2M11.3507.6

Université de Montréal

Apprentissage à base de gradient pour l'extraction de caractéristiques dans les signaux sonores complexes

par
Alexandre Lacoste

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

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

Décembre, 2006

© Alexandre Lacoste, 2006.

Faculté des études supérieures
Grade conféré à compter du
2007 JUIN 07
Université de Montréal

QA
76
U54
2007
v.022

Université de Montréal

Direction des bibliothèques

AVIS

L'auteur a autorisé l'Université de Montréal à reproduire et diffuser, en totalité ou en partie, par quelque moyen que ce soit et sur quelque support que ce soit, et exclusivement à des fins non lucratives d'enseignement et de recherche, des copies de ce mémoire ou de cette thèse.

L'auteur et les coauteurs le cas échéant conservent la propriété du droit d'auteur et des droits moraux qui protègent ce document. Ni la thèse ou le mémoire, ni des extraits substantiels de ce document, ne doivent être imprimés ou autrement reproduits sans l'autorisation de l'auteur.

Afin de se conformer à la Loi canadienne sur la protection des renseignements personnels, quelques formulaires secondaires, coordonnées ou signatures intégrées au texte ont pu être enlevés de ce document. Bien que cela ait pu affecter la pagination, il n'y a aucun contenu manquant.

NOTICE

The author of this thesis or dissertation has granted a nonexclusive license allowing Université de Montréal to reproduce and publish the document, in part or in whole, and in any format, solely for noncommercial educational and research purposes.

The author and co-authors if applicable retain copyright ownership and moral rights in this document. Neither the whole thesis or dissertation, nor substantial extracts from it, may be printed or otherwise reproduced without the author's permission.

In compliance with the Canadian Privacy Act some supporting forms, contact information or signatures may have been removed from the document. While this may affect the document page count, it does not represent any loss of content from the document.

Université de Montréal
Faculté des études supérieures

Ce mémoire intitulé:

Apprentissage à base de gradient pour l'extraction de caractéristiques dans les signaux sonores complexes

présenté par:

Alexandre Lacoste

a été évalué par un jury composé des personnes suivantes:

Max Mignotte
président-rapporteur

Douglas Eck
directeur de recherche

Yoshua Bengio
membre du jury

Mémoire accepté le 23 avril 2007

RÉSUMÉ

Dans ce mémoire, nous présentons des techniques à base de gradient pour résoudre des tâches dans le domaine de la recherche d'information musicale. L'emphase est mise sur le réseau de neurones à convolution afin de déterminer son habileté à résoudre les problèmes avec un minimum de prétraitements spécifiques.

Dans un premier temps, nous présentons les travaux préliminaires qui nous ont menés à ces recherches. Il s'agit de la détection automatique d'évènements musicaux à l'aide de réseaux de neurones *feedforward*. Les résultats obtenus lors du concours MIREX 2005 montre une robustesse et une flexibilité accrue par rapports aux autres algorithmes. Des performances de 80 % sont obtenues pour notre méthode alors que les autres participants se situent sous 75 %.

Pour approfondir le sujet, nous avons, par la suite, utilisé le réseau de neurones à convolution sur la même tâche. L'algorithme est testé sur un ensemble de données construit pour ce travail et obtient des résultats supérieurs à la méthode précédente. Une performance de 89 % est obtenue pour celui-ci alors que le réseau de neurones *feedforward* se limite à 86 %. D'autres tests montrent qu'il est possible, moyennant une légère baisse de performances, de minimiser le traitement pour obtenir une vitesse de calcul 100 fois plus élevée que la première version de l'algorithme.

Finalement, nous avons aussi appliqué le réseau de neurones à convolution sur la tâche de classification du genre musical. Des résultats acceptables sont obtenus mais ne compétitionnent pas avec les algorithmes modernes de ce domaine. Notre méthode obtient 72 % de classification alors qu'il est possible d'obtenir 78 % avec les méthodes proposées dans la littérature. Néanmoins, en combinant les deux méthodes, nous obtenons un résultat de 81 %.

Mots clés: Apprentissage statistique, Réseau de neurones à convolution, Recherche d'information musicale, Détection automatique d'évènements musicaux, Classification du genre musical.

ABSTRACT

In this Master Thesis, we present gradient based methods for solving music information retrieval tasks. We mainly focus on the convolutional neural network architecture, to evaluate its ability at solving those tasks with relatively little of prior knowledge.

In the first part, we present preliminary works which have led us in that direction. It concerns methods for extracting note onsets from a musical piece, using a feedforward neural network. The results obtained from the MIREX 2005 contest show a robustness and flexibility higher than other participants. Our algorithm scored 80 % of F-measure when other participants were limited to 75 % of F-measure.

As a second approach, we have applied the convolutional neural network on the same task. For a more accurate evaluation, we have tested our algorithm on a custom dataset, built for this purpose. The obtained results are in favor of the convolutional neural network with 89 % of F-measure compared to 86 % for the feed forward neural network. Other tests show that it is possible, while loosing some performance, to accelerate the speed of the algorithm by a factor 100 by reducing the size of the filters.

Finally, we also applied the convolutional neural network to the music genre classification task. While acceptable results are obtained, state of the art algorithms are more appropriate. Our approach obtained 72 % of classification rate while it is possible to obtain 78 % with state of the art techniques. However, by combining the two approaches, we have obtained 81 % of classification rate.

Key words : Machine learning, Feedforward neural network, Convolutional neural network, Music information retrieval, Note onset detection, Music genre classification.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES

DAEM	:	Détection automatique d'évènements musicaux
DCT	:	Délai de la courbe tutrice
HIE	:	Histogramme d'inter-évènements
HMM	:	Modèle de Markov caché
Log-mel-TFL	:	Spectrogramme à échelle mel et amplitude logarithmique
LT#	:	Largeur temporelle du filtre de la couche #
NF#	:	Nombre de filtres pour la couche #
NFM	:	Nombre de fréquences MEL extraites
RIM	:	Recherche d'information musicale
RNC	:	Réseau de neurones à convolution
RNDT	:	Réseau de neurones à délai temporel
RNF	:	Réseau de neurones feedforward
SVM	:	Machine à vecteurs de support
TFL	:	Transformée de Fourier locale
TQC	:	Transformée à Q-constant
TS#	:	Nombre de translations spectrales pour la couche #

NOTATION

Dans ce document, à moins d'indications contraire, à l'intérieur d'un contexte mathématique, nous utilisons les lettres en caractère gras pour représenter des vecteurs ($\mathbf{x}$), des majuscules pour représenter des matrices (A) et des minuscules pour représenter un variable scalaire (y).

Les expressions anglaises où une traduction adéquate n'a pu être trouvée sont écrites en *caractère italique*.

REMERCIEMENTS

Je souhaite remercier particulièrement James Bergstra qui, à mainte reprise, a eu la patience de m'écouter parler de théories prématurées. Malgré que ces idées n'ont pas toutes portées fruit, plusieurs sont à l'origine de ce mémoire et n'auraient pas vu le jour sans le soutient de ce collègue.

Plusieurs personnes m'ont aussi aidées pour la correction de ce mémoire. Parmi celles-ci, je nomme Julie Carreau, Catherine Coulombe, Hugo Larochelle, Jean Lacroix, Estelle Bouthillier, Thierry Bertin-Mahieux, Sean Wood et Stanislas Lauly.

Finalement, j'aimerais remercier, Douglas Eck, mon directeur de recherches pour son initiative dans le domaine de la recherche d'information musicale ainsi que pour son appui financier.

CHAPITRE 1

INTRODUCTION

Écouter de la musique est une action que nous effectuons tous les jours. Il s'agit en général d'une activité plaisante voir même relaxante. Pourtant, pour apprécier une mélodie, notre cerveau doit effectuer un travail laborieux. Plusieurs dizaines de milliers de variables doivent être combinées de manière appropriée afin de reconnaître les notes qui la composent. Par la même occasion, il nous est possible de percevoir plusieurs autres caractéristiques telles que le type d'instruments utilisés, le style de l'artiste, le genre musical ...

Une question intéressante se pose alors : le cerveau est-il le seul mécanisme capable de percevoir ces propriétés ? Évidemment, il est le seul à pouvoir les apprécier. Cependant, il existe déjà plusieurs algorithmes informatiques capables d'extraire des notes d'un signal musical digitalisé. Par la suite, d'autres algorithmes sont capables d'analyser ces séquences de notes pour les catégoriser ou les comparer entre elles.

Malheureusement, ces algorithmes sont généralement très sensibles au bruit et ne fonctionnent que pour un ensemble restreint de pièces musicales. Pour ces raisons, le présent travail explore de nouvelles méthodes pour extraire des caractéristiques sonores afin de faire évoluer le domaine de la recherche d'information musicale (RIM).

1.1 Motivation

L'univers de la musique est souvent perçu comme étant principalement récréatif et le manque de rationalisme attire rarement l'intérêt des scientifiques. D'un autre point de vue, les artistes sont souvent réticents à laisser la science se mêler de la musique de peur de voir la créativité et l'imagination disparaître.

Cependant, nous vivons actuellement un rapprochement important entre ces deux domaines. De nouveaux sons, apparaissant grâce à des algorithmes informatiques, permettent une plus grande créativité pour les artistes. De nouveaux algorithmes d'identification musicale permettent de retrouver presque instantanément le titre d'une chanson à partir de son contenu en recherchant parmi plusieurs millions de pièces musicales.

Ces algorithmes sont intéressants mais l'application la plus en demande à l'heure actuelle est certainement la recommandation musicale. De nouveaux algorithmes sont maintenant capables de faire découvrir de nouvelles pièces musicales à des auditeurs à partir de leurs préférences. Certains systèmes utilisent uniquement les préférences d'une collectivité de membres mais, avec les développements en RIM, il est maintenant possible d'utiliser le contenu de la pièce musicale pour orienter la recommandation. Cette dernière amélioration permet à des artistes moins connus de se faire découvrir, favorisant la musique créative au détriment de la musique populaire. La demande pour un tel système est élevée au point tel que les compagnies informatiques se bousculent pour prendre le marché.

La transcription automatique de pièces musicales constitue aussi un domaine important en RIM. Cette tâche consiste à trouver la séquence de notes qui est jouée dans une chanson. Cette information peut alors être utilisée pour produire la partition, pour compresser l'extrait musical ou pour rechercher une mélodie similaire dans une base de données. Cette dernière utilisation pourrait même servir à retrouver une chanson dans une base de données en sifflant son refrain.

D'un point de vue scientifique, la recherche en musique apporte un tout autre environnement pour faire évoluer les algorithmes d'apprentissage statistique. Ceux-ci bénéficient de la diversité des tâches à résoudre pour faire évoluer leur flexibilité. De plus, des techniques développées pour la musique peuvent être réutilisées pour des tâches connexes telles que la reconnaissance de la parole ou le traitement de séquences vidéo.

1.2 Problématique

Le domaine de la RIM comprend plusieurs défis majeurs. Parmi ceux-ci, une problématique importante est liée à la quantité massive d'information contenue dans une chanson, offrant un défi tant au niveau calculatoire qu'au niveau algorithmique. Un autre défi intéressant consiste à développer des algorithmes flexibles et facilement ajustables, pour pouvoir les utiliser dans une plus grande variété de contexte. Commençons tout d'abord par discuter des difficultés concernant la robustesse des algorithmes.

1.2.1 Robustesse

Les techniques de traitement de signaux généralement utilisées sont souvent simples et rapides. Mais elles sont, pour la plupart, développées manuellement et testées sur un très petit ensemble d'exemples. Ce handicap restreint généralement les algorithmes à ne fonctionner que pour des cas simples et peu bruités. Par exemple, si la variation d'amplitude sonore est utilisée pour détecter les évènements dans un signal, les variations de tonalités ne seront pas détectées et le bruit pourra déclencher des évènements non désirés.

De plus, à l'aide de techniques de traitements de signaux, il est généralement difficile de détecter des caractéristiques abstraites ou complexes. Par exemple, si nous voulons détecter le genre musical, il nous faut combiner, de manière abstraite, plusieurs variables décrivant le timbre des instruments ainsi que le rythme de la chanson.

Pour palier à ces deux problèmes, il est préférable d'utiliser le domaine de l'apprentissage statistique. Grâce à ces outils, il est possible à partir d'exemples identifiés par des humains, de trouver automatiquement une fonction adéquate. Si l'ensemble d'exemples est suffisant riche et si l'algorithme d'apprentissage possède une flexibilité appropriée, il sera possible d'obtenir une fonction robuste capable de combiner l'information de manière abstraite.

1.2.2 Haute dimensionnalité

Pour évaluer la quantité d'informations contenues dans une pièce musicale, considérons d'abord le processus de digitalisation des signaux sonores. Le théorème de Nyquist-Shannon nous indique qu'il nous faut au minimum deux échantillons par cycle pour percevoir une fréquence. De ce fait, si nous mesurons périodiquement la pression de l'air à un rythme deux fois plus élevé que la fréquence maximale que nous voulons percevoir, nous pouvons capturer notre signal sonore. Cette technique d'échantillonnage est nommée "modulation d'impulsion codée" (MIC).

D'autre part, des études psycho-acoustiques démontrent que l'oreille humaine moyenne perçoit un spectre sonore allant de 20 Hz jusqu'à 20 KHz. Il nous faut donc 40000 mesures par secondes pour capturer l'information perçue par l'oreille et donc 0.4 million d'échantillons pour encoder une mélodie de 10 secondes.

Cette quantité d'information cause un problème majeur pour les algorithmes d'apprentis-

sage statistique. En général, plus la taille de l'entrée est élevée, plus l'algorithme est complexe et une complexité accrue augmente les problèmes de généralisation (Section 2.3.1). Une solution fréquente pour ce problème est d'utiliser du prétraitement, pour réduire la taille de l'entrée ou simplement extraire de l'information pertinente. Malheureusement, une grande perte d'information accompagne souvent cette méthode. Pour minimiser la perte d'information, le développeur doit investiguer le domaine entourant la tâche en question et explorer différentes méthodes de prétraitement. Ceci augmente le temps de développement et diminue la flexibilité de l'algorithme, tel que discuté dans la section qui suit.

1.2.3 Réutilisation

Tel que présenté à la section 5.3, le domaine de la RIM est composé d'une multitude de tâches différentes. Plusieurs algorithmes de base ont déjà été proposés dans ce domaine mais la plupart d'entre eux ont été développés dans un contexte particulier et sont difficilement réutilisables. Prenons par exemple le cas d'un algorithme de détection d'évènements sonores. S'il a été développé pour bien fonctionner avec des instruments harmoniques, il se retrouvera probablement désorienté lorsque d'autres types d'instruments seront présents. Tout simplement, il est possible qu'un autre utilisateur ait une définition différente d'évènements, désirant ignorer les variations de tonalités, pour ne considérer que les variations d'amplitude.

La problématique majeure est que ces algorithmes possèdent généralement une grande dose de traitements très spécifiques au problème ainsi que des paramètres sensibles ajustés par le concepteur. Par conséquent, pour pouvoir appliquer le programme à une variante différente de la tâche, il faut faire appel à l'auteur ou même faire appel à un autre expert pour reconcevoir l'algorithme.

Pour ces raisons, dans nos travaux, nous considérons les techniques d'apprentissage statistique qui minimisent la quantité de prétraitement requis. Plus précisément, nous tentons d'éviter d'inclure dans l'architecture les connaissances spécifiques que nous avons sur le problème. Cette direction favorise énormément la réutilisation de l'algorithme. Cependant, ce gain a un coût car il est fréquent d'observer une performance plus élevée pour les algorithmes ayant subi un développement approfondi.

1.3 Aperçu des chapitres

Dans le présent travail, nous visons donc à développer des algorithmes ayant de meilleures performances dans certaines tâches de RIM. Cependant, il ne s'agit pas de l'objectif principal. Nous visons essentiellement à tester le potentiel du réseau de neurones à convolution (RNC) (Section 2.4) pour la résolution de problèmes en RIM. Cet algorithme, décrit en détails dans le prochain chapitre, utilise le concept de partage de paramètres pour éviter d'augmenter la complexité de l'algorithme lorsque la taille de l'entrée augmente. Cette caractéristique permet alors d'éviter le prétraitement spécifique à une tâche et favorise grandement la réutilisation de l'algorithme.

La première partie de ce mémoire se concentre sur les préalables nécessaires pour comprendre les outils utilisés au cours des autres chapitres. Par la suite, nous présentons un travail préliminaire effectué sur la détection d'évènements dans les signaux sonores. Au chapitre 4, nous reprenons les travaux effectués sur la détection d'évènements à l'aide du RNC. Finalement, nous analysons le comportement du RNC sur la tâche de classification du genre musical.

1.3.1 Chapitre 2 : Préalables

Dans ce chapitre, nous décrivons premièrement les techniques d'apprentissage statistique utilisées au cours de ce mémoire. Ces sections couvrent ce qui est nécessaire pour comprendre comment optimiser un réseau de neurones *feedforward (RNF)* ainsi qu'un RNC. Pour ce faire, nous décrivons le concept d'apprentissage à base de gradient, les méthodes d'optimisation ainsi que le concept de généralisation en apprentissage statistique.

Par la suite, nous décrivons les techniques de prétraitement utilisées. Nous nous sommes limités à représenter notre signal sous forme de spectrogramme mais nous avons exploré trois différents types de spectrogrammes. Le premier étant la transformée de Fourier locale, décrivant les fréquences selon une échelle linéaire. Le deuxième étant la transformée à Q-constant, décrivant les fréquences selon une échelle logarithmique. Finalement, nous expliquons le spectrogramme à échelle de MEL, une échelle semi-linéaire et semi-logarithmique s'inspirant de la perception humaine.

1.3.2 Chapitre 3 : Réseaux Neuronaux Feedforward pour la détection d'évènements

Ce chapitre correspond à une traduction de [LE06], un article sur la détection d'évènements, publié à la suite du concours de détection d'évènements MIREX 2005. La méthode consiste essentiellement à appliquer un RNF sur une courte séquence de trames d'un spectrogramme pour évaluer si cet instant contient un évènement. En évaluant chaque instant, nous obtenons une courbe de crêtes déterminant la position des évènements.

1.3.3 Chapitre 4 : Réseaux Neuronaux à convolutions pour la détection d'évènements

La technique décrite au chapitre 3 correspond essentiellement à la convolution d'un RNF. Dans le chapitre 4, nous explorons le gain apporté par un vrai RNC où chaque couche est convoluée séparément. Plusieurs tests sont effectués pour mettre en valeur les propriétés du RNC. Finalement, les résultats démontrent que la quantité réduite de paramètres dans le cas du RNC permet d'obtenir une meilleure généralisation.

1.3.4 Chapitre 5 : Réseaux Neuronaux à convolutions pour la classification musicale

Dans ce dernier chapitre, nous nous intéressons essentiellement à savoir si le RNC peut être appliqué à la tâche de classification de genre et obtenir une performance équivalente aux algorithmes modernes. Pour ce faire, nous comparons les performance obtenues par le RNC à un algorithme développé dans [Ber06], s'inspirant des techniques modernes. Les résultats montrent une performance significativement inférieure pour le RNC mais suffisamment élevée pour le considérer comme un candidat intéressant dans le domaine de la RIM. Les résultats montrent aussi qu'en combinant les deux algorithmes, nous pouvons obtenir une performance supérieure.

CHAPITRE 2

PRÉALABLES

Le domaine de l'apprentissage statistique nous offre une vaste gamme d'algorithmes d'apprentissage. Cependant, pour notre travail, nous explorons des problèmes d'apprentissage supervisé. De plus, nous voulons explorer la possibilité de minimiser la réduction de dimensionnalité manuellement. Pour ce faire, notre algorithme doit être capable de supporter une grande quantité de variables d'entrée, tout en gardant un temps d'apprentissage raisonnable. Compte tenu des ces restrictions, le réseau de neurones *feedforward* (RNF) s'avère être un bon candidat.

Dans cette section, nous décrivons le fonctionnement du RNF ainsi que différents algorithmes servant à optimiser leurs paramètres. Nous introduisons aussi une catégorie plus spécifique de RNF, le réseau de neurones à convolution (RNC). Ce dernier permet de réduire significativement la quantité de paramètres requis pour résoudre un problème, en réutilisant ceux-ci à plusieurs endroits différents dans un signal. Cependant, cette réduction de paramètres n'est possible que pour les signaux possédant une invariance de translation comme présentés à la section 2.4.

Bien que nous voulions minimiser la réduction de dimensionnalité manuelle, nous utilisons quelques formes de prétraitement générique, permettant de présenter les signaux sonores à notre RNF sous une forme plus appropriée. Ces manipulations, grandement inspirées de la perception psycho-acoustique de l'être humain, transforment le signal d'une forme temporelle vers une forme spectro-temporelle. Cette dernière est effectuée en transformant une courte période de temps de notre signal en son équivalant spectrale. Dans la section 2.5, nous présentons différentes méthodes pour obtenir une telle représentation.

2.1 Réseau de neurones *feedforward*

Le réseau de neurone *feedforward* (RNF) est une fonction $g : \mathbb{R}^m \rightarrow \mathbb{R}^n$ où son comportement exact est défini par l'intermédiaire d'un ensemble de paramètres $\mathbf{w}$. Si nous possédons une fonction nous permettant d'évaluer le coût en fonction de $\mathbf{w}$, nous pouvons alors cher-

cher la valeur des paramètres qui nous conviens le mieux. Dans cette section, nous décrivons la structure de g alors que dans la section suivante, nous décrivons comment chercher une valeur optimale pour $\mathbf{w}$.

Malgré que le RNF s'inspire du modèle connexionniste, il est mathématiquement plus simple de le percevoir comme une séquence de transformations affines entrelacées de fonctions non linéaires appelées fonction d'activation.

Une couche de neurones peut alors s'exprimer ainsi :

$$c(\mathbf{x}; A, \mathbf{b}) = a(A\mathbf{x} + \mathbf{b}) \tag{2.1}$$

où a est la fonction d'activation, A est une matrice de paramètres de taille m par n, appliquée sur l'entrée $\mathbf{x}$ de taille m. $\mathbf{b}$ est un vecteur de paramètres de taille n permettant d'appliquer un biais. Dans le modèle connexionniste, m correspond au nombre de synapses d'un neurone et n, le nombre de neurones sur cette couche. La connectivité globale du RNF impose que le nombre de synapses soit de même dimensionnalité que le vecteur d'entrée $\mathbf{x}$.

Pour obtenir un RNF, il suffit tout simplement d'imbriquer plusieurs couches.

$$g(\mathbf{x}; \mathbf{w}) = (c_k \circ c_{k-1} \circ \cdots \circ c_1)(\mathbf{x}) \tag{2.2}$$

où $\circ$ correspond à l'opérateur de composition de fonctions et $\mathbf{w}$ correspond à l'ensemble des paramètres A et b de chaque couche. La dernière couche est communément appelée la couche de sortie et les autres, appelées : couches cachées.

L'ancêtre du RNF est le Perceptron, un RNF à une seule couche [MP69]. Ce dernier, étant limité à une séparation linéaire de son domaine, est incapable d'implémenter certaines fonctions simples telles que le "ou exclusif". Pour pallier à ce problème, une composition de plusieurs Perceptrons a été implémentée, donnant naissance au Perceptron à plusieurs couches, synonyme de RNF. Il a été démontré par [HSW89] que le RNF à une ou plusieurs couches cachées correspond à un approximateur universel (c.-à-d. capable d'approximer arbitrairement toute fonction continue, pourvu que le nombre de neurones soit suffisamment grand).

Bien qu'une seule couche cachée soit généralement suffisante, il est commun d'observer

de meilleurs résultats avec deux couches cachées. Cependant, plus de deux couches cachées apportent rarement un bénéfice. Ce résultat est contre-intuitif puisqu'en informatique, il est fréquent de résoudre des problèmes avec une composition profonde de fonctions simples. De plus, plusieurs problèmes informatiques possèdent une borne inférieure sur leur profondeur de circuit pour obtenir une réponse exact. Le problème de parité est un exemple de fonction qui se résoud beaucoup mieux avec un réseau profond. Ce problème consiste à trouver la parité d'une chaîne de d bits et requiert $O(d^2)$ paramètres pour être résolu à l'aide d'une seule couche cachée alors qu'il en faut $O(d)$ pour le résoudre à l'aide de d couches cachées. Il est même possible de résoudre ce problème avec $O(1)$ paramètres si un réseau de neurones récurrent est utilisé.

Il a récemment été identifié dans [BLPL07] que la baisse de performance des réseaux profonds peut provenir d'une mauvaise optimisation des paramètres. Un réseau génératif profond ([HOT06]) est alors utilisé pour initialiser le réseau de neurones dans un état plus approprié. Par la suite, l'entraînement se continue par une rétropropagation de l'erreur. Cette technique permet d'augmenter le nombre de couches et d'obtenir de meilleurs performances.

2.2 Fonction de coût

Pour pouvoir entraîner notre réseau (c.-à-d. trouver des valeurs de paramètres appropriées), il nous faut obtenir une valeur de performance globale, nous permettant de choisir parmis les différents ensemble de paramètres. Pour ce faire, nous utilisons un ensemble de paires $\mathbf{x}_i$ et $\mathbf{t}_i$ où $\mathbf{x}_i$ est notre vecteur de variables d'entrée et $\mathbf{t}_i$ désigne la sortie désirée de notre réseau. À l'aide de ce dernier, nous pouvons utiliser une fonction de coût, pour évaluer l'erreur sur un exemple d'apprentissage.

$$e_i = f\left(g\left(\mathbf{x}_i; \mathbf{w}\right), \mathbf{t}_i\right)$$

où $f(\cdot)$ est la fonction de coût. Nous pouvons combiner plusieurs erreurs e_i en moyennant sur différents éléments de notre ensemble de données, pour obtenir une valeur scalaire décrivant la performance de notre modèle.

$$E = \langle e_i \rangle$$

À l'aide de cet indice de performance, nous pouvons modifier la valeur des paramètres $\mathbf{w}$, et choisir celle qui convient le mieux.

Si les fonctions f et g sont différentiables, nous pouvons trouver le gradient de E par rapport à $\mathbf{w}$, nous procurant la direction et l'amplitude de la pente la plus élevée. Il suffit donc de se déplacer dans la direction inverse pour obtenir une erreur plus faible.

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \frac{\partial E}{\partial \mathbf{w}} \tag{2.3}$$

où α est un paramètre déterminant la vitesse d'apprentissage. Évidemment, cette méthode, appelée descente de gradient, ne fait que trouver un minimum local dans le voisinage du point de départ, mais ce résultat est régulièrement suffisant pour un apprentissage de nos paramètres $\mathbf{w}$.

Dans plusieurs cas, descendre le gradient peut constituer une méthode très lente pour optimiser notre fonction f. Heureusement, la littérature sur l'apprentissage statistique nous offre plusieurs alternatives pour en accroître la vitesse. Dans la section suivante, nous discuterons quelques-unes de ces méthodes, qui seront utilisées au cours de ce document.

2.3 Algorithmes d'optimisation

Dans la section précédente, nous avons montré que le RNF constitue une fonction continue et dérivable, qui, à l'aide d'un ensemble de valeurs désirées, peut être englobée dans une fonction qui retourne une valeur scalaire, qualifiant le système. Ce type de problème peut être représenté comme suit :

$$E = f(D; \mathbf{w})$$

où $\frac{\partial E}{\partial \mathbf{w}}$ existe sur tous le domaine. D constitue notre ensemble de données où, dans le cas du RNF, il s'agit de l'ensemble des paires $\{\mathbf{x}_i, \mathbf{t}_i\}$.

Dans cette section, nous décrivons quelques algorithmes conçus pour optimiser des fonctions possédant ces caractéristiques. Mais tous d'abord, il nous faut déterminer ce que nous voulons optimiser.

2.3.1 Généralisation

Si nous optimisons notre modèle de sorte à minimiser l'erreur sur tous les exemples que nous possédons, il est fort probable que nous rencontrions le problème de sur-apprentissage. C'est-à-dire que notre modèle aura la possibilité de très bien s'ajuster aux exemples que nous avons, tout en négligeant les exemples que nous pourrions rencontrer. La figure 2.1a présente un exemple de sur-apprentissage en une dimension. La courbe pointillée représente le vrai modèle alors que la courbe pleine représente le modèle appris à l'aide des points bleus. Si notre modèle possède suffisamment de flexibilité, il est possible qu'il s'adapte à tous les points que nous possédions, mais rien ne garantit qu'il sera capable de généraliser sur d'autres points.

Ceci nous amène à un point important : le but en apprentissage statistique n'est pas de minimiser l'erreur sur les exemples que nous possédons, mais plutôt de minimiser l'erreur sur les exemples que nous ne possédons pas. Une méthode simple pour améliorer la situation est de cacher certains exemples à notre algorithme d'optimisation. Ceci nous permettra d'estimer comment il se comportera en présence d'exemples inconnus.

Cette mesure donne place à plusieurs façons d'améliorer l'erreur de généralisation. Une méthode nommée arrêt prématuré permet de couper l'optimisation avant que le sur-apprentissage devienne trop important. Pour ce faire, il suffit de valider notre modèle à chaque étape par des exemples inconnus de notre optimiseur et d'arrêter l'algorithme au moment où l'erreur de validation atteint son minimum (figure 2.2).

Une autre méthode pour atténuer le sur-apprentissage consiste à minimiser à la fois l'erreur et la norme du vecteur de paramètres.

$$L = E + \lambda |\mathbf{w}|^2$$

où λ est un hyper-paramètre permettant de quantifier l'importance de minimiser la norme des paramètres. L devient alors la nouvelle valeur à minimiser. Cette technique peut ne pas être avantageuse dans toutes les circonstances. Par exemple, en appliquant une restriction sur la norme des paramètres, nous affirmons que notre modèle est plus susceptible de bien fonctionner pour de petites valeurs de $\mathbf{w}$, ce qui peut ne pas être le cas. Nous pouvons alors

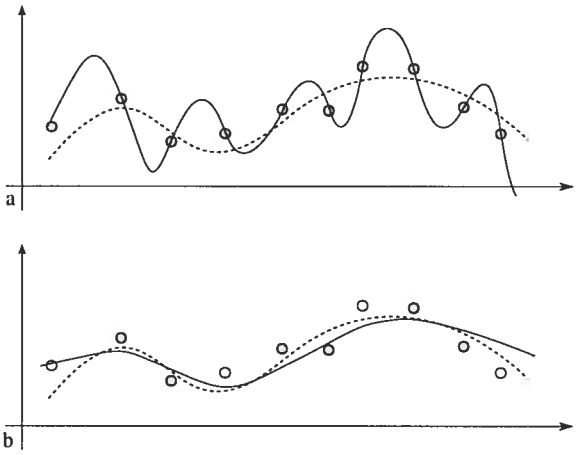

FIG. 2.1 – La courbe pointillée représente le vrai modèle alors que la courbe pleine représente la courbe apprise à partir des points bleus. L'image *a* nous présente un exemple de sur-apprentissage. La courbe apprise passe par tous les points, permettant d'obtenir une erreur d'optimisation presque nulle. Dans l'image *b*, les points blancs représentent des exemples retirés de l'ensemble d'apprentissage pour pouvoir arrêter l'optimisation avant le sur-apprentissage.

ajuster la valeur de α pour optimiser l'erreur de validation. Cette valeur ne peut pas être ajustée sur l'ensemble d'entraînement en même temps que les autres paramètres, puisqu'elle convergerait automatiquement vers 0. C'est pourquoi elle constitue un hyper-paramètre.

2.3.2 Algorithmes profitant de la Hessienne

L'algorithme de la descente de gradient exprimé à l'équation 2.3, possède quelques inconvénients, qui peuvent parfois le rendre très lent. Dans le cas où la courbure selon un axe est

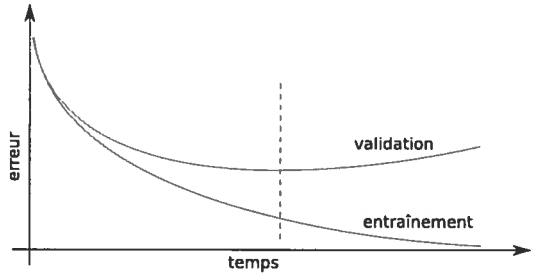

FIG. 2.2 – Exemple d'une courbe d'entraînement accompagnée de la courbe de validation. Nous pouvons voir clairement que passé un certain temps, l'erreur de validation commence à augmenter.

beaucoup plus élevée que dans un autre, le gradient pointera dans une direction très différente de celle du minimum (figure 2.3). Pour pallier à ce problème, plusieurs algorithmes tels que Levenberg Marquadt [Mar63] et Quasi-Newton [Bis95], utilisent l'information apportée par la Hessienne pour effectuer des déplacements plus appropriés. La Hessienne est calculée à l'aide de la dérivée seconde et contient l'information sur la courbure de l'espace.

$$H_{ij} = \frac{\partial^2 E}{\partial w_i \partial w_j}$$

Ces techniques permettent d'optimiser la fonction en beaucoup moins d'étapes, cependant, le calcul de la Hessienne coûte un temps $O(n^2)$ où n est la cardinalité de $\mathbf{w}$, alors que le calcul du gradient ne coûte que $O(n)$. Il s'ensuit que, pour un problème relativement complexe où n est élevé, ces deux dernières méthodes deviennent impraticables.

Il existe cependant l'algorithme du Gradient Conjugué [Bis95], qui profite implicitement de l'information sur la Hessienne et qui consomme un temps $O(n)$ pour chaque étape. L'algorithme trouve d'abord une direction *conjuguée* et performe une optimisation unidimensionnelle selon cet axe. Deux directions sont considérées *conjuguées* lorsque :

$$\mathbf{d}_i^{\mathrm{T}} H \mathbf{d}_j = 0 \qquad j \neq i \tag{2.4}$$

où $\mathbf{d}_i$ correspond à la direction d'optimisation de l'étape i. Dans le cas où la Hessienne correspond à l'identité, l'équation 2.4 force la recherche dans des directions qui sont mutuellement perpendiculaires. Cependant, si une courbure est plus prononcée selon un axe, la Hessienne ne sera plus l'identité et l'équation 2.4 mettra l'emphase sur l'axe élancé, ce qui résoud essentiellement le problème principal de la descente de gradient. L'avantage avec cet algorithme, c'est qu'il n'est pas nécessaire de connaître explicitement H.

$$\mathbf{d}_{i+1} = -\mathbf{g}_{i+1} + \beta_i \mathbf{d}_i \tag{2.5}$$

$$\beta_i = \frac{\mathbf{g}_{i+1}^{\mathrm{T}} (\mathbf{g}_{i+1} - \mathbf{g}_i)}{\mathbf{g}_i^{\mathrm{T}} \mathbf{g}_i} \tag{2.6}$$

où l'équation 2.6 correspond à la version *Polak-Ribiere* du gradient conjugué. L'équation 2.5

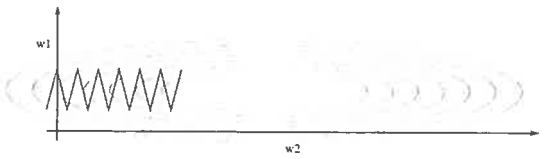

FIG. 2.3 – Courbes de niveaux représentant un espace de minimisation quadratique en deux dimensions où la courbure d'un axe est beaucoup plus prononcée que l'autre. La trajectoire oscillante représente le déplacement du vecteur de paramètres $\mathbf{w}$. Dans une situation de courbure inégale, la progression vers le minimum peut devenir très lente.

nous indique qu'il suffit de garder en mémoire l'ancien gradient ($\mathbf{g}_i$) et l'ancienne direction de recherche pour trouver la nouvelle direction. Finalement, l'évolution des paramètres s'écrit comme suit :

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \alpha_i \mathbf{d}_i$$

Il est possible d'estimer α_i à l'aide de la Hessienne, mais comme nous voulons éviter de calculer celle-ci, il nous faut effectuer une minimisation unidimensionnelle le long de l'axe $\mathbf{d}_i$. Pour cette étape, plusieurs algorithmes sont disponibles, mais le plus couramment utilisé est [Cha92].

2.3.3 Méthodes stochastiques

La méthode du *Gradient Conjugué,* présentée dans la section précédente permet de profiter de l'information sur la Hessienne, tout en gardant une complexité linéaire. Cependant, pour fonctionner, l'ensemble de donnés D_t doit être le même que D_{t+1}. Dans le cas contraire, les directions ne sont plus conjuguées. Cette restriction nous oblige à calculer le gradient sur toutes nos données avant de procéder à une modification de notre vecteur de paramètres. Comme la taille de notre ensemble de données grandit généralement avec la complexité du problème à résoudre, il s'ensuit que le gradient Conjugué peut devenir inapproprié pour certains problèmes.

Dans les circonstances où nous voulons utiliser un grand ensemble de données, il est généralement préférable d'utiliser des algorithmes dits stochastique. C'est-à-dire des algorithmes qui ne peuvent fonctionner qu'avec une approximation du gradient, effectuée sur un sous-ensemble de nos données. Malheureusement, la littérature sur l'apprentissage statistique nous offre peu de variantes à ce sujet. Il nous faut donc repartir avec des algorithmes similaires

à la descente du gradient. Une légère modification peut être appliquée à celui-ci pour en améliorer les performances. Il s'agit de représenter le problème d'optimisation comme une bille subissant la gravité dans un monde multidimensionnel et visqueux. Dans cette nouvelle représentation, la descente du gradient n'est que le cas spécifique où la masse de la bille est nulle. Il suffit donc d'attribuer une masse à notre bille, pour qu'elle puisse accélérer vers le minimum et acquérir une quantité de mouvement, d'où l'attribut *momentum* caractérisant cet algorithme.

Dans cette méthode, le gradient correspond à l'accélération due à la gravité, mais il y a aussi une force de frottement due à la viscosité, qui est proportionnelle à la vitesse. Ces deux informations nous permettent de calculer la vitesse de la bille à l'aide d'une estimation numérique de l'intégrale.

$$\mathbf{v}_{t+1} = \mathbf{v}_t - \mu \mathbf{g}_t - \gamma \mathbf{v}_t \tag{2.7}$$

où μ est un paramètre qui détermine l'influence de la gravité et γ, un paramètre déterminant la viscosité. La position de la bille est tout simplement l'intégrale de la vitesse, qui peut être, encore une fois, estimée numériquement par :

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \epsilon \mathbf{v}_{t+1} \tag{2.8}$$

Nous pouvons simplifier en introduisant l'équation 2.7 dans l'équation 2.8, pour obtenir :

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \mathbf{g}_t + \beta \mathbf{v}_t$$

où α est notre vitesse d'apprentissage, au même titre que dans la version classique de la descente de gradient. β correspond indirectement à la masse de la particule et permet de déterminer l'importance de l'accumulation des gradients antérieurs. Dans le cas où β est nul, nous retrouvons le cas du gradient classique. Pour simplifier les équations, nous n'avons pas tenu compte de la composition des paramètres α et β, car ils sont généralement ajustés empiriquement afin d'obtenir une bonne convergence.

2.4 Réseau de neurones à convolution

Dans la section 2.1, nous décrivions le RNF comme étant un approximateur universel. Cependant, pour résoudre certains problèmes, la quantité de paramètres nécessaire peut être trop élevée, entraînant généralement le problème de sur-apprentissage. C'est le cas des situations où le nombre de variables d'entrée est très élevé. Généralement des techniques de réduction de dimensionnalité sont utilisées. Cependant, il est possible pour certains types de problèmes de réutiliser les mêmes paramètres pour plusieurs sous-ensembles de variables.

Le réseau de neurones à convolution (RNC), est une architecture qui permet une grande réutilisation de paramètres, pour autant que nous puissions appliquer les deux restrictions suivantes à notre problème. (A) Les propriétés de bas niveaux ne peuvent être extraites que localement et (B), le problème possède une invariance de translation. La première restriction nous permet de concentrer l'apprentissage sur un sous-ensemble des variables et la deuxième restriction nous permet de réutiliser la fonction apprise sur d'autres sous-ensembles de variables.

Le principe consiste à convoluer une banque de filtre avec le signal, où chaque filtre est appris et une fonction d'activation est appliquée pour chaque valeur du signal de sortie. Cette opération est l'équivalent d'une couche d'un RNF où les connections sont locales et les paramètres réutilisés sur plusieurs neurones. Par la suite, d'autres couches à convolution peuvent êtres appliquées, pour obtenir un traitement plus complexe. La figure 2.4 donne un exemple unidimensionnel du RNC. Le principe peut être généralisé à n dimensions, en considérant le signal d'entrée comme étant n-dimensionnel, convolué avec k filtres n-dimensionnel, retournant k signaux n-dimensionnel.

2.4.1 Littérature du RNC

Suite à l'apparition de l'algorithme de rétropropagation vers la fin des années 1980, une grande variété d'architectures de réseaux de neurones émergent. Alors que de par sa simplicité et de par sa flexibilité, le RNF prend de l'ampleur, de nombreux résultats démontrent qu'une connectivité locale et une réutilisation des paramètres peuvent être avantageuses [LeC86, LeC89, Moz87, Fuk80]. En 1990, ces indices permettent à Yann Lecun de développer une version complète du RNC [LBD+90]. Celui-ci est utilisé pour classifier des chiffres écrits à la

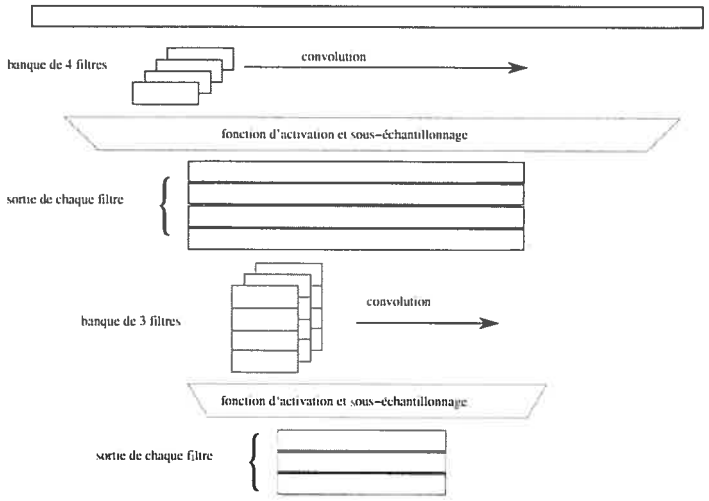

FIG. 2.4 – Représentation unidimensionnelle du réseau à convolution. Tout d'abord, le signal d'entrée est convolué avec une banque de filtres, pour générer plusieurs signaux filtrés. Par la suite, une fonction d'activation est appliquée sur chaque sortie et les signaux sont sous-échantillonnés, dans le but d'obtenir une réduction de dimensionnalité. Finalement, ce procédé peut être appliqué à plusieurs reprises pour obtenir une transformation plus complexe.

main et est capable de généraliser sur une grande variété d'écritures (figure 2.5b).

En parallèle, une version différente de l'algorithme est développée par différents auteurs. Il s'agit du réseau de neurone à délai temporel (RNDT), une convolution unidimensionnelle appliquée sur des signaux temporels. Ce réseau permet d'obtenir des résultats significativement meilleurs que le modèle de Markov caché (HMM), pour la classification de phonèmes dans les signaux vocaux [WHH+89].

Progressivement, plusieurs applications en traitement d'image bénéficient des avantages du RNC. Particulièrement quand il s'agit de classifier des objets qui peuvent se trouver à plusieurs endroits dans l'image. Un premier exemple est la classification d'objets, indépendamment de l'éclairage et de l'arrière-plan [LHB04]. Dans ce problème, les images sont stéréographiques et il y a 5 catégories d'objets tridimensionels et une sixième catégorie pour l'arrière-plan (figure 2.5a). Une comparaison est faite avec l'algorithme du *k plus proche voisin* et la *machine à vecteurs de support* et pour cette tâche, le RNC atteint une performance significativement plus élevée.

Une deuxième application similaire à la détection d'objets, est la reconnaissance de vi-

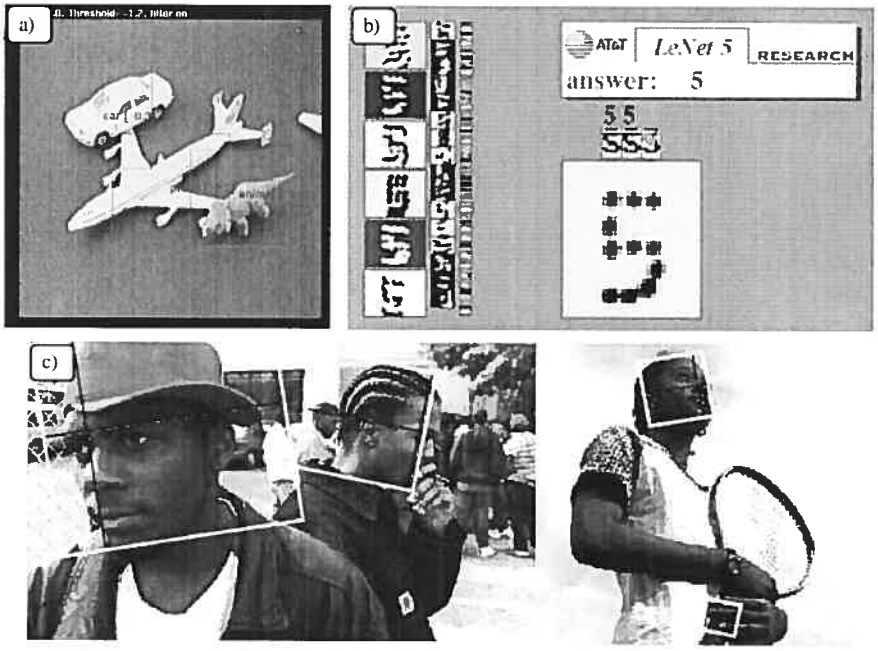

FIG. 2.5 – Résultats de différentes applications du RNC : a) Malgré la possibilité de superposition, le RNC classifie plusieurs objets dans la même image et découvre leur position ; b) Le RNC classifie des chiffres écrits à la main et permet de généraliser sur une très grande variété de styles ; c) Le RNC permet de trouver les visages dans une images, tout en estimant les angles de rotation.

FIG. 2.6 – Estimation de la performance à contourner les obstacles. La courbe grise pale est l'angle suggéré par un humain alors que la courbe pleine représente l'angle employé par le robot.

sages. Mais cette fois-ci, les angles de rotation de la tête sont aussi évalués (figure 2.5c). Pour faciliter la tâche au réseau, les deux angles sont déployés comme étant une variété dans un espace à neuf dimensions et l'absence de visage est étiquetée comme étant un point éloigné de cette variété. Les résultats démontrent que l'apprentissage des angles et de la détection à la fois aide significativement la détection de visage.

Une application, cette fois-ci plus technique, est l'esquive d'obstacles [LMB+05]. Un petit camion équipé de deux caméras doit se déplacer de manière à éviter les objets. Son seul degré de liberté est l'angle de rotation des roues. Pour apprendre celui-ci, un ensemble de séquences vidéo est manuellement identifié par des humains. L'entrée du RNC correspond à une paire d'images stéréographiques en couleur, divisé en 2 plans de luminance et 4 plans de chrominance (2*YUV). Les performances, pour un tel type de problème, sont difficiles à décrire, puisque le robot peut choisir un angle de rotation différent de celui du tuteur, tout en ayant fait un bon choix. La figure 2.6 procure un aperçu des performance du système.

2.4.2 Implémentation du RNC

Dans cette section, nous décrivons les étapes pour obtenir les équations qui nous permettent d'ajuster les poids d'une couche de RNC. La règle de dérivée en chaîne, nous permet de pouvoir se concentrer sur une couche à la fois. C'est-à-dire :

$$\frac{\partial L}{\partial w_j} = \sum_i \frac{\partial L}{\partial y_i} \frac{\partial y_i}{\partial w_j} \tag{2.9}$$

où les y_i représentent le vecteur de sortie d'une couche et $\frac{\partial L}{\partial y_i}$ est retourné par la couche qui suit, durant l'étape de rétropropagation. Utilisant cette technique, il nous suffit de déterminer

FIG. 2.7 – Différents types de corrélations représentés en une dimension. L'exemple du haut représente une corrélation interne où le filtre n'est déplacé que là où toutes les valeurs sont définies. Dans le cas où une convolution est effectuée, le filtre est inversé. C'est-à-dire : $w_i^{conv} = w_{n-i}^{corr}$. Dans le cas d'une corrélation complète, le filtre est déplacé en dehors des limites du signal et les valeurs non définies sont interprétées comme des zéros.

y_i, $\frac{\partial L}{\partial w_i}$ et $\frac{\partial L}{\partial x_i}$ où le dernier terme sera rétropropagé en tant que $\frac{\partial L}{\partial y_i}$ pour la couche précédente.

Tout d'abord, définissons quelques termes. Dans notre cas, la convolution est un abus de langage. Nous effectuons plutôt une corrélation (figure 2.7). Pour effectuer une convolution, il nous faut inverser notre filtre. C'est-à-dire : $w_i^{conv} = w_{n-i}^{corr}$. Évidemment, comme notre filtre est appris, qu'il soit inversé ou pas n'a aucune importance. Cependant, cette nomenclature nous permet de distinguer ces opérations, qui seront toutes deux utilisées dans cet ouvrage.

Dans la circonstance où les signaux à corréler sont de taille finie, il nous faut traiter les extrémités comme un cas particulier. Ici, deux méthodes différentes sont utilisées : la corrélation *interne* et la corrélation *complète* (figure 2.7). Lorsque nous parlons de corrélation *interne*, il s'agit de déplacer le filtre seulement aux endroits où toutes les valeurs sont définies. Dans le cas d'une corrélation *complète*, le filtre est déplacé en dehors des limites du signal et les valeurs non définies sont interprétées comme des zéros. Évidemment, il est inutile de continuer lorsque plus aucune valeur ne se chevauche.

L'entrée d'une couche de RNC est un tenseur tridimensionnel. C'est-à-dire, différentes images de même taille placées dans un bloc où trois indices sont nécessaires pour localiser une valeur. La sortie sera aussi tridimensionnelle mais de taille différente. Les paramètres sont composés d'une banque de filtres, où chaque filtre est aussi tridimensionnel et peut donc être vu comme un tenseur quadridimensionnel. Alors, l'équation de convolution peut être

écrite comme suit :

$$y_{ijk} = \sum_{mnp} w_{mnpk} x_{(i+m)(j+n)p} \tag{2.10}$$

Si w est de taille $M \cdot N \cdot P \cdot K$ et x de taille $Q \cdot T \cdot P$ alors, y sera de taille $(Q - M + 1) \cdot (T - N + 1) \cdot K$.

Commençons par trouver la dérivée de la sortie par rapport à l'entrée

$$\begin{aligned} \frac{\partial y_{ijk}}{\partial x_{qrs}} &= \sum_{mnp} w_{mnpk} \delta_{(i+m)q} \delta_{(j+n)r} \delta_{ps} \\ &= w_{(q-i)(r-j)sk} \end{aligned} \tag{2.11}$$

où δ_{ij} est le delta de Kronecker ; $\delta_{ij} = \begin{cases} 1 \text{ si } i = j \\ 0 \text{ si } i \neq j \end{cases}$. À l'aide de l'équation 2.11 et du principe de dérivée en chaîne, nous pouvons trouver :

$$\begin{aligned} \frac{\partial L}{\partial x_{qrs}} &= \sum_{ijk} \frac{\partial L}{\partial y_{ijk}} \frac{\partial y_{ijk}}{\partial x_{qrs}} \\ &= \sum_{ijk} \frac{\partial L}{\partial y_{ijk}} w_{(q-i)(r-j)sk} \end{aligned} \tag{2.12}$$

cette équation est similaire à l'équation 2.10. Cependant, le signe négatif dans les indices de w indique qu'il s'agit d'une convolution et non d'une corrélation. De plus, nous partons d'un signal de la taille de la sortie, vers un signal de la taille de l'entrée. Ceci indique qu'il s'agit d'une convolution complète.

Une procédure similaire nous permet de trouver le gradient :

$$\begin{aligned} \frac{\partial L}{\partial w_{qrst}} &= \sum_{ijk} \frac{\partial L}{\partial y_{ijk}} x_{(i+q)(j+r)s} \delta_{kt} \\ &= \sum_{ij} \frac{\partial L}{\partial y_{ijt}} x_{(i+q)(j+r)s} \end{aligned} \tag{2.13}$$

si nous considérons $\frac{\partial L}{\partial y_{ijt}}$ comme étant une liste d'images et x_{ijs} comme étant aussi une

liste d'images, alors, l'équation 2.13 correspond à une corrélation interne entre chaque paire d'images st. Les images de $\frac{\partial L}{\partial y_{ijt}}$ étant plus petites que celles de x_{ijs} , elles sont considérées comme les filtres.

2.5 Prétraitements et spectrogrammes

Dans cette section, nous décrivons quelques méthodes de prétraitement génériques, qui s'inspirent d'études psycho-acoustiques et qui peuvent être appliquées à tout signal sonore. Ces méthodes transforment le signal dans le domaine spectro-temporel où une réduction de dimensionnalité apporte généralement une perte d'information moins importante. Par exemple, le format de compression *mp3* utilise une transformation similaire, pour obtenir une réduction de l'ordre d'un facteur 10 avec des pertes presque inaudibles par l'être humain. De plus, les transformations que nous nous apprêtons à décrire, permettent de rapatrier localement les variables que nous voulons corréler. Telle que décrite dans la section 2.4, cette propriété est hautement désirée pour une architecture de RNC.

2.5.1 Transformée de Fourier locale

La transformée de Fourier locale (TFL), calcule une transformée de Fourier pour chaque moment d'un signal. Ceci nous permet d'obtenir une évolution du contenu spectral dans le temps (figure 2.8). Pour procéder ainsi, une fenêtre lisse est déplacée le long du signal afin d'en extraire une trame pour chaque instant. Finalement une transformée de Fourier est appliquée sur chacune de ces trames.

$$TFL\left(t,\omega\right)=\int_{-\infty}^{\infty}x\left(\tau\right)w^{*}\left(\tau-t\right)e^{-j\omega\tau}d\tau \tag{2.14}$$

où $w\left(t\right)$ est la fonction qui applique la fenêtre au moment t. $x(t)$ est le signal à transformer qui, dans notre cas, est le signal audio en format PCM.

La version discrète de la TFL est :

$$TFL\left[n,k\right]=\sum_{m=-\infty}^{\infty}x\left[n+m\right]w\left[m\right]e^{-jkm} \tag{2.15}$$

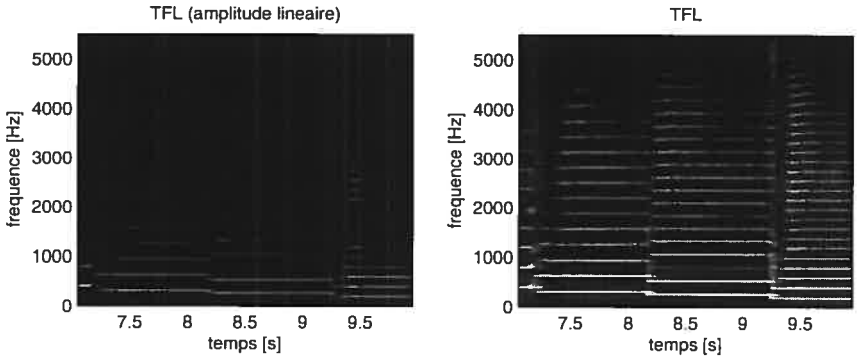

FIG. 2.8 – Le plan de magnitude de la TFL d'un enregistrement de saxophone. La largeur de la fenêtre spectrale est de 100 ms et le facteur de chevauchement est de 0.9. L'image de gauche représente une amplitude restée linéaire alors que l'image de droite représente une amplitude logarithmique pour faire ressortir les sons plus faibles.

Le choix du type de fenêtre à utiliser dépend du type d'application, la plus couramment utilisée est la fenêtre de Hamming ou de Hanning. Dans le cas où une fenêtre rectangulaire est utilisée, une série de hautes fréquences, de fortes magnitudes sont induites dans le signal, ce qui n'est généralement pas désiré.

La largeur de la fenêtre permet de faire un compromis entre la résolution spectrale et temporelle. Une fenêtre plus large permet d'obtenir une résolution spectrale accrue au dépend de la résolution temporelle.

2.5.2 Amplitude logarithmique

Le premier spectrogramme de la figure 2.8 contient la même information que le deuxième. Seulement, les harmoniques aiguës sont plus faibles et peu visibles. Dans le deuxième spectrogramme, le logarithme de l'amplitude est appliqué afin de faire ressortir les sons plus faibles. De plus, une étude psycho-acoustique existe, pour appuyer l'avantage d'extraire le logarithme de l'amplitude [Moo95, Kla99]. Afin d'éviter de trop amplifier le bruit et les artéfacts numériques, une constante additive est ajoutée à l'intérieur de la fonction logarithmique.

$$I = log(A + \epsilon)$$

Dans cet ouvrage, nous utilisons une valeur de ϵ de l'ordre de 1 % de la valeur maximale de A.

2.5.3 Échelle logarithmique

La TFL est probablement la méthode la plus répandue pour obtenir un spectrogramme à partir d'un signal sonore. Cependant, il est fréquent d'observer que beaucoup moins d'information se retrouve dans les hautes fréquences (figure 2.8). Cela peut être expliqué par le fait que les harmoniques sont plus espacées pour les notes aiguës. Par exemple, prenons une note ayant f comme fréquence fondamentale, les autres harmoniques seront de fréquence kf où $k = 1, 2, 3...$ Alors, l'espacement entre deux harmoniques consécutives sera de f, entraînant un plus grand espacement entre les harmoniques pour les notes aiguës. Si nous choisissons une échelle logarithmique pour les fréquences, nous avons une distance inter-harmoniques indépendante de la fréquence.

$$\begin{aligned} \Delta_{log} &= \log(kf) - \log((k-1)f) \\ &= \log(k) + \log(f) - \log(f) - \log(k-1) \\ &= \log(k) - \log(k-1) \end{aligned}$$

Cependant, il est intéressant de noter qu'à présent, l'espacement entre deux harmoniques consécutives dépend du numéro de l'harmonique. La figure 2.10 nous donne un exemple visuel d'un espacement logarithmique.

Une étude de l'oreille humaine nous suggère aussi d'adopter une échelle logarithmique pour notre spectrogramme. Par exemple, l'échelle mel [SVN37] est développée à la suite de tests psycho-acoustiques effectués sur des individus. L'étude porte à évaluer la distance en Hertz qu'il doit y avoir entre les notes, pour que le pas soit jugé constant par l'oreille humaine. Il s'avère que la distance dépend de la fréquence et que l'échelle est quasi-logarithmique, du moins pour les hautes fréquences. Nous pouvons obtenir une correspondance entre l'échelle mel et la valeur en Hertz à l'aide de l'équation suivante :

$$m = 1127.01048 \log_e \left(1 + \frac{f}{700}\right) \tag{2.16}$$

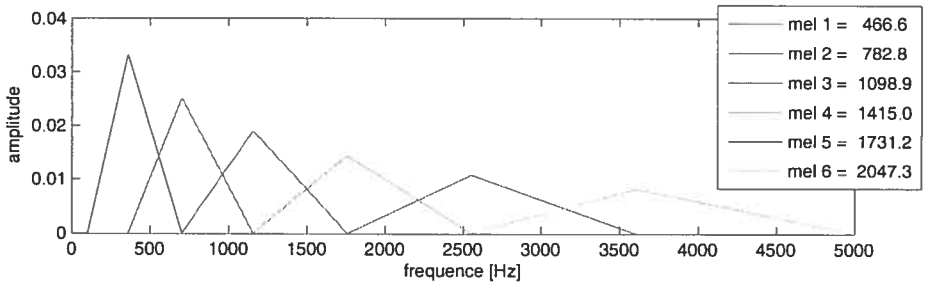

FIG. 2.9 – Une représentation graphique d'une banque de 6 filtres mel dans le domaine spectral. L'espacement entre les fréquence centrales est logarithmique et l'amplitude du filtre décroît linéairement à partir de la fréquence centrale n jusqu'à la fréquence centrale $n \pm 1$.

Cet espacement n'est pas exactement logarithmique. La constante additive à l'intérieur du logarithme provoque un espacement progressivement linéaire vers les basses fréquences. Le premier spectrogramme de la figure 2.10 en donne un aperçu.

2.5.4 Banque de filtres mel

Il est possible de convertir notre TFL vers une échelle logarithmique ou plutôt *quasi* logarithmique, en utilisant une banque de filtres mel. Alors que dans le domaine temporel, un filtre doit être convolué avec notre signal, dans le domaine spectral, un simple produit scalaire est suffisant. La figure 2.9 donne une représentation graphique d'une banque de 6 filtres mel. L'espacement entre les fréquence centrales est logarithmique et l'amplitude du filtre décroît linéairement à partir de la fréquence centrale n jusqu'à la fréquence centrale $n \pm 1$. Nous pouvons observer le résultat d'un tel filtre à la figure 2.10. Dans ce document, nour référencerons le spectrogramme à échelle mel et amplitude logarithmique par Log-mel-TFL.

2.5.5 Transformée à Q-constant (TQC)

La TQC [Bro91] constitue une méthode directe d'obtenir un spectrogramme à échelle logarithmique. Au même titre que la transformée de Fourier, la TQC possède un équivalent rapide [BP92]. Dans un tel spectrogramme, la résolution spectrale à haute fréquence est volontairement plus faible, il est donc possible de diminuer la largeur de la fenêtre, pour obtenir une meilleure résolution temporelle. Ceci constitue une flexibilité qui n'est pas disponible lorsque nous convertissons une TFL vers une échelle logarithmique à l'aide de filtres

mel. Cet avantage peut s'avérer utile pour certains problèmes tels que la détection d'évènements musicaux où une résolution temporelle élevée est un atout.

Le nom de cet algorithme provient du fait que le ratio entre la fréquence et la résolution spectrale à cette fréquence est une valeur constante.

$$Q = \frac{f_k}{f_{k+1} - f_k} = \left(2^{\frac{1}{b}} - 1\right)^{-1} \tag{2.17}$$

où b représente le nombre de fréquences par octave et k, l'index de la fréquence. Si nous choisissons $b = 12$ et une valeur appropriée pour f_0, nous retrouvons alors la gamme occidentale de 12 notes par octave. La deuxième image de la figure 2.10 expose un exemple de TQC.

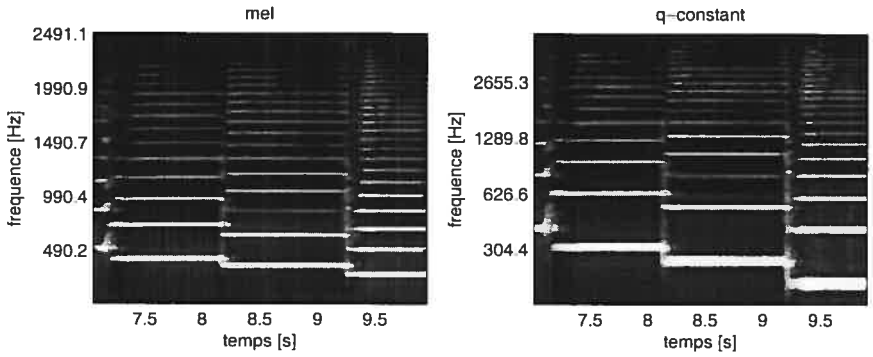

FIG. 2.10 – Les deux images représentent des spectrogrammes à échelle logarithmique du même échantillon sonore que dans la figure 2.8. Le premier étant effectué en convertissant une TFL à l'aide d'un filtre mel alors que le deuxième est effectué à l'aide de la TQC . La fenêtre spectrale est fixée à 100 ms et la résolution spectrale est de 48 pixels par octave.

Une analyse des deux spectrogrammes affichés à la figure 2.10 nous permet de faire une comparaison entre l'échelle mel et une échelle purement logarithmique. Nous pouvons voir que, dans le cas de la TQC, la résolution spectrale à basse fréquence est potentiellement trop élevée et nous obtenons une redondance qui peut être indésirable. L'échelle mel permet d'obtenir un espacement de plus en plus linéaire dans les basses fréquences, réduisant ainsi cette redondance. Cependant, la TQC possède une invariance de translation intéressante selon l'axe des fréquences. Malgré que les harmoniques en basses fréquences soient plus épaisses, l'ensemble des espacements pour une note ne dépend pas de sa tonalité. Cela permet de réutiliser des fonctions similaires à des fréquences différentes.

CHAPITRE 3

DÉTECTION AUTOMATIQUE D'ÉVÈNEMENTS MUSICAUX PAR RÉSEAUX DE NEURONES ARTIFICIELS

3.1 Introduction

Tel que discuté dans le chapitre 1, nous voudrions résoudre des problèmes de reconnaissance musicale en évitant d'effectuer une grande réduction de dimensionnalité manuelle. Étant donné qu'une chanson complète possède généralement une très grande quantité de variables, nous voudrions, dans un premier temps, travailler sur un problème relativement simple. La détection automatique d'évènements musicaux (DAEM) s'avère être un bon choix. Ce problème consiste à trouver le commencement de chaque note jouée dans une pièce musicale. L'information concernant un évènement se retrouve essentiellement localisée dans le temps, ceci permet de se concentrer sur un sous-ensemble de variables, plutôt que de traiter toute la chanson à la fois.

Malgré que la plus grande partie de l'information concernant un évènement se retrouve dans un ensemble restreint de variables, la plupart des travaux effectués dans la littérature de la DAEM effectuent une très grande quantité de réduction de dimensionnalité. Cette situation va nous permettre d'analyser s'il est possible d'obtenir un gain de performance en utilisant toutes les variables qui sont à notre disposition.

Ce chapitre présente les résultats d'un travail effectué pour le concours de détection d'évènements musicaux MIREX 2005. Ce concours a permis d'évaluer les résultats de 9 différents algorithmes sur un même ensemble de données, inconnu des participants. Le contenu de ce chapitre est une version adaptée de [LE06], un article publié à la suite de ce concours. Dans un premier temps, nous explorons les différents algorithmes disponibles dans la littérature. Par la suite, nous donnons une description détaillée de deux versions différentes de notre algorithme. Pour terminer, nous discutons des résultats obtenus selon les différents choix d'architecture.

3.2 Littérature sur la détection d'évènements

Les premiers algorithmes de détection d'évènements se concentraient principalement sur la variation de l'amplitude du signal dans le domaine du temps. [Sch98] a démontré qu'une grande partie de l'information contenue dans le signal peut être jetée, tout en conservant la majeure partie de l'information rythmique. Sur un ensemble de quelques chansons, Scheirer a séparé celles-ci en plusieurs bandes de fréquences distancées logarithmiquement. Par la suite, il n'a récupéré que l'enveloppe énergétique de chaque bande en appliquant un filtre passe bas sur le signal rectifié. À cette étape, l'ensemble des enveloppes contient l'information rythmique de la pièce musicale. Pour vérifier le résultat, chaque enveloppe est modulée séparément avec une source de bruit. Par la suite, elles sont additionnées ensemble pour former un nouveau signal musical (Figure 3.1). Finalement, le simple fait d'écouter les chansons est concluant pour vérifier que l'information rythmique est majoritairement présente, malgré que l'information spectrale soit principalement disparue.

Une deuxième expérience démontre qu'il faut être prudent lors de la combinaison. Les différentes enveloppes doivent être gardées séparées pendant la modulation avec le bruit. Dans le cas où elles sont combinées avant la modulation, une grande partie de l'information rythmique est perdue.

[Kla99] a utilisé le modèle psycho-acoustique de Scheirer pour développer un détecteur d'évènements robuste. Pour obtenir une meilleure résolution en fréquence, il a utilisé une banque de 21 filtres. De plus, l'auteur précise que le plus petit changement d'intensité que l'oreille humaine peut percevoir est proportionnel à l'intensité du signal [Moo95]. Alors, $\frac{\Delta I}{I}$ est constant, où I est l'intensité du signal. Par conséquent, pour faire ressortir les évènements, au lieu d'utiliser l'équation $\frac{d}{dt}A$ où A est l'amplitude de l'enveloppe, il utilise

$$\frac{1}{A}\left(\frac{d}{dt}A\right) = \frac{d}{dt}\log\left(A\right). \tag{3.1}$$

Ceci fait ressortir les pics d'évènements de manière beaucoup plus stable et permet aux pics plus faibles de ressortir plus facilement. De plus, la dérivée du logarithme de l'amplitude n'obtient pas son maximum au même endroit que la simple dérivée de l'amplitude. Une étude psycho-acoustique [Moo95] démontre que la crête obtenue à l'aide de 3.1 correspond

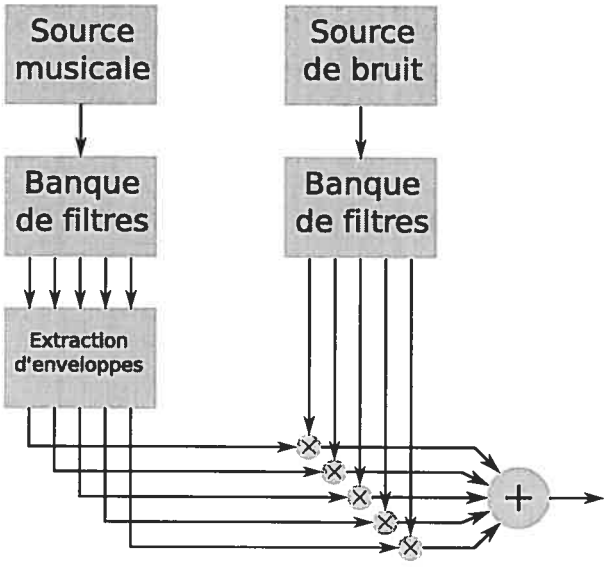

FIG. 3.1 – Moduler une source de bruit avec l'enveloppe énergétique de différentes bandes de fréquences conserve la majeure partie de l'information rythmique de la pièce musicale.

plus précisément à la position perçue par l'oreille humaine.

Quelques années plus tard, Klapuri utilise le même genre de prétraitement [KEA06] et gagne le concours d'induction de tempo de ISMIR 2004 [GKD+05].

3.2.1 Détection d'évènements à l'aide de la phase

Contrairement aux travaux de Scheirer et Klapuri, J. P. Bello et C. Duxbury [DBDS03b, DBDS03a, BS03, BDDS04] profitent de l'information contenue dans la phase du signal pour trouver les évènements. Ils ont trouvé qu'à l'équilibre, les oscillateurs ont une phase prévisible, ce qui n'est pas le cas au moment d'une percussion. Il suffit donc d'utiliser le manque de prédictibilité de la phase comme un indice de percussion. Tout d'abord, l'accélération de la phase est calculée :

$$\alpha_{k,n} = \text{princarg}\left[\varphi_{k,n} - 2\varphi_{k,(n-1)} + \varphi_{k,(n-2)}\right] \tag{3.2}$$

où $\varphi_{k,n}$est la phase du spectrogramme pour la fréquence k à l'instant n. L'opérateur "princarg" ramène l'angle dans le domaine $[-\pi, \pi]$. Finalement, pour mesurer le manque de prédictibilité, des statistiques sur l'accélération de la phase sont calculées le long de l'axe des

fréquences. Les statistiques utilisées sont la moyenne, la variance et le kurtosis. Cette mesure est calculée pour chaque instant du spectrogramme et produit une trace d'évènements, qui peut être analysée avec un simple algorithme d'extraction de pics. L'auteur a aussi combiné la phase et l'énergie dans le domaine complexe, pour une détection plus robuste. Des résultats sur des chansons monophoniques et polyphoniques démontrent un accroissement des performances pour la phase contre l'énergie et des résultats encore supérieurs en combinant les deux méthodes.

3.2.2 Détection d'évènements utilisant l'apprentissage statistique

Très peu de travail a été effectué sur la détection d'évènements musicaux à l'aide d'apprentissage statistique. Récemment [KP04] a utilisé une machine à vecteurs de support (SVM) sur des caractéristiques spectrales, pour détecter l'occurrence d'un évènement entre deux trames choisies. En utilisant cette méthode de manière hiérarchique, ils peuvent repérer le moment de l'évènement. Cette approche est particulièrement robuste pour les changements d'états très lents, comme ceux que l'on retrouve souvent dans les pièces musicales de violon ou de chant.

[DG02] a aussi développé un algorithme de segmentation utilisant les SVMs. Il classifie les trames comme faisant partie d'un évènement ou non. Le SVM a été utilisé pour trouver l'hypersurface délimitant la zone caractérisant les évènements du reste. Malheureusement, aucun test concluant n'est effectué pour démontrer la performance du modèle.

[MKP02] utilisent un réseau de neurones artificiels pour détecter la présence d'évènements. Leur approche s'apparente à la nôtre par leur utilisation de réseaux neuronaux, mais en tout autres points, elle en est différente. Leur algorithme utilise le modèle de Schierer [Sch98] avec 22 filtres comme prétraitement. Un réseau "integrate-and-fire" est utilisé séparément sur chaque enveloppe pour déclencher l'évènement et, finalement, un perceptron à plusieurs couches est utilisé pour valider la réponse du réseau précédent. Les résultats indiquent une performance élevée, mais le modèle a été testé uniquement sur des extraits de piano.

3.3 Description de l'algorithme

Dans cette section, nous décrivons deux variantes de notre algorithme. Les deux utilisent des réseaux de neurones artificiels pour classifier les trames comme étant un évènement ou non. La première variante, SINGLE-NET, suit le procédé décrit à la Figure 3.2. La deuxième variante, MULTI-NET, combine l'information de (A) plusieurs instanciations de SINGLE-NET, chacune entraînée avec différents hyper-paramètres et (B) une trace de tempo calculé à partir de la trace de détection d'évènements procurée par SINGLE-NET. Chacune de ces différentes sources d'information est ensuite combinée sous forme de matrice de taille similaire au spectrogramme original, qui est ensuite fournie en entrée à un autre réseau de neurones artificiels de même architecture que celui de SINGLE-NET. Voir Figure 3.3.

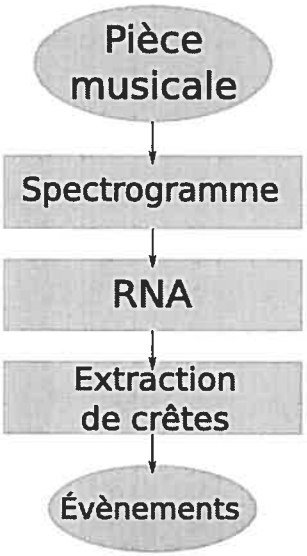

FIG. 3.2 – Diagramme d'exécution de SINGLE-NET. Cette variante de l'algorithme transforme d'abord le signal temporel vers le domaine spectral, en utilisant une des transformations décrites à la Section 3.3.1. Par la suite, le réseau de neurones artificiels est appliqué sur cette représentation, pour produire une trace d'évènements. Cette trace sera finalement analysée par l'algorithme d'extraction de pics, pour produire la liste des évènements.

3.3.1 Prétraitement

Tel que discuté dans la section 2.5, transformer notre signal dans le domaine spectro-temporel nous apporte plusieurs avantages. D'autant plus que, pour ce problème, le plan

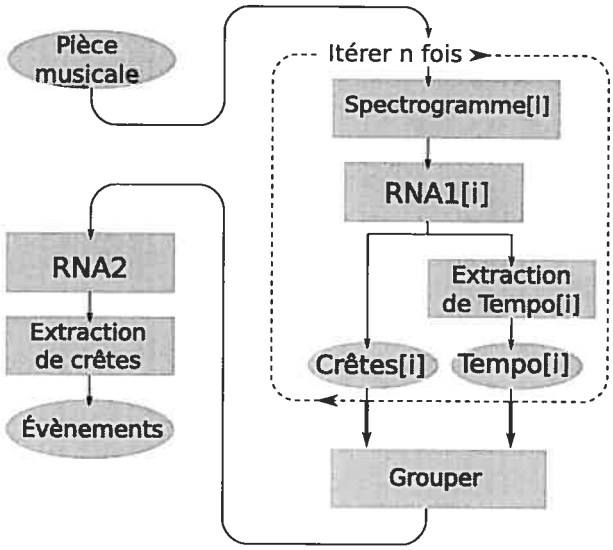

FIG. 3.3 – Diagramme d'exécution de MULTI-NET. La variante SINGLE-NET est répétée plusieurs fois avec différents hyper-paramètres. Par la suite, une trace de tempo est induite à partir de chaque sortie de SINGLE-NET. Finalement les n traces d'évènements et les n traces de tempo sont combinés en une matrice, pour être fournies en entrée à un algorithme similaire à SINGLE-NET.

de magnitude ne nous permet de trouver une grande partie des évènements qu'à l'aide d'un simple détecteur de pente positive. Pour ce travail, nous explorerons si le choix du type de spectrogramme peut avoir une influence sur le résultat. Pour ce faire, nous testerons la TFL et la TQC. Nous aimerions aussi voir s'il est possible de profiter de l'information contenue dans le plan de phase.

Dans la section suivante, nous explorons différentes manipulations effectuées sur le plan de phase, qui permettent de faire ressortir des indices concernant la position des évènements. Ceci nous permettra de combiner cette information avec la magnitude pour voir s'il est possible d'obtenir de meilleurs résultats.

3.3.1.1 Traitement du plan de phase

La TFL et la TQC sont tous les deux des transformées complexes. Elle possèdent donc chacune un plan de magnitude et un plan de phase. Nous pouvons déjà constater que le plan de magnitude possède de l'information pertinente (Figure 3.7). Il n'est pas tout aussi évident que le plan de phase contienne de l'information pertinente (Figure 3.4).

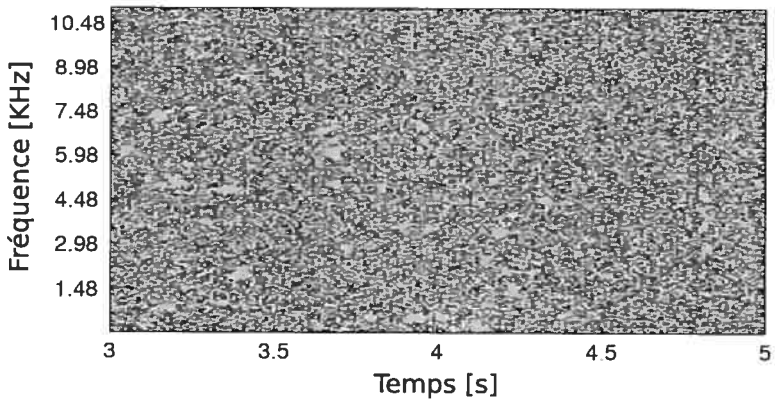

FIG. 3.4 – Le plan de phase de la TFL de la figure 3.7. Sans aucune manipulation, le plan de phase s'apparente beaucoup à une matrice de bruit.

Une manière potentiellement intéressante de traiter cette information est en utilisant l'équation 3.2. Les expériences de [BS03] démontrent que la distribution de probabilités des accélérations de phase sur l'axe des fréquences change beaucoup lors d'un évènement. Cependant, dans certain cas, les patterns d'évènements sont plutôt absents (Figure 3.5) et les résultats de la Table 3.1 démontrent que notre algorithme est presque incapable de les percevoir.

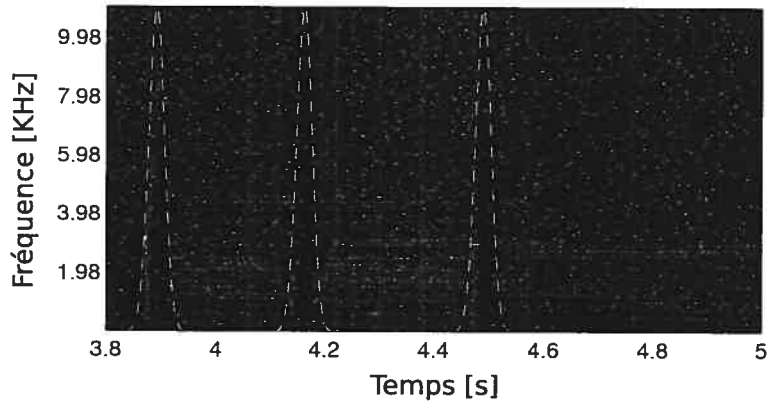

FIG. 3.5 – Plan de phase de la TFL de la figure 3.7, transformée selon l'équation 3.2. La courbe pointillée jaune représente la position des évènements. Selon cette représentation, les indices d'évènements sont très difficiles à percevoir.

Jusqu'à présent, rien n'indique que le plan de phase peut être utilisé d'une quelconque

manière. Cependant, si nous calculons la différence de phase le long de l'axe des fréquences, nous obtenons des résultats différents.

$$\varpi_{k,n} = \text{princarg}\left[\varphi_{k,n} - \varphi_{(k-1),n}\right] \tag{3.3}$$

où $\varpi_{k,n}$ représente la différence de phase entre la fréquence k et la fréquence $k-1$ au temps n, que nous nommerons déphasage spectral. Cette opération génère des indices prononcés qui corrèlent fortement avec la position des évènements (Figure 3.6). Cette approche semble plus prometteuse selon l'architecture de notre algorithme. Les résultats de la Table 3.1 démontrent que cette méthode est capable de performer presque aussi bien que sur le plan de magnitude.

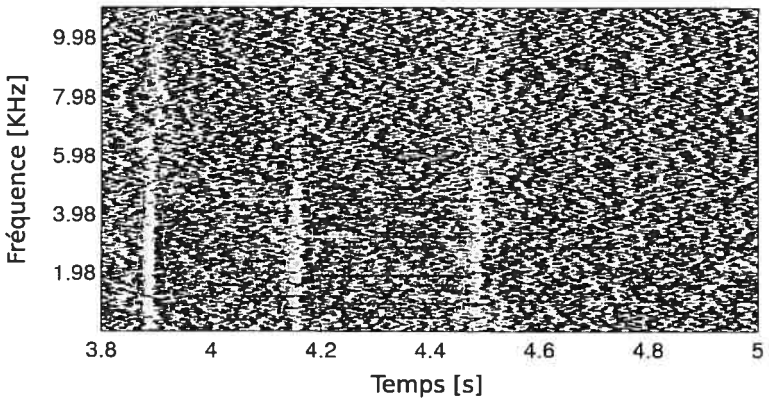

FIG. 3.6 – Le plan de phase de la TFL de la figure 3.7, transformé selon l'équation 3.3. La courbe pointillée jaune représente la position des évènements.

3.3.2 Apprentissage supervisé pour la mise en évidence des évènements.

Notre approche consiste à utiliser un réseau de neurones artificiels (RNF) pour déterminer la probabilité qu'un évènement se soit produit à un instant particulier. Pour ce faire, le RNF intègre l'information contenue à l'intérieur d'une courte période autour de l'instant désigné. Le but de cette étape est de produire le signal idéal pour l'algorithme d'extraction de pics.

Dans les sections qui suivent, nous décrirons comment le RNF est appliqué sur le spectrogramme, quelle architecture nous avons utilisée et comment nous l'avons entraînée.

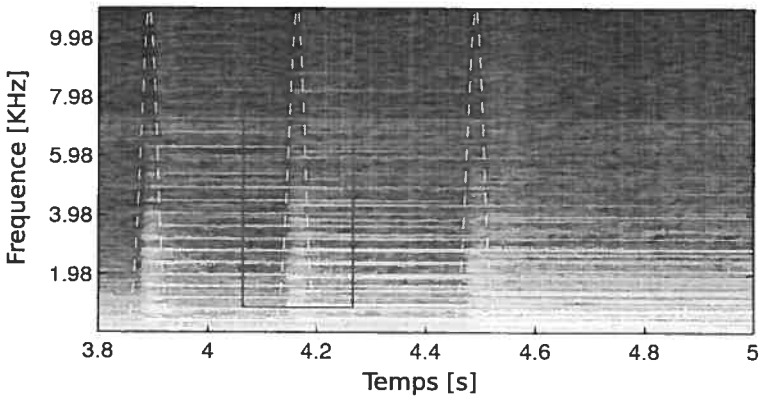

FIG. 3.7 – Le rectangle rouge représente la fenêtre perçue par l'algorithme pour un instant particulier et une fréquence particulière. Elle doit être suffisamment large pour pouvoir couvrir toute l'information pertinente. Dans le cas présent la largeur de la fenêtre est de 200 ms.

3.3.2.1 Les variables d'entrée

Tel qu'énoncé ci-dessus, l'information pertinente à la classification d'une trame se trouve dans un voisinage près de celle-ci. À cet effet, nous choisissons les variables d'entrée à l'intérieur d'une fenêtre centrée au moment spécifié et d'une largeur d'environ 200 ms (Figure 3.7).

Pour le cas du problème de détection d'évènements, nous pouvons considérer $x \in \mathbb{R}^n$ comme étant invariant de translation avec x correspondant à l'ensemble des n variables contenu à l'intérieur de la fenêtre. Autrement dit :

$$p(X = x|T = t) = p(X = x) \tag{3.4}$$

Ceci permet de réduire de beaucoup la complexité du problème, en ne se concentrant que sur un court instant à la fois. Malheureusement, l'axe des fréquences ne possède pas une telle invariance de translation. Si nous considérons notre fenêtre de variables d'entrée plus petite que la hauteur du spectrogramme (Figure 3.7), nous pouvons la déplacer dans l'axe des fréquences. Malheureusement, la distribution de probabilité sur x change en fonction de la fréquence.

$$p(X = x|F = f) \neq p(X = x) \tag{3.5}$$

où f est la fréquence centrale de la fenêtre d'entrées. Par exemple dans le cas de la TFL, une note ayant sa fondamentale plus élevée, aura ses harmoniques plus espacées que son équivalent à basse fréquence. Une TQC semble régler le problème, parce que l'espacement des harmoniques devient invariant de la note. Cependant, les harmoniques en basse fréquence sont beaucoup plus floues (Figure 2.10).

Malgré cette dépendance, un léger décalage en fréquence n'introduit qu'un petit changement sur la distribution de probabilité.

$$|f_1 - f_2| < \epsilon \Rightarrow p(x|f_1) \simeq p(x|f_2) \tag{3.6}$$

où ϵ est positif et suffisamment petit.

Pour éviter d'avoir à rajouter des valeurs dans la zone externe au spectrogramme, la fenêtre d'entrées ne peut être déplacée qu'aux endroits où elle figure complètement à l'intérieur du spectrogramme. Par conséquent, une hauteur de fenêtre de 100 % ne permet aucune translation. En réduisant la hauteur de la fenêtre à 90 %, nous pouvons alors effectuer une légère translation qui satisfait l'équation (équation 3.6). Par exemple, pour 200 fréquences, la hauteur de la fenêtre d'entrée serait de 180 fréquences et il y aurait 21 translations. Pour des raisons d'efficacité, nous n'avons choisi que 10 translations espacées uniformément. Il y a trois raisons principales à effectuer une translation selon l'axe des fréquences. Premièrement, la fenêtre d'entrées est plus petite, ce qui fait moins de paramètres à apprendre. De plus, chaque translation verticale procure une différente trace d'évènements, ce qui augmente la robustesse. Finalement, chaque translation verticale correspond à un ensemble d'entraînement supplémentaire, ce qui diversifie nos données.

Évidemment, la translation verticale ne procure qu'une très faible réduction de dimensionnalité et le nombre de variables d'entrée demeure encore très grand. Cependant, la représentation que nous avons choisi possède une certaine redondance et le plan de magnitude est suffisamment lisse pour qu'il soit peu coûteux de simplement ignorer des variables. Par conséquent, nous avons choisi le nombre de variables désiré aléatoirement à l'intérieur de la fenêtre. La sélection s'est faite selon une distribution uniforme pour l'axe des fréquences et selon une loi normale pour l'axe du temps. La Table 3.2 liste les résultats pour différentes quantités de variables. En général, 200 variables sont suffisantes mais des valeurs plus élevées

améliorent les résultats.

3.3.2.2 Structure du réseau de neurones artificiels

Le but premier de l'entraînement supervisé est de trouver une fonction qui mette en évidence les évènement dans une courbe simple, qui peut ensuite être analysée par un simple algorithme d'extraction de pics. Pour obtenir une telle fonction, nous avons utilisé un RNF à deux couches cachées et un seule neurone de sortie. Les couches cachées utilisent comme fonction d'activation tanh et la couche de sortie utilise la sigmoïde logistique. Nous avons aussi testé le système avec une seule couche cachée ainsi qu'aucune couche cachée. Les résultats pour différentes architectures sont énoncés dans la Table 3.3.

3.3.2.3 Courbe tutrice

Notre objectif étant de produire la courbe idéale pour l'algorithme d'extraction de pics, nous allons construire une courbe tutrice déterminant les moments auquel le RNC doit répondre. Il s'ensuit que notre tuteur soit une trace contenant des pics à la position des évènements préalablement identifiés. Pour augmenter le nombre de valeurs différentes de zéro, il nous faut attribuer une certaine largeur à ces évènements. Pour ce faire, nous avons choisi d'utiliser la fonction Gaussienne avec une variance de 10 ms. Notre tuteur correspond donc à une mixture de Gaussiennes, comme décrite par l'équation suivante :

$$T_s(t) = \sum_i \exp^{-\frac{(\tau_{s,i}-t)^2}{\sigma^2}} \tag{3.7}$$

où $\tau_{s,i}$ correspond à la position temporelle du i^{eme} évènement manuellement identifié de la pièce s.

Notre RNF prédit une valeur pour chaque instant dans chaque pièce de musique de notre ensemble de données. L'erreur totale est simplement la somme des erreurs au carré pour tous ces évènements.

$$E = \sum_{s,j} \left(T_s(t_j) - O_s(t_j)\right)^2 \tag{3.8}$$

où $O_s(t_j)$ est la sortie du réseau pour l'entrée j de la pièce s.

3.3.2.4 Optimisation

Pour entraîner notre réseau, nous avons utilisé la variante de Polak-Ribiere du gradient conjugué. Plus précisément, il s'agit de l'implémentation faisant parti du Neural Network Toolbox de Matlab.

Pour minimiser les chances de surapprentissage, nous avons utilisé la méthode d'arrêt prématuré. Selon cette technique, l'apprentissage se termine lorsque les performances diminuent sur un ensemble de données hors-entraînement réservé à cet effet [Bis95].

Pour obtenir des valeurs de performance plus représentatives, nous avons utilisé la technique de validation croisée sur notre ensemble d'entraînement. Pour plus de détails concernant la validation croisée, voir section 3.5 et concernant l'ensemble de données, voir section 3.4.

3.3.3 Extraction de pics

La dernière étape de notre algorithme consiste à déterminer quels pics correspondent à des évènements et en extraire leur position. Notre approche se divise en trois étapes. Dans un premier temps, les différentes traces d'évènements produites par la translation en fréquence sont fusionnées. Par la suite, les pics sont extraits de cette trace fusionnée. Finalement, une optimisation du paramètre de sélection des pics est effectuée.

3.3.3.1 Fusion des traces d'évènements

Tel qu'expliqué à la 3.3.2.1, la translation verticale de notre classifieur génère plusieurs traces d'évènements. Afin de regrouper les résultats, nous avons simplement extrait la moyenne, pour ne produire qu'une seule trace d'évènements. La figure 3.8 affiche un exemple du résultat.

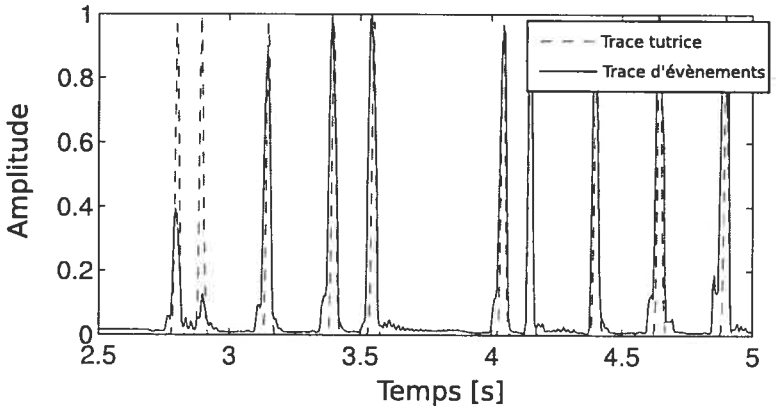

FIG. 3.8 – La courbe en pointillé représente la courbe tutrice. La courbe noire est un exemple de trace d'évènements produite par la fusion des sorties du réseau.

3.3.3.2 Extraction de pics

Comme les pics sont composés essentiellement de hautes fréquences, il est peu coûteux d'appliquer un filtre passe haut, pour éliminer les courbes de basses fréquences, qui peuvent distortionner la hauteur relative des pics. Pour ce faire, nous avons d'abord extrait l'arrière plan en appliquant un filtre Gaussien sans délai de 500 ms d'écart type avec notre signal. Par la suite, nous avons soustrait l'arrière-plan de la trace d'évènements pour ne récupérer que l'avant-plan. Finalement, nous soustrairons de l'avant-plan une valeur seuil, qui déterminera quels seront les pics acceptés (équation 3.9).

$$\rho_s(t) = O_s(t) - u_s(t) - K \tag{3.9}$$

où

$$u_s(t) = g * O_s(t) \tag{3.10}$$

où g est le filtre Gaussien, K est la valeur seuil et ρ_s est la trace de pics de la chanson s.

En utilisant cette approche, chaque croisement de pente positive avec l'abscisse correspond au début d'un évènement et chaque croisement de pente négative en indique la fin. La position centrale est estimée en calculant le centre de masse de l'ensemble des valeurs comprises dans cette intervalle.

$$\tau_{s,i} = \frac{\sum_{j \in p_i} t_j \rho_s(t_j)}{\sum_{j \in p_i} \rho_s(t_j)} \tag{3.11}$$

où $\tau_{s,i}$ est la valeur temporelle du i^{eme} évènement de la pièce musicale s et j est l'index qui parcoure toutes les valeurs contenues dans l'intervalle i.

3.3.3.3 Optimisation du seuil

Pour optimiser les performances, la valeur du seuil K dans l'équation 3.9 est apprise à partir de l'ensemble d'entraînement. Pour pouvoir faire une telle optimisation, il nous faut d'abord une méthode pour mesurer la performance globale de notre algorithme. À cet effet, nous avons utilisé la F-mesure[1], elle correspond à la moyenne harmonique entre le rappel et la précision.

$$\begin{aligned} P &= \frac{n_{cd}}{n_{cd} + n_{fp}} \\ R &= \frac{n_{cd}}{n_{cd} + n_{fn}} \\ F &= \frac{2PR}{P + R} \end{aligned} \tag{3.12}$$

où n_{cd} correspond au nombre d'évènements correctement détectés, n_{fn} correspond au nombre de faux négatifs et n_{fp} correspond au nombre de faux positifs. Une performance parfaite procure une F-mesure de 1 et pour un nombre fixe d'erreurs, la F-mesure est optimale quand le nombre de faux positifs est égal au nombre de faux négatifs.

Par cause de non continuité dans la fonction d'extraction de pics, nous ne pouvons pas utiliser de méthode de descente de gradient pour optimiser notre valeur seuil. Cependant, nous n'avons qu'une seule valeur à optimiser. Nous avons donc effectué une recherche exhaustive en estimant la F-mesure pour 25 valeurs de K dans l'intervalle $[0.02, 0.5]$ et choisi la valeur optimale.

[1] La F-mesure a aussi été utilisée pour le concours de détection d'évènements musicaux de MIREX 2005.

3.3.4 Variante MULTI-NET

Une exploration de différentes architectures et de différents spectrogrammes nous a permis d'observer qu'aucun ensemble d'hyper-paramètres n'était parfait. En fait, certaines configurations étaient complémentaires. Dans le but d'augmenter la robustesse de l'algorithme, nous avons combiné le résultat de plusieurs architectures différentes de SINGLE-NET.

De plus, pour chaque architecture, nous avons rajouté une estimation locale du tempo ainsi qu'une mesure de rythmicité. Cette trace de tempo est estimée à partir de la trace d'évènements fournis par chaque architecture. Pour combiner toutes ces informations, nous avons utilisé un RNF supplémentaire.

En considérant la F-mesure, cette complexité supplémentaire semble avantageuse. Par exemple, les résultats obtenus lors du concours de détection d'évènements musicaux de MIREX 2005 (décris brièvement à la Section 3.5.1), démontre une hausse de performance de 1.7 % par rapport à SINGLE-NET.

Les sections suivante décrivent comment obtenir la trace de tempo à partir de la trace d'évènements, ainsi que la méthode utilisée pour combiner les différentes sources d'information.

3.3.4.1 Extraction du tempo

La variante SINGLE-NET a accès uniquement à l'information contenue dans un voisinage près de l'évènement. Par conséquent, elle est incapable de découvrir les régularités qui apparaissent à plus grande échelle comme le tempo. Pour la variante SINGLE-NET, nous calculons une trace de tempo. Cette nouvelle information estime, pour un instant donné, si nous attendons un évènement ou pas, en fonction uniquement des évènements précédents et futurs

Nous obtenons cette trace de tempo Γ, en corrélant l'histogramme d'inter-évènements (HIE) d'un point particulier sur la trace d'évènements avec l'histogramme d'inter-évènements de toutes les paires d'évènements. Si les deux histogrammes sont corrélés, cela indique que cet instant est en phase avec le tempo.

$$\Gamma(t) = h\left(\{\mu_i - \mu_j\}_{ij}\right) \cdot h\left(\{\mu_i - t\}_i\right) \tag{3.13}$$

où $\Gamma(t)$ est la trace de tempo au temps t, $h(S)$ l'histogramme de l'ensemble de points S et μ_i est le i^{eme} évènement. Le produit scalaire entre les deux histogrammes correspond à la mesure de corrélation.

Cette méthode calcule n histogrammes, où chacun requiert un temps $O(n)$ à s'exécuter. Par conséquent, l'algorithme requiert un temps $O(n^2)$. De plus, chaque erreur produite lors de l'extraction des pics se répercute sur ce résultat. Pour améliorer la méthode, la Section 3.3.5.1 introduit une méthode pour calculer la trace de tempo directement à partir de la trace d'évènements, en calculant la corrélation croisée entre celle-ci et sa propre auto-corrélation. Cette méthode se calcule en un temps $O(n \log(n))$. La Figure 3.9 expose un exemple de trace de tempo.

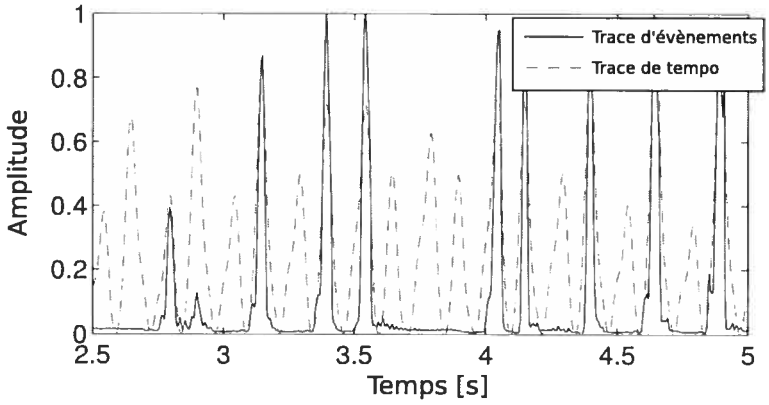

FIG. 3.9 – La trace d'évènements est le résultat de la fusion des différentes sorties du réseau de neurones, tel qu'exposé dans la figure 3.8. La trace de tempo démontre la corrélation croisée entre la trace d'évènements et sa propre auto-corrélation.

3.3.4.2 Rythmicité

La trace de tempo permet d'indiquer localement l'information sur le tempo global. Cependant, dans le cas où la trace d'évènements ne contient pas de répétition dans l'espacement de ses pics, la trace de tempo fournira de l'information impertinente. Il faut donc procurer en entrée un indice supplémentaire concernant la rythmicité globale de la trace de tempo. Pour ce faire, nous calculons l'entropie normalisée de l'histogramme d'inter-évènements.

$$H(T) = \frac{1}{\log_2 n} \sum_{i=1}^{n} p(t_i) \log_2 p(t_i) \tag{3.14}$$

Le facteur de normalisation permet de s'assurer que chaque mesure d'entropie se retrouve entre 0 et 1, indépendamment de la longueur du signal.

3.3.4.3 Groupement de l'information

Pour combiner l'information fournie au dernier réseau neuronal, nous avons simplement empilé toutes les traces fournies par les différents SINGLE-NET, ainsi que les traces de tempo et de rythmicité. Par exemple, les 10 translations en fréquences avec la trace de tempo et celle de rythmicité produisent 12 rangées par modèle. En utilisant 7 architectures différentes, nous obtenons une matrice de 84 rangées.

Cette information rapatriée produit une matrice avec un nombre de trames par seconde égale au spectrogramme original. Le reste de l'algorithme est identique à SINGLE-NET, excepté que le spectrogramme est remplacé par cette nouvelle matrice. Cependant, dans ce cas-ci, une translation verticale de la fenêtre d'entrée n'apporte aucune cohérence. C'est pourquoi la fenêtre couvre les 84 rangées.

3.3.5 Trace de tempo

Dans cette section, nous décrivons une méthode pour calculer la trace de tempo, s'exécutant en un temps $O(n \log n)$. Tout d'abord, nous montrons que la trace de tempo peut être obtenue en corrélant la trace d'évènements avec l'HIE. Par la suite, nous montrons comment obenir cet HIE à l'aide de l'autocorrélation de la trace d'évènements.

3.3.5.1 Trace de tempo par corrélation croisée

Soit :

$$h_a(t) = \gamma \star \gamma \tag{3.15}$$

où $h_a(t)$ est l'HIE pour un temps d'inter-évènement t, γ est la trace d'évènements et $\star$ est l'opérateur de corrélation croisée. En utilisant ceci, nous pouvons réécrire l'équation 3.13 :

$$\begin{aligned}\Gamma(t) &= \int h_a(t'')(\gamma \star \delta_t)\,dt'' \\ &= \int h_a(t'')\left(\int \gamma(t')\,\delta(t'-t+t'')\,dt'\right)dt'' \\ &= \int h_a(t'')\gamma(t+t'')\,dt'' \\ &= (\gamma \star \gamma) \star \gamma \end{aligned} \tag{3.16}$$

où $\Gamma(t)$ est la trace de tempo au temps t et $\delta_t \equiv \delta(\tau - t)$ où δ est le delta Dirac.

Par conséquent, la trace de tempo peut être calculée en corrélant la trace d'évènements 3 fois avec elle-même. Cette opération prends un temps $O(n \log n)$, ce qui est significativement plus rapide que $O(n^2)$, le temps requis par l'équation 3.13.

3.3.5.2 Histogramme d'inter-évènements par auto-corrélation

Tel qu'énoncé ci-haut, nous allons maintenant démontrer que l'HIE d'une courbe composée de pics est équivalent à l'auto-corrélation de cette courbe. Pour procéder ainsi, nous démontrons que l'auto-corrélation d'une somme de fonctions est la somme de toutes les paires de corrélation croisée possibles entre ces fonctions.

$$\begin{aligned} f(t) &\equiv \sum_i g_i(t) \\ f(t) \star f(t) &= \mathfrak{F}\left[|F(k)|^2\right] \\ &= \mathfrak{F}\left[\sum_{ij} \overline{G_i(k)} G_j(k)\right] \\ &= \sum_{ij} g_i(t) \star g_j(t) \end{aligned} \tag{3.17}$$

où $F(k)$ et $G_i(k)$ sont respectivement les résultats des transformées de Fourier de $f(t)$ et $g_i(t)$. $\mathfrak{F}$ est l'opérateur de transformée de Fourier.

Un résultat connu nous indique que la corrélation croisée de deux Gaussiennes est une autre Gaussienne avec une nouvelle moyenne correspondant à $\mu_1 - \mu_2$ et une nouvelle variance

correspondant à $\sigma_1^2 + \sigma_2^2$.

$$N(t; \mu_1, \sigma_1) \star N(t; \mu_2, \sigma_2) = N\left(t; (\mu_1 - \mu_2), \sqrt{\sigma_1^2 + \sigma_2^2}\right) \tag{3.18}$$

où

$$N(t; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(t-\mu)^2}{\sigma^2}} \tag{3.19}$$

Si nous approximons notre trace d'évènements par une mixture de Gaussiennes,

$$\gamma(t) = \sum_i \alpha_i N(t; \mu_i, \sigma_i) \tag{3.20}$$

alors, à l'aide de l'équation 3.17 et de l'équation 3.20, nous pouvons réécrire l'auto-corrélation de la trace d'évènements comme suit :

$$\gamma(t) \star \gamma(t) = \sum_{ij} (\alpha_i N(t; \mu_i, \sigma_i)) \star (\alpha_j N(t; \mu_j, \sigma_j)) \tag{3.21}$$

et en utilisant l'équation 3.18, l'équation 3.21 devient

$$\sum_{ij} \alpha_i \alpha_j N\left(t; (\mu_i - \mu_j), \sqrt{\sigma_i^2 + \sigma_j^2}\right) \tag{3.22}$$

qui est un cas plus générique d'un histogramme à fenêtre de Parzen. Le cas traditionnel est lorsque α_i et σ_i demeurent constants pour tous les points. Cette perte d'information survient lorsque nous extrayons les pics de la trace d'évènements, gardant uniquement la position et ignorant la largeur et la hauteur des pics.

3.4 Ensemble de données

Pour apprendre cette tâche correctement, nous avons besoin d'un ensemble de données annoté avec précision, couvrant une grande variété de styles musicaux. Une précision sur l'annotation est particulièrement importante pour cette tâche. Dans le cas où l'erreur sur la position est plus grande que l'écart type utilisé pour générer la courbe tutrice, l'algorithme sera fortement puni pour avoir prédit l'évènement à la bonne position et sera également puni

de ne pas avoir prédit là où il n'y avait pas d'évènement.

Nous avons cherché des ensembles de données disponibles gratuitement sur Internet. Le jeu de données le plus intéressant que nous ayons trouvé constitue celui de Pierre Leveau [PLR04]. Cependant, il n'est constitué que d'un petit nombre de pièces monophoniques. Cela le rend donc inapproprié pour notre tâche.

Nous avons donc décidé d'annoter notre propre ensemble de chansons. Pour avoir la possibilité partager ce travail, nous avons choisi d'annoter l'ensemble de pièces "Ballroom" utilisé lors de MIREX 2004. Ces pièces sont déjà annotées pour les tempos, mais ne contiennent pas les évènements individuels. Ce jeu de données est composé de 698 fichiers d'environ 30 secondes chacun, enregistré dans un format de modulation d'impulsion codée. Un annotation complète de ces données serait trop exigeant et n'est pas nécessaire. Nous avons donc annoté 59 segments de 10 secondes. La plupart de ces extraits sont des compositions polyphoniques complexes avec du chant.

L'annotation s'est effectuée manuellement à l'aide d'un outil graphique développé en Matlab. L'avantage de cet outil est de procurer des indices visuels et auditifs sur la position des évènements, pour pouvoir annoter avec précision. L'annotateur peut alors réécouter au ralenti un segment pour s'assurer de la présence d'un évènement.

3.5 Résultats

Pour mesurer la performance sur différentes méthodes et différents hyperparamètres, nous avons utilisé la validation croisée. Pour ce faire, 15 chansons sur 69 sont utilisées pour le test et le reste, utilisées pour l'entraînement. 3 différentes séparations de notre ensemble de données ont été effectuées pour la validation croisée, nous fournissant une performance moyenne ainsi qu'une variance.

Nos spectrogrammes contiennent de l'ordre de 200 trames par seconde et chaque pièce musicale dure 10 secondes. En tenant compte des 10 translations en fréquences que nous effectuons, nous obtenons un ensemble de données de l'ordre de 1000000 images. Pour ce travail, nous avons décidé d'utiliser le gradient conjugué, qui ne fonctionne qu'en lot, c'est-à-dire qu'il doit itérer sur tout l'ensemble de données avant de performer un changement de paramètres. Une telle quantité d'échantillons provoque un apprentissage plutôt lent. Pour

accélérer le processus, nous avons choisi de n'utiliser que 5 % de ces données.

Les résultats qui suivent présentent les performances effectuées sur différents hyperparamètres. Les valeurs qui ne sont pas affichées explicitement prennent les valeurs par défaut suivantes : la taille de la fenêtre d'entrée est 150 ms, le nombre de trames par seconde est 200 Hz, le nombre de variables d'entrées est de 150, le taille de la première couche cachée est de 18 neurones, la deuxième est de 15 neurones et la longueur de la fenêtre de Hamming est de 30 ms.

Le premier test effectué est pour déterminer quel plan est le plus approprié pour détecter les évènements. Nous avons testé la TFL, la TQC, l'accélération de la phase et la différence de phase selon l'axe spectral. Pour chacun de ces tests, nous avons aussi évalué les performances pour différentes largeur de fenêtre spectrale (Table 3.1). La meilleure performance est atteinte à l'aide de la TQC, mais la différence avec la TFL n'est pas très significative. Les résultats montrent que la largeur de la fenêtre spectrale ne semble pas avoir un impact important, pour autant qu'elle soit suffisamment petite pour produire une résolution temporelle adéquate. La plan d'accélération de phases n'a performé que légèrement mieux qu'un plan de bruit, démontrant que cette représentation n'est pas appropriée. Cependant, le plan de différence spectrale de la phase réussit à performer presque aussi bien que les plans de magnitude, mais cette fois-ci, avec une grande dépendance sur la largeur de la fenêtre spectrale. Cette dépendance laisse croire qu'un traitement plus approprié du plan de phase pourrait apporter des résultats plus stables et potentiellement plus performants.

Le prochain test évalue l'effet de la largeur de la fenêtre d'entrée ainsi que le nombre de variables extraites de cette fenêtre. Ce test est effectué sur le plan de magnitude de la TFL. La table 3.2 démontre que la largeur de la fenêtre d'entrée n'a pas d'impact significatif tant qu'elle reste suffisamment grande. Cependant, un nombre de variables élevé permet d'obtenir de meilleures performances, avec une saturation autour de 400 variables.

Dans la table 3.3, nous affichons les résultats pour différentes architectures de RNF. Lorsque deux couches cachées sont utilisées, les performances sont légèrement plus élevées que pour une seule couche. Aussi, nous pouvons remarquer, qu'un petit nombre de neurones est suffisant pour obtenir de bonnes performances. Ce nombre peut être aussi petit qu'un seul neurone et atteindre des performances acceptables.

Plan	Fenêtre Spectrale	F-mesure Entraînement	F-mesure Test
TFL mag	10 ms	86 ± 2	86 ± 5
TFL mag	30 ms	86 ± 1	86 ± 5
TFL mag	100 ms	84 ± 2	83 ± 8
TQC mag	10 ms	86 ± 2	86 ± 5
TQC mag	30 ms	**87 ± 2**	**87 ± 5**
TQC mag	100 ms	84 ± 2	84 ± 6
TFL accél. ph	10 ms	49 ± 2	49 ± 4
TFL accél. ph	30 ms	47 ± 1	47 ± 5
TFL accél. ph	100 ms	49 ± 4	47 ± 6
TFL diff. spectrale ph	10 ms	62 ± 2	61 ± 6
TFL diff. spectrale ph	30 ms	80 ± 1	79 ± 4
TFL diff. spectrale ph	100 ms	74 ± 2	73 ± 6
Bruit	—	40 ± 2	40 ± 6

TAB. 3.1 – Résultats obtenus après avoir entraîné notre algorithme sur différentes représentations de la chanson. La TQC obtient le meilleur résultat, mais la faible différence de performance avec la TFL ne permet pas de tirer conclusion. L'accélération de la phase n'obtient que légèrement mieux qu'un plan de bruit alors que la différence spectrale de la phase obtient des résultats presque aussi valables que les plans de magnitude. L'écart type sur la F-mesure est calculée à partir de la validation croisée.

Largeur de la fenêtre d'entrée	Nombre de variables d'entrée	F-mesure Entraînement	F-mesure Test
450 ms	200	86 ± 2	86 ± 6
300 ms	200	86 ± 2	86 ± 6
150 ms	200	86 ± 2	86 ± 5
75 ms	200	85 ± 2	84 ± 5
300 ms	100	84 ± 2	84 ± 6
300 ms	200	86 ± 2	86 ± 6
300 ms	400	87 ± 2	87 ± 5
300 ms	800	87 ± 2	87 ± 6

TAB. 3.2 – Résultats obtenus pour différentes largeurs de fenêtres et différents nombres de variables. La largeur de la fenêtre d'entrée n'a pas d'impact significatif tant qu'elle reste suffisamment grande. Cependant, un nombre de variables élevé permet d'obtenir de meilleures performances, avec une saturation autour de 400 variables.

1^{re} couche	2^{e} couche	F-mesure Entraînement	F-mesure Test
50	30	87 ± 2	87 ± 5
20	15	87 ± 1	87 ± 4
10	5	87 ± 2	87 ± 5
10	0	86 ± 2	86 ± 4
5	0	86 ± 2	85 ± 3
2	0	85 ± 2	85 ± 5
1	0	83 ± 2	83 ± 4

TAB. 3.3 – Résultats obtenus pour différentes architectures de RNF. L'absence de neurone sur la deuxième couche représente un RNF à une seule couche cachée.

Nb variables d'entrée	Taille de la fenêtre de Hamming	F-mesure Entraînement	F-mesure Test
100	30 ms	85 ± 2	84 ± 5
100	50 ms	85 ± 1	84 ± 7
100	100 ms	80 ± 2	79 ± 8
200	30 ms	86 ± 2	86 ± 5
200	50 ms	86 ± 2	85 ± 6
200	100 ms	84 ± 2	84 ± 7

TAB. 3.4 – Résultats obtenus pour les tests combinant le plan de magnitude de la TFL avec son plan de phase transformé selon l'équation 3.3. Malheureusement, cette combinaison ne semble pas apporter de gain de performance.

Les résultats de la table 3.1 nous laisse croire qu'en combinant le plan de magnitude et le plan de phase, nous pourrions obtenir des meilleurs résultats. Nous avons effectué le test pour différents nombres de variables d'entrées et différentes largeurs de fenêtre spectrale où le nombre de variables correspond au nombre extrait dans chacun des deux plans utilisés. Pour obtenir une représentation adéquate du plan de phase, nous l'avons transformé selon l'équation 3.3. Malheureusement, cette combinaison ne semble pas apporter de gain de performance (table 3.4).

3.5.1 Résultats de MIREX 2005

Nous avons soumis les deux variantes de notre algorithme au concours MIREX 2005. L'ensemble des données à évaluer pour ce concours est composé de 30 extraits de tambours en solo, 30 extraits à tonalité monophonique en solo, 10 extraits à tonalité monotimbrale et

Type d'algorithme	MULTI-NET	SINGLE-NET
F-mesure moyenne	80.07 %	78.35 %
Précision moyenne	79.27 %	77.69 %
Rappel moyen	83.70 %	83.27 %
Nombre total bien identifié	7974	7884
Nombre de faux positifs	1776	2317
Nombre de faux négatifs	1525	1615
Nombre d'évènements fusionnés	210	202
Nombre d'évènements dédoublés	53	60
Temps d'exécution (s)	4713	1022

TAB. 3.5 – Résultats obtenus lors du concours de détection automatique d'évènements de MIREX 2005. Les deux première places sont obtenues par les deux variantes de notre algorithme décrites dans ce chapitre.

15 extraits polyphoniques complexes. Sur cet ensemble de données, la variante MULTI-NET a obtenu une performance légèrement supérieure à celle de la variante SINGLE-NET. MULTI-NET a obtenu 80.07 % alors que SINGLE-NET a obtenu 78.35 %. Ces résultats ont été classés comme étant respectivement la meilleure et la deuxième meilleure performance du concours (table 3.6).

Les deux variantes de notre algorithme ont été développées pour bien performer sur une grande variété de données, principalement sur le type polyphonique complexe. Néanmoins, les résultats de la table 3.7 indiquent que certains participants ont obtenu des résultats significativement plus élevés dans certaines catégories. Notre algorithme a cependant montré la meilleure balance entre la précision et le rappel. Cette caractéristique est probablement due au seuil appris pour la partie d'extraction de pics.

3.6 Discussion

Une analyse plus approfondie des erreurs produites sur notre ensemble de données nous montre qu'une grande quantité de faux négatifs sont produits lors de sons aigus possédant des harmoniques minces. Nous croyons que la raison principale est due au faible pourcentage de variables actives lorsque les sons sont très harmonieux. Bien que l'amplitude de ces sons soit élevée, le logarithme de l'amplitude atténue significativement cet indice. Cette faille expliquerait les faibles résultats obtenus lors du concours MIREX 2005 pour la catégorie

percussions métalliques. Notre entrée a obtenu une performance de 86.55 % alors que les meilleurs résultats pour cette catégorie sont de 99.28 %. Concernant les faux positifs, ils sont majoritairement générés par le vibrato du chant, mais notre algorithme reste tout de même robuste dans cette catégorie.

Il y a aussi des effets de bords. Au début et à la fin de chaque séquence, le réseau a fréquemment eu des problèmes à bien percevoir les évènements. Une solution au problème serait d'entraîner trois réseaux différents. Un premier, utilisant uniquement l'information provenant du futur pour pouvoir trouver les évènements en début de segments. Un deuxième, utilisant uniquement l'information provenant du passé pour trouver les évènements en fin de segments. Finalement, un troisième, qui s'occupe des autres évènements. De plus, celui qui utilise l'information provenant uniquement du passé pourrait être utilisé pour la détection en temps réel.

3.6.1 Travaux futurs

Malgré que nos résultats soient relativement bons, il reste beaucoup de place à l'amélioration. Travailler sur un algorithme qui trouve les évènements harmonieux améliorerait de beaucoup les performances. L'utilisation de boosting [DSS93] permettrait de combiner cet algorithme avec d'autres, pour obtenir une meilleure méthode de combinaison d'algorithmes.

L'utilisation du RNC pourrait être très appropriée pour ce problème. En utilisant cette méthode, la taille des filtres de la première couche pourrait être grandement réduite, tout en ayant la possibilité d'intégrer sur une grande plage de temps.

Dans ce chapitre, nous avons appliqué une translation spectrale de notre RNF selon l'affirmation que la tâche possédait une invariance de translation pour de petits déplacements. Cependant, aucun test concluant n'a été effectué pour vérifier si cette technique apporte un avantage quelconque. Il serait donc intéressant de vérifier les résultats pour différentes valeurs de translations.

3.7 Conclusion

Nous avons présenté un algorithme qui profite de l'apprentissage supervisé, pour faire ressortir les évènements d'un signal sonore complexe. De plus, nous avons montré que le choix

du seuil pour l'extraction des pics peut aussi être appris sur l'ensemble d'entraînement. Notre variante SINGLE-NET utilise un simple RNF pour faire ressortir les évènements du signal et simplifier la tâche de l'extracteur de pics. La variante MULTI-NET combine les réponses provenant de plusieurs différents SINGLE-NET ainsi qu'une trace de tempo. Malgré que la variante MULTI-NET apporte des résultats légèrement meilleurs, nous croyons que la variante SINGLE-NET mérite plus d'attention due à sa simplicité. Nous apportons aussi des évidences que notre algorithme fonctionne bien en le comparant aux autres algorithmes de dernière génération. Nous concluons alors que l'utilisation d'apprentissage supervisé constitue une approche appropriée à la tâche de détection d'évènements sonores.

3.8 Appendice

3.8.1 Résumé des résultats obtenus lors du concours de détection d'évènements musicaux MIREX 2005

Le but de ce concours était d'évaluer et de comparer différents algorithmes de détection d'évènements musicaux. L'ensemble de données utilisé pour évaluer ces algorithmes contenait 85 extraits musicaux composés de 9 différentes classes : complexes, polyphoniques, percussions métalliques, cuivres, tambours, cordes pincées, chants, cordes soutenues et instruments à vent. Cette information est résumée à partir de `http://www.music-ir.org/evaluation/mirex-results/audio-onset/index.html`.

	Participants	**F-mesure**	**Précision**	**Rappel**
1	Lacoste & Eck (MULTI-NET)	80.07 %	79.27 %	83.70 %
2	Lacoste & Eck (SINGLE-NET)	78.35 %	77.69 %	83.27 %
3	Ricard, J.	74.80 %	81.36 %	73.70 %
4	Brossier, P.	74.72 %	74.07 %	81.95 %
5	Röbel, A. (2)	74.64 %	83.93 %	71.00 %
6	Collins, N.	72.10 %	87.96 %	68.26 %
7	Röbel, A. (1)	69.57 %	79.16 %	68.60 %
8	Pertusa, Klapuri, & Iñesta	58.92 %	60.01 %	61.62 %
9	West, K.	48.77 %	48.50 %	56.29 %

TAB. 3.6 – Résultats cumulatifs du concours de détection d'évènements de MIREX 2005. Le cumulatif est effectué en utilisant une moyenne pondérée des 9 catégories selon leur nombre de fichiers.

	Complexes (15)	Polyphoniques (10)	Percussions métalliques (4)	Cuivres (2)	Tambours (30)	Cordes pincées (9)	Chants (5)	Cordes soutenues (6)	Vents (4)
MULTI-NET	**78.85**	**86.31**	86.55	70.25	91.40	81.84	**45.33**	56.68	58.75
SINGLE-NET	77.02	85.93	86.37	67.88	89.91	83.49	34.35	52.87	56.48
Ricard, J.	71.90	83.26	87.17	**72.66**	90.97	77.85	27.59	38.45	38.57
Brossier, P.	76.16	80.88	73.97	64.88	86.28	79.99	22.16	**57.92**	52.08
Röbel, A. (2)	62.84	76.24	90.34	68.32	89.96	**84.20**	40.68	36.18	**66.01**
Collins, N.	60.25	75.70	**99.28**	69.09	**92.31**	81.97	29.34	14.74	47.57
Röbel, A. (1)	59.76	69.29	97.92	61.87	86.29	77.58	42.69	17.35	51.13
Pertusa et al.	50.16	59.37	60.22	54.41	77.22	67.74	11.12	38.45	25.59
West, K.	47.13	39.98	34.58	33.94	71.61	39.85	12.07	32.12	18.11

TAB. 3.7 – F-mesure en pourcentage pour chacune des 9 classes d'évènements pour le concours MIREX 2005. La meilleure performance obtenue pour chaque classe est en caractère gras et le nombre d'extraits pour chaque classe est entre parenthèses.

CHAPITRE 4

RÉSEAU DE NEURONES À CONVOLUTION POUR LA DÉTECTION D'ÉVÈNEMENTS.

4.1 Introduction

Dans le chapitre précédent, nous avons exploré la possibilité de convoluer un RNF sur le spectrogramme. Cette méthode s'est avérée très performante par rapport aux méthodes généralement utilisées dans la littérature du traitement de signal.

Cependant, le nombre de synapses nécessaires pour intégrer sur une période de temps suffisamment grande est particulièrement élevé. Cette caractéristique apporte deux principaux désavantages. Premièrement, la vitesse de l'algorithme en est affectée tant au niveau de l'exécution qu'au niveau de l'apprentissage. De plus, tel qu'expliqué à la section 2.3.1, un nombre élevé de paramètres apporte fréquemment des problèmes de généralisation.

Dans ce chapitre, nous explorons la possibilité de pallier à ces deux problèmes par l'utilisation du RNC. Tel qu'expliqué à la section 2.4, le RNC permet d'effectuer progressivement une réduction de dimensionalité. Cela permet de réduire significativement la taille des filtres de la première couche, pour laisser aux autres couches la tâche d'intégrer sur une plus grande période de temps.

Nous voulons principalement évaluer si le RNC apporte un avantage particulier par rapport au RNF. Cependant, plusieurs autres questions peuvent être adressées en testant l'algorithme sur différents hyperparamètres. Premièrement, nous aimerions savoir si le prétraitement effectué est avantageux ou si le RNC peut travailler directement sur le spectrogramme. Un autre test permet d'évaluer s'il est possible de profiter des invariances de translations contenues dans l'axe des fréquences. Finalement, nous voulons évaluer la perte de performance lorsque l'algorithme est utilisé dans un environnement temps réel, c'est-à-dire lorsque uniquement l'information provenant du passé est accessible.

4.2 Détails de l'algorithme

L'application du RNC pour la détection d'évènements est plutôt directe. Il suffit de considérer le spectrogramme comme une image et d'appliquer l'algorithme tel que décris à la section 2.4. La sortie imposée correspond à la courbe tutrice, construite de la même manière que présenté à la section 3.3.2.3.

Dans cette section, nous décrivons quel type de spectrogramme nous avons utilisé, quelques détails sur l'architecture du RNC ainsi que la méthode utilisée pour que notre algorithme puisse fonctionner en temps réel. Nous décrivons aussi quel algorithme d'optimisation nous avons utilisé pour apprendre les paramètres du RNC.

4.2.1 Spectrogramme

Le spectrogramme le plus approprié pour cet algorithme s'avère être la TFL réduite à l'échelle mel (Section 2.5.4). Le nombre de fréquences extraites se situe généralement entre 8 et 60 et la densité est fixée à 200 trames par seconde. La largeur de la fenêtre spectrale est aussi fixée à 36 ms et la fonction logarithme est appliquée à la magnitude du spectrogramme.

Malgré que la TFL sans réduction à l'échelle mel fonctionne très bien, la grande quantité de fréquences contenues dans ce spectrogramme affecte légèrement la vitesse d'exécution ainsi que la performance des résultats.

4.2.2 Architecture du RNC

Pour nos tests, nous avons limité la profondeur à 2 couches cachées et une couche de sortie. Cependant, les résultats démontrent que la deuxième couche cachée n'apporte que très peu d'avantages.

Malgré que la flexibilité du RNC nous permette d'effectuer une translation de nos filtres selon l'axe des fréquences, nous utilisons rarement cette option. Par le fait même, nous fixons la taille verticale des filtres de la première couche au nombre de fréquences de notre spectrogramme. Cela impose une taille verticale de 1 pour les filtres des autres couches.

La largeur temporelle de nos filtres est généralement fixée à 4 ms pour la première couche et 20 ms pour la dernière couche. Le sous-échantillonnage n'est généralement pas utilisé,

puisque nous voulons garder une haute résolution temporelle et nous n'effectuons pas de translation spectrale.

4.2.3 Délai de la courbe tutrice

Nous avons aussi ajouté un autre paramètre, qui nous permet d'appliquer un délai à la courbe tutrice. Cette flexibilité, nous permet de déterminer le moment auquel le RNC doit répondre.

La figure 4.1 montre les différents cas de délais possibles. Pour connaître les conséquences du délai, il nous faut connaître la taille totale du RNC, c'est-à-dire, le nombre minimal de trames nécessaires pour produire un point dans la courbe de sortie. Cette taille se calcule comme suit :

$$\left(\sum_{i=1}^{N} n_i\right) - (N-1)$$

où n_i correspond à la largeur du filtre de la couche i et N, le nombre total de couches.

Selon notre architecture, un délai de 0 permet au RNC d'utiliser l'information provenant du passé et du futur de l'évènement. Plus précisément, pour des filtres de taille 8, 20 et 40 et une résolution temporelle de 200 trames par seconde, le RNC perçoit 165 ms dans le futur et 165 ms dans le passé. Dans le cas où le délai est de 165 ms, le RNC perçoit 330 ms dans le passé. Cela permet à l'algorithme de fonctionner en temps réel, sans avoir à attendre après l'information provenant du futur. Inversement, si le délai est de -165 ms, le RNC perçoit uniquement l'information provenant du futur. Cette dernière configuration peut être profitable dans le cas où nous voudrions découvrir des évènements en début de signal.

La majeure partie de nos test sont effectués avec un délai de 0, mais un résultat à la section 4.3 démontre que les performances sont équivalentes pour un délai plus petit que la moitié de la taille du RNC.

4.2.4 Optimisation

Pour apprendre la valeur de nos paramètres, nous avons utilisé le gradient conjugué (Section 2.3.2). Ce choix est justifié par la petite taille de notre ensemble de donnée, soit 60

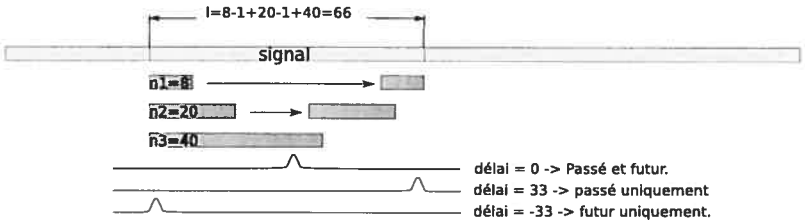

FIG. 4.1 – Délai de la courbe tutrice. Dans le cas présent, la largeur du RNC est de 66 trames. Un délai de 33 trames sur la courbe tutrice restreint le RNC à ne percevoir que le passé et le présent de l'évènement.

chansons de 10 secondes, comme présenté à la section 3.4.

Nous avons utilisé la version du gradient conjugué implémenté en Matlab par Carl Edward Rasmussen. Cette version utilise l'équation de Polack-Ribiere pour trouver la direction de recherche et l'optimisation unidimensionnelle s'effectue à l'aide d'approximation cubique et quadratique de l'erreur et du gradient.

Pour éviter le surapprentissage, nous avons utilisé la technique de la dégradation des poids ainsi que l'arrêt prématuré (Section 2.3.1). L'arrêt prématuré s'est effectué en choisissant l'ensemble de paramètres les plus performants sur les 10 chansons contenues dans l'ensemble de validation.

4.2.5 Extraction des pics

Pour ce qui est de l'extraction des pics, nous avons utilisé le même algorithme que dans la section 3.3.3.2. C'est-à-dire : un filtre passe haut suivi d'un seuil appris sur l'ensemble d'entraînement.

4.3 Expériences

Dû à la taille réduite des filtres dans le cas du RNC, nous savons déjà que la vitesse de l'algorithme s'est beaucoup améliorée. Cependant, nous sommes intéressés à savoir si cette réduction de paramètres affecte la performance. Il est possible que la flexibilité réduite empêche de percevoir certains évènements, mais il est aussi possible que cette même baisse de flexibilité aide à la généralisation.

Par conséquent, nous avons effectué une batterie de tests, qui nous permet de comparer la

différence de performance entre le RNF et le RNC. Nous avons aussi profité de l'installation pour tester l'algorithme sur plusieurs autres hyper-paramètres afin de répondre à d'autres questions. Ces tests supplémentaires nous ont permis, entre autres, de mesurer l'importance du prétraitement, de vérifier si la translation spectrale peut être avantageuse et, finalement, si l'algorithme peut fonctionner en temps réel.

4.3.1 Méthodologie

Pour mesurer la performance de notre algorithme, nous avons utilisé la F-mesure tel qu'expliqué à la section 3.3.3.3. Cette mesure correspond à la moyenne harmonique entre le ratio de faux positifs et le ratio de faux négatifs.

Afin d'obtenir un meilleur estimé de notre F-mesure, tous nos tests sont effectués en utilisant la technique de validation croisée sur 6 segmentations différentes de notre ensemble de données. Pour obtenir une valeur sommaire de la validation croisée, nous calculons la moyenne des résultats obtenus. Pour estimer l'erreur sur notre mesure, nous avons utilisé la loi de Student, qui est décrite par l'équation suivante :

$$s_n = A\frac{\sigma_n}{\sqrt{n}}$$

où n est la taille de l'ensemble de points, σ_n en est la variance et A est un facteur proportionnel à l'intervalle de confiance. Cette fonction permet de mesurer l'erreur sur l'estimation de la moyenne. Pour $n = 6$ et un intervalle de confiance de 90 %, $A \simeq 1.476$. Ce qui nous donne $s_n \simeq 0.6\sigma_n$.

Dans le chapitre 3, nous avons utilisé une tolérance de 50 ms pour l'extraction des pics. C'est-à-dire, la distance maximale entre l'évènement tuteur et l'évènement trouvé, pour qu'ils soient considérés corrélés. Cette valeur de tolérance provient de MIREX 2005. Cependant, nous avons jugé qu'une valeur plus faible serait plus appropriée et ferait ressortir la précision de l'algorithme. De plus, lorsqu'un extrait musical contient une haute densité d'évènements, une valeur élevée de la tolérance permet à une réponse aléatoire d'être très souvent associée à un pic. Pour améliorer la situation, nous avons décidé d'utiliser une valeur de tolérance de 30 ms.

4.3.2 Paramètres par défaut

Afin de simplifier la notation, nous décrivons ici les valeurs par défauts utilisées pour notre architecture. Entre parenthèses, se trouve un acronyme pour chaque paramètre qui servira pour des références futures.

- Nombre de filtres sur la première couche (NF1) : 6
- Nombre de filtres sur la deuxième couche (NF2) : 0 (1 couche cachée)
- Largeur temporelle de la première couche (LT1) : 40 ms (8 trames)
- Largeur temporelle de la deuxième couche (LT2) : 200 ms (40 trames)
- Largeur temporelle de la couche de sortie (LT3) : non utilisée
- Nombre de translations spectrales couche 1 (TS1) : 0
- Nombre de translations spectrales couche 2 (TS2) : 0
- Nombre de fréquences mel extraites (NFM) : 60
- Nombre de trames par seconde (NTS) : 200
- Délai de la courbe tutrice (DCT) : 0 ms

4.3.3 RNC vs RNF

Le premier test que nous avons effectué sert à comparer la différence de performance entre le RNC et le RNF. Étant donné que l'environnement de test est relativement différent que celui décrit dans le chapitre 3, il est inapproprié de directement comparer ces résultats avec ceux obtenus pour le RNC. Cependant, il est possible de simuler l'architecture du RNF avec celle du RNC. Il suffit de choisr des filtres de taille 1 pour toutes les couches, excepté pour la première couche. Les paramètres choisis pour simuler le RNF sont donc :

- LT1 = 200 ms (40 trames)
- LT2 = 0 ms (1 trame)
- Ainsi que tous les autres paramètres par défaut.

La figure 4.2 présente la différence de performance pour ces deux architectures. Il est donc évident que le RNC apporte un avantage significatif par rapport au RNF. Pour faire ressortir la différence de performance, nous avons fait varier la taille de l'ensemble d'entraînement. Nous pouvons observer que les performances du RNC se dégradent moins rapidement que celles du RNF et il continue de bien se comporter pour des ensembles de données aussi petits

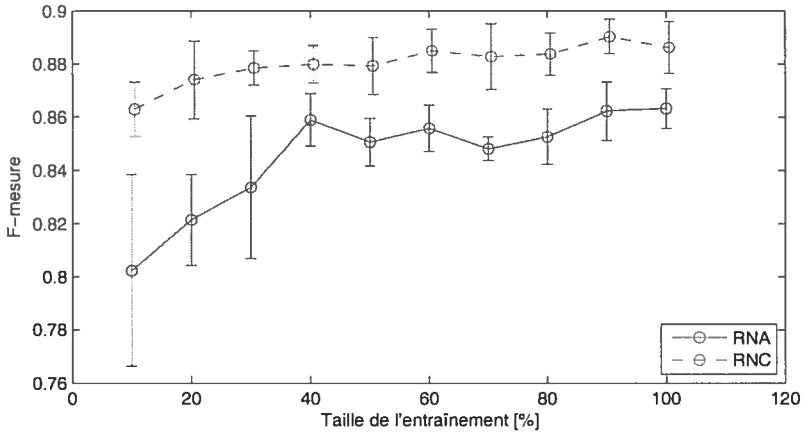

FIG. 4.2 – Test démontrant la différence de performance entre le RNC et le RNF. La taille de l'ensemble d'entraînement est progressivement réduite, pour faire ressortir la différence.

que 10 % de la taille originale, soit 4 chansons de 10 secondes.

La différence de performance peut provenir de deux endroits différents. Premièrement, le nombre de paramètres pour le RNF est de $40 \times 60 \times 6 + 6 \times 1 \times 1 = 14406$ alors que le RNC n'en possède que $8 \times 60 \times 6 + 40 \times 6 \times 1 = 3120$. Le RNC possède donc presque 5 fois moins de paramètres à optimiser, ce qui lui permet de généraliser plus facilement. Deuxièmement, l'architecture du RNC est aussi un facteur qui peut expliquer sa performance. Par exemple, la deuxième couche du RNC traite 240 variables de haut niveau contrairement au RNF qui n'en traite que 6.

4.3.4 Influence du prétraitement

Dans cette section, nous désirons explorer l'importance du prétraitement ainsi que les valeurs optimales. Pour ce faire, nous avons d'abord évalué les performances pour un différent nombre de fréquences mel (figure 4.3). Étonnamment, une seule bande de fréquence est suffisante pour obtenir une F-mesure de 84. Évidemment, en augmenter le nombre améliore la performance, pour autant que nous n'en rajoutions pas trop. En fait, l'apogée est atteint à un nombre de 8 bandes de fréquences, pour finalement redescendre légèrement.

Nous croyons que cette baisse de performance n'est pas due au nombre accru de paramètres, mais plutôt à l'amplitude logarithmique du spectrogramme. L'amplitude logarithmique permet de faire ressortir les faibles sons, pour percevoir les évènements plus subtils.

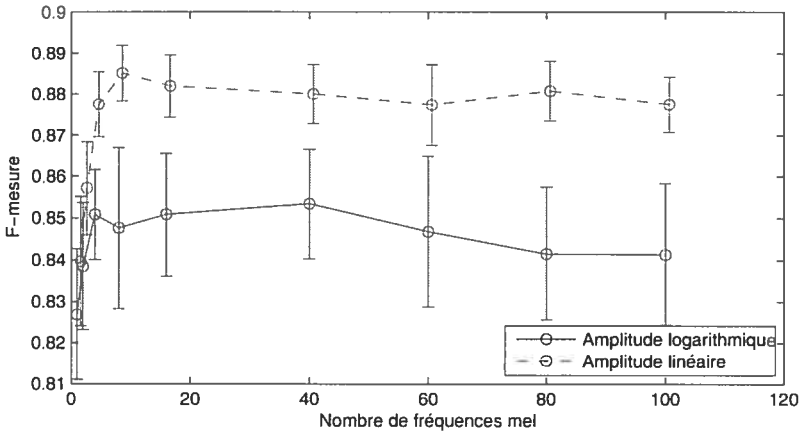

FIG. 4.3 – Test démontrant la performance en fonction du nombre de fréquences mel extraites, ainsi que l'utilité d'appliquer la fonction logarithmique sur l'amplitude du spectrogramme.

Cependant cette méthode pollue aussi les amplitudes plus élevées.

Considérons le cas extrême où un évènement est produit par un son sinusoïdal de haute fréquence et de haute amplitude. Malgré que l'intensité soit élevée, très peu de fréquences seront actives dans la TFL. Supposons qu'une fois réduite à l'échelle mel, une seule bande soit active. Alors pour 100 fréquences mel, il n'y aura que 1 % des bandes qui seront actives contrairement à 10 % dans le cas de 10 fréquences mel. À cette étape, nous n'avons toujours pas appliqué la fonction logarithme, alors il est probablement encore possible de distinguer l'évènement pour un nombre de 100 fréquences mel, dû à l'intensité élevée de l'évènement. Cependant, une fois le logarithme appliqué, cette bande de fréquence ressort beaucoup moins et ne constitue qu'un indice très faible parmi toutes les autres variables. Ce phénomène expliquerait pourquoi nous avons eu des performances beaucoup plus faibles dans la catégorie percussions métalliques du concours MIREX 2005 (tableau 3.7).

La courbe pleine de la figure 4.3 affiche les résultats obtenus si nous enlevons la fonction logarithmique. Les faibles performances obtenues témoignent de l'importance d'utiliser ce traitement. Cependant, le fait que la valeur optimale du nombre de fréquences mel se situe autour de 40, appuie l'énoncé précédent sur les sons aigus.

4.3.5 Translations spectrale

Dans la section 3.3.2.1 nous avons appliqué une translation spectrale au RNF selon l'affirmation que le tâche possédait une invariance de translation pour de faibles déplacements. Dans cette section, nous effectuons un test, pour vérifier si cette translation spectrale peut s'avérer utile. La figure 4.4 affiche les résultats obtenus en faisant varier le nombre de translations appliquées. Le paramètre qui nous permet d'ajuster le nombre de translations spectrales effectuées est la taille verticale de notre filtre. Pour ce test, nous avons utilisé un nombre de fréquences mel de 100, alors, pour un filtre de taille 99, 1 translation verticale est effectuée et pour un filtre de taille 20, 80 translations sont effectuées. Nous avons aussi utilisé un sous-échantillonnage de 2 pour réduire la quantité de variables fournies à la deuxième couche.

La figure 4.4 montre que pour une faible valeur de translations, le RNC performe moins bien que le cas où aucune translation n'est appliqué (c'est-à-dire : Une F-mesure de 0.87 par rapport au 0.88). Par contre, pour des valeurs suffisamment élevées de translations, la performance est rapidement rétablie, voire même légèrement supérieure, pour une telle quantité de fréquences mel. De plus, pour une valeur de translations de 30, des performances de l'ordre de 0.895 sont atteintes, ce qui constitue la meilleure performance obtenue sur cet ensemble de données.

Par contre, la forte erreur obtenue sur notre mesure de performance nous empêche d'affirmer avec certitude que les translations spectrales sont effectivement avantageuses. De plus, le temps d'exécution de l'algorithme augmente linéairement avec le nombre de translations, ce qui le rend 30 fois plus lent pour obtenir la performance optimale.

4.3.6 Délai de la courbe tutrice

En appliquant un délai sur la courbe tutrice, nous forçons le RNC à répondre en avance ou en retard par rapport au centre de ses filtres (section 4.2.3). Si le délai correspond à la moitié de la largeur totale du RNC, alors il est restreint à utiliser uniquement l'information provenant du passé et peut alors être utilisé dans un environnement temps réel.

Dans cette section, nous regardons l'impact de ce délai sur les performances. Les paramètres du RNC pour ce test sont :

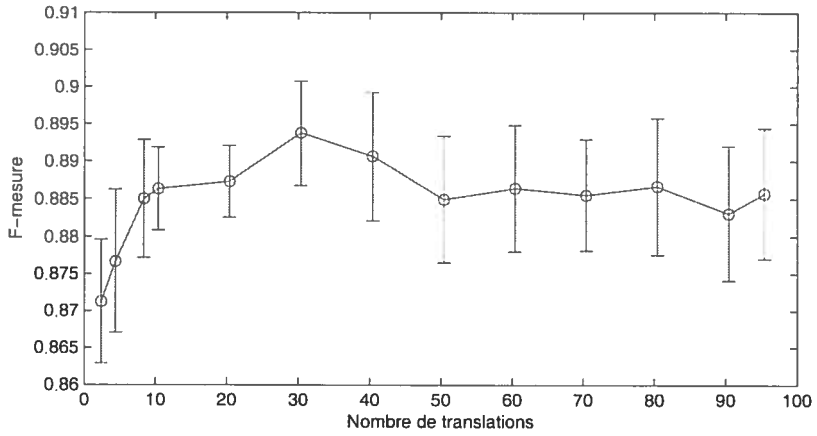

FIG. 4.4 – Résultats pour la translation des filtres selon l'axe des fréquences.

- LT1 = 40 ms (8 trames)
- LT2 = 100 ms (20 trames)
- LT3 = 200 ms (40 trames)

La largeur totale du RNC est donc de $40+20-1+8-1=66$ trames (section 4.2.3) et donc de 330 ms. Par conséquent, un délai positif de 165 ms permet d'utiliser l'algorithme en temps réel. La figure 4.5 montre la performance pour différentes valeurs du délai. Nous pouvons voir que les performances restent stables jusqu'à 160 ms et diminuent très rapidement par la suite. Le graphique ne montre pas la performance pour un délai de 165 ms, mais un délai de 160 ms est suffisant pour considérer notre algorithme temps réel, puisque de toute façon, la fenêtre spectrale utilisée est de 36 ms et requiert donc 17 ms d'information provenant du futur.

Un autre résultat intéressant concerne le délai négatif. Il est aussi possible de bien découvrir les évènements en n'utilisant que l'information provenant du futur. Cependant, la figure 4.5 montre une perte de performance plus rapide et plus importante pour un délai négatif. De plus, après 140 ms, les performances se dégradent très rapidement. Mais un tel algorithme serait suffisant pour découvrir les évènements en début de segments, lorsque très peu d'information sur le passé est disponible.

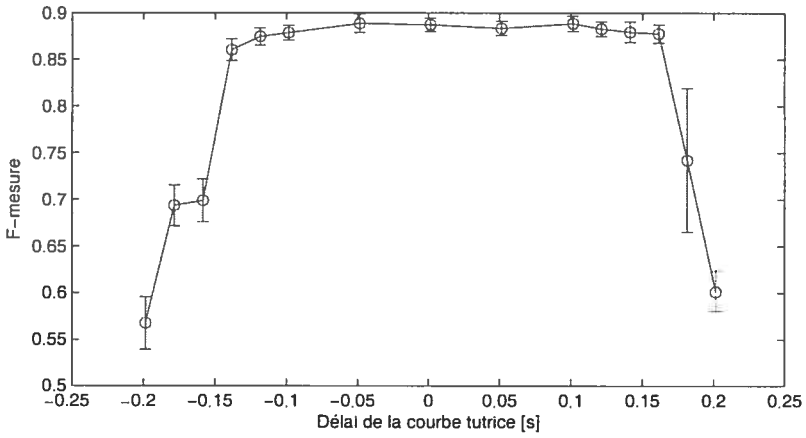

FIG. 4.5 – Test démontrant la limite du délai de la courbe tutrice tant au niveau positif qu'au niveau négatif.

4.3.7 À propos de la vitesse d'exécution

Dans la section 4.3.4, nous avons vu que seulement 8 fréquences mel étaient nécessaires pour obtenir les performances optimales. De plus, la section 4.3.3 montre que nous pouvons significativement diminuer la quantité de multiplications à effectuer en utilisant un RNC au lieu d'un RNF. Dans cette section, nous diminuons progressivement le nombre de trames par seconde de notre spectrogramme ainsi que le nombre de neurones de la première couche. De plus, nous ajustons quelques autres paramètres :

- LT1 = 20 ms
- LT2 = 100 ms
- NFM = 7
- DCT = 50 ms (temps réel)

Toutes ces modifications ont pour but de trouver un algorithme rapide fonctionnant en temps réel. La figure 4.6 montre les résultats obtenus pour différents nombre de neurones ainsi que pour différents nombres de trames par seconde. Nous pouvons observer que deux neurones ainsi que 80 trames par seconde sont suffisants pour rester près d'une F-mesure de 88.

Nous avons aussi effectué des tests pour mesurer les différences de vitesse. Les tests sont effectués sur un processeur AMD Athlon 64 3200+ avec 512 KB de cache, cadencé à 2.2 GHz. La fonction pour calculer les spectrogrammes est specgram de Matlab et les convolutions sont effectuées à l'aide de multiplications matricielles appelant la librairie ATLAS.

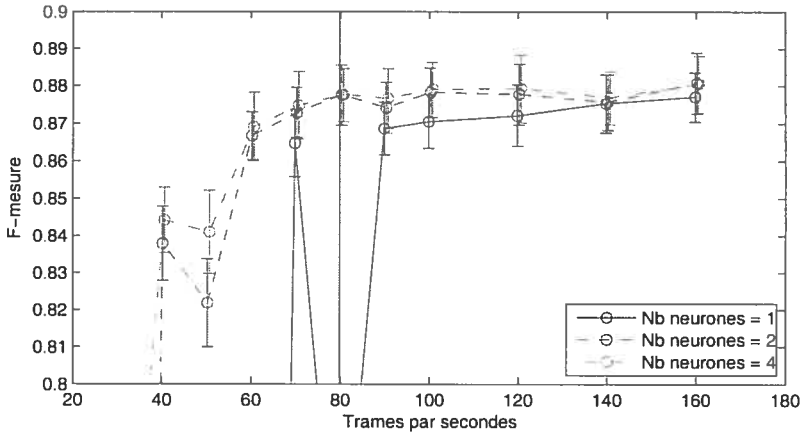

FIG. 4.6 – Test démontrant la baisse de performance lors que le nombre de trames par seconde est réduit ainsi que lorsque le nombre de neurones est réduit.

RNF	30.3 ms
RNC-défaut	8.3 ms
RNC-rapide	1.1 ms
RNF-trans	89.3 ms

TAB. 4.1 – Mesure de vitesse d'exécution pour différentes architectures. La mesure correspond au temps nécessaire pour découvrir 1 seconde d'évènements et n'inclut pas le temps de calcul du spectrogramme.

Pour calculer un spectrogramme de 200 trames par seconde, 36 ms de fenêtre spectrale ainsi que 400 fréquences, il faut 32.3 ms pour chaque seconde de signal alors que pour un spectrogramme de 80 trames par seconde, il en faut 12.8 ms. En réduisant la largeur de la fenêtre spectrale ainsi que le nombre de fréquences, nous aurions probablement une bonne accélération sans trop de pertes de performances, mais nous en sommes resté là.

Pour ce qui est de la vitesse d'exécution du RNC, nous avons mesuré la vitesse de quatre architectures différentes :

- RNF (paramètres de la section 4.3.3).
- RNC-défaut (paramètres de la section 4.3.2).
- RNC-rapide (paramètres de la section 4.3.7).
- RNF-trans (paramètres de la section3.3).

Les résultats de la table 4.1 montrent un progrès énorme de la vitesse d'exécution en fonction de la simplicité du modèle. L'algorithme présenté à la section 4.3.7 s'exécute presque

100 fois plus rapidement que celui utilisé au concours MIREX 2005. Cependant, ces mesures n'incluent pas le temps de calcul du spectrogramme. Alors que, dans les premières versions, le temps d'exécution était dominé par le RNF, dans le cas du RNC, le temps d'exécution est dominé par le calcul du spectrogramme. Il est cependant plausible de trouver une méthode beaucoup plus rapide pour calculer un spectrogramme contenant uniquement 7 bandes de fréquences.

4.4 Travaux futurs

Malgré que les résultats sont très intéressants, la F-mesure demeure encore sous 0.9. Pour certaines applications, ce taux d'erreur est encore trop élevé. Cependant, ces applications sont généralement plus spécifiques et la définition d'évènements est potentiellement plus concise.

Dans notre cas, nous avons construit notre ensemble de données dans le but de détecter les évènements dans les signaux sonores complexes. Cette situation rend plusieurs évènements difficiles à détecter. Mais le plus important est que cette situation rends plusieurs évènements difficiles à annoter. Un cas fréquent constitue le chant. Certaines notes vocales sont très percussives, alors que d'autres progressent lentement tout en changeant de fréquences et contiennent parfois du vibrato. Comme il y a un continuum entre ces deux cas, il est souvent difficile de trancher, à savoir, s'il s'agit d'un évènement ou pas.

La meilleure solution pour pallier à ce problème consiste à entraîner l'algorithme pour la tâche désirée en vérifiant les erreurs qu'il performe et en les corrigeant au besoin. Cependant, pour ce faire, l'utilisateur doit construire son propre ensemble de données et posséder un utilitaire lui permettant de corriger les erreurs.

Pour aller chercher un gain de performance supplémentaire, sans avoir à construire un ensemble de données différent pour chaque situation, il serait intéressant de considérer le boosting [DSS93] où chaque RNC serait considéré comme un appreneur faible. De cette manière, l'algorithme pourrait se concentrer sur certains types d'évènements généralement difficiles à trouver.

Pour accélérer le calcul du spectrogramme, la branche des transformées d'ondelettes pourrait être considérée. De manière générale, les transformées d'ondelettes représentent une banque de filtres espacés logarithmiquement. Un avantage important de cette technique est la

rapidité à laquelle une transformée d'ondelette discrète (TOD) est exécutée. Contrairement à la transformée de Fourier, rapide, qui s'exécute en un temps $O(n \log n)$, la TOD s'exécute en un temps $O(n)$ où n est la longueur du signal à transformer. Cependant cette technique possède deux désavantages principaux. Premièrement, une TOD ne possède qu'une résolution spectrale d'un octave, ce qui restreint le spectrogramme à environ 8 bandes de fréquences. Heureusement, les résultats de la section 4.3.4 nous indique que cette restriction n'est pas un problème. Le deuxième désavantage est cependant plus difficile à contourner : la TOD ne calcule pas de partie imaginaire. Il n'est donc pas possible d'extraire l'amplitude du spectrogramme. Pour pallier à ce problème, il faudrait explorer la transformée d'ondelettes complexe à arbre dual [SBK05]. Cette technique pourrait nous permettre d'obtenir très rapidement un spectrogramme de 8 bandes de fréquences espacées logarithmiquement et, par conséquent, permettrait de détecter les évènements d'un segment de 1 seconde d'audio en moins de 2 ms.

4.5 Conclusion

Dans ce chapitre, nous avons confirmé l'avantage du RNC par rapport au RNF pour la détection d'évènements. Le RNC permet d'atteindre des performances de 0.895 de F-mesure alors que le RNF se limite aux alentours de 0.86 de F-mesure. De plus, nous avons démontré à la section 4.3.6 que notre algorithme pouvait fonctionner en temps réel, en n'utilisant que l'information provenant du passé. Finalement, en permettant de diminuer légèrement les performances de l'algorithme, nous avons trouvé les paramètres pour obtenir un algorithme s'exécutant en 1.1 ms pour 1 seconde de signal sonore à partir du spectrogramme.

Ces résultats dépassent de beaucoup les performances des algorithmes modernes. Nous croyons aussi que les performances obtenues sont suffisantes pour plusieurs tâches de détection d'évènements. Cet algorithme pourrait donc être mis à la disposition sous forme de librairie, pour être utilisé dans des problèmes de traitements sonores plus complexes.

CHAPITRE 5

RÉSEAU DE NEURONES À CONVOLUTION POUR LA RECONNAISSANCE DE GENRE.

5.1 Introduction

Dans le chapitre 4 nous avons vu que le RNC pouvait apporter des bénéfices face à l'architecture plus traditionnelle du RNF et ce, sans pour autant effectuer un prétraitement spécifique à la détection d'évènements. Aussi, contrairement au RNF, le RNC peut s'appliquer à des signaux de plus forte taille sans avoir une augmentation dramatique du nombre de paramètres à ajuster. Dans ce chapitre, nous nous intéressons à savoir si le RNC peut s'appliquer avec succès à d'autres tâches en recherche d'information musicale. Plus précisément, nous analysons les résultats obtenus lorsque le RNC est appliqué au problème de reconnaissance du genre musical.

Malgré que nous n'ayons recueilli que des résultats préliminaires, nous avons comparé les résultats obtenus à un algorithme moderne décrit dans [Ber06], pour établir la différence de performance. De plus, nous avons construit un modèle hybride composé de ces deux algorithmes, pour combiner leurs avantages.

Dans ce chapitre, nous révisons brièvement certains travaux importants effectués dans la littérature de la classification musicale. Par la suite, nous décrivons les 3 différents algorithmes analysés. Finalement, pour évaluer le potentiel du RNC, nous discutons les résultats obtenus sur un jeu de données de 10 genres (section 5.5.1).

5.2 Survol sur la classification du genre musical

Alors que la classification de genre musical est une tâche récente, elle s'inspire grandement des techniques de classification utilisées en reconnaissance de la parole, une tâche beaucoup plus mature. Ces techniques utilisent généralement un algorithme d'apprentissage sur des statistiques de coefficients spectraux pour discriminer les signaux linguistiques des signaux musicaux.

S'inspirant de ces travaux sur la reconnaissance de la parole, Tzanetakis et Cook [TC02] produisent un algorithme catégorisant les signaux sonores en 10 genres musicaux. L'algorithme extrait d'abord plusieurs caractéristiques spectrales ainsi que quelques caractéristiques sur la structure temporelle. La moyenne et la variance des caractéristiques spectrales sont extraites sur plusieurs trames afin d'obtenir une description globale de l'extrait musical. Utilisant un mélange de Gaussienne pour estimer la distribution de probabilité pour chaque classe, l'algorithme obtient une performance de $61 \pm 4\%$ de pièces correctement classifiés.

Par la suite, [LOL03] entreprends une exploration plus approfondie de l'importance des différentes caractéristiques extraites, ainsi que la performance de différents algorithmes de classification. En utilisant des caractéristiques similaires extraites du même jeu de données que dans [TC02], Tao Li et Al. obtiennent des performances de $72.1 \pm 4.68\%$ en utilisant la machine à vecteurs de support (SVM) comme algorithme de classification. Ils montrent, par la même occasion, que le ton ainsi que le rythme sont très peu discriminatifs voir même nuisibles. De plus, ils apportent un nouvel ensemble de caractéristiques, calculées en effectuant des statistiques sur les coefficients d'une transformée d'ondelettes. À l'aide de cette nouvelle représentation, ils obtiennent une performance de $78.5 \pm 4.07\%$ sur le jeu de données de [TC02].

Récemment, lors du concours de classification de genre de MIREX 2005, des algorithmes plus performants font surface. Les vainqueurs du concours rapportent dans [BCE+06], des performances de 83% avec leur algorithme sur le jeu de donnés de [TC02]. Pour ce faire, l'algorithme poursuit une stratégie similaire à [TC02, LOL03]. Une grande quantité de caractéristiques spectrales sont d'abord extraites, des statistiques sont effectuées sur ces caractéristiques et finalement AdaBoost ([FS95]) est utilisé pour classifier ces descripteurs. La force de l'algorithme réside dans la très grande quantité de caractéristiques spectrales extraites combiné avec AdaBoost, capable d'ignorer les caractéristiques peu discriminatives.

Dans [Ber06], une étude plus approfondie des caractéristiques utilisées dans [BCE+06] est effectuée. Cette fois-ci, l'algorithme d'apprentissage utilisé est le RNF. De plus, dans l'extraction des statistiques sur les coefficients, la covariance est aussi utilisée, ce qui génère $\frac{n(n+1)}{2}$ coefficients pour la matrice de covariance ainsi que n autres coefficients pour la moyenne où n correspond au nombre de coefficients spectraux extraits d'une trame. De plus,

il démontre que si les statistiques sont extraites sur des séquences de plus de 2 secondes, les performances diminuent. Finalement, il montre qu'à l'aide de statistiques sur les coefficients Log-mel-TFL, il obtient une performance de 80.9 % de pièces correctement classifiés sur le jeu de données de [TC02].

5.3 Motivation

Un élément important se retrouve en commun avec tous ces algorithmes de classification de genre. Très peu, sinon aucune, information concernant la structure temporelle de la pièce musicale est utilisée. Dans [LOL03], un test effectué suggère que les caractéristiques temporelles extraites apportent peu de gain et sont même potentiellement nuisible. Dans [Ber06] un test montre que si les statistiques spectrales sont effectuées sur une fenêtre de plus de 2 secondes, les performances diminuent, ce qui pousse à prédire le genre sur une très courte durée. Après quoi, les réponses obtenues pour l'ensemble des fenêtres de 2 secondes dans un extrait musical sont combinées à l'aide d'une simple moyenne, ce qui, encore une fois, élimine toute information concernant la structure musicale.

Malgré les résultats obtenus par ces tests, il est connu que plusieurs genres musicaux possèdent une structure temporelle particulière (par exemple : reggae, valse). De plus, une autre tâche importante en RIM consiste à prédire le genre d'une composition uniquement à partir de sa partition ([MF05]) et des performances supérieures à 80 % sont obtenues. Il est donc envisageable qu'un algorithme capable de traiter cette information puisse obtenir une meilleure performance de classification. Le RNC est un bon candidat pour cette tâche. Cependant, le type de statistiques extraites sur les notes dans [MF05] est peut-être plus complexe que ce que le RNC peut extraire.

5.4 Description des architectures

Dans cette section, nous décrivons les architectures des trois algorithmes implémentés. La première est celle décrite dans [Ber06], qui nous permettra d'obtenir une valeur de référence sur la performance de notre algorithme. La deuxième architecture correspond au RNC. Finalement, nous avons combiné les deux architectures afin de profiter de leurs avantages complémentaires. Aux fins de références ultérieures, nous les avons nommés respectivement

Stats-RNF, *RNC-Genre* et *Hybrid*.

Dans tous les cas, nous avons utilisé la fonction d'activation softmax afin d'obtenir une distribution de probabilité sur les 10 genres :

$$a_i(\mathbf{x}) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

où, dans notre cas, $\mathbf{x}$ correspond à la sortie de la dernière couche. La fonction de coût utilisé est le Log-vraisemblance, qui correspond à la divergence de Kullback-Leibler entre la distribution de probabilité fourni par l'algorithme et la distribution tutrice. Cette dernière étant construite en attribuant une probabilité de 1 à la classe préalablement identifié par un humain.

Pour optimiser nos paramètres, nous avons, dans un premier temps, utilisé la descente de gradient stochastique [Bot04] avec *momentum* [OL94]. Ce choix est justifié par la taille substantielle de notre ensemble de données (5.5.1). Cependant, les hyperparamètres de cet optimiseur semblaient avoir beaucoup d'influence sur les résultats de test et il nous fallait effectuer plusieurs optimisations complètes, pour trouver les valeurs appropriées. Pour obtenir une convergence plus stable, nous avons alors décidé d'utiliser le gradient conjugué, qui prends plus de temps à converger mais qui trouve les paramètres optimaux en une seule exécution. L'implémentation utilisée est celle de Carl Edward Rasmussen décrite à la section 4.2.4.

Afin d'atténuer le sur-apprentissage, nous avons utilisé la technique de l'arrêt prématuré ainsi que la minimisation de la norme du vecteur de paramètres, tel que discuté à la section 2.3.1.

5.4.1 Architecture *Stats-RNC*

Tel que décrit à la section 5.2, les statistiques sur les coefficients spectraux apportent une information importante sur le genre musical. Nous avons donc choisi d'implémenter un des algorithmes présentés dans [Ber06] pour comparer les résultats (figure 5.1a). Dans cette méthode, l'extrait musical est d'abord transformé sous la représentation Log-mel-TFL. De là, le spectrogramme est séparé en n segments de t secondes, catégorisés séparément. Pour obtenir une valeur unique par extrait musical, la moyenne des prédictions est effectuée

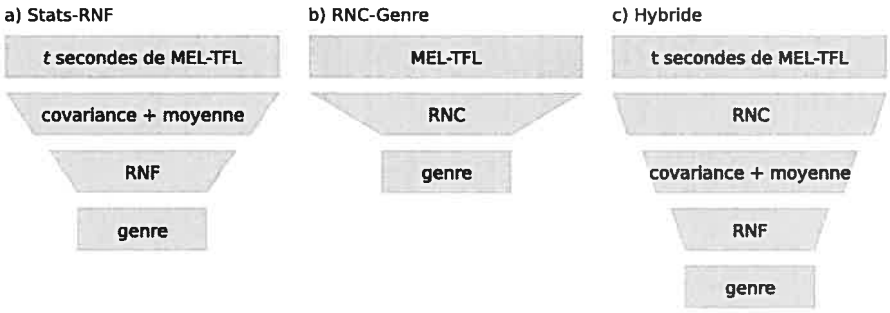

FIG. 5.1 – Représentation des différentes architectures implémentées : a) Implémentation d'un l'algorithme décris dans [Ber06] ; b) Implémentation de RNC pour le genre ; c) Modèle hybride combinant les deux architectures.

sur l'ensemble des segments. Pour catégoriser un segment, la moyenne et la matrice de covariance sont extraites sur l'ensemble des trames contenues dans le segment. Pour des trames de taille k, ces statistiques nous procurent $k + (k+1)\,k/2$ caractéristiques décrivant un segment. Finalement, ces statistiques sont présentées en entrée à un RNF afin de trouver une transformation adéquate, estimant le genre d'un segment.

5.4.2 Architecture *RNC-Genre*

La deuxième architecture est implémentée à l'aide du RNC (figure 5.1b). Encore une fois, nous utilisons la représentation Log-mel-TFL, mais cette fois-ci, l'extrait sonore n'est pas segmenté. Le RNC est donc appliqué directement sur le spectrogramme. Le sous-échantillonnage nous permet d'obtenir progressivement une représentation de haut niveau mais comme l'extrait musical peut être de taille variable, il n'est pas possible de retourner une estimation unique pour la chanson. Pour ce faire, nous retournons une estimation pour chaque moment de l'extrait et nous calculons la moyenne des prédictions afin d'obtenir une valeur globale.

5.4.3 Architecture *Hybride*

Afin de profiter des avantages des deux algorithmes, nous les avons combinés. Pour ce faire, nous avons tout simplement rajouté un RNC entre le spectrogramme et les statistiques dans l'architecture 5.4.1 (figure 5.1). Cet ajout permet d'effectuer une transformation plus appropriée de notre spectrogramme avant d'y extraire nos statistiques.

5.5 Résultats obtenus

Afin d'évaluer le comportement de nos architectures, nous avons effectué des tests sur le jeu de données de [TC02]. Ces tests nous permettent de comparer nos algorithmes entre eux ainsi qu'aux autres algorithmes discutés dans la section 5.2, testés sur le même jeu de données.

5.5.1 Jeu de données

Le jeu de données de [TC02] est constitué de 1000 extraits musicaux d'une durée de 30 secondes chacun, enregistrés selon le format de modulation d'impulsion codée avec une fréquence d'échantillonnage de 22050 Hz. Ces extraits musicaux ont été catégorisés selon les genres suivants : blues, classique, country, disco, hip-hop, jazz, métal, pop, reggae et rock.

5.5.2 Méthodologie

Afin d'estimer la performance de notre algorithme, nous avons utilisé la technique de validation croisée présentée à la section 2.3.1. Les tests sont performés sur 4 séparations de notre jeu de données, en utilisant 20 % pour le test, 10 % pour la validation et 70 % pour l'entraînement. La séparation de notre ensemble de données est effectuée au niveau des extraits musicaux et non au niveau des segments de 2 ± 1 secondes utilisés pour la classification. Dans le cas contraire, un même extrait aurait pu se retrouver séparé entre l'ensemble d'entraînement et l'ensemble de test, provoquant une valeur erronée de la mesure de généralisation.

La mesure de performance pour un apprentissage est calculée à partir du nombre moyen d'exemples classifiés correctement sur l'ensemble de test. Finalement, dans le but d'obtenir une valeur sommaire de la performance sur la validation croisée, nous avons extrait la moyenne et l'écart type de l'ensemble des tests.

5.5.3 Performances

Pour chacun des trois algorithmes, nous avons effectué une recherche sur les paramètres afin de trouver une architecture convenable. La table 5.1 montre les résultats obtenus pour les paramètres choisis (Section 5.5.4).

Stats-RNF	78.2 ± 4.3 %
RNC-Genre	72.3 ± 4.8 %
Hybride	80.5 ± 4.2 %

TAB. 5.1 – Résultats obtenus pour les 3 architectures sur la tâche de classification de genre. L'incertitude est représentée par l'écart type sur la validation croisée.

Les résultats montrent que le RNC à lui seul n'atteint pas les performances obtenues par les algorithmes modernes de classification de genre. Cependant, les résultats sont supérieurs à ceux obtenus dans [TC02] et comparables à ceux obtenus dans [LOL03]. De plus, le modèle *Hybride*, qui applique d'abord le RNC avant d'extraire la matrice de covariance, permet, selon nos tests, d'améliorer les performances du modèle proposé dans [Ber06]. Par contre, il est à noter que nous n'avons pas réussi à obtenir exactement les mêmes résultats que ceux présentés dans [Ber06], qui sont de 80.9 % au lieu de 78.2 %.

5.5.4 Paramètres des architectures

Dans cette section, nous discutons des paramètres qui ne sont pas appris par descente de gradient mais par une recherche exhaustive en effectuant un apprentissage complet pour différentes valeurs de ces paramètres. Évidemment, le nombre de configurations augmente exponentiellement avec le nombre de paramètres, rendant l'optimisation presque impossible. Pour ce faire, nous avons guidé la recherche intuitivement en choisissant manuellement les valeurs en fonctions des résultats obtenus à l'itération précédente.

Pour le calcul du spectrogramme, nous avons choisi une fréquence d'échantillonnage de 22050 Hz, une fenêtre spectrale de 40 ms et un facteur d'enchevêtrement de 0.7. Pour la projection sur l'échelle mel, nous avons choisi 30 bandes de fréquences. Pour les deux architectures où le spectrogramme est segmenté, nous avons utilisé 30 segments de 1 seconde par extrait musical.

5.5.4.1 Paramètres pour l'architecture *Stats-RNF*

Pour cette architecture, nous avons utilisé 30 neurones sur la première couche cachée ainsi que 20 sur la deuxième. La couche de sortie étant fixée à 10 par le nombre de classes.

	couche #1	couche #2	couche de sortie
NF	16	10	10
LT	10 trames (130 ms)	15 trames (390 ms)	20 trames (1 s)
SE	2 trames	2 trames	4 trames

TAB. 5.2 – Liste des paramètres choisis pour l'architecture RNC-Genre

5.5.4.2 Paramètres pour l'architecture *RNC-Genre*

Pour cette architecture, le nombre de couches cachées optimal s'est avéré être 2. La deuxième augmente légèrement la performance par rapport à une seule couche alors que la troisième semble diminuer la performance. La table 5.2 affiche, pour chaque couche, le nombre de filtres utilisés (NF), la largeur temporelle des filtres (LT) ainsi que le sous-échantillonnage (SE).

5.5.4.3 Paramètres pour l'architecture *Hybride*

Les meilleures performances pour cette architecture, sont obtenues en n'utilisant qu'une seule couche à convolution de 20 filtres à l'entrée ainsi qu'une seule couche cachée de 20 neurones pour le RNF. Comme les statistiques sont effectuées après la couche à convolution, la quantité de sous-échantillonnage est fixée à une trame. Pour ce qui est de la largeur des filtres, nous avons choisi 10 trames (130 ms).

5.6 Discussion

Tel que discuté à la section 5.3, une des raisons principales motivant l'utilisation du RNC pour la classification de genre est le potentiel de pouvoir intégrer l'information sur une plus grande échelle de temps. Malheureusement, la recherche de paramètres énoncée à la section 5.5.4.2, montre que la largeur optimale du RNC est de moins de 2 secondes, ce qui ne permet pas de tenir compte de l'architecture temporelle de la chanson.

Il est donc possible qu'une couche de RNC ne permette pas une réduction de dimensionnalité suffisamment rapide, pour obtenir une largeur temporelle acceptable, avant d'atteindre une profondeur de réseau trop élevée.

La largeur choisie du RNC permet au mieux de tenir compte de la structure temporelle des instruments, une caractéristique impossible selon le modèle *Stats-RNF*. Cependant, ce

dernier a beaucoup mieux performé que le *RNC-Genre*. Le nombre de paramètres étant équivalent pour les deux algorithmes, laisse croire que la complexité des algorithmes n'est pas la cause de la différence de performances. Ceci suggère que le RNC a de la difficulté à extraire des caractéristiques équivalentes à la matrice de covariance des trames et que l'ordre des trames est de peu d'importance.

Les résultats obtenus par le modèle *Hybride* montre que le RNC peut être utilisé comme prétraitement pour améliorer les performances. Cet ajout fait cependant augmenter la profondeur de l'algorithme. Pour compenser il faut alors diminuer la profondeur dans les autres modules, pour les raisons discutées à la section 2.1.

Cette expérience montre que le RNC possède un potentiel intéressant pour être appliqué directement sur un spectrogramme mais qu'un prétraitement accru et une architecture appropriée peut apporter une différence significative de performance.

CHAPITRE 6

CONCLUSION

Dans ce mémoire, nous avons présenté des techniques à base de gradient pour résoudre des tâches en RIM. L'emphase est mise sur le RNC afin de déterminer son habileté à résoudre ces problèmes sans avoir à effectuer un prétraitement spécifique à la tâche.

Dans le chapitre 3, nous avons décrit une méthode utilisant le RNF pour résoudre la tâche de la DAEM. Cette méthode s'inspire des avantages du RNC pour faire convoluer un RNF sur le spectrogramme du signal sonore. Les résultats montrent que cette méthode est plus robuste et plus flexible que les méthodes modernes de traitements de signaux et obtient une F-mesure de 80.07 % par rapport à 74.80 % au concours de MIREX 2005. Cet avantage est cependant obtenu au dépend d'un temps de calcul beaucoup plus élevé. D'autres résultats montrent que l'information contenue dans la phase du spectrogramme peut permettre d'obtenir des performances comparables à celle contenue dans l'amplitude.

Dans le chapitre 4, nous avons démontré que le RNC est plus avantageux que le RNF dans le domaine de la RIM. Dans un premier temps, il obtient une F-mesure de l'ordre de 89 % alors que le RNF se limite à 86 % sur notre ensemble de données. De plus, le temps de calcul est significativement réduit par rapport à celui du RNF, permettant d'obtenir une vitesse comparable à celle obtenue par les autres algorithmes du concours de DAEM de MIREX 2005. Finalement, si une légère perte de F-mesure est tolérée, nous pouvons obtenir un algorithme s'exécutant en temps réel et ayant un temps d'exécution presque 100 fois plus faible que l'algorithme original utilisant le RNF.

Ces avantages proviennent principalement du fait que les couches sont convoluées indépendamment dans le cas du RNC. Ceci permet de faire une réduction de dimensionnalité progressive à chaque couche, contrairement au RNF, qui doit effectuer la majeure partie de sa réduction sur la première couche.

Finalement, nous avons appliqué le RNC à la tâche de la classification du genre musical. Pour ce problème, les algorithmes modernes sont mieux adaptés. Sur l'ensemble de données utilisé, le RNC obtient une performance de 72 % de classification alors qu'un algorithme moderne obtient une performance de 78 % de classification. Cependant, une combinaison de

ces deux algorithmes permet d'obtenir une performance de 81 %.

Il est à noter que les performances de 72 % obtenues pour la classification de genre se comparent très bien à beaucoup d'algorithmes dans l'histoire de cette tâche. Pour ces raisons, nous ne rejetons pas le potentiel du RNC à bien résoudre plusieurs autres problèmes en RIM. Il va de soi que la faible quantité de tâches explorées dans ce travail ne permet pas de tirer une conclusion.

6.1 Travaux futures

Il reste encore plusieurs autres tâches pouvant être abordées avec le RNC en RIM. Parmi celles-ci, nous pourrions explorer l'extraction de tempo, la reconnaissance d'artistes et la transcription automatique. Toutes ces tâches pourraient bénéficier du RNC, tout en apportant de la lumière sur son comportement.

Malgré les avantages nombreux du RNC, au cours de nos travaux, nous lui avons noté une faiblesse potentielle pour la tâche de la RIM. Il s'agit de la profondeur de réseau qui est limitée, tel que discuté à la section 2.1. Dans le chapitre 5, nous avons vu que de petits sous-échantillonnages sont requis pour obtenir une performance optimale. Il est possible que ce soit spécifique à la classification de genre mais s'il s'agit d'une caractéristique du RNC, il serait alors difficile d'extraire des caractéristiques de haut niveau avec seulement 2 couches cachées. Pour augmenter le nombre de couches cachées, nous pourrions faire appel aux réseaux génératifs profonds pour initialiser notre réseau ([BLPL07]) .

Finalement, il serait intéressant d'explorer d'autres avenues que le RNC pour résoudre des problèmes de RIM. Des algorithmes pouvant détecter d'autres types d'invariances dans le signal amélioreraient assurément le potentiel de généralisation.

BIBLIOGRAPHIE

[BCE+06] J. Bergstra, N. Casagrande, D. Erhan, D. Eck, and B. Kegl. Aggregate features and AdaBoost for music classification. *Machine Learning*, 2006. accepted.

[BDDS04] Juan Pablo Bello, Chris Duxbury, Mike Davies, and Mark Sandler. On the use of phase and energy for musical onset detection in the complex domain. *IEEE Signal Processing Letters*, 11(6) :553–556, 2004.

[Ber06] J. Bergstra. Algorithms for classifying recorded music by genre. Master's thesis, University of Montreal, 2006.

[Bis95] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, 1995.

[BLPL07] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*. MIT Press, Cambridge, MA, 2007.

[Bot04] Léon Bottou. Stochastic learning. In Olivier Bousquet and Ulrike von Luxburg, editors, *Advanced Lectures on Machine Learning*, number LNAI 3176 in Lecture Notes in Artificial Intelligence, pages 146–168. Springer Verlag, Berlin, 2004.

[BP92] Judith C. Brown and Miller S. Puckette. An efficient algorithm for the calculation of a Constant Q transform. *Journal of the Acoustical Society of America*, 92(5) :2698–2701, 1992.

[Bro91] Judith C. Brown. Calculation of a Constant Q spectral tansform. *Journal of the Acoustical Society of America*, 89(1) :425–434, 1991.

[BS03] Juan Pablo Bello and Mark Sandler. Phase-based note onset detection for music signals. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing. ICASSP-03*, pages 441–444, 2003.

[Cha92] C. Charalambous. Conjugate gradient algorithm for efficient training of artificial neural networks. *Circuits, Devices and Systems, IEEE Proceedings*, 139(3) :301–310, June 1992.

[DBDS03a] Chris Duxbury, Juan Pablo Bello, Mike Davies, and Mark Sandler. A combined phase and amplitude based approach to onset detection for audio segmentation. In *Proceedings of the 4th European Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS-03)*, pages 275–280, London, UK, 2003.

[DBDS03b] Chris Duxbury, Juan Pablo Bello, Mike Davies, and Mark Sandler. Complex domain onset detection for musical signals. In *Proc. of the 6th Int. Conference on Digital Audio Effects (DAFx-03)*, pages 90–93, London, UK, September 8-11 2003.

[DG02] Manual Davy and Simon Godsill. Detection of abrupt spectral changes using support vector machines. an application to audio signal segmentation. In *IEEE ICASSP 2002*, pages 1313–1316, Orlando, USA, May 2002.

[DSS93] Harris Drucker, Robert E. Schapire, and Patrice Simard. Improving performance in neural networks using a boosting algorithm. In *Advances in Neural Information Processing Systems 5, [NIPS Conference]*, pages 42–49, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[FS95] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *European Conference on Computational Learning Theory*, pages 23–37, 1995.

[Fuk80] K. Fukushima. Neocognitron : A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36 :193–202, 1980.

[GKD+05] F. Gouyon, A. Klapuri, S. Dixon, M. Alonso, G. Tzanetakis, C. Uhle, and P. Cano. An experimental comparison of audio tempo induction algorithms, 2005.

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

[HSW89] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Netw.*, 2(5) :359–366, 1989.

[KEA06] A. Klapuri, A. Eronen, and J. Astola. Analysis of the meter of acoustic musical signals. *IEEE Trans. Speech and Audio Processing*, 14(1) :342–355, 2006.

[Kla99] Anssi Klapuri. Sound onset detection by applying psychoacoustic knowledge. In *IEEE Conference on Acoustics, Speech, and Signal Processing (ICASSP'99)*, volume 6, pages 3089–3092, Phoenix, Arizona, Mars 1999.

[KP04] Emir Kapanci and Avi Pfeffer. A hierarchical approach to onset detection. In *Proceedings of the International Computer Music Conference*, pages 438–441, Miami, Florida, 2004.

[LBD+90] Y. LeCun, B. Boser, J. S Denker, D. Henderson, R. E.. Howard, W. Howard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems II (Denver 1989)*, pages 396–404. Morgan Kaufmann, San Mateo, CA, 1990.

[LE06] A. Lacoste and D. Eck. A supervised classification algorithm for note onset detection. *EURASIP Journal on Applied Signal Processing*, 2006. Acceptï¿½avec modifications.

[LeC86] Y. LeCun. Learning processes in an asymmetric threshold network. In E. Bienenstock, F. Fogelman-Soulié, and G. Weisbuch, editors, *Disordered systems and biological organization*, pages 233–240, Les Houches, France, 1986. Springer-Verlag.

[LeC89] Y. LeCun. Generalization and network design strategies. In R. Pfeifer, Z. Schreter, F. Fogelman, and L. Steels, editors, *Connectionism in Perspective*, Zurich, Switzerland, 1989. Elsevier. an extended version was published as a technical report of the University of Toronto.

[LHB04] Yann LeCun, Fu-Jie Huang, and Leon Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of CVPR'04*, volume 2, pages 97–104. IEEE Press, July 2004.

[LMB+05] Y. LeCun, U. Muller, J. Ben, E. Cosatto, and B. Flepp. Off-road obstacle avoidance through end-to-end learning. In *Advances in Neural Information Processing Systems (NIPS 2005)*. MIT Press, 2005.

[LOL03] T. Li, M. Ogihara, and Q. Li. A comparative study on content-based music genre classification. In *Proceedings of the International ACM SIGIR Conference*

on Research and Development in Information Retrieval, pages 282–289, Toronto, Canada, 2003.

[Mar63] D. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *SIAM J. Appl. Math.*, 11 :431–441, 1963.

[MF05] C. McKay and I. Fujinaga. Automatic music classification and the importance of instrument identification. In *Proceedings of the Conference on Interdisciplinary Musicology (CIM05), Montreal, Canada*, 2005.

[MKP02] M. Marolt, A. Kavcic, and M. Privosnik. Neural networks for note onset detection in piano music. In *Proceedings of the International Computer Music Conference*, 2002. Unpublished.

[Moo95] B. Moore. *Handbook of Perception and Cognition*. Academic Press, 2nd edition, 1995.

[Moz87] Michael C. Mozer. *The perception of multiple objects : a parallel, distributed processing approach*. PhD thesis, Institute for Cognitive Science, University of California, San Disgo, 1987. Available as ICS Technical Report 8803.

[MP69] M. Minsky and S. Papert. Perceptrons. *MIT Press,Cambridge, MA*, 1969.

[OL94] Genevieve B. Orr and Todd K. Leen. Momentum and optimal stochastic search. In M.C. Mozer, P. Smolensky, D.S. Touretsky, J.L Elman, and A. S. Weigend, editors, *Prokceedings of the 1993 Connectionist Models Summer School*, pages 351–357. Erlbaum, 1994.

[PLR04] Laurent Daudet Pierre Leveau and Gael Richard. Methodology and tools for the evaluation of automatic onset detection algorithms in music. In *Proc. of the 5th International Conference on Music Information Retrieval (ISMIR)*, Barcelona, October 2004.

[SBK05] I. W. Selesnick, R. G. Baraniuk, and N. C. Kingsbury. The dual-tree complex wavelet transform. *Signal Processing Magazine, IEEE*, 22(6) :123–151, 2005.

[Sch98] E. Scheirer. Tempo and beat analysis of acoustic musical signals. *Journal of the Acoustical Society of America*, 103(1) :588–601, 1998.

[SVN37] S.S. Stevens, J. Volkmann, and E.B. Newmann. A scale for the measurement of a psychological magnitude : Pitch. *Journal of the Acoustical Society of America*, 8 :185–190, January 1937.

[TC02] G. Tzanetakis and P. Cook. Musical genre classification of audio signals. *IEEE Trans. on Speech and Audio Processing*, 10(5) :293–302, 2002.

[WHH+89] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang. Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech, and Signal processing*, 37 :328–339, 1989.