(\mathbf{v}) = \underbrace{\sum_i \frac{(v_i - b_i)^2}{2}}_{B} \underbrace{-\sum_k \ln \left(1 + e^{h_k (c_k + (\mathbf{W}\mathbf{v})_k)}\right)}_{H} + \underbrace{\ln Z}_{C} \tag{14}$$

2.3 Approximations

2.3.1 Algorithme de divergence contrastive

L'algorithme de divergence contrastive (Contrastive Divergence) (CD) est le point clef permettant d'entrainer les RBMs. C'est Geoffrey Hinton qui a développé cette technique qui apparait dans ces travaux à partir de 2001. L'article fondateur est écrit en 2002 [20] et démontre l'amélioration des résultats sur des tâches de reconnaissance de chiffres (base MNIST [6]) en utilisant cette technique pour entrainer des RBM.

Revenons à l'équation 10. Le terme de phase négative n'est pas calculable à cause de l'espérance E_p impliquant une somme sur tous les $\mathbf{v}$ possibles. La *première approximation* de l'algorithme de CD consiste à remplacer cette somme par un terme, calculé avec un vecteur visible généré par le réseau $\mathbf{v}^*$ selon la probabilité $p(\mathbf{v})$:

$$E_p\left[\frac{\partial \mathcal{F}(\mathbf{v})}{\partial \theta}\right] = \sum_{\mathbf{v}} p(\mathbf{v}) \frac{\partial \mathcal{F}(\mathbf{v})}{\partial \theta} \simeq \frac{\partial \mathcal{F}(\mathbf{v}^*)}{\partial \theta}$$

Cette approximation peut se justifier car d'une part les mises à jours par mini-batchs (voir partie 2.2.1) vont moyenner cette estimation de phase négative sur plusieurs chaînes de sampling et d'autre part les mises à jour sont très rapprochées, ce qui a encore un effet lissant.

Le problème n'est pas résolu pour autant car pour calculer la probabilité $p(\mathbf{v})$, il faut calculer la fonction de partition Z (voir équation 3) qui implique une somme sur tous les $\mathbf{v}$ et tous les $\mathbf{h}$ possibles. Cependant, il existe une technique d'échantillonnage, le *Gibbs sampling* (voir partie 2.3.3), permettant d'obtenir un sample généré par le réseau selon la probabilité $p(\mathbf{v})$ sans la connaître explicitement.

La *seconde approximation* de l'algorithme CD est la suivante : pour obtenir un sample $\mathbf{v}^*$ selon $p(\mathbf{v})$, il faudrait théoriquement une infinité d'itérations de l'algorithme de Gibbs sampling. L'approximation consiste à n'en faire qu'un nombre k fini, souvent de l'ordre de la dizaine, en prenant un vecteur d'entraînement comme état initial. Geoffrey Hinton a même remarqué ([20]) qu'une seule itération suffisait parfois à obtenir des résultats satisfaisants

Avant de rentrer dans les détails de l'algorithme d'échantillonnage de Gibbs, ont peut relever quelques remarques concernant la convergence de cet algorithme qui ont été établies dans [47]. La convergence de la procédure de contrastive est encore un sujet ouvert. Il a été prouvé qu'il n'existe aucune fonction dont cette mise à jour est le gradient. On ne peut donc pas analyser son comportement simplement comme dans des algorithmes d'optimisation de fonction de coût. Ilya Sutskever et Tijmen Tieleman ont même exhibé un cas où l'algorithme boucle indéfiniment et dont la non convergence est assurée. La bonne nouvelle de ces travaux est qu'il existe de manière certaine des points fixes et donc l'algorithme peut converger.

6. voir http://yann.lecun.com/exdb/mnist/

2.3.2 Les chaînes de Markov de Monte Carlo

Le gibbs sampling est une méthode de la famille des Monte Carlo Markov Chain (MCMC). Ces approches sont hybrides :

Méthodes de Monte Carlo L'objectif est de calculer des quantités intéressantes (espérance, variance...) d'une distribution de probabilité inconnue. L'hypothèse est d'être capable de tirer des échantillons de cette distribution. Les quantités à estimer s'expriment sous forme d'intégrales :

$$I = \int g(X)p(X)dX$$

avec g une fonction ($g(X) = X$ pour l'espérance, $g(X) = (X - E(X))^2$ pour la variance par exemple). L'estimateur de Monte Carlo est le suivant :

$$\hat{I}_M = \frac{1}{M}\sum_{i=1}^{M} g(X^{(i)})$$

La convergence de $\hat{I}_M$ vers I quand M tend vers l'infini est assurée par la loi des grands nombres forte (SLLN pour Strong Law of Large Numbers). Cette loi affirme que si $X_1, X_2, ...$ est une séquence de variables aléatoires indépendantes et identiquement distribuées dont la moyenne est μ, alors

$$\frac{X_1 + X_2 + ... + X_M}{M} \to \mu \text{ quand } M \to \infty$$

Chaîne de Markov Une Chaîne de Markov est un processus aléatoire dont les états futurs sont indépendants des états passés sachant l'état présent.

On note $\Pi^{(t)}$ l'état de la chaîne de Markov à l'instant t :

$$\Pi^{(t)} = (p(X^{(t)} = X_1), p(X^{(t)} = X_2), \ldots, p(X^{(t)} = X_N)$$

L'intérêt de cette formulation est que $\Pi^{(t)}$ suit une loi géométrique de raison P la matrice des probabilités de transition :

$$P = \Big(p(X^{(t+1)} = X_j \mid X^{(t)} = X_i)\Big)_{(i,j)\in N^2}$$

Ainsi,

$$\Pi^{(t+1)} = \Pi^{(t)}P$$

On définit la *distribution stationnaire* π d'une chaîne de Markov comme un point d'arrêt de cette suite, i.e. $\pi = \pi P$.

Hybride Les méthodes de Monte Carlo nécessitent, d'après la SLLN, des variables aléatoires indépendantes. Ce n'est pas le cas des chaînes de Markov. Nous allons donc utiliser un autre théorème appelé *théorème ergodique*.

Si une chaîne de Markov est apériodique, irréductible et positive récurrente alors :

$$\frac{1}{M}\sum_{i=1}^{M} g(X_i) \to \int_X g(X)\pi(X)dX$$

quand $M \to \infty$ avec π la distribution stationnaire.

Ainsi, les échantillons obtenus en utilisant une chaîne de Markov, bien que dépendants, convergent vers la distribution stationnaire de cette chaîne. On peut ne garder que tous les d échantillons pour obtenir des données plus indépendantes, et on peut ignorer un certain nombre d'échantillons au début pour s'éloigner de l'état initial.

2.3.3 Échantillonnage de Gibbs

Nous allons définir ce qu'est la méthode d'échantillonnage de Gibbs appliqué à un champs de Markov, c'est-à-dire montrer que l'on peut échantillonner les variables aléatoires d'un MRF en utilisant une MCMC dont la distribution stationnaire est la distribution de Gibbs (voir 2.1.1). Ensuite nous verrons comment l'appliquer dans le cas concret d'une machine de Boltzmann restreinte.

On définit une chaîne de Markov dont les vecteurs d'état sont les variables aléatoires d'un champs de Markov $\mathbf{X}^{(t)} = (X_1^{(t)}, \ldots, X_N^{(t)})$ à un instant t. On note $\mathbf{X}_{-i} = (X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_N)$ le vecteur $\mathbf{X}$ auquel on enlève X_i et $p(\mathbf{X})$ la distribution de Gibbs associée. L'échantillonnage de Gibbs se déroule en deux étapes :

1. Choisir $i \in \{1, \ldots, N\}$ selon la loi $q(i)$ qui est strictement positive, par défaut une loi uniforme

2. Mettre à jour X_i en utilisant la probabilité $p(X_i|\mathbf{X}_{-i})$.

Alors on peut définir la probabilité $P_{\mathbf{X}\mathbf{X}'}$ de transition d'un état $\mathbf{X}$ à un état $\mathbf{X}'$:

$$P_{\mathbf{X}\mathbf{X}'} = \begin{cases} q(i)p(X_i'|\mathbf{X}_{-i}) & \text{si } \exists\, i \in \{1, \ldots, N\} \text{ tel que } \mathbf{X}_{-i} = \mathbf{X}'_{-i} \\ 0 & \text{sinon.} \end{cases} \tag{15}$$

Montrons maintenant que la distribution $p(\mathbf{X})$ est la distribution stationnaire de cette chaîne de Markov. Pour cela, on définit la condition de bilan détaillé. Pour deux états $\mathbf{X}$ et $\mathbf{X}'$, la distribution π satisfait cette condition si :

$$\pi(\mathbf{X})P_{\mathbf{X}\mathbf{X}'} = \pi(\mathbf{X}')P_{\mathbf{X}'\mathbf{X}} \tag{16}$$

Montrons que la condition 16 est suffisante pour assurer que la distribution π est stationnaire. Soit N le nombre d'états possible, indexés par les entiers naturels dans

$\{1, \ldots, N\}$, la propriété 16 devient :

$$\pi(i)P_{ij} = \pi(j)P_{ji}$$

Donc,

$$16 \Rightarrow \sum_{i=1}^{N} \pi(i)P_{ij} = \sum_{i=1}^{N} \pi(j)P_{ji} \Rightarrow \sum_{i=1}^{N} \pi(i)P_{ij} = \pi(j) \Rightarrow \pi P = \pi$$

Or, comme démontré dans [15], la chaîne de Gibbs décrite précédemment est satisfait la condition de bilan détaillé. Soient $\mathbf{X}, \mathbf{X'}$ et i tel que $\mathbf{X'}_{-i} = \mathbf{X}_{-i}$:

$$\begin{aligned}
p(\mathbf{X})P_{\mathbf{X}\mathbf{X'}} &= p(\mathbf{X})q(i)p(X'_i|\mathbf{X}_{-i}) \\
&= p(\mathbf{X}_i, \mathbf{X}_{-i})q(i)\frac{p(X'_i, \mathbf{X}_{-i})}{p(\mathbf{X}_{-i})} \\
&= \frac{p(\mathbf{X}_i, \mathbf{X'}_{-i})}{p(\mathbf{X'}_{-i})}q(i)p(X'_i, \mathbf{X}_{-i}) \\
&= p(X_i|\mathbf{X'}_{-i})q(i)p(\mathbf{X'}) \\
&= p(\mathbf{X'})P_{\mathbf{X'}\mathbf{X}}
\end{aligned}$$

Donc la distribution p définie plus haut est bien la distribution stationnaire π de la chaîne de Gibbs.

Nous avons vu dans la partie 2.3.2 que pour assurer la convergence d'une chaîne de Markov vers sa distribution stationnaire π, il fallait que la chaîne soit apériodique, irréductible et positive récurrente. Ces propriétés sont vérifiées et se déduisent en partie de la stricte positivité de $q(i)$ et de $p(X'_i|\mathbf{X}_{-i})$ (pour plus de détails, voir [15])

Ainsi, le théorème ergodique (partir 2.3.3) s'applique à cette chaîne. Cela signifie que la méthode d'échantillonnage de Gibbs peut être utilisée pour obtenir des échantillons de la distribution de Gibbs $p(\mathbf{X}) = \frac{e^{-E(\mathbf{X})}}{Z}$. Il faudrait en théorie une infinité d'itérations pour obtenir des samples de la distribution exacte.

Application au cas des RBM Dans le cas des machines de Boltzmann restreintes, la procédure peut être simplifiée. Soient $\mathbf{X}$ et $\mathbf{X'}$ deux états et $i \in \{1, \ldots, N\}$ tels que $\mathbf{X}_{-i} = \mathbf{X'}_{-i}$. On peut développer $\mathbf{X}$ (resp. $\mathbf{X'}$) de la manière suivante :

$$\mathbf{X} = (v_1, \ldots, v_N, h_1, \ldots, h_K)$$

Dans le cas où la variable aléatoire à mettre à jour $X_i = v_n$ est visible, on peut écrire :

$$\mathbf{X}_{-i} = (v_1, \ldots, v_{n-1}, v_{n+1}, \ldots, v_N, h_1, \ldots, h_K)$$

Dans un RBM, il n'existe aucune connexion entre les unités d'une même couche. Donc v_n est indépendante de $(v_1, \ldots, v_{n-1}, v_{n+1}, \ldots, v_N)$. L'équation 15 devient donc :

$$P_{\mathbf{X}\mathbf{X'}} = q(i)p(v'_n|\mathbf{h})$$

Avec $\mathbf{h} = (h_1, \ldots, h_K)$.
De même, si $X_i = h_k$ est une unité cachée, on peut écrire :

$$P_{\mathbf{X}\mathbf{X}'} = q(i)p(h'_k|\mathbf{v})$$

Ainsi, la mise à jour d'une unité visible est indépendante de toutes les autres unités visibles. On peut donc effectuer la mise à jour de toutes les unités visibles en une seule étape, selon un vecteur de probabilités :

$$\mathbf{P}_{\mathbf{v}|\mathbf{h}} = (p(v_1|\mathbf{h}), \ldots, p(v_N|\mathbf{h}))$$

De même pour les unités cachées :

$$\mathbf{P}_{\mathbf{h}|\mathbf{v}} = (p(h_1|\mathbf{v}), \ldots, p(h_K|\mathbf{v}))$$

En reprenant les notations de [2], où $\sim$ signifie "est échantillonné à partir" et $\hat{\mathbf{P}}(\mathbf{x})$ signifie la probabilité empirique, k itérations de l'algorithme d'échantillonnage de Gibbs peuvent être résumées de la manière suivante :

$$\begin{aligned}
\mathbf{v}_1 &\sim \hat{\mathbf{P}}(\mathbf{v}) \\
\mathbf{h}_1 &\sim \mathbf{P}_{\mathbf{h}|\mathbf{v}_1} \\
\mathbf{v}_2 &\sim \mathbf{P}_{\mathbf{x}|\mathbf{h}_1} \\
\mathbf{h}_1 &\sim \mathbf{P}_{\mathbf{h}|\mathbf{v}_2} \\
&\vdots \\
\mathbf{x}_{k+1} &\sim \mathbf{P}_{\mathbf{x}|\mathbf{h}_k}
\end{aligned}$$

Calcul des probabilités Exprimons maintenant $\mathbf{P}_{\mathbf{v}|\mathbf{h}}$ en fonction des états cachés, visibles et des paramètres. Pour ce faire, nous allons d'abord calculer $p(v_i|\mathbf{h})$ en factorisant $p(\mathbf{v}|\mathbf{h})$

$$\begin{aligned}
p(\mathbf{v}|\mathbf{h}) &= \frac{p(\mathbf{v}, \mathbf{h})}{p(\mathbf{h})} \\
&= \frac{\exp(-E(\mathbf{v}, \mathbf{h}))}{\sum_{\mathbf{v}} e^{-E(\mathbf{v}, \mathbf{h})}} \\
&= \frac{\exp(\mathbf{b}'\mathbf{v} + \mathbf{c}'\mathbf{h} + \mathbf{v}'\mathbf{W}\mathbf{h})}{\sum_{\mathbf{v}} \exp(\mathbf{b}'\mathbf{v} + \mathbf{c}'\mathbf{h} + \mathbf{v}'\mathbf{W}\mathbf{h})} \\
&= \prod_{i=1}^{N} \frac{\exp(v_i(b_i + (\mathbf{W}\mathbf{h})_i))}{\sum_{v_i} \exp(v_i(b_i + (\mathbf{W}\mathbf{h})_i))}
\end{aligned}$$

Or nous savons par la formulation du produit d'experts (voir partie 2.1.2) que

$$p(\mathbf{v}|\mathbf{h}) = \prod_{i=1}^{N} p(v_i|\mathbf{h})$$

Chaque terme du produit ne dépend que d'une unité visible v_i. Donc on peut faire l'identification suivante

$$p(v_i|\mathbf{h}) = \frac{\exp(v_i(b_i + (\mathbf{Wh})_i))}{\sum_{v_i} \exp(v_i(b_i + (\mathbf{Wh})_i))} \tag{17}$$

Dans le cas binaire, les seules valeurs possibles de v_i sont 0 et 1. Donc

$$p(v_i = 1|\mathbf{h}) = \frac{\exp(b_i + (\mathbf{Wh})_i)}{1 + \exp(b_i + (\mathbf{Wh})_i)} = sigm(b_i + (\mathbf{Wh})_i)$$

Avec le même raisonnement, on obtient aussi

$$p(h_k = 1|\mathbf{v}) = sigm(c_k + (\mathbf{vW})_k)$$

Notons $\otimes$ le produit de vecteurs termes à termes. Alors les vecteurs de probabilités $\mathbf{P_{h|v}}$ et $\mathbf{P_{v|h}}$ valent :

$$\mathbf{P_{h|v}} = sigm(\mathbf{c} + \mathbf{vW}) \tag{18}$$

$$\mathbf{P_{v|h}} = sigm(\mathbf{b} + \mathbf{Wh}) \tag{19}$$

Dans le cas gaussien, les unités visibles prennent leurs valeurs dans $\mathbb{R}$ et le terme de biais visible vaut $\frac{1}{2}(v_i - b_b)^2$. L'équation 17 devient donc

$$p(v_i|\mathbf{h}) = \frac{\exp(-\frac{1}{2}(v_i - b_i)^2 + v_i(\mathbf{Wh})_i)}{\int_{v_i} \exp(-\frac{1}{2}(v_i - b_i)^2 + v_i(\mathbf{Wh})_i) dv_i} \tag{20}$$

L'exposant de l'exponentielle se refactorise de la manière suivante :

$$\frac{1}{2}(v_i - b_i)^2 - v_i(\mathbf{Wh})_i = \frac{1}{2}[(v_i - (b_i + (\mathbf{Wh})_i))^2 + K_i]$$

avec

$$K_i = b_i^2 - (b_i + (\mathbf{Wh})_i)^2$$

L'équation 20 devient donc

$$p(v_i|\mathbf{h}) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(v_i - (b_i + (\mathbf{Wh})_i))^2} = \mathcal{N}(b_i + (\mathbf{Wh})_i, 1)$$

On note $\mathcal{N}(\nu, \sigma)$ la loi normale On a donc bien une distribution gaussienne. Le vecteur $\mathbf{P_{v|h}}$ s'exprime donc

$$\mathbf{P_{v|h}} = \mathcal{N}(\mathbf{b} + \mathbf{Wh}, \mathbf{1})$$

2.4 L'algorithme en pratique

Initialisation Dans [14], les poids sont initialisés grâce à des échantillons tirés aléatoirement selon une loi uniforme sur l'intervalle $[-1/\sqrt{k}, 1/\sqrt{k}]$ où k est le nombre de connections entrantes (fan-in) venant de la couche précédente.

Hinton recommande une initialisation des poids aléatoire selon une loi gaussienne centrée de variance $0,01$, ou nulle si le processus d'apprentissage est stochastique, ce qui est notre cas.

Le biais visible peut être initialisé à 0. Le biais caché est également initialisé à 0 sauf si l'on applique une contrainte de parcimonie de cible t (voir partie 2.4.1), auquel cas on peut initialiser le biais caché $\mathbf{c}$ à $\log[t/(1-t)]$ ([21]).

2.4.1 Améliorations

L'algorithme présenté plus haut constitue est améliorable grâce à plusieurs procédés. Certains s'expriment comme des contraintes sur les poids, d'autres sont des modifications légères du processus d'entraînement. Cette partie présente ceux qui ont été utilisés pendant ce stage, les ordres de grandeur et conseils de réglage proviennent essentiellement de [21].

Contrainte de poids faible La contrainte dite de weight decay (contrainte de poids faible en français) est une contrainte très répandue. Dans certaines formulations, elle fait même partie de l'algorithme de divergence contrastive ([47] par exemple). Elle se présente comme un terme de pénalisation des poids larges dans la fonction de coût. Le plus souvent, ce terme est proportionnel la norme $\mathcal{L}_2$ des poids.

$$\alpha_{wc} \sum_{i,j} w_{i,j}^2$$

Le weight-cost α_{wc} ajuste l'importance de la pénalité. En général,
$\alpha_{wc} \in [10^{-5}, 10^{-2}]$.
Cette contrainte permet de lisser les poids et de limiter les poids inutiles, donc de faciliter leur interprétation. Elle sert aussi à éviter qu'un poids diverge. Hinton [21] affirme que cette contrainte permet également de limiter le surapprentissage et d'augmenter la vitesse de convergence de la chaine de Gibbs.

Moment La technique du moment est une amélioration classique de l'algorithme de descente du gradient. Elle consiste à ajouter un terme de mémoire dans l'expression de la mise à jour. On note $\mathcal{C}$ la fonction de coût, θ le paramètre mis à jour, α_m de coefficient de moment et $\Delta\theta(t)$ la t-ième mise à jour de θ.

$$\Delta\theta(t) = \alpha_m \Delta\theta(t-1) - \epsilon \frac{d\mathcal{C}}{d\theta}(t)$$

Ce procédé permet de donner une certaine inertie au système. Imaginons que l'on puisse faire rouler une bille sur la surface d'erreur. La position de cette bille représente le

jeu de paramètres du réseau. Si la surface d'erreur est trop irrégulière et que cette bille est trop légère, le parcours s'arrêtera très rapidement dans un creux même peu profond. Ce minimum local n'est peut-être pas la valeur optimale. Si la bille est plus lourde, son inertie lui évitera de rester coincée dans un minimum non optimal et elle aura donc plus de chances de finir à une altitude plus faible, voir un minimum global. Le principe du moment est le même, on espère trouver un meilleur minimum en gardant en partie la direction précédente.

On règle généralement : $\alpha_m = 0,5$. On peut améliorer l'apprentissage en augmentant ce paramètre aux alentours de 0,9 lorsque l'erreur de reconstruction ne diminue plus trop ([21]).

Échantillonnage Selon le modèle présenté plus haut, les valeurs des neurones cachés sont toujours échantillonnées selon une loi de probabilité. Les valeurs des neurones visibles sont soit échantillonnées (échantillonnage de Gibbs, voir partie 2.3.3) lors de la phase négative, soit fixées selon les vecteurs d'entrainement lors de la phase positive. En pratique, on peut utiliser la valeur des probabilités d'activation de la couche visible dans le cas binaire lors de l'échantillonnage de Gibbs afin d'accélérer l'apprentissage : on réduit le bruit d'échantillonnage. Les couches cachées, elles, doivent être échantillonnées durant l'apprentissage car cette réduction de l'information (information bottleneck) constitue un régularisateur important pour la convergence. Une fois le réseau entraîné, on peut utiliser directement les probabilités de la couche cachée comme vecteur de sortie.

Parcimonie La parcimonie tend à réduire le nombre d'unités cachées activées en même temps. Cela permet d'interpréter plus facilement la valeur des poids et il a constaté dans [35] que cette contrainte pouvait améliorer les performances de discrimination.

L'idée est de fixer une probabilité d'activation cible p pour les neurones cachés et d'ajouter un terme pénalisant la distance entre q la probabilité d'activation des unités cachées et p à la fonction de coût. On estime $\hat{q}$ pour chaque unité cachée sur un minibatch d'entrainement selon la formule :

$$\hat{q} = \lambda \hat{q}_{old} + (1 - \lambda) q_{minibatch}$$

avec $\hat{q}_{old}$ l'ancienne valeur de l'estimateur $\hat{q}$, $q_{minibatch}$ est la moyenne de probabilité d'activation de l'unité cachée sur un minibatch et λ le pas de décroissance de l'estimateur.

La distance utilisée pour le terme de pénalité est l'entropie croisée entre la probabilité cible et la probabilité estimée. La fonction de coût est donc de la forme :

$$\mathcal{C} + \alpha_p(-p \log q - (1 - p) \log(1 - q))$$

Cette pénalité est réglée par un facteur α_p : le coût de parcimonie.

Il y a donc trois paramètres à régler. Les ordres de grandeur conseillés sont les suivants :

- $p \in [0,01; 0,1]$
- $\lambda \in [0,9; 0,99]$

- Le coût de parcimonie se règle de manière à ce que $q \simeq p$ mais sans trop interférer dans l'apprentissage. On peut contrôler l'effet de cette contrainte à partir des histogrammes des activités moyennes.

2.4.2 Contrôler l'apprentissage

Erreur de reconstruction L'erreur de reconstruction est la distance (souvent norme $\mathcal{L}^2$) entre un vecteur d'entraînement et sa reconstruction après un certain nombre d'itérations de la chaîne de Gibbs. Cet indicateur ne doit être interprété que comme un signal d'alarme en cas de divergence. L'erreur doit diminuer rapidement au début de l'entraînement puis stagner. Si on utilise la technique du moment, il peut y avoir quelques oscillations. L'intérêt est que cette grandeur est facile à calculer mais peu fiable car la vitesse de convergence d'une chaîne de Markov est indépendante de l'adéquation du modèle aux données. Si la chaîne converge très lentement, le sample généré ne sera pas très éloigné du sample de départ même si le modèle est très mal représenté.

Éviter le sur apprentissage Un cas dit de sur-apprentissage est un cas où les performances du système sont très bonnes sur la base d'apprentissage mais mauvaises sur de nouveaux vecteurs. Cela signifie que le système a bien appris mais qu'il est incapable de généraliser son fonctionnement. Un exemple de sur-apprentissage rencontré au sein de l'équipe analyse synthèse a été rencontré sur une tâche de distinction d'instruments. La base d'entrainement contenait des exemples de guitare et de piano, et les performances sur cette base avoisinaient les 100% tandis que sur de nouveaux vecteurs, les résultats étaient totalement aléatoires. Ils s'avère que la base d'apprentissage comportait un biais important : le volume des enregistrements de guitare était moins important que le volume des ceux de piano. Le système avait simplement appris à calculer un seuil sur la puissance moyenne du signal. Dans cet exemple, le problème vient de la base d'apprentissage, mais il peut aussi être causé par un mauvais réglage du système. Notamment en dimensions : si le système est trop expressif (trop d'unités cachées) où que l'entrainement dure trop longtemps par rapport à la quantité de données d'apprentissage disponible.

Les RBMs sont des modèles probabilistes. Ils apprennent la distribution de probabilité des données d'entrée. Un raisonnement intuitif pour éviter le sur-apprentissage est de comparer la probabilité de données d'entrainement avec la probabilité de données de validation. On sélectionne un sous-ensemble de vecteurs d'apprentissage et un sous-ensemble de vecteurs de validation qui n'ont pas servi à l'optimisation des paramètres du système. Puis on calcule les probabilités de chaque vecteur des deux bases et on moyenne pour chaque base. On suit alors la différence entre ces deux valeurs de probabilités moyennes. Lorsque celle ci devient trop élevée, cela signifie que le réseau connaît trop bien la base d'entrainement par rapport à une base de vecteurs différents et qu'il est temps d'arrêter l'apprentissage. Mais le temps de calcul de la probabilité d'un vecteur dans un RBM est très élevé à cause de la fonction de partition (voir équation 2). Comme suggéré dans [21], on peut se contenter du calcul de l'énergie libre si l'on ne s'intéresse qu'à la différence entre deux énergies libres car les termes de fonction de partition s'annulent.

FIGURE 2 – Ces images, tirées du blog de recherche de google, représente sur la ligne du haut des images qui ont servi d'état initial de la couche visible d'un réseau de neurones de classification dont la classe était fixée à "Towers & Pagodas", "Buildings" et "Birds & Insects" et leur reconstructions par le réseau sur la ligne du bas. Outre l'aspect esthétique, elles permettent de voir ce que le réseau a appris.

$$\frac{p(\mathbf{v}_{entraînement})}{p(\mathbf{v}_{validation})} = \frac{\exp(-\mathcal{F}(\mathbf{v}_{entraînement}))}{\exp(-\mathcal{F}(\mathbf{v}_{validation}))} = \exp(\mathcal{F}(\mathbf{v}_{validation}) - \mathcal{F}(\mathbf{v}_{entraînement}))$$

Ainsi, il suffit de calculer la différence des moyennes des énergies libres de deux bases fixes au cours de l'apprentissage, l'une contenant des vecteurs d'entraînement et l'autre contenant des vecteurs de validation qui n'ont pas servi à entrainer le réseau.

Histogrammes des poids Il est intéressant de contrôler l'histogramme des paramètres (poids et biais) pour vérifier qu'il n'y a pas de divergence. De plus, comparer l'histogramme des poids et celui de leurs mises à jour permet d'ajuster le pas d'apprentissage. Geoffrey Hinton suggère, dans [21], de régler le pas d'apprentissage de manière à ce que les mises à jour soient de l'ordre de 10^{-3} fois les poids.

Observation de statistiques négatives Les RBMs sont des modèles dits génératif : ils peuvent générer des vecteurs selon la loi de probabilité qu'ils ont appris. C'est l'algo-

rithme d'échantillonnage de Gibbs qui est utilisé pour générer de tels vecteurs au cours de l'apprentissage lors de la phase négative (voir partie 2.3.1). Ces vecteurs générés, aussi appelés statistiques négatives, en apprennent beaucoup sur ce que le réseau a appri : ils sont sensés ressembler à des vecteurs d'entraînement. Pour prendre un exemple visuel, on peut citer le travail récent mené par des ingénieurs de Google : deepdream [7] (voir figure 2). L'idée est la même sur un réseau de classification conséquent entrainé sur une grande quantité d'images. L'idée est de faire générer une image au réseau en donnant comme vecteur initiale une image réelle et en fixant la classe désirée. Le réseau reconstruit donc l'image avec des objets élémentaires de la classe demandée. Cela permet de voir comment il pense.

Champs récepteurs Il est intéressant, lorsque les données comportent une structure (spatiale ou temporelle), de visualiser les poids associés à chaque unité cachée. Cela permet de voir les descripteurs appris par le réseau. On les appelle parfois les champs récepteurs en référence aux neurones sensoriels, ce terme qualifie la zone sensorielle (portion d'image, bande de fréquence...) qui active un certain neurone. Par exemple dans [30], la visualisation des champs récepteurs montre que certains neurones détectent des phonèmes bien précis. Cela dit mieux vaut rester prudent pour cet exemple car le réseau est convolutif, ce qui signifie que les poids peuvent être translatés le long du spectre.

Observations des activations d'unités cachées Une manière de contrôler l'apprentissage est de prendre un batch d'entrainement et de regarder les activations de la couche cachée correspondant à chaque example d'entrainement. Cela permet de voir si certains neurones ne sont jamais utilisés par exemple, ou de contrôler la parcimonie.

7. Voir http://googleresearch.blogspot.fr/

3 Réseau de neurones profond

Cette partie présente les réseaux de neurones tels qu'ils ont été largement utilisés entre les années 80 et 90 puis leur évolution plus récente : les réseaux de neurones profonds.

3.1 Les réseaux de neurones

3.1.1 Origine

Les premières formulations du réseau de neurones remontent à 1943, dans les travaux de Mc Culloch et Pitts [33]. L'idée est de reproduire le fonctionnement d'un neurone humain, comme représenté sur la figure 3. Le fonctionnement d'un neurone formel est simple : c'est une somme pondérée d'entrées à laquelle on applique une fonction d'activation. Les coefficients de pondération sont appelés poids synaptiques et la fonction d'activation utilisée était initialement un seuil : la sortie valant 1 si la somme pondérée dépasse le seuil et 0 sinon. On note b la valeur du seuil.

$$Y = \begin{cases} 1 & \text{si } \sum_k W_k i_k > b \\ 0 & \text{sinon.} \end{cases} \tag{21}$$

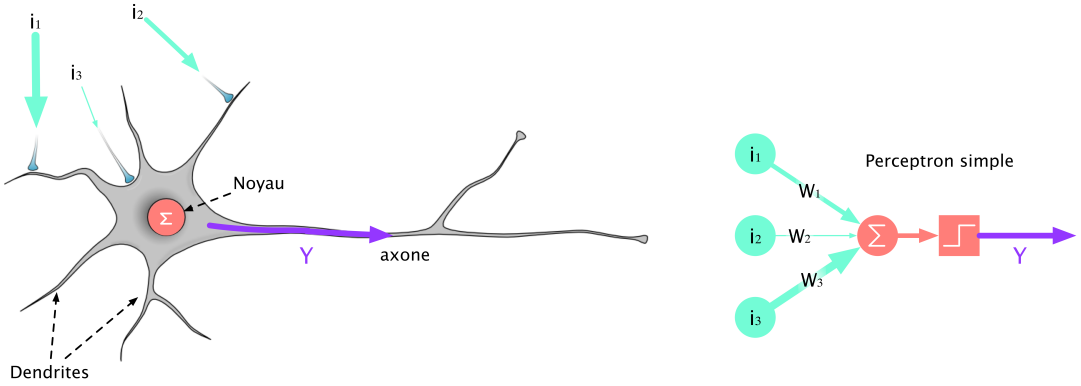


FIGURE 3 – Schéma d'un neurone à gauche et représentation d'un neurone formel à droite

On peut ensuite utiliser plusieurs neurones en parallèle, on obtient donc un réseau avec un vecteur de sortie comportant autant d'éléments que de neurones. Ce système, baptisé percetron, constitue l'un des premier systèmes artificiels capable d'apprendre sur des exemples. Frank Rosenblatt proposa une méthode pour optimiser les poids synaptiques. Mais quelques années plus tard, en 1969, Marvin Lee Minsky et Seymour Papert ont publié un livre appelé Perceptrons [34] dans lequel ils mettent en avant les limites du modèle. En effet, appliqué un seuil sur un vecteur pondéré revient à découper linéairement l'espace des vecteurs d'entrée en deux avec un hyperplan. Ce travail, trop

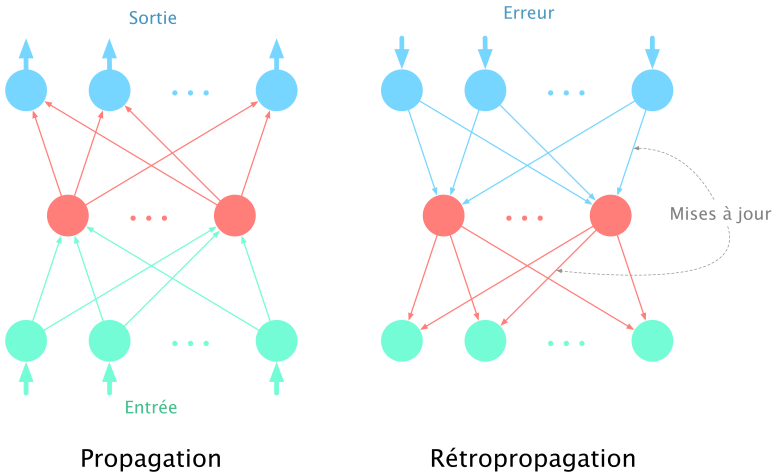


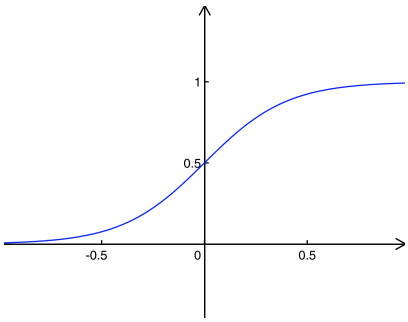
FIGURE 4 – Schéma des deux phases de l'algorithme de back-propagations.

rapidement généralisé à tous les types de réseaux de neurones, a considérablement réduit les efforts dans le domaine qui n'a pas vu d'avancée majeure pendant une dizaine d'années.

3.1.2 Perceptron multicouche, algorithme de back propagation

Un nouveau type de réseau permettant de dépasser les limites du simple perceptron a été proposée dans les années 80 : les perceptron multicouche (Multi Layer Perceptron) (MLP). En réalité, cette formulation date de 1969 [8] mais a été réellement utilisée à partir de 1986 grâce à l'article de Rumelhart et Hinton [41]. Le principe est de superposer plusieurs perceptrons dont la fonction d'activation est la fonction sigmoïde.

$$sigm(x) = \frac{1}{1 + e^{-x}}$$



L'algorithme d'entrainement des MLP est communément appelé algorithme de back-propagation (ou rétropropagation du gradient). C'est une descente de gradient sur la fonction d'erreur quadratique entre la sortie et la sortie désirée. Cet entraînement est supervisé, il nécessite donc de connaître pour chaque vecteur d'entrée, un vecteur de sortie désirée. Pour une tâche de classification par exemple, cela revient à connaître la

classe du vecteur d'entraînement. Il se déroule en deux étapes (voir figure 4) :

1. La propagation du vecteur d'entrée, c'est à dire le calcul du vecteur de sortie du réseau en fonction du vecteur d'entrée.

2. La rétropropagation du gradient, c'est à dire le calcul du gradient de la fonction d'erreur quadratique par rapport à chaque paramètre du réseau puis la mise à jour de ces paramètres.

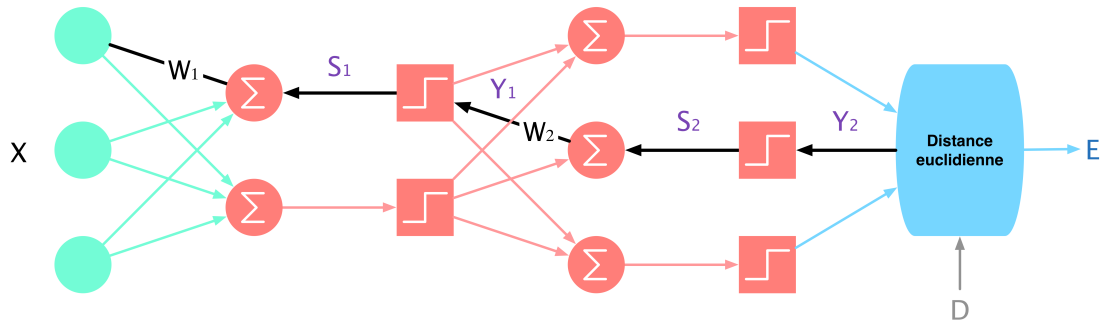


FIGURE 5 – Phase de rétropropagation détaillée

Propagation Le calcul du vecteur de sortie se fait en calculant les sorties successives des différentes couches, la sortie d'une couche servant d'entrée à la couche suivante. Ce vecteur est ensuite comparée au vecteur de sortie désiré : on calcule le vecteur d'erreur quadratique servant pour la phase de rétropropagation. Dans un cas de classification, chaque neurone de la dernière couche du MLP représente une classe : il vaut 1 si le vecteur d'entrée appartient à cette classe, 0 sinon.

Rétropropagation Le gradient en fonction de chaque paramètre se calcule assez facilement grâce à une propriété intéressante de la fonction sigmoïde : sa dérivée s'exprime en fonction d'elle-même.

$$\text{sigm}'(x) = \text{sigm}(x)(1 - \text{sigm}(x))$$

Détaillons un peu le développement sur un exemple : en prenant les notations de la figure 5 où $\mathbf{X}$ est le vecteur d'entrée, S_1 et S_2 sont les sorties des sommes de la première et de la deuxième couche, y_1 et y_2 sont les sorties des fonctions d'activations de la première et la deuxième couche, $\mathbf{D}$ est le vecteur de sortie désiré et E est l'erreur quadratique entre la sortie et la sortie désirée. w_1 et w_2 sont des poids des couches 1 et 2 (x est la composante de $\mathbf{X}$ correspondant à w_1 et d la composante de $\mathbf{D}$ correspondant à y_2). Alors on a

$$\frac{\partial E}{\partial w_1} = \underbrace{\frac{\partial E}{\partial y_2}}_{2(y_2-D)} \times \frac{\partial y_2}{\partial w_1}$$
$$\frac{\partial y_2}{\partial w_1} = \underbrace{\frac{\partial y_2}{\partial S_2}}_{y_2(1-y_2)} \times \frac{\partial S_2}{\partial w_1}$$
$$\frac{\partial S_2}{\partial w_1} = \underbrace{\frac{\partial S_2}{\partial y_1}}_{w_2} \times \frac{\partial y_1}{\partial w_1}$$
$$\frac{\partial y_1}{\partial w_1} = \underbrace{\frac{\partial y_1}{\partial S_1}}_{w_1} \times \underbrace{\frac{\partial S_1}{\partial w_1}}_{x}$$

Donc

$$\frac{\partial E}{\partial w_1} = 2(y_2 - D)y_2(1 - y_2)w_2 w_1 x$$

On peut ainsi obtenir simplement l'expression du gradient en fonction de tous les paramètres.

Les limites du perceptron multicouche Une des limitations que rencontre le MLP est que si l'architecture est trop profonde, l'optimisation des paramètres mène bien souvent à des minimas locaux non optimaux. En effet, comme expliqué dans [14], [2] la surface d'erreur n'est pas convexe et est d'autant plus irrégulière que réseau est profond. Par ailleurs, il semblerait que l'entrainement des couches les plus basses (proches du vecteur visible) soient peu efficaces dans un MLP profond ([2], [14]). L'intuition est que l'information de la mise à jour des paramètres est de moins en moins pertinente à mesure que l'on s'enfonce dans les couches les plus basses.

3.2 Deep belief network

On appelle un réseau comportant au moins trois couches. Cette appellation implique souvent que l'optimisation du réseau comporte une phase d'entrainement couche par couche, c'est-à-dire une phase durant laquelle chaque couche est entrainée de manière non supervisée, la première prenant en entrée les vecteurs d'entrainement, la seconde prenant en entrée la sortie de la première etc...

L'intérêt des réseau de neurones profond (Deep Neural Network) (DNN) réside dans leur pouvoir de représentation, c'est-à-dire la complexité des fonction de probabilité qu'ils sont capables de représenter. En effet, d'après [2], [3] et [17], ce dernier travail prouve notamment que la suppression d'une couche peut entraîner une augmentation exponentielle du nombre de neurones nécessaires à la même tâche. Mais l'optimisation de telles architectures est un problème complexe et non convexe (voir [2]). Les approches

utilisées sont des heuristiques dont les propriétés de convergence ne sont pas prouvées [47].

Nous nous intéressons dans ce stage aux DBN qui sont des réseaux de neurones profonds interprétables comme un empilement de RBM ou comme un perceptron multicouche. L'entrainement se déroule en deux phases :

1. pré-entrainement non supervisé : l'idée est d'entrainer chaque couche comme un RBM avec les techniques décrites plus haut. l'entrainement commence par les couches les plus basses (proches du vecteur d'entrée visible) et remonte jusqu'au vecteur de sortie. La couche à entrainer est la couche cachée et la couche précédemment entrainée est la couche visible. Les paramètres de poids $\mathbf{W}$ et de biais caché $\mathbf{c}$ sont utilisés pour initialiser les paramètres d'un MLP ayant les mêmes caractéristiques (nombre de couches, nombre de neurones...).

2. entrainement final supervisé : On entraîne le perceptron multicouche selon l'algorithme de back-propagation présenté dans la partie 3.1.2.

De nombreux tests sont interprétés dans [14], où différents réseaux sont entraînés sur des tâches de classification simples (formes géométriques, chiffres écrits à la main). Les conclusions principales qui en ressortent sont que les performances de classifications sont meilleures avec un pré-entraînement non supervisé à condition que le réseau comporte suffisamment de neurones. Même sur des réseaux peu profonds, les performances sont améliorées. Une autre conclusion est que les trajectoires des paramètres au cours de l'apprentissage sont disjointes et arrivent dans des domaines différents. Cela signifie que les solutions trouvées avec un pré-apprentissage ne sont qualitativement pas les mêmes que celles trouvées sans, malgré une deuxième phase de réglage fin identique.

Interprétation image concepts haut niveau.

Définir l'énergie libre pour un DBN (Pour l'utilisation d'un DBN par classe + regression logistique sur énergies libres)

4 Expérimentations

4.1 Données

4.1.1 Base tzanetakis

Nous utilisons la base constituée par George Tzanetakis pour [48] qui est devenu une base de référence en classification en genre musical. Ce succès est sans doutes dû au fait que la base est équilibrée : elle contient exactement 100 extraits de 30 secondes pour chacun des 10 genres représentés : blues, classical, country, disco, hiphop, jazz, metal, pop, reggae, rock.

Cette base comporte un certain nombre de défauts qui sont listés par Bob L. Sturm dans [43],[46]. Notamment des doublons exacts (5% de l'ensemble), des doublons légèrement différents (versions remasterisée, nouvelle version de la chanson), un morceau totalement corrompu, des genres majoritairement composés d'un artiste, un mauvais étiquetage (10,6% de l'ensemble)... L'ordre de grandeur important à retenir est que le système de classification parfait obtiendrait, en tenant compte des mauvais étiquetages et des défauts, un F-score de 94,5%.

Lorsqu'on utilise un réseau de neurones profonds, la question de l'information fournie au réseau est fondamentale. En effet, un réseau suffisamment expressif est capable d'effectuer lui-même les opérations nécessaires pour extraire l'information pertinente. Dans [24] par exemple, un parallèle est établi entre le nombre d'étapes des traitements et extractions de descripteurs conçus à la main et la profondeur d'un réseau de neurone. Ce point de vue inciterait donc à introduire le maximum d'information possible dans un réseau comportant beaucoup de paramètres et de laisser l'entraînement faire le travail. Le risque est d'avoir alors un temps de calcul trop important ou de rencontrer des minima locaux très sous-optimaux. Plus on effectue de prétraitement, plus on assure un résultat correct mais en laissant moins de liberté au réseau.

Nous avons donc choisi d'entraîner le réseau sur des vecteurs de transformée à Q constant (constant Q transform) (CQT) (voir partie 4.1.2). Comme ces vecteurs ne fournissent qu'une information spectrale, nous les complétons avec des informations sur l'évolution temporelle de deux manières différentes afin de former deux ensembles d'entrainements :

- le vecteur de CQT concaténé à un vecteur de variation spectrale (voir 4.1.3)
- 4 vecteurs de CQT alignés sur le temps (ou beat-synchrone) consécutifs concaténés. Le premier vecteur est le premier temps de la mesure.

Pour calculer la deuxième base, on a besoin de la position des temps. Plusieurs approches de classification utilisent des descripteurs dits *beat-synchrones*, notamment dans [12] et [42]. Les informations de rythme qu'ils utilisent sont obtenues à partir d'algorithmes de détection appliqués au signal. Ces approches fournissent de bons résultats dans la plupart des cas, mais ne fonctionnent pas du tout sur certains morceaux de jazz ou de musique classique notamment.

J'ai donc réalisé des annotations à la main à l'aide du logiciel *Audiosculpt*, développé par l'ircam, et notamment de l'algorithme de détection des temps intégré : *ircambeat*. Ces annotations étant très longues à effectuer, je n'ai pu annoter lors de ce stage que la moitié de la base Tzanetakis, ce qui constitue déjà un bon ensemble d'entraînement (500 extraits de 30 secondes). J'ai également annoté les temps forts (premiers temps de la mesure) de manière à ce que les concaténations des 4 vecteurs se fasse en commencant au début de la mesure. De plus, la majorité des extraits de la base tzanetakis ont des mesures à 4 temps.

4.1.2 Constant Q transform

Comme évoqué dans la partie précédente, nous utilisons des vecteurs de transformée à Q constant pour alimenter le réseau. Présentons brièvement cette opération.

La transformée à Q constant est une convolution du signal avec un banc de filtres dont les fréquences de résonance f_k suivent une échelle logarithmique et dont les longueurs sont inversement proportionnelles aux fréquences f_k. Notons $k \in \{0, \dots, N\}$ l'indice du filtre, Q son facteur de qualité (identique pour tous les filtres), L_k la longueur du filtre en échantillons et f_e la fréquence d'échantillonnage du signal.

$$f_k = f_0 * 2^{k/12}$$

$$L_k = Q\frac{f_e}{f_k}$$

f_0, N et Q sont des paramètres de la CQT. Dans librosa, la librairie utilisée pour le calcul, le paramètre Q n'est pas entré directement mais calculé à partir de deux autres paramètres : `resolution` et `bins_per_octave` selon la formule suivante :

$$Q = \frac{\texttt{resolution}}{2^{1/\texttt{bins\_per\_octave}-1}}$$

Le paramètre `resolution` influe donc directement sur la longueur des fenêtres. Le paramètre `bins_per_octave` précise le nombre de filtres par octaves souhaité. On utilise souvent 12 filtres par octaves, un filtre par note.

On peut observer ces filtres sur la figure 6 et leur réponse en fréquence sur la figure 7.

L'intérêt de cette représentation est d'abord la dimension résultante qui est bien plus faible que celle d'une transformée de Fourier. De plus, les longueurs de filtres s'adaptant avec la fréquence, on obtient une bonne résolution fréquentielle pour les basses fréquences (fenêtre longue) et une bonne résolution temporelle pour les hautes fréquences (fenêtre courte). La notion de note apparait ce qui est pertinent puisqu'on analyse des signaux musicaux. Enfin, l'échelle des fréquences est, comme l'oreille humaine, logarithmique.

4.1.3 Pré traitement

Nous avons vu dans la partie 2.2.4 que les variables à valeurs continues pouvaient être modélisées par des unités visibles gaussiennes. Mais les composantes des vecteurs de

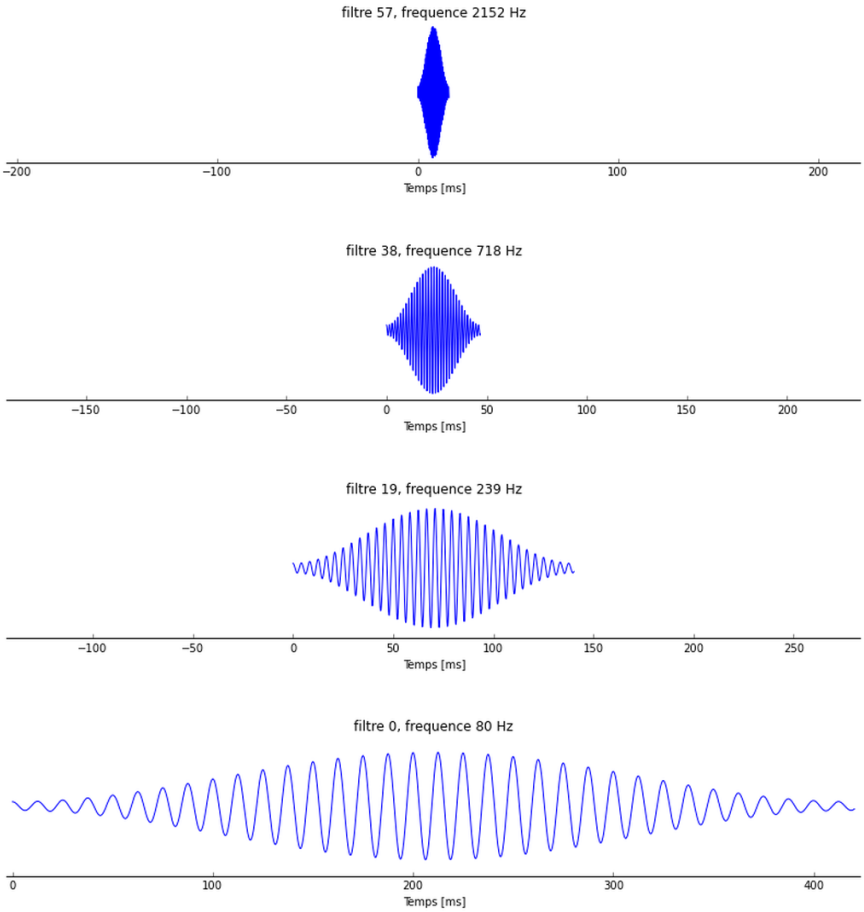


FIGURE 6 – Représentation de plusieurs filtres à Q constant calculés avec $f_0 = 80$, $\texttt{bins\_per\_octave} = 12$, $N = 60$, $f_e = 22050$, $\texttt{resolution} = 2$. On observe que les filtres des hautes fréquences sont plus courts que les filtres des basses fréquences. L'abscisse représente le temps en ms et l'ordonnée est normalisée, on ne voit donc pas l'effet de la normalisation par L_k la longueur du filtre.

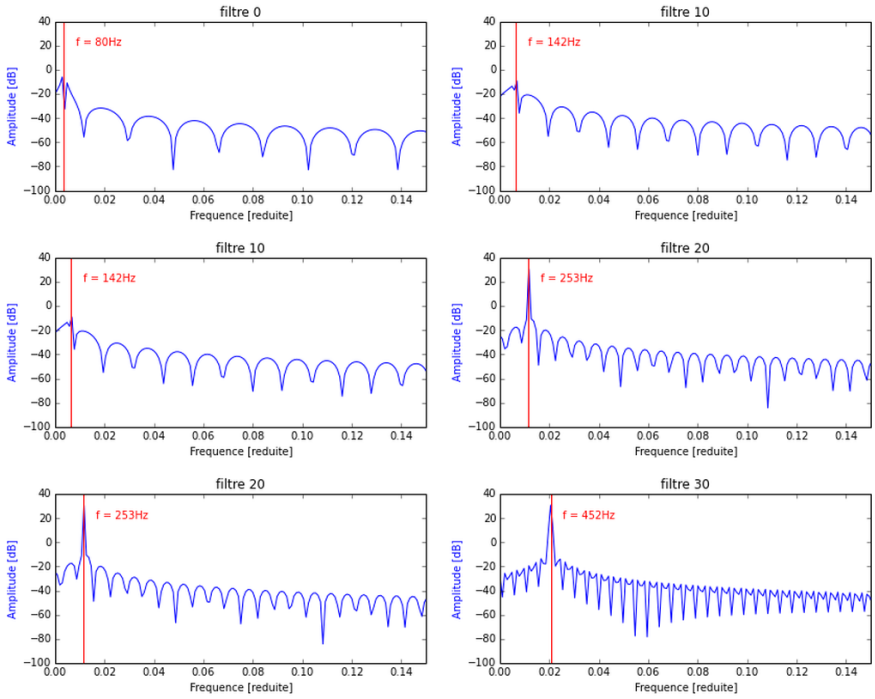


FIGURE 7 – Représentation des réponses en fréquence de plusieurs filtres, calculés avec les mêmes paramètres que pour la figure 6. L'abscisse représente la fréquence normalisée par f_e la fréquence d'échantillonnage et l'ordonnée représente l'amplitude en dB.

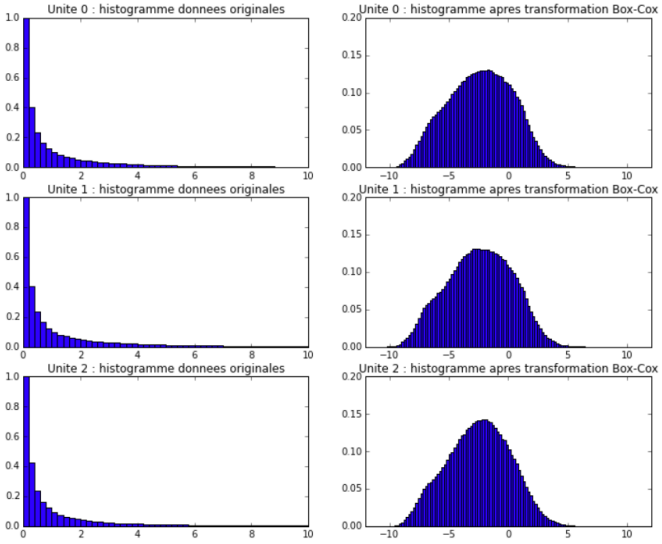


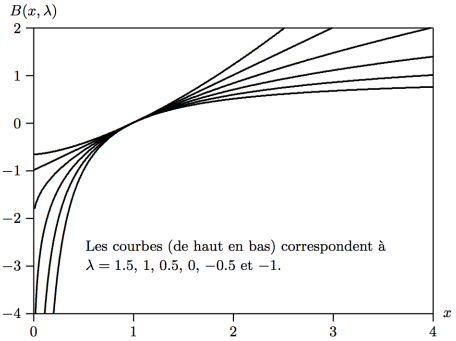
FIGURE 8 – Exemples d'histogrammes calculés à partir de l'ensemble des vecteurs de CQT de la base Tzanetakis. Les trois lignes représentent trois composantes de vecteur différentes, la colonne de gauche représente la distribution originale et la colonne de droite la distribution transformée.

CQT sont-elles modélisables par des gaussiennes ? On observe sur la colonne de gauche de la figure 8 que ce n'est pas le cas.

Transformation de Box-Cox Pour rendre les données plus proches du modèle gaussien, nous allons appliquer une transformée de *Box-Cox* [7].

Cette transformée s'exprime simplement, en fonction du vecteur d'entrée x et du paramètre λ selon la formule suivante (on peut voir à droite une représentation de $B(x, \lambda)$ pour plusieurs valeurs de λ, tirée d'un cours) :

$$B(x,\lambda) = \begin{cases} \frac{x^\lambda - 1}{\lambda} & \text{si } \lambda \neq 0 \\ \ln(x) & \text{sinon} \end{cases}$$



L'optimisation du paramètre lambda est faite par la fonction python `stats.boxcox` de manière à obtenir une distribution la plus proche possible d'une gaussienne (voir colonne de droite de la figure 8). Cela permet d'obtenir des données modélisables par des unités gaussiennes.

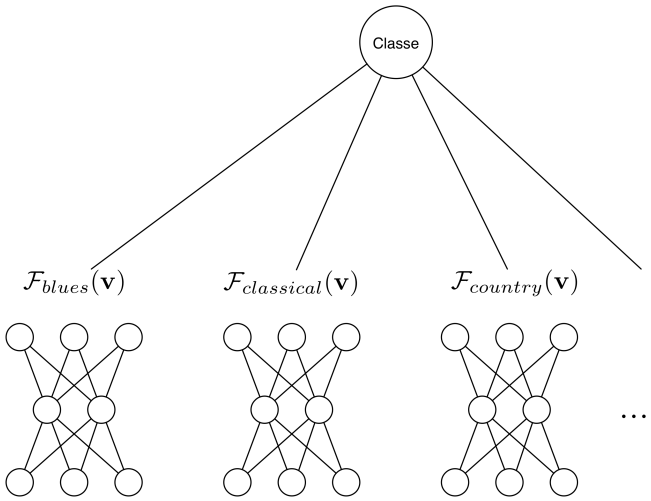


FIGURE 9 – L'architecture possédant autant que de réseaux que de classes. La couche supérieure représente un entraînement supervisé sur l'énergie libre des couches supérieurs de ces réseaux.

Variation spectrale Pour donner de l'information sur l'évolution temporelle du signal, une des possibilités envisagées est de concaténer le vecteur avec sa variation spectrale, c'est-à-dire la convolution entre v_t le vecteur à l'instant t et v_{t+1} le vecteur à l'instant $t+1$.

$$var_{spec} = (v_t * v_{t-1})(d) \text{ avec } d \in \{-5, \ldots, 6\}$$

On ne garde que 12 valeurs de la convolution pour couvrir une octave. La structure du spectre rend moins utile les octaves suivantes car les harmoniques se superposent.

4.2 Architectures

La première architecture testée dans ce stage est l'architecture "classique", c'est-à-dire l'application d'un DBN à une tâche de classification : on l'entraîne en deux étapes (pré-entrainement non supervisé puis entrainement supervisé) puis on entraîne un sur une ou plusieurs couches de ce réseau. C'est la technique utilisée dans beaucoup d'applications des DBN, et notamment dans [16].

Nous proposons une autre architecture, inspirée de [21]. Geoffrey Hinton y proposait une utilisation des RBM pour la classification consistant à entrainer un RBM différent sur chaque classe de manière à ce que chacuns de ces réseaux apprennent à représenter au mieux un type de donnée. il utilisait ensuite l'énergie libre de chaque RBM en entrée d'une couche de régression logistique afin d'effectuer une phase d'entrainement supervisée. La proposition que nous avons est d'appliquer le meme principe mais avec des DBN (voir schéma 9). Cette méthode n'a malheureusement pas pu être implémentée à temps pour ce mémoire.

4.3 Résultats et interprétation

Les résultats obtenus ne sont malheureusement pas satisfaisants, de l'ordre de 40% de précision/rappel [8] en moyenne. Ces résultats ne remettent pas en question la méthode utilisée, ils sont dûs à de mauvais choix des outils pour les implémenter et une sous-estimation du temps nécessaire au bon paramétrage. En effet, l'implémentation a été faite à l'aide de theano, une librairie python optimisée pour le calcul sur processeur graphique. C'est une librairie de calcul symbolique, un paradigme intéressant mais long à prendre en main et à débuger. De plus, dans le but de comprendre l'algorithme en profondeur, j'ai fais le choix de réimplémenter les RBM moi-même, ce qui m'a prit beaucoup de temps.

Si les résultats absolus ne sont pas exploitables, on peut malgré tout évaluer les hypothèses faites pour ce stage. Ces essais sont menés sur une base de vecteurs concaténés, c'est à dire que chaque vecteur d'entraînement est composé de 4 vecteurs de transformée à Q constant de 60 bins consécutifs. Le réseau comporte donc 240 unités visibles.

Concernant la transformée de BoxCox par exemple, nous avons entraîné un réseau avec 3 couches cachées de respectivement 300, 100 et 10 neurones sur 15 epochs [9] de pré-entrainement non supervisé et 40 epochs d'entrainement supervisé. Les résultats de classification par le réseau sont meilleurs avec la transformée : on obtient un rappel moyen de 23 % sans la transformée et de 29% avec. Cette comparaison est discutable puisque cette différence est quasiment annulée après l'étape de classification avec le SVM.

Concernant l'utilisation de descripteurs beat-synchrones, on observe sur le même type de réseau une amélioration de l'ordre de 10 % en rappel moyen : on passe de 39% de rappel moyen à 48% en version beat-synchrone.

La figure 10 représente quelques filtres pris au hasard parmis les unités cachées de réseau précédemment cité, entrainé sur les données beat-synchrones. Les 4 colonnes représentent donc les 4 temps de la mesure, l'ordonnée représente les bins de la CQT, donc la fréquence.

8. La précision pour une classe i désigne le rapport du nombre d'éléments correctement attibués à la classe i sur le nombre d'éléments attribués à la classe i (bons et mauvais). Le rappel pour une classe i est le rapport du nombre d'éléments correctement attibués à la classe i sur le nombre d'éléments appartenant réellement à la classe i.

9. un *epoch* est une expression désignant une itération de l'algorithme d'apprentissage sur l'ensemble des minibatchs, c'est-à-dire l'ensemble de la base de donnée.

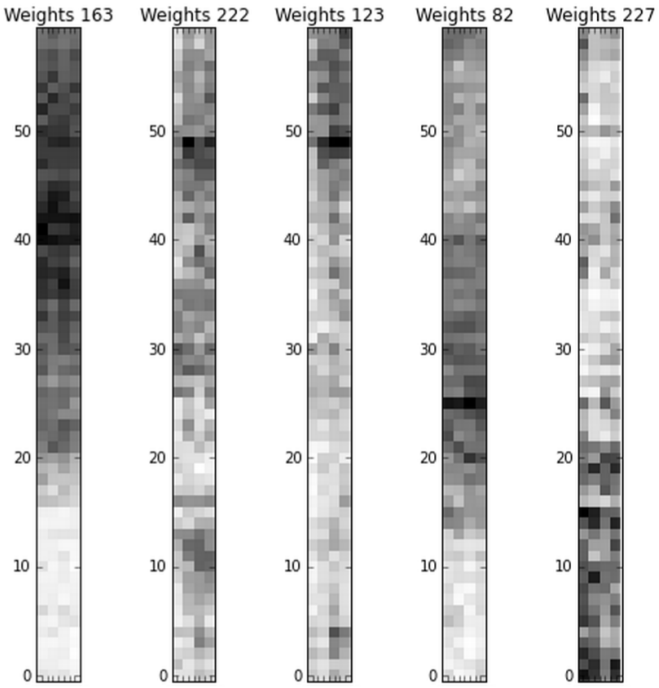


FIGURE 10 – Filtres d'unités cachées pris au hasard dans la première couche d'un DBN entrainé sur des vecteurs de CQT beat-synchrones et calés sur la mesure. Les poids sont réarrangés pour représenter les 4 vecteurs de CQT concaténés. Les 4 colonnes représentent donc les 4 temps de la mesure. On voit que certains filtres diffus captent l'énergie sur tout la mesure, d'autres plus précis marquent les temps forts.

5 Conclusion

5.1 Récapitulatif

Durant ce stage au sein de l'équipe analyse-synthèse, nous avons pu étudier en détail le fonctionnement des réseaux de neurones profonds, en particulier les deep belief networks. Nous avons donc pu comprendre les modèles probabilistes et les approximations qui constituent ces algorithmes d'apprentissage et comprendre quels étaient les moyens de contrôler cet apprentissage.

Nos contributions sont premièrement l'application de ces réseaux à des données beat-synchrone annotées à la main afin d'avoir non seulement les temps mais aussi les premiers temps de chaque mesure de manière certaine. La deuxième contribution est l'application de la transformée de Box-Cox à chacune des composantes du vecteur d'entrée afin de se rapprocher de l'hypothèse faite pour la première couche du réseau : les unités visibles modélisent des variables aléatoires suivant une loi gaussienne. Cette transformation semble améliorer légèrement le résultat (voir partie 4.3).

5.2 Perspectives

La première chose à faire pour compléter ce travail est de passer un temps conséquent à essayer différentes configurations de paramètres afin d'obtenir des résultats de classification plus proches de l'état de l'art. Le temps de calcul nécessaire à l'entraînement des réseaux de neurones profonds est important.

L'approche beat-synchrone est une bonne piste qui peut être améliorée. En effet, elle accentue une information musicale de haut niveau : les temps forts de la mesure par exemple, mais elle omet toute information non stationnaire à l'échelle du temps (au sens musical). Il serait intéressant de trouver un moyen de conserver cette information qui varie rapidement, comme s'y attache Stéphane Mallat dans [1]. De plus, l'annotation de bases de données peut donner des indications dans un cadre exploratoire, mais à grande échelle, une automatisation est nécessaire.

Concernant le système, plusieurs étapes peuvent être modifiées. Notamment l'étape de classification, qui utilise systématiquement une machine à vecteurs de support. Il est suggéré dans [1] que d'autres classificateurs pourraient être plus adaptés à l'audio, adaBoost notamment (voir [5]).

Références

[1] Joakim Andén and Stéphane Mallat. Multiscale scattering for audio classification. In Klapuri and Leider [25], pages 657–662.

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

[3] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In Bernhard Schölkopf, John C. Platt, and Thomas Hoffman, editors, *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006*, pages 153–160. MIT Press, 2006.

[4] Yoshua Bengio, Dale Schuurmans, John D. Lafferty, Christopher K. I. Williams, and Aron Culotta, editors. *Advances in Neural Information Processing Systems 22 : 23rd Annual Conference on Neural Information Processing Systems 2009. Proceedings of a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada*. Curran Associates, Inc., 2009.

[5] James Bergstra, Norman Casagrande, Dumitru Erhan, Douglas Eck, and Balázs Kégl. Aggregate features and adaboost for music classification. *Mach. Learn.*, 65(2-3) :473–484, December 2006.

[6] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[7] G. E. P. Box and D. R. Cox. An analysis of transformations. *Journal of the Royal Statistical Society. Series B (Methodological)*, 26(2) :pp. 211–252, 1964.

[8] Arthur E. Bryson and Yu-Chi Ho. *Applied optimal control, optimization, estimation, and control*. Blaisdell Pub. Co Waltham, Mass, 1969.

[9] Kaichun K. Chang, Jyh-Shing Roger Jang, and Costas S. Iliopoulos. Music genre classification via compressive sampling. In Downie and Veltkamp [13], pages 387–392.

[10] Ronan Collobert and Jason Weston. A unified architecture for natural language processing : deep neural networks with multitask learning. In *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*, pages 160–167, 2008.

[11] Sander Dieleman, Philemon Brakel, and Benjamin Schrauwen. Audio-based music classification with a pretrained convolutional network. In Klapuri and Leider [25], pages 669–674.

[12] Sander Dieleman and Benjamin Schrauwen. Accelerating sparse restricted boltzmann machine training using non-gaussianity measures. In *Deep Learning and Unsupervised Feature Learning (NIPS-2012)*, 2012.

[13] J. Stephen Downie and Remco C. Veltkamp, editors. *Proceedings of the 11th International Society for Music Information Retrieval Conference, ISMIR 2010, Utrecht, Netherlands, August 9-13, 2010*. International Society for Music Information Retrieval, 2010.

[14] Dumitru Erhan, Pierre-Antoine Manzagol, Yoshua Bengio, Samy Bengio, and Pascal Vincent. The difficulty of training deep architectures and the effect of unsupervised pre-training. In David A. Van Dyk and Max Welling, editors, *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, AISTATS 2009, Clearwater Beach, Florida, USA, April 16-18, 2009*, volume 5 of *JMLR Proceedings*, pages 153–160. JMLR.org, 2009.

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

[16] Philippe Hamel and Douglas Eck. Learning features from music audio with deep belief networks. In Downie and Veltkamp [13], pages 339–344.

[17] Johan Håstad and Mikael Goldmann. On the power of small-depth threshold circuits. *Computational Complexity*, 1 :113–129, 1991.

[18] G E Hinton and R R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786) :504–507, July 2006.

[19] Geoffrey E. Hinton. Products of experts. pages 1–6, 1999.

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

[21] Geoffrey E. Hinton. A practical guide to training restricted boltzmann machines. In Grégoire Montavon, Genevieve B. Orr, and Klaus-Robert Müller, editors, *Neural Networks : Tricks of the Trade (2nd ed.)*, volume 7700 of *Lecture Notes in Computer Science*, pages 599–619. Springer, 2nd edition, 2012.

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

[23] Geoffrey E. Hinton and Ruslan R Salakhutdinov. Using deep belief nets to learn covariance kernels for gaussian processes. In J.C. Platt, D. Koller, Y. Singer, and S.T. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1249–1256. Curran Associates, Inc., 2008.

[24] Eric J. Humphrey, Juan Pablo Bello, and Yann LeCun. Moving beyond feature design : Deep architectures and automatic feature learning in music informatics. In Fabien Gouyon, Perfecto Herrera, Luis Gustavo Martins, and Meinard Müller, editors, *ISMIR*, pages 403–408. FEUP Edições, 2012.

[25] Anssi Klapuri and Colby Leider, editors. *Proceedings of the 12th International Society for Music Information Retrieval Conference, ISMIR 2011, Miami, Florida, USA, October 24-28, 2011*. University of Miami, 2011.

[26] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models - Principles and Techniques*. MIT Press, 2009.

[27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors, *NIPS*, pages 1106–1114, 2012.

[28] Honglak Lee, Chaitanya Ekanadham, and Andrew Y. Ng. Sparse deep belief net model for visual area V2. In John C. Platt, Daphne Koller, Yoram Singer, and Sam T. Roweis, editors, *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, pages 873–880. Curran Associates, Inc., 2007.

[29] Honglak Lee, Roger B. Grosse, Rajesh Ranganath, and Andrew Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In Andrea Pohoreckyj Danyluk, Léon Bottou, and Michael L. Littman, editors, *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*, volume 382 of *ACM International Conference Proceeding Series*, pages 609–616. ACM, 2009.

[30] Honglak Lee, Peter T. Pham, Yan Largman, and Andrew Y. Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In Bengio et al. [4], pages 1096–1104.

[31] Stefaan Lippens, Jean-Pierre Martens, and Tom De Mulder. A comparison of human and automatic musical genre classification. In *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2004, Montreal, Quebec, Canada, May 17-21, 2004*, pages 233–236. IEEE, 2004.

[32] Benyamin Matityaho and Miriam Furst. Neural network based model for classification of music type. In *Electrical and Electronics Engineers in Israel, 1995., Eighteenth Convention of*, pages 4–3. IEEE, 1995.

[33] W. S. McCulloch and W. Pitts. A logical calculus of the idea immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5 :115–133, 1943.

[34] Marvin Lee Minsky. Perceptrons : an introduction to computational geometry. 1990.

[35] Vinod Nair and Geoffrey E. Hinton. 3d object recognition with deep belief nets. In Bengio et al. [4], pages 1339–1347.

[36] François Pachet and Daniel Cazaly. A taxonomy of musical genres. In Joseph-Jean Mariani and Donna Harman, editors, *Computer-Assisted Information Retrieval (Recherche d'Information et ses Applications) - RIAO 2000, 6th International Conference, College de France, France, April 12-14, 2000. Proceedings*, pages 1238–1245. CID, 2000.

[37] Yannis Panagakis and Constantine Kotropoulos. Music genre classification via topology preserving non-negative tensor factorization and sparse representations. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2010, 14-19 March 2010, Sheraton Dallas Hotel, Dallas, Texas, USA*, pages 249–252. IEEE, 2010.

[38] Yannis Panagakis, Constantine Kotropoulos, and Gonzalo R. Arce. Music genre classification using locality preserving non-negative tensor factorization and sparse representations. In *Proceedings of the 10th International Society for Music Information Retrieval Conference*, pages 249–254, Kobe, Japan, October 26-30 2009. `http://ismir2009.ismir.net/proceedings/PS2-10.pdf`.

[39] Geoffroy Peeters and Enrico Marchetto. Evaluation of late fusion for classification by genre. Internal report, 2014.

[40] Marc'Aurelio Ranzato, Christopher S. Poultney, Sumit Chopra, and Yann LeCun. Efficient learning of sparse representations with an energy-based model. In *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006*, pages 1137–1144, 2006.

[41] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Parallel distributed processing : Explorations in the microstructure of cognition, vol. 1. chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.

[42] Erik Schmidt and Youngmoo Kim. Learning rhythm and melody features with deep belief networks. In Alceu de Souza Britto Jr., Fabien Gouyon, and Simon Dixon, editors, *ISMIR*, pages 21–26, 2013.

[43] Bob L. Sturm. An analysis of the GTZAN music genre dataset. In Cynthia C. S. Liem, Meinard Müller, Steven K. Tjoa, and George Tzanetakis, editors, *Proceedings of the second international ACM workshop on Music information retrieval with user-centered and multimodal strategies, MIRUM '12, Nara, Japan, October 29 - November 02, 2012*, pages 7–12, 2012.

[44] Bob L. Sturm. A survey of evaluation in music genre recognition. *Adaptive Multimedia Retrieval*, October 2012.

[45] Bob L. Sturm. Classification accuracy is not enough : On the evaluation of music genre recognition systems. *Journal of Intelligent Information Systems*, 41(3) :371–406, 2013.

[46] Bob L. Sturm. The GTZAN dataset : Its contents, its faults, their effects on evaluation, and its future use. *CoRR*, abs/1306.1461, 2013.

[47] Ilya Sutskever and Tijmen Tieleman. On the convergence properties of contrastive divergence. In Yee Whye Teh and D. Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, volume 9 of *JMLR Proceedings*, pages 789–795. JMLR.org, 2010.

[48] George Tzanetakis and Perry R. Cook. Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing*, 10(5) :293–302, 2002.