Master ATIAM
Acoustique, Traitement du signal, Informatique Appliqués à la Musique
Année 2013/2014

Université Pierre et Marie Curie

Mémoire de stage

Extraction d'information dans la musique pour l'automatisation de la séparation de la voix chantée

Vers des méthodes d'apprentissage profond

Auteur :
Simon Leglaive

Sous la direction de :
Romain Hennequin

Document confidentiel

Stage effectué du 3 mars au 14 août 2014

Audionamix
171 quai de Valmy, 75010 Paris

Résumé

L'ajout d'information extérieure permet en général d'améliorer fortement les performances des systèmes de séparation de sources, notamment pour une tâche d'extraction de la voix chantée dans un morceau de musique. Dans le cadre de certains modèles paramétriques de sources basés sur des représentations temps/fréquence, la mélodie vocale constitue le choix le plus pertinent pour guider le processus de séparation, car elle encode à la fois l'information fréquentielle et temporelle. Estimer la mélodie vocale c'est connaître la succession de notes portées par la voix, mais c'est avant tout savoir quand ces notes sont chantées. Les méthodes de l'état de l'art en estimation de mélodie relèguent souvent l'étape de détection de voix au second plan ; nous montrons qu'il est pourtant primordial de détecter la voix de façon robuste pour obtenir de bonnes performances globales. Cette information d'activité vocale étant essentielle pour l'estimation de mélodie, elle l'est a fortiori pour les systèmes de séparation de la voix chantée qui utilisent comme *a priori* la ligne mélodique ; et de façon générale elle peut être utilisée *a posteriori* dans n'importe quelle approche de séparation de la voix chantée de l'accompagnement.

Les méthodes de détection de la voix chantée se basent généralement sur l'élaboration de descripteurs très haut-niveau. Elles utilisent ensuite des modèles de classification à court-terme et de lissage temporel qui sont difficilement optimisables conjointement. Or l'aspect temporel est essentiel dans la musique, chaque vecteur de descripteur issu d'une analyse à court-terme ne peut être considéré indépendemment des vecteurs précédents et suivants. Nous proposons ici une nouvelle méthode de détection de la voix chantée basée sur une approche par apprentissage profond. Nous mettons en oeuvre un réseau de neurones récurrent bi-directionnel composé de blocs longue mémoire à court-terme. Pour chaque vecteur d'entrée, le système tient compte d'un contexte temporel passé et futur pour décider de l'absence ou de la présence de voix, la taille de ce contexte étant apprise automatiquement lors de la phase d'entraînement. Ce système de classification permet donc de prendre en compte l'aspect séquentiel inhérent à l'extraction de descripteurs à court-terme dans un morceau de musique. L'architecture étant composée de plusieurs couches cachées, le réseau est capable d'extraire à partir de descripteurs bas niveau une représentation simple et adaptée à notre tâche.

Les résultats que nous obtenons permettent d'améliorer significativement l'état de l'art de la détection de la voix chantée sur une base de données de référence. L'intégration de cette méthode dans un système d'estimation de mélodie permet également d'améliorer ses performances globales. Cette nouvelle méthode que nous proposons est en ce sens encourageante pour continuer l'exploration de l'apprentissage profond dans le domaine de l'extraction automatique d'information dans la musique.

Mots-clés : détection de la voix chantée, apprentissage profond, réseaux de neurones, estimation de la mélodie, séparation de sources

Abstract

Adding information is a common procedure that aims to improve the performance of a source separation system, particularly in the context of extracting singing voice in a piece of music. For some parametric models of sources, based on time/frequency representations, the singing melody is the most relevant choice to guide the source separation process. Indeed, the melody encodes both pitch information and temporal activations. Thus, estimating singing melody consists in identifying the succession of pitches carried by the voice, and above all, detecting when these notes are sung. State-of-the-art methods for melody estimation usually do not consider the importance of voicing detection, pushing this step of the process into the background. However, we show that a solid voicing detection is essential to achieve good global performance for melody estimation. Consequently, this information of vocal activity is also fundamental for singing voice separation systems that use the pitch line as prior; and more generally, it can also be used as posterior for any kind of approach to separate singing voice from accompaniment.

Singing voice detection techniques usually design high-level handcrafted features. They then make use of models for frame classification and temporal smoothing that cannot be easily optimized simultaneously. But temporal aspect is essential in music, each feature vector extracted by a short-term analysis cannot be considered as independent from the previous and the next vectors. We propose in this master's thesis a new method for singing voice detection, based on a deep learning approach. We build a bidirectional recurrent neural network with long short-term memory blocks. For each input vector, the system takes into account a past and future temporal context to decide between presence and absence of the voice. The length of this context is learned automatically during the training procedure. This classifier then uses the inherent sequential aspect of a short-term feature extraction in a piece of music. The network contains several hidden layers, so it is able to extract from low-level features a simple representation fitted to our task.

The results we obtain significantly outperform state-of-the-art algorithms on a common database. When this method is integrated in a melody estimation system, global performance is also improved. The new method we propose thus motivates future exploration of deep learning for other tasks in music information retrieval.

Keywords: singing voice detection, deep learning, neural networks, melody estimation, source separation

Table des matières

Liste des sigles et acronymes

BLSTM Bi-directionnel à Longue Mémoire à Court-Terme (*Bidirectionnal Long Short-Term Memory*). 46, 47, 49, 51–54, 56, 59–62, 70

BPTT rétropropagation dans le temps (*Backpropagation Through Time*). 42, 44, 46, 53

BRNN Réseau de Neurones Récurrent Bi-directionnel (*Bidirectional Recurrent Neural Network*). 44

CQT Transformation à Q-Constant (*Constant-Q Transform*). 7, 15, 17, 18, 47

EM Espérance-Maximisation (*Expectation-Maximisation*). 51

GMM Modèle de Mélange de Gaussiennes (*Gaussian Mixture Model*). 7, 24, 51

HA-LFPCs *Harmonic Attenuated* LFPC. 23

HMM Modèle de Markov Caché (*Hidden Markov Model*). 8, 9, 24, 41, 51

HPSS *Harmonic/Percussive Source Separation*. 6, 10, 15–19, 22, 24, 26, 52, 53, 58

LFPCs *Log Frequency Power Coefficients*. 23, 24

LPCs Coefficients de Prédiction Linéaire (*Linear Predictive Coefficients*). 23

LSTM Longue Mémoire à Court-Terme (*Long Short-Term Memory)*. 45–47, 51–55, 58

MAP Maximum A Posteriori. 7

MBR Machines de Boltzmann Restreintes. 48

MFCCs Coefficients Cepstraux à Echelle Mel (*Mel Frequency Cepstral Coefficients*). 8, 23–25, 47, 51, 56

MIDI *Musical Instrument Digital Interface*. 1

MIR *Music Information Retrieval*. 2, 3, 40, 47, 69

MIREX *Music Information Retrieval Evaluation eXchange*. 5, 6, 8, 10, 13, 19–22, 62, 69

MRFFT *Multi-Resolution Fast Fourier Transform*. 7

MSE Erreur Quadratique Moyenne (*Mean Squared Error*). 32

NMF Factorisation en Matrices Non-Négatives (*Non-negative Matrix Factorization*). 1–3, 9

PLCA Analyse Probabiliste en Composantes Latentes (*Probabilistic Latent Component Analysis*). 1

PLPs Coefficients de Prédiction Linéaire Perceptuels (*Perceptual Linear Predictive Coefficients*). 23, 24

RMSE Racine Carrée de l'Erreur Quadratique Moyenne (*Root Mean Squared Error*). 32

RNN Réseau de Neurones Récurrent (*Recurrent Neural Network*. 41, 42, 44–48

SGSMM *Smooth-Gaussian Scaled Mixture Model*. 9

SIMM *Smooth-Instantaneous Mixture Model*. 9

SSE Somme des Carrés des Résidus (*Sum of Squared Error*). 32

SVM Machine à Vecteurs de Support (*Support Vector Machine*). 9, 24, 47

TFCT Transformée de Fourier à Court Terme. 7, 9, 10, 15, 17, 21, 52

TFD Transformée de Fourier Discrète. 7, 10

ZCR *Zero Crossing Rate*. 23

Chapitre 1

Introduction

1.1 Cadre du stage

Ce stage de recherche s'est déroulé dans le cadre du Master Acoustique Traitement du Signal et Informatique Appliqués à la musique (ATIAM) de l'Université Pierre et Marie Curie. Il a été effectué chez Audionamix, une entreprise experte dans la séparation de sources sonores pour la musique, le cinéma et la télévision.

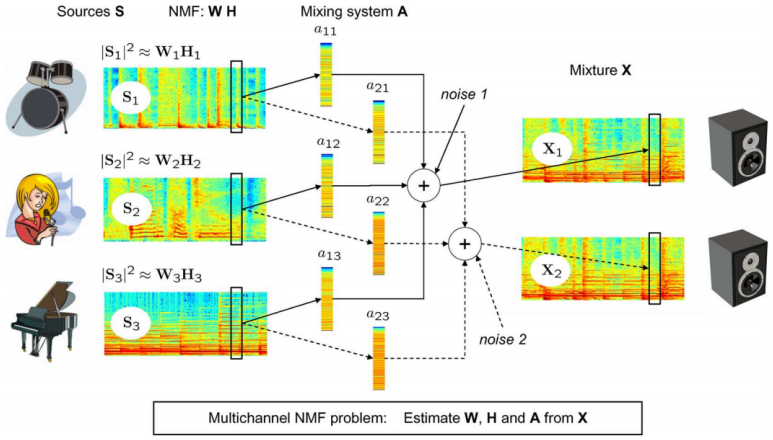
Les outils technologiques développés par l'équipe de Recherche et Développement située à Paris étaient jusqu'à présent destinés au bureau de production de Los Angeles exlusivement. Ces outils ont par exemple permis la ré-édition du film *Psycho* d'Alfred Hitchcok en son multicanal ou l'utilisation de la voix isolée d'Edith Piaf dans *Inception* de Christopher Nolan.

Aujourd'hui, l'expertise technique d'Audionamix s'ouvre aux professionnels du son et aux artistes grâce au logiciel ADX TRAX permettant d'isoler la voix chantée de l'accompagnement dans la musique. Ce stage porte sur l'automatisation de cette application afin de limiter l'effort utilisateur.

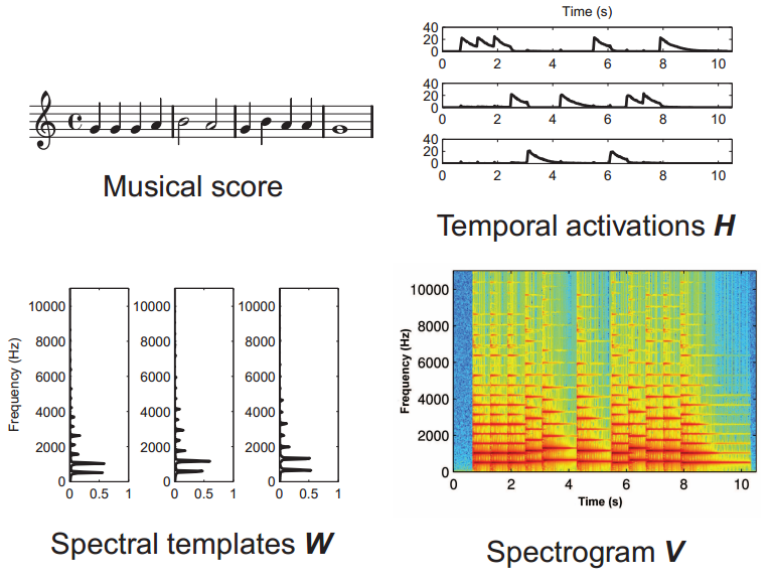
1.2 Contexte et problématique

Le problème de séparation de sources appliqué à la musique consiste à estimer les signaux associés aux différents instruments composant la pièce de musique à partir d'un ou plusieurs mélanges de ces signaux sources. Lorsque l'on dispose d'un nombre de mélanges inférieur au nombre de sources à séparer, le problème est dit sous-déterminé. La figure 1.1a illustre par exemple le problème de séparation de sources sous-déterminé (deux mélanges, 3 sources) en milieu réverbérant (canal acoustique modélisé par un filtrage), dans le cadre d'un modèle de sources par Factorisation en Matrices Non-Négatives (*Non-negative Matrix Factorization*) (NMF). Une approche courante est d'introduire des connaissances supplémentaires sur le mélange et/ou sur les sources à séparer, surtout lorsque le problème est sous-déterminé. On souhaite de cette façon guider le processus de séparation par l'ajout de contraintes sur la nature et le nombre de sources à séparer, sur leur localisation temporelle et fréquentielle, etc. C'est dans cette logique que certaines approches exploitent des modèles paramétriques de sources, permettant de contraindre les paramètres par l'ajout d'*a priori* sur les sources. On peut par exemple citer les modèles non-négatifs, utilisés par exemple dans le cadre de la NMF [1] pour décomposer des spectrogrammes en produit de matrices non-négatives et le modèle source/filtre [2] qui permet de modéliser la physique de production des sources. Dans le cas d'un modèle de sources par NMF comme illustré figure 1.1b, le spectrogramme de puissance ou d'amplitude $\mathbf{V}$ de chaque source est représenté par la factorisation de deux matrices non-négatives, une matrice d'atomes fréquentiels $\mathbf{W}$ et une matrice d'activations temporelles $\mathbf{H}$: $\mathbf{V} \approx \mathbf{WH}$. La contrainte de non-négativité de cette décomposition permet aux atomes extraits d'être généralement interprétés comme ayant un sens perceptif, ils peuvent par exemple correspondre aux notes de musique présentes dans le morceaux. Il est également possible d'utiliser des données extérieures pour guider la séparation, dans [3] une partition de musique (sous forme de fichier au format *Musical Instrument Digital Interface* (MIDI)) est utilisée afin d'initialiser la décomposition par NMF. Dans [4], l'approche se base sur une décomposition du mélange par Analyse Probabiliste en Composantes Latentes (*Probabilistic Latent Component Analysis*) (PLCA) où un *a priori* est introduit grâce à une imitation du signal source à séparer fournie par l'utilisateur. On peut également imaginer utiliser des informations telles que la nature des sources à séparer, le genre du locuteur pour la voix chantée, le style de la musique, permettant l'ajout de contraintes ou d'*a priori* concernant les paramètres des modèles utilisés.

Les travaux que nous allons présenter ici se placent dans le cadre d'ADX TRAX, une application logicielle destinée à séparer la voix chantée de l'accompagnement dans un morceau de musique, développée chez Audionamix. Contrairement au formalisme générale de la séparation de sources que nous avons présenté précédemment, on s'intéresse plutôt dans notre cas à une séparation de *stems*. C'est-à-dire que l'on cherche à séparer le signal de voix chantée avec les différents traitements qui lui ont été appliqués avant d'être mixé aux autres sources, ces



(a) Représentation du système de mélange convolutif sous-déterminé et formulation du problème par NMF, extrait de [1] avec l'autorisation des auteurs



(b) Illustration de la NMF, extraite des planches de présentation de [5] avec l'autorisation de l'auteur

FIGURE 1.1 – Séparation de sources

traitements peuvent donc correspondre à la propagation naturelle du signal dans le canal acoustique, mais aussi à des effets artificiels qui auraient été appliqués à la voix lors du mixage, par exemple un ajout de réverbération. Un utilisateur d'ADX TRAX peut guider manuellement le processus de séparation en indiquant l'évolution de la fréquence fondamentale de la voix chantée au cours du temps, permettant ainsi d'améliorer significativement les résultats de la séparation. L'ambition posée par ce stage est d'automatiser le processus de séparation par de l'information extraite via des méthodes d'analyse automatique de la musique, ceci afin de limiter l'effort de l'utilisateur.

Lorsque l'on travaille sur une représentation temps/fréquence du signal, il semble pertinent de chercher en premier lieu à caractériser la piste de voix selon ces deux dimensions. Ce qui nous amène à considérer deux tâches appartenant au domaine de l'extraction automatique d'information dans la musique (*Music Information Retrieval* (MIR)) : la détection automatique de voix chantée et l'extraction automatique de la mélodie vocale. La première tâche peut-être vue comme une sous-tâche de la seconde, on peut en effet considérer que l'absence de voix fait partie de la description mélodique que l'on recherche.

Dans le cas d'un modèle de sources par NMF, on comprend grâce à la figure 1.1b l'intérêt de connaître à quels instants la voix est présente, cette information étant totalement reliée à la matrice d'activations temporelles **H**. Bien que cela soit moins immédiat, le contenu mélodique que véhicule la voix est également une information pertinente qui peut-être reliée à la matrice **W**, si en plus certaines contraintes imposent à ses colonnes de correspondre effectivement à des notes. Finalement, comme ces deux matrices constituent les paramètres du modèle à estimer, il est possible de les contraindre afin qu'elles prennent en compte ces connaissances supplémentaires caractérisant le signal de voix chantée à séparer.

Si connaître la mélodie vocale paraît être intéressant pour informer la séparation dans les cas particuliers d'un modèle de source par NMF ou pour le modèle source/filtre, l'information de présence de voix est plus générale, dans le sens où elle est indépendante du modèle choisi. Elle pourra être utilisée *a priori* pour guider le processus, mais également *a posteriori* pour, par exemple, s'assurer que le signal de voix soit nul si celle-ci est estimée absente, c'est ce qui est fait dans [6]. Détecter la présence de voix chantée semble donc être une tâche primordiale et non spécifique à l'algorithme mis en place. C'est une information simple mais également puissante et générale.

De la même façon, l'information de présence de voix est essentielle à un système d'estimation de la mélodie vocale. Il s'agit en effet d'estimer correctement la ligne mélodique, liée à la source principale, mais cette mélodie n'est définie que lorsque la voix est présente et de façon complémentaire, il ne faut pas estimer de fréquences fondamentales quand la voix est absente. Nous proposons dans ce rapport de mettre en évidence la nécessité d'une détection de voix robuste pour obtenir de bonnes performances globales d'estimation de mélodie. Pour ce faire, nous mettons en place une nouvelle méthode de détection de la voix chantée : Des descripteurs bas-niveau sont extraits après une étape de pré-traitement par double séparation harmonique/percussif et un réseau de neurones récurrent comprenant des blocs longue mémoire à court-terme permet d'effectuer la classification en présence/absence de voix. Ces architectures de réseaux font partie du domaine de l'apprentissage profond (*Deep Learning*) qui suscite depuis quelques années un grand intérêt en traitement de l'information, en traitement du signal ou en vision par ordinateur (voir une revue récente de l'apprentissage profond et de ses applications dans [7]). Cet engouement est dû au fait que ces méthodes présentent des résultats compétitifs avec l'état de l'art sur de nombreuses tâches. L'apprentissage profond est un domaine de recherche actif, où beaucoup de travail reste à effectuer, notamment sur le plan théorique. Il n'est encore que peu ancré dans le domaine de l'extraction automatique d'information dans la musique. Nous allons voir que la méthode de détection de la voix chantée mise en place dans ce rapport dépasse significativement les résultats de l'état de l'art et permet d'améliorer les performances globales d'une méthode d'estimation de mélodie. Ces résultats sont permettent de motiver de futurs travaux liés au domaine du MIR utilisant des approches similaires, en particulier pour l'estimation automatique de mélodie.

Les chapitres 2 et 3 permettent de dresser l'état de l'art de l'estimation automatique de mélodie et de la détection de la voix chantée. Dans le chapitre 2, nous décrivons deux méthodes de référence que nous avons implémentées. Les résultats de l'évaluation de celles-ci montrent l'importance de la détection de voix pour obtenir de bonnes performances globales. Dans le chapitre 4, nous détaillons le fonctionnement et une technique d'apprentissage des réseaux de neurones dans le cadre d'un système de classification. Nous présentons les architectures neuronales sans et avec connexions récurrentes et nous introduisons l'utilisation des blocs longue mémoire à court-terme. Finalement, dans le chapitre 5, nous décrivons et évaluons la nouvelle méthode de détection de la voix chantée que nous avons mise en place et permettant d'améliorer l'état de l'art.

Chapitre 2

Etat de l'art de l'estimation automatique de mélodie

Dans ce chapitre nous allons décrire l'état de l'art pour la tâche d'estimation automatique de mélodie. Les méthodes que nous allons présenter ici ont toutes participé au cours des dix dernières années au concours annuel *Music Information Retrieval Evaluation eXchange* (MIREX) dans la catégorie *Audio Melody Extraction*. Ce concours sert de référence pour de multiples applications liées à l'extraction automatique d'information dans la musique car il permet de comparer les différents algorithmes sur un ensemble de critères d'évaluation communs et pour des bases de tests identiques [8]. Grâce au cadre commun posé par le concours MIREX, il est possible de comparer objectivement les algorithmes mais également de dégager les grands axes qui regroupent ou à l'inverse différencient ces méthodes, sur ce principe de comparaison on pourra notamment se référer à [9].

Deux grandes catégories de méthodes d'extraction de mélodie existent, celles qui se basent sur le calcul d'une fonction de saillance, c'est-à-dire une représentation du signal dans le plan temps/fréquence qui met en avant les fréquences fondamentales et leurs harmoniques, permettant ainsi d'extraire l'ensemble des notes candidates à la mélodie. La seconde approche met en œuvre des techniques de séparation de sources afin d'extraire en premier lieu un signal où la source portant la mélodie est isolée.

Parmi ces approches, nous avons pu relever deux algorithmes présentant de bons résultats, nous les avons implémentés afin d'obtenir deux méthodes de référence. Nous les décrivons dans ce chapitre et nous présentons les résultats de leur évaluation sur un banc d'essai que nous avons mis en place. Nous évaluons également la méthode actuellement utilisée dans ADX TRAX pour la comparer aux deux méthodes de référence.

A partir de cette évaluation, nous avons pu dégager les pistes de travail pour ce stage, notamment en définissant quel type d'information permettrait de faire le plus grand pas vers une automatisation de la séparation effectuée dans ADX TRAX. Il faut donc voir ce chapitre d'état de l'art de l'estimation automatique de mélodie comme l'étape nous ayant permis de poser le problème d'automatisation auquel nous souhaitons faire face.

2.1 Position du problème d'estimation de mélodie

L'estimation automatique de mélodie peut être définie dans un premier temps comme la tâche permettant d'obtenir automatiquement une séquence de valeurs de fréquences fondamentales représentant les notes de la mélodie principale dans un morceau de musique. D'un point de vue musicologique, il n'est pas évident de définir ce qu'est la mélodie principale d'une pièce de musique. Si dans la musique populaire il est fréquent d'avoir un instrument ou une voix prédominante portant cette mélodie principale, ce n'est en revanche pas le cas pour un très grand nombre d'autres styles de musique où plusieurs mélodies, portées par plusieurs instruments, se superposent. Dans un tel cas de figure, il y a de grandes chances pour qu'un ensemble d'auditeurs identifie de façon différente la mélodie du morceau, du fait du caractère très subjectif lié à celle-ci. Il nous est alors nécessaire de se soustraire au maximum de cet aspect subjectif afin de définir une tâche plus restreinte et donc comportant certaines hypothèses. Ceci amène à considérer la mélodie principale comme appartenant à une unique source. De façon à lever l'ambiguïté concernant cette source, il est nécessaire de traiter des pièces de musique où est clairement défini une voix ou un instrument principal. Dans le contexte de ce stage, nous nous intéressons à la mélodie portée par la voix chantée.

Ces considérations peuvent être résumées par la définition donnée dans [9] : l'extraction automatique de mélodie consiste à estimer la séquence de fréquences fondamentales prédominantes correspondant à une unique source à partir de l'enregistrement d'un morceau de musique polyphonique comprenant une voix ou un instrument principal. Le terme polyphonique se référant au cas où plusieurs notes sont jouées simultanément. Estimer la mélodie principale est donc une tâche proche de l'estimation de fréquences fondamentales multiples, car plusieurs fréquences fondamentales sont également concurrentes à chaque instant, mais il ne s'agit pas ici

d'identifier correctement toutes ces notes à chaque instant, il faut seulement estimer la séquence correspondant à la mélodie.

Pour estimer la mélodie correctement, il est aussi nécessaire de détecter les instants pour lesquels la voix ou l'instrument principal est présent (ou absent). Ceci amène donc à considérer une sous-tâche au problème qui est d'estimer quand la mélodie est présente ou absente. Cette partie de l'algorithme s'appelle détection de voix, bien que cela réfère indifféremment à la détection de la voix chantée ou d'un instrument. Les performances concernant la détection de voix s'avèrent très importantes pour obtenir une bonne estimation de mélodie, c'est donc tout une partie du processus de traitement sur laquelle il est possible de travailler pour améliorer un système d'extraction de mélodie. En effet, s'il est possible de considérer le problème de détection de voix comme indépendant de l'estimation de mélodie, l'inverse ne l'est pas. Comme nous le verrons, certaine méthodes détectent la présence de mélodie implicitement lorsqu'ils estiment cette dernière, et d'autres séparent totalement les deux tâches.

2.2 Approches générales

Le problème d'estimation de mélodie tel que présenté dans [9] se base sur une analyse des différentes méthodes présentées au concours MIREX depuis 2005. Les algorithmes considérés peuvent être regroupés en quelques catégories principales, définissant alors les approches envisageables pour aborder ce problème. Cette présentation assez globale que nous allons détailler ici permet de bien dégager les différents axes empruntés par le grand nombre de méthodes qui composent la littérature dans ce domaine, et elle constitue donc une bonne façon d'appréhender dans un premier temps ce problème.

2.2.1 Méthodes basées saillance

Les méthodes qui se basent sur le calcul d'une fonction de saillance sont les plus nombreuses. Elle peuvent se décomposer de manière générale en plusieurs blocs de traitement successifs, comme illustré figure 2.1. Nous allons détailler ici ces différentes étapes.

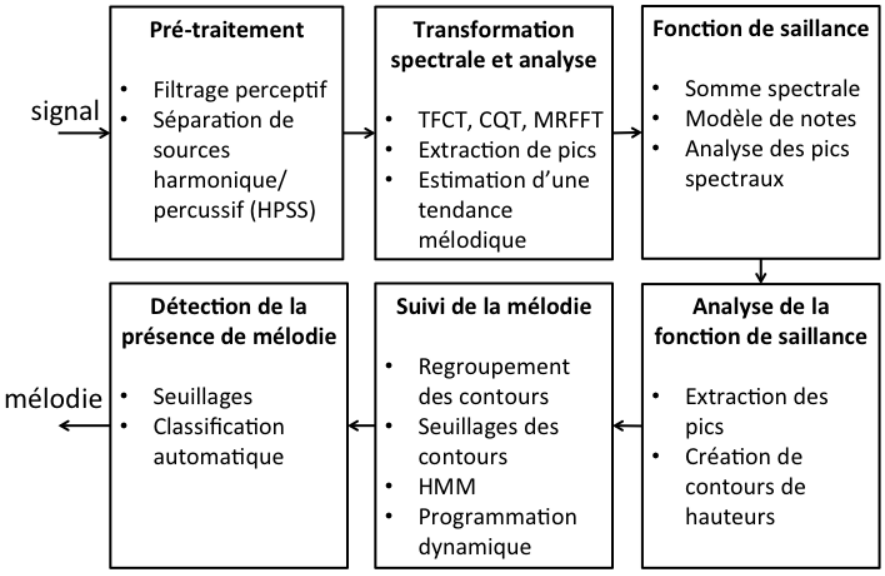


FIGURE 2.1 – Méthodes basées saillance

Pré-traitement :

Dans [10], les auteurs effectuent en premier lieu un filtrage basé sur une approche perceptive, le filtre est en effet construit selon l'inverse des courbes d'isosonie [11], afin de renforcer les fréquences auxquelles l'oreille est plus sensible. Pour garder une certaine indépendance vis-à-vis du niveau du signal d'entrée, le filtre correspond à l'inverse de la moyenne des courbes d'isosonie. D'autres méthodes [12, 13] utilisent une technique de séparation de sources (*Harmonic/Percussive Source Separation* (HPSS) [14]) afin d'obtenir un signal pour lequel la mélodie est mise en avant.

Transformation spectrale et analyse :

La majorité des méthodes de détection de mélodie travaillent sur une représentation temps/fréquence du signal, telle que la Transformée de Fourier à Court Terme (TFCT) :

$$X(l,k) = \sum_{n=0}^{M-1} x(n+lH)w(n)e^{-j2\pi\frac{kn}{N}} \tag{2.2.1}$$

$$l = 0, 1, ... \text{ et } k = 0, 1, ..., N-1$$

où $x(n)$ est le signal dans le domaine temporel, $w(n)$ la fenêtre d'analyse, l l'indice de trame, M la longueur de la fenêtre en échantillons, N est le nombre de points de la Transformée de Fourier Discrète (TFD) et H l'incrément entre deux trames en nombre d'échantillons.

Après cette transformation, il est commun d'effectuer une sélection des pics spectraux dans le plan temps/-fréquence. Si la TFCT est la transformation la plus usuelle [15, 16, 10], ses limitations en terme de précision et de résolution sont un problème pour estimer précisément la mélodie. C'est en ce sens que certaines méthodes effectuant une détection des pics spectraux estiment ensuite les fréquences instantanées associées à ces derniers. Pour cela, les méthodes utilisent le principe du vocodeur de phase et la relation différentielle qui lie phase et fréquence instantanée [17, 10]. D'autres méthodes se basent sur des transformations à multi-résolution pour contrer les limitations de la TFCT, elles utilisent notamment une *Multi-Resolution Fast Fourier Transform* (MRFFT) [17, 12, 13] ou une Transformation à Q-Constant (*Constant-Q Transform*) (CQT) [18]. En basse-fréquence, il est en effet nécessaire d'avoir une meilleure résolution fréquentielle qu'en haute-fréquence, car les interférences entre harmoniques correspondant à différentes notes jouées simultanément sont plus nombreuses.

Si pour l'étape d'analyse, la plupart des méthodes effectuent uniquement une extraction des pics spectraux, éventuellement suivie d'un filtrages de ces derniers, d'autres cherchent à extraire plus d'information directement à partir de la représentation temps/fréquence calculée. Hsu et al. [13] estiment par exemple une zone réduite à un octave évoluant dans le temps, dans laquelle la mélodie devant être estimée a de grandes chances de se trouver. Cette zone sera ensuite utilisée pour tronquer la fonction de saillance.

Fonction de saillance :

La détection des pics spectraux permet de connaître pour chaque trame les fréquences associées à la présence d'énergie. Il est alors nécessaire de savoir parmi celles-ci, lesquelles constitueraient de bons candidats pour appartenir à la mélodie. Pour cela, on définit la saillance de chaque fréquence fondamentale, c'est-à-dire sa propension à appartenir à la mélodie dans la trame considérée. La fonction de saillance correspond ainsi à une représentation à court-terme de la saillance des multiples fréquences fondamentales envisageables pour appartenir à la mélodie. Les pics de cette fonction de saillance sont ensuite généralement extraits et constituent finalement l'ensemble des candidats définitifs pour appartenir à la mélodie.

La majorité des méthodes se basent sur le calcul d'une somme spectrale afin de définir la fonction de saillance [10, 16, 13, 12]. Pour chaque pic fréquentiel dans une trame, un calcul de la somme pondérée des amplitudes de ses harmoniques permet de définir la saillance associée à cette fréquence. Goto [19] et Marolt [15] ont une approche différente, ils apprennent un ensemble de modèles de notes grâce au spectre, puis l'estimation du Maximum A Posteriori (MAP) d'un modèle dont la fréquence fondamentale correspond à une note candidate est considérée comme la saillance associée à cette fréquence. Dressler [17] effectue une analyse par paire de chaque pic fréquentiel afin de calculer la fonction de saillance.

Suivi :

Une fois que l'on dispose des pics de la fonction de saillance, il faut déterminer lesquels appartiennent effectivement à la mélodie.

Il est alors possible d'effectuer directement un suivi de la mélodie à partir de l'extraction des pics de la fonction de saillance, ou bien, on peut regrouper ces pics au cours du temps afin de créer préalablement un ensemble de trajectoires dans le plan temps/fréquence [10, 15, 19, 17]. Dans ce cas, c'est finalement à partir de ces trajectoires de hauteurs que la mélodie est extraite. Afin de regrouper les pics spectraux, il est commun d'utiliser des contraintes de continuité et des descripteurs basés sur les trois dimensions temps/fréquence/saillance.

Afin d'obtenir la mélodie finale, plusieurs techniques de suivi sont envisagées, que ce soit au niveau des pics de la fonction de saillance ou au niveau des trajectoires construites préalablement. Marolt [15] utilise des Modèles de Mélange de Gaussiennes (*Gaussian Mixture Models*) (GMMs) afin de regrouper les fragments mélodiques. Un groupe formé de plusieurs fragments est décrit par une loi normale multidimensionnelle apprise par un GMM à partir de quatre descripteurs caractérisant les différents fragments mélodiques. Goto [19] et Dressler [17] définissent un ensemble d'heuristiques qui permet à des agents d'effectuer un suivi des pics de la fonction de saillance afin d'extraire la mélodie. Salamon et Gómez [10] procèdent quant à eux par élimination

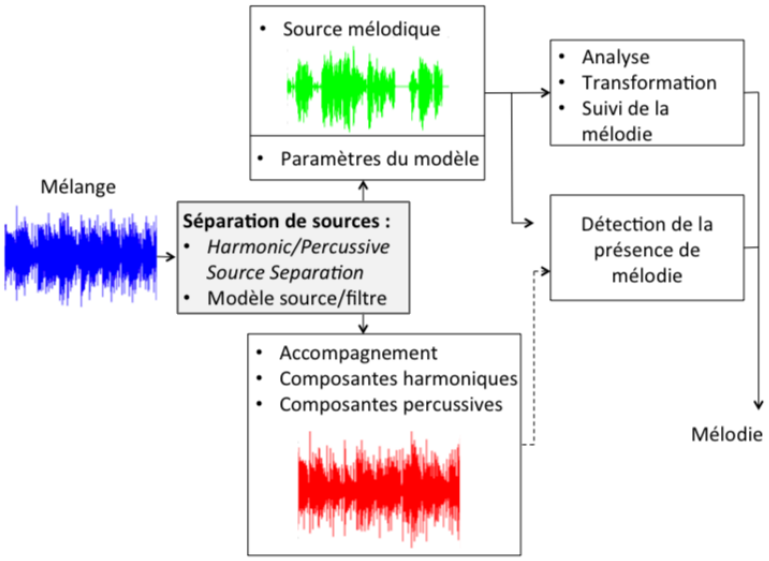


FIGURE 2.2 – Méthodes basées séparation de sources

des trajectoires de hauteurs qui n'appartiennent pas à la mélodie, ceci par une succession de seuillages basés sur des caractéristiques de ces trajectoires dans le plan temps/fréquence. Hsu [13] estime la séquence de notes la plus vraisemblable par programmation dynamique en ayant préalablement calculé un ensemble de descripteurs qui correspondent à "l'énergie aux demi-tons d'intérêt". Yeh [12] met en place une approche hybride, c'est-à-dire que trois estimations sont combinées (valeur médiane des trois mélodies estimées) pour pallier mutuellement les inconvénients de chacune. Les deux premières estimations correspondent à une passe avant et une passe arrière (axe temporel inversé) de la méthode d'Hsu [13], la dernière estimation correspond à un suivi de la mélodie basé sur un Modèle de Markov Caché (*Hidden Markov Model*) (HMM) dont les états correspondent aux différents *bins* fréquentiels. Les observations considérées pour ce HMM sont les amplitudes des deux pics spectraux les plus saillants dans chaque trame, ainsi que leurs positions fréquentielles. La séquence optimale est alors décodée par l'algorithme de Viterbi.

Détection de la présence de la mélodie :

Le plus souvent, la dernière étape pour la tâche d'extraction automatique de mélodie correspond à la détection de la présence ou de l'absence de la mélodie. Cette étape influencera fortement les résultats généraux, car il s'agit effectivement d'estimer correctement la mélodie, mais cela implique de savoir quand elle est présente. Une bonne détection de voix est donc essentielle, cependant elle est souvent reléguée au second plan dans les méthodes de l'état de l'art. Hsu [13] met en place un réseau de neurone à une couche cachée afin d'effectuer la classification présence/absence de mélodie, 39 Coefficients Cepstraux à Echelle Mel (*Mel Frequency Cepstral Coefficients*) (MFCCs) (13 coefficients et leurs dérivées première et seconde) étant extraits pour chaque trame. Marolt [15] et Dressler [17] effectuent un seuillage par trame basé sur l'énergie dans la fonction de saillance. Salamon et Gómez [10] ont une approche particulière car la détection de voix fait partie intégrante du système d'estimation de mélodie. Ils effectuent un seuillage sur les contours de hauteurs préalablement construits, leur élimination dans les zones où la mélodie est absente se base sur la distribution de la saillance moyenne des contours.

2.2.2 Méthodes basées séparation de sources

Les méthodes d'extraction automatique de mélodie par séparation de sources sont les plus récentes et donc les moins nombreuses. Les deux à avoir participé au concours MIREX sont la méthode de Durrieu et al. [20] et celle de Tachibana et al. [14]. Ces méthodes cherchent dans un premier temps à isoler de façon plus ou moins précise la source sonore correspondant à l'instrument principal. C'est sur ce signal qu'est ensuite estimée la mélodie, comme illustré figure 2.2. Il est rare que la séparation soit parfaite, néanmoins ces méthodes permettent d'obtenir un signal dans lequel la source considérée pour l'extraction de la mélodie est mise en avant, et l'accompagnement est quant à lui fortement atténué. C'est notamment cette caractéristique de mise en relief de la source portant la mélodie principale qui a conduit les auteurs dans [13, 12] à utiliser la méthode proposée par Tachibana [14] comme pré-traitement. Nous détaillerons plus amplement cette technique qui fait partie de nos

méthodes de référence dans la section 2.3.2.

La méthode de Durrieu exploite des modèles de spectrogrammes par NMF tels que nous avons pu les introduire brièvement section 1.2. Le spectrogramme de puissance du signal est décomposé comme la somme instantanée de deux contributions (deux matrices), la première liée au signal de l'instrument principal et la seconde à l'accompagnement : $\mathbf{S^X} = \mathbf{S^V} + \mathbf{S^M}$.

La contribution de l'instrument portant la mélodie principale est ensuite représentée par un modèle source/-filtre, permettant dans le cas de la voix chantée, de représenter le fait que le signal soit produit par une excitation harmonique, caractérisée par sa fréquence fondamentale, et que celle-ci soit ensuite filtrée par le conduit vocal. Le modèle ainsi utilisé ne permet pas de prendre en compte les sons non-voisés, où l'excitation n'est plus harmonique mais correspond à du bruit. Seuls les voyelles peuvent donc être représentées par cette approche, et non les consonnes. Ce modèle peut être étendu aux instruments de musique pour lesquels la partie filtre est interprétée comme caractérisant le timbre de l'instrument et la partie source reste une excitation harmonique.

Le modèle source/filtre s'écrit de la façon suivante en décomposant le spectrogramme d'amplitude ou de puissance associé à l'instrument porteur de la mélodie, où $\odot$ représente le produit terme à terme :

$$\mathbf{S^V} = \mathbf{S^\Phi} \odot \mathbf{S^{F_0}} = (\mathbf{W^\Phi H^\Phi}) \odot (\mathbf{W^{F_0} H^{F_0}})$$

La matrice $\mathbf{W^\Phi}$ contient les atomes fréquentiels associés à des filtres contraints à être lisses en fréquence, $\mathbf{H^\Phi}$ est la matrice d'activations temporelles de ces atomes. La matrice $\mathbf{W^{F_0}}$ contient les peignes harmoniques associés à chaque note envisagées pour la mélodie, $\mathbf{H^{F_0}}$ est la matrice d'activations temporelles associée à ces atomes.

Deux modèles sources/filtre sont considérés, le premier, *Smooth-Instantaneous Mixture Model* (SIMM), correspond exactement au modèle décrit ci-dessus, c'est à dire qu'à chaque instant, la source principale est modélisée par un mélange de l'ensemble des différentes notes sources possibles, représentées par les colonnes de $\mathbf{W^{F_0}}$, le mélange et les activations de ces atomes étant caractérisés par $\mathbf{H^{F_0}}$. Le second modèle, *Smooth-Gaussian Scaled Mixture Model* (SGSMM), est plus réaliste dans le sens où il considère qu'à un instant donné, l'instrument principal est représenté par un unique couple source-filtre actif (k, u) où k est l'indice des colonnes de $\mathbf{W^\Phi}$ et u celui des colonnes de $\mathbf{W^{F_0}}$. Cet caractéristique peut être vue comme une contrainte imposée sur les matrices d'activations $\mathbf{H^{F_0}}$ et $\mathbf{H^\Phi}$.

Afin d'extraire la mélodie, Durrieu définit un HMM dont les états cachés u correspondent à l'ensemble des notes possibles de la source, soit les différentes colonnes de $\mathbf{W^{F_0}}$. Le signal observé est la TFCT du signal de départ, soit $\mathbf{S^X}$. La probabilité a posteriori de l'état de la source associée à l'instrument principal est directement estimée, pour chaque trame, à partir des modèles définis et de leurs paramètres. Le suivi se fait ensuite par l'algorithme de Viterbi.

La détection de voix se fait dans le cas du SGSMM en augmentant le modèle source/filtre d'un état "silence", c'est-à-dire en autorisant le modèle à représenter une trame du spectrogramme de l'instrument principal comme nulle. Cette représentation n'est pas possible pour le modèle SIMM comme la source principale est modélisée à chaque instant par un mélange de l'ensemble des différentes notes sources possibles et non par un unique couple source/filtre. De ce fait, la détection de présence de mélodie se fait par un seuillage trame par trame sur l'énergie du signal associé à l'instrument principal.

2.2.3 Autres approches

Bien que les approches par calcul d'une fonction de saillance ou par séparation de sources soient les plus nombreuses, d'autres stratégies sont également envisagées dans la littérature.

Dans [21], les auteurs ont une approche par apprentissage statistique, ils utilisent en effet des Machines à Vecteurs de Support (*Support Vector Machines*) (SVMs) pour estimer la mélodie sans ou avec réduction à l'octave. Dans le premier cas, ils entraînent un unique SVM afin d'effectuer un classification selon 60 notes MIDI, c'est donc un problème à 60 classes. Dans le cas où la mélodie est réduite à l'octave, ils entraînent 12 SVMs dans le cadre d'une classification binaire. A partir de ces SVMs ils déduisent une estimation de la probabilité a posteriori de chaque classe ce qui permet de construire un *posteriorgram*, c'est-à-dire une représentation à court-terme où pour une trame de signal est représenté l'ensemble des probabilités a posteriori pour les 12 notes de l'octave. Le vecteur d'entrée utilisé pour la classification correspond aux amplitudes associées à 256 *bins* fréquentiels (entre 0 et 2kHz) issus du calcul de la TFCT, la classification se fait donc trame par trame. Plusieurs stratégies de normalisation des descripteurs sont étudiées, celle qui présente les meilleurs résultats correspond à une normalisation de la TFCT pour chaque trame de telle sorte à centrer et réduire les amplitudes sur une fenêtre temporelle de 51 trames, permettant ainsi de réduire l'influence de la diversité des signaux entre la base d'apprentissage et la base de test.

L'approche mise en place dans [22] combine grâce à un HMM deux estimateurs monophoniques de mélodie, c'est-à-dire deux systèmes de détection de fréquence fondamentale pour des signaux où une seule note est

présente à un même instant. Le premier estimateur correspond à la méthode *Two-Way Mismatch* [23]. Le second applique une méthode temporelle de détection de fréquence fondamentale, par l'analyse de la fonction d'auto-corrélation sur des fenêtres successives de 50ms de signal. L'idée est qu'en combinant ces deux estimation de mélodie, le résultat sera plus précis, même sur un signal pour lequel plusieurs instruments jouent en même temps.

2.3 Méthodes de référence

Dans cette partie nous allons présenter deux méthodes que nous avons implémentées comme référence. Nous avons considéré plusieurs critères afin de les sélectionner, notamment des critères spécifiques à l'application que nous souhaitons faire de l'extraction de la mélodie.

Premièrement, nous nous attachons à extraire une mélodie associée à la voix chantée. Certaines méthodes, comme celle de Dressler [17] présentent par exemple une certaine capacité à extraire la mélodie avec de bons résultats, tant pour un instrument que pour la voix chantée, ce qui dans notre cas n'est pas le plus pertinent.

L'évaluation d'une méthode d'extraction de la mélodie porte généralement sur deux aspects considérés dans un premier temps indépendamment : une détection précise de la présence de voix et une estimation fidèle de la mélodie. Et bien sûr, afin d'évaluer la méthode dans sa globalité, l'intersection de ces deux mesures est également à prendre en compte.

Dans notre cas, il est nécessaire de prêter attention à un autre aspect important, le temps de calcul. En effet on veut que cette extraction de mélodie se fasse rapidement, car elle sert de pré-traitement à la séparation de sources et a pour objectif à terme d'être implantée dans une application de séparation automatique de la voix chantée et de l'accompagnement.

Finalement, en comparant les différentes méthodes proposées ainsi que leurs résultats dans le cadre du concours MIREX, nous avons pu identifier deux algorithmes à implémenter et évaluer : la méthode de Salamon et Gómez [10] qui est basée saillance, et la méthode de Tachibana et al. [14] qui utilise la technique de séparation de sources HPSS pour effectuer ensuite un suivi de mélodie dans le signal résultant pour lequel la mélodie vocale est mise en avant.

2.3.1 Salamon et Gómez

La méthode de Salamon et Gómez [10] présente actuellement les meilleurs résultats au concours MIREX. Il est intéressant de noter que cet algorithme a été présenté en 2010 et 2011 ; en 2010 ses performances étaient inférieures à la méthode de Dressler [17], mais en 2011 il a surpassé tous les autres algorithmes, ceci grâce à une optimisation poussée des représentations et paramètres mis en jeu [24]. Cette méthode peut se décomposer en plusieurs étapes que nous allons détailler.

2.3.1.1 Extraction des sinusoïdes

En premier lieu, les auteurs effectuent une extraction des sinusoïdes. Cette extraction se déroule en trois étapes :

- Un filtrage perceptif du signal selon l'inverse des courbes d'isosonie [11] permet de mettre en avant les fréquences auxquelles l'oreille est plus sensible et en atténuant celles où elle ne l'est pas. Ces courbes décrivent la sensibilité de l'oreille par rapport au niveau sonore selon la fréquence. Pour être indépendant du niveau du signal en entrée, la forme du filtre est estimée en effectuant une moyenne des courbes d'isosonie pour différentes intensités. Le gain est unité à 3kHz. Ce pré-traitement est implémenté d'après [25] grâce à un filtre à réponse impulsionnelle infinie d'ordre 10 cascadé avec un filtre de Butterworth passe-haut d'ordre 2.

- Il s'agit ensuite de passer à une représentation temps/fréquence du signal. Cette représentation est calculée à l'aide de la TFCT (équation (2.2.1)). Etant donnée la TFD d'une trame de signal $X(l,k)$, les pics spectraux p_i sont ensuite extraits en estimant les maxima locaux k_i dans le module du spectre $|X(l,k)|$.

- La précision en ce qui concerne la position des pics spectraux étant limitée par les paramètres de la TFCT (taille de la fenêtre temporelle d'analyse, nombre de points de la TFD), la fréquence instantanée des pics et leur amplitude sont estimés finement à partir du spectre de phase $\Phi(l,k)$ en utilisant la relation différentielle qui lie phase et fréquence instantanée, selon le principe du vocodeur de phase.

2.3.1.2 Calcul de la fonction de saillance

Les pics extraits à partir de la TFCT sont utilisés pour estimer la fonction de saillance. Les pics de cette fonction constitueront l'ensemble des fréquences candidates à la mélodie. Le calcul de la fonction de saillance se base

sur une somme spectrale, où la saillance d'un pic correspond à la somme pondérée des énergies des pics aux multiples de sa fréquence.

La fonction de saillance couvre une plage de fréquence allant de 55Hz à 1.76kHz, échantillonnée sur 600 *bins* à partir d'une échelle en cents où 1 bins représente 10 cents soit un dixième de demi-ton. La correspondance fréquence/*bin* se fait par :

$$B(f) = \lfloor \frac{1200 log_2(\frac{f}{55})}{10} + 1 \rfloor$$

Pour chaque trame, la fonction de saillance $S(b)$ est calculée à partir des pics p_i caractérisés par leur fréquence f_i et leur amplitude a_i, où $i = 1...I$ est l'indice du pic dans la trame.

Les pics d'énergie trop faibles ne sont pas pris en compte dans le calcul de la fonction de saillance, ceux-ci sont éliminés trame par trame si leur amplitude est inférieure de γ dB par rapport au pic d'amplitude maximale a_M dans la trame considérée. Ce seuillage est défini par la fonction :

$$e(a_i) = \begin{cases} 1 & \text{si } 20 log_{10}(\frac{a_M}{a_i}) < \gamma \\ 0 & \text{sinon} \end{cases}$$

Chaque pic p_i, lorsqu'il est supposé correspondre à l'harmonique h du *bin* b, est pondéré par la fonction :

$$g(b, h, f_i) = \begin{cases} cos^2(\delta \frac{\pi}{2}) \alpha^{h-1} & \text{si } \delta \leq 1 \\ 0 & \text{si } \delta > 1 \end{cases}$$

où $\delta = \frac{|B(\frac{f_i}{h}) - b|}{10}$ est une distance en demi-ton. Si δ est inférieur à un demi-ton, alors la fréquence f_i est effectivement considérée comme l'harmonique h du *bin* b. Du fait de cette tolérance d'un demi-ton, il est possible qu'une fréquence corresponde à la même harmonique pour plusieurs *bins*, ce qui permet notamment de prendre en compte l'inharmonicité. Cependant, la pondération favorisera le fait que f_i soit proche de la fréquence exacte correspondant à l'harmonique h du *bin* b. α est un coefficient de pondération pris dans l'intervalle $[0, 1]$ qui réduit l'influence des harmoniques d'ordre élevé dans le calcul de la fonction de saillance.

Ces considération amènent à la fonction de saillance suivante :

$$S(b) = \sum_{h=1}^{N_h} \sum_{i=1}^{I} e(a_i) g(b, h, f_i) (a_i)^{\beta}$$

où β correspond à un facteur de compression sur l'amplitude du pic.

Il est intéressant de noter que plusieurs paramètres sont mis en jeu dans le calcul de la fonction de saillance : α, β, γ, N_h. Ces paramètres ont une influence certaine sur les résultats d'extraction de mélodie, c'est pour cette raison qu'ils ont fait l'objet d'une optimisation dans [24]. Les paramètres induisant les meilleurs résultats, sur la base de donnée sur laquelle ils ont été optimisés, sont : $\alpha = 0.8$, $\beta = 1$ et $\gamma = 40$ dB, $N_h = 20$ harmoniques.

Figure 2.3, on peut observer le résultat du calcul de la fonction de saillance sur un extrait de morceau de jazz vocal. On repère clairement la mélodie vocale centrée autour du *bin* 300, mais on remarque également que les *bins* en rapport harmoniques sont très présents. Ce phénomène, directement dû au calcul de la fonction de saillance tel que nous venons de le présenter, induit des erreurs communes à l'estimation de mélodie, ce sont les erreurs d'octave, où la fréquence estimée à un instant donné pour faire partie de la mélodie ne correspond pas à la note jouée mais à l'octave inférieur ou supérieur. Ce type de confusion est d'autant plus fréquent pour certains styles de musique où la mélodie est portée par la voix chantée, comme l'opéra. En effet, le chanteur ou la chanteuse utilise le formant du chanteur. Cet effet correspond à un renforcement énergétique dans le spectre autour de 3kHz, zone fréquentielle la plus sensible de l'oreille. Il permet au chanteur ou à la chanteuse de passer par dessus l'orchestre. Cette énergie localisée dans le spectre pourra amener l'algorithme à estimer la mélodie à un octave supérieur. En effet, la pondération des harmoniques prises en compte dans la fonction de saillance décroit avec l'ordre h de l'harmonique (coefficient de pondération α^{h-1} avec $\alpha \in [0, 1]$), l'énergie présente dans la zone fréquentielle correspondant au formant du chanteur aura donc tendance à renforcer la saillance de l'octave (voire du double-octave pour les tessitures vocales assez basses) de la mélodie.

2.3.1.3 Création des contours de hauteurs

Une détection de pics dans la fonction de saillance calculée précédemment permet d'obtenir l'ensemble des fréquences fondamentales candidates à la mélodie. Ces pics sont ensuite regroupés en trajectoires ou contours, c'est-à-dire en une séquence de pics saillants continue en temps et en fréquence. La création des ces contours peut se décomposer en plusieurs étapes :

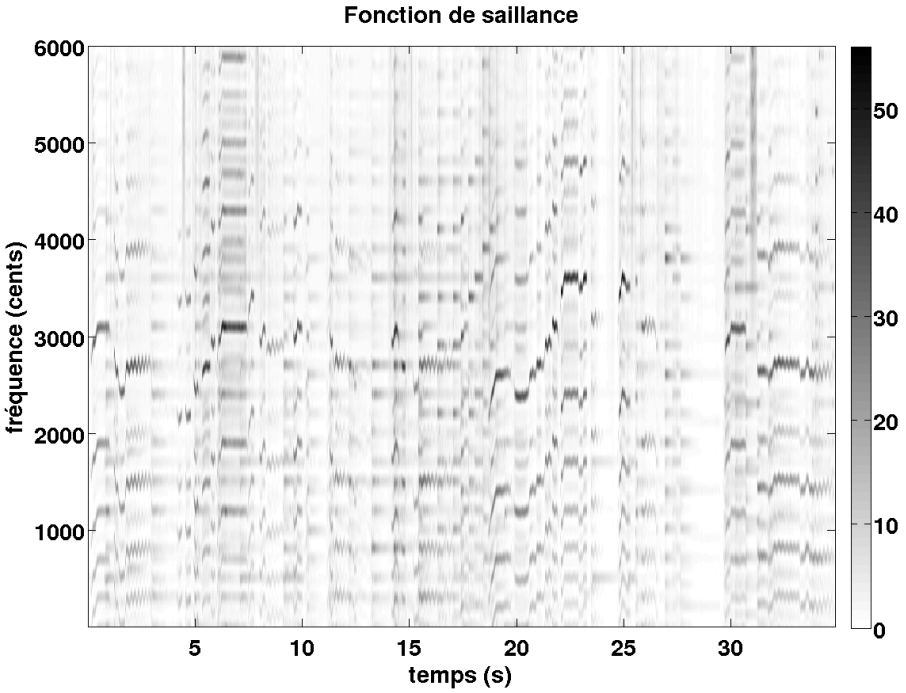


FIGURE 2.3 – Fonction de saillance

Filtrage des pics de la fonction de saillance :

En premier lieu, un filtrage des pics non-saillants permet de minimiser la création de contours qui n'appartiennent pas à la mélodie. Le processus de filtrage est le suivant :

- Analyse par trame : Les pics sont filtrés en comparant leur saillance à celle du pic le plus saillant dans la trame courante, ceux dont la saillance est inférieure à un certain seuil τ_+ sont éliminés.
- Analyse globale : La moyenne μ_s et l'écart-type σ_s de la saillance de l'ensemble des pics présents à toutes les trames sont calculés. Les pics dont la saillance est inférieure à $\mu_s - \tau_\sigma \sigma_s$ sont alors filtrés. Le seuil τ_σ détermine une certaine déviation autorisée autour de la saillance moyenne.

Suivi des pics spectraux :

Cette étape permet de créer des contour de hauteurs, c'est-à-dire un ensemble de flux de pics. Soient S^+ les pics temps/fréquence non-filtrés précédemment, et S^- ceux filtrés.

L'algorithme sélectionne le pic le plus saillant de $S+$; en cherchant successivement les pics les plus saillants dans les trames suivantes et précédentes (continuité temporelle), le contour de hauteurs est créé. Certaines contraintes sont à respecter : Pour être ajouté au contour en cours de création, un pic ne doit pas se trouver à plus de 40 cents de celui précédemment choisi (continuité fréquentielle). Quand un pic est sélectionné, il est retiré de S^+. Afin d'éviter de couper un contour qui devrait être unique, on autorise le processus à prendre des pics dans S^-, cependant on fixe une limite de durée de 100 ms, au delà de laquelle la construction du contour courant est arrêtée. Une fois que le suivi est terminé, le contour ainsi créé est sauvegardé et le processus est entièrement réitéré jusqu'à ce qu'il n'y ait plus de pic dans S^+. On observe figure 2.4 les contours ainsi créés.

On remarque également qu'à cette étape du processus d'extraction de mélodie plusieurs paramètres doivent être optimisés, ceci implique l'utilisation d'une base de donnée pour laquelle la vérité terrain est connue. Les paramètres utilisés ici sont alors $\tau_+ = 0.9$ et $\tau_\sigma = 0.9$ pour le filtrage des pics, la différence en cents autorisée pour la sélection de pics successifs et la durée maximale autorisée pour prendre en compte des pics de S^- ont déjà été détaillés dans le cas de l'algorithme de suivi.

Caractérisation des contours :

Une fois les contours de hauteurs construits, il est nécessaire de déterminer lesquels appartiennent à la mélodie. Pour cela on utilise un ensemble de descripteurs de contours afin de guider le processus de sélection. Les caractéristiques extraites pour les contours sont :

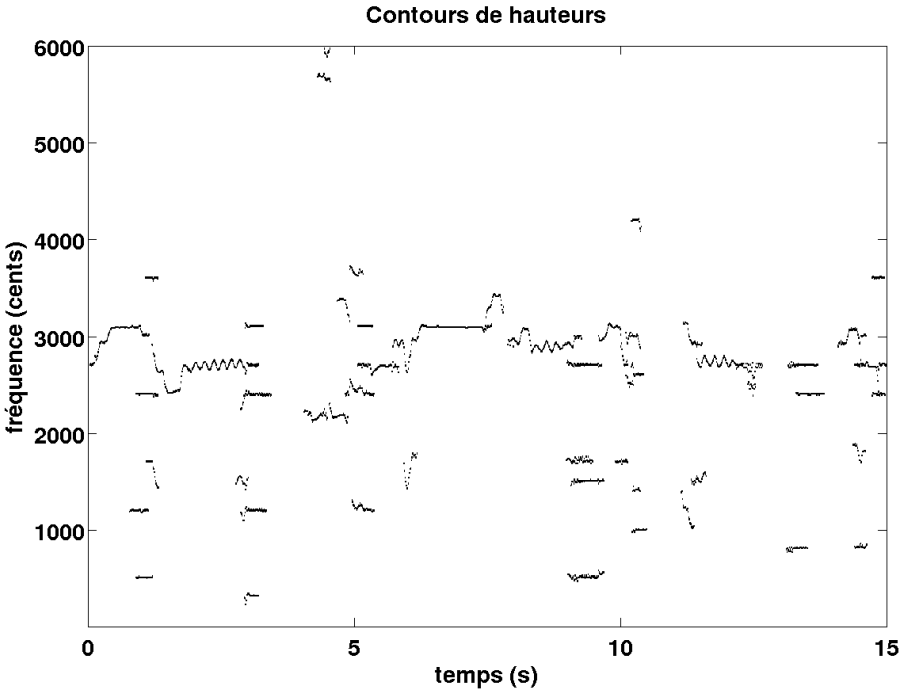


FIGURE 2.4 – Contours de hauteurs

- **Hauteur moyenne** $C_{\overline{f}}$: Moyenne des fréquences des pics qui composent le contour.
- **Ecart-type de la hauteur** C_{σ_f} : Ecart-type des fréquences des pics qui composent le contour.
- **Saillance moyenne** $C_{\overline{s}}$: Moyenne de la saillance des pics qui composent le contour.
- **Saillance totale** $C_{\sum_s}$: Somme de la saillance des pics qui composent le contour.
- **Ecart-type de la saillance** C_{σ_s} : Ecart-type de la saillance des pics qui composent le contour.
- **Longueur** C_l : Longueur du contour.
- **Présence de vibrato** C_v : Présence de vibrato détectée si le contour contient une composante sinusoïdale importante entre 5 et 8Hz.

A partir de l'observation de ces descripteurs pour plusieurs morceaux, les auteurs ont remarqué des caractéristiques spécifiques aux contours appartenant à la mélodie, notamment une variance fréquentielle plus grande dans le cas d'une mélodie vocale, des contours plus longs, une localisation dans les fréquences moyennes du spectre et une plus grande saillance. Afin d'avérer ces intuitions, les distributions de ces descripteurs sont calculées pour les contours appartenant à la mélodie et ceux appartenant à l'accompagnement, ceci à partir d'une base de donnée composée d'extraits proches de ceux utilisés pour l'évaluation MIREX. Ces distributions confirment les premières observations et permettront dans la suite de la méthode de sélectionner les contours appartenant à la mélodie. En ce qui concerne le vibrato, il est montré à partir de la base de donnée que lorsqu'un contour présente cette caractéristique, il appartient dans 95% des cas à la mélodie.

2.3.1.4 Sélection de la mélodie

L'étape finale de cette méthode d'extraction de la mélodie utilise les distributions des descripteurs estimées précédemment afin de filtrer les contours n'appartenant pas à la mélodie. Ce processus se décompose en trois sous-étapes : détection de présence de mélodie, réduction des erreurs d'octave et élimination des contours singuliers puis sélection finale de la mélodie.

Détection de présence de mélodie :

Les distributions de la moyenne de la saillance des contours de la mélodie et de l'accompagnement s'apparentent toutes deux à des gaussiennes, mais de moyenne et variance différentes (figure 3.c de [10]).

Si ces deux distributions ne sont pas parfaitement séparables, il est néanmoins possible d'effectuer une classification en fixant un seuil afin d'éliminer un grand nombre de contours appartenant à l'accompagnement et non à la mélodie. Les contours dont la saillance moyenne est inférieure à un certain seuil τ_v sont alors éliminés. Ce seuil est également un paramètre critique qu'il est nécessaire de définir de façon empirique.

Comme nous l'avons déjà mentionné, un contour présentant du vibrato ($C_v = vrai$) sera dans 95% des cas un contour mélodique. De plus, le calcul des distributions de l'écart-type de la hauteur des contours met en évidence que lorsque ce descripteur est supérieur à 40 cents ($C_{\sigma_f} > 40$), la probabilité que le contour appartienne à la mélodie est supérieure à 95%. Ces observations sont ainsi utilisées afin d'immuniser de tout filtrage, à ce stade, les contours présentant une saillance moyenne relativement faible mais de fortes caractéristiques mélodiques.

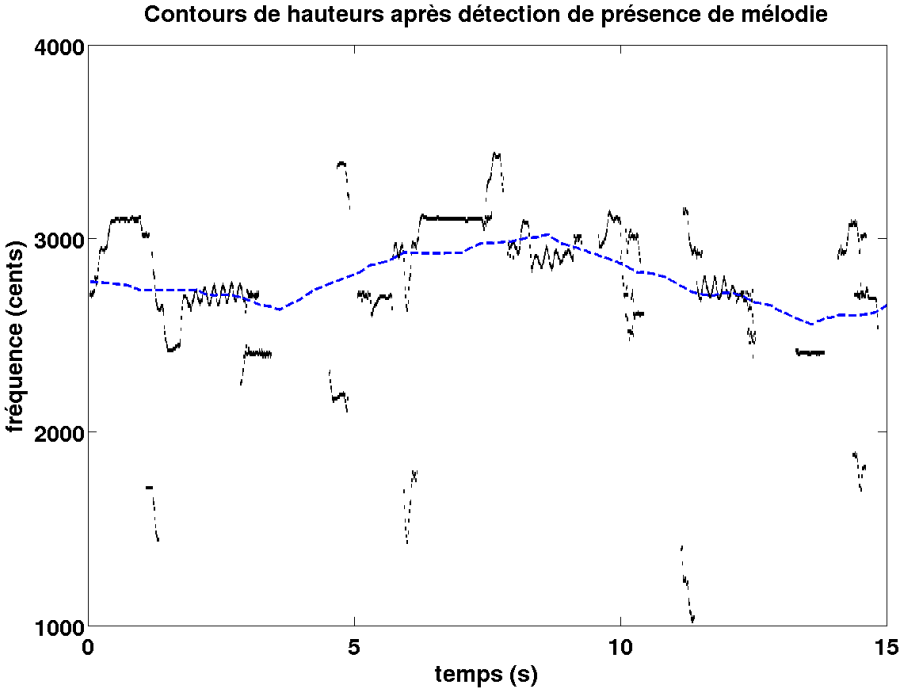


FIGURE 2.5 – Contours de hauteurs après détection de présence de mélodie. La courbe en tireté représente la moyenne lissée $\overline{P(t)}$

Réduction des erreurs d'octave et élimination des contours singuliers :

Comme nous l'avons remarqué lors du calcul de la fonction de saillance, une des erreurs les plus fréquentes correspond aux erreurs d'octave. La méthode présentée ici pour minimiser les erreurs d'octave se base sur l'étude des trajectoires en fréquence des contours de hauteurs. Afin de détecter les contours à l'octave, et pour décider lequel appartient à la mélodie, les contours voisins sont pris en compte, l'hypothèse sous-jacente étant que la mélodie a tendance à présenter une trajectoire continue en fréquence, le contour choisi sera donc celui qui évite les discontinuités fréquentielles. L'élimination des contours singuliers suit ce même principe de continuité fréquentielle de la mélodie. Une seconde hypothèse, pour décider quel contour parmi ceux à l'octave appartient à la mélodie, est que le plus souvent, le contour correct aura une saillance moyenne supérieure.

Le processus de réduction des erreurs d'octave et d'élimination des contours singuliers est le suivant, où $P(t)$ est l'évolution fréquentielle du contour au cours du temps :

1. Calculer $\overline{P(t)}$ pour chaque trame comme la moyenne pondérée des fréquences de chaque contour présent dans la trame. La pondération se fait par la saillance totale des contours afin de pénaliser les contours courts et non-saillants.

2. Lisser $\overline{P(t)}$ sur une fenêtre de 5 secondes. On observe figure 2.5 les contours après détection de la présence de mélodie et pour lesquels la moyenne lissée $\overline{P(t)}$ est représentée en tireté.

3. Détecter les contours à l'octave et pour chaque paire éliminer celui qui est le plus loin de $\overline{P(t)}$.

4. Répéter les étapes 1) et 2) pour mettre à jour $\overline{P(t)}$.

5. Eliminer les contours plus loin d'un octave de $\overline{P(t)}$, ils correspondent aux contours singuliers.
6. Répéter les étapes 1) et 2) pour mettre à jour $\overline{P(t)}$.
7. Répéter les étapes 3) à 6) deux fois. Considérer pour chaque itération l'ensemble des contours après détection de présence de mélodie mais en utilisant la dernière mise à jour de $\overline{P(t)}$.

Sélection de la mélodie finale :

Le résultat de l'ensemble des contours après la succession de filtrages effectués précédemment est représenté figure 2.6a. Pour la sélection finale de la mélodie, lorsqu'il y a encore plus d'un seul contour dans une trame, on choisit celui qui a la plus grande saillance totale. Si aucun contour n'est présent dans la trame, on considère que la mélodie n'y est pas présente. Afin d'évaluer l'estimation de mélodie indépendamment de la détection de présence de voix, une fréquence fondamentale est tout de même estimée pour ces trames en sélectionnant le pic du contour le plus saillant qui était présent avant les filtrages. Le résultat de l'estimation de la mélodie est présenté avec la vérité terrain sur la figure 2.6b, cette dernière a été décalée vers le bas pour plus de clarté.

2.3.2 Tachibana et al.

Nous avons également choisi la méthode d'extraction de mélodie basée sur la technique de séparation de sources harmonique/percussif HPSS [14] comme référence car elle présente également de bons résultats au niveau de l'estimation de la mélodie, mais surtout, elle introduit l'HPSS qui est utilisée dans [13] et [12] comme prétraitement à la détection de mélodie. L'HPSS se base sur une idée principale : les sons harmoniques sont lisses temporellement car ils sont maintenus et périodiques pendant une certaine durée, et les sons percussifs sont eux lisses fréquentiellement car ils sont instantanés. L'HPSS exploite donc la propriété anisotropique de ces deux types de sons, c'est-à-dire le fait que leur caractère lisse dépende de la direction considérée, fréquentielle ou temporelle. En appliquant à la suite deux HPSS à partir du signal de départ et en utilisant successivement des spectrogrammes avec différentes résolutions, il est alors possible comme nous allons le voir d'obtenir un signal pour lequel la mélodie est mise en avant, et à l'inverse, ce qui correspondait à des signaux percussifs ou harmoniques est atténué. Finalement, une dernière étape de suivi par programmation dynamique à partir de la CQT du signal obtenu par double HPSS permet d'extraire la mélodie principale.

2.3.2.1 Séparation de Sources Harmonique/Percussif (HPSS)

L'algorithme cherche à décomposer le spectrogramme d'amplitude du signal de départ calculé par TFCT $\mathbf{W} = \{|W_{n,k}|\}_{(n,k)\in\Omega}$ où $\Omega = \{(n,k)|0 \leq n \leq N-1, 0 \leq k \leq K-1\}$ en une partie harmonique $\mathbf{H} = \{|H_{n,k}|\}_{(n,k)\in\Omega}$ lisse temporellement et une partie percussives $\mathbf{P} = \{|P_{n,k}|\}_{(n,k)\in\Omega}$ lisse fréquentiellement, sous la contrainte que la somme de $\mathbf{H}$ et $\mathbf{P}$ soit égale au spectrogramme original $\mathbf{W}$. L'HPSS est alors exprimée comme un problème d'optimisation, on cherche à minimiser :

$$J(\mathbf{H},\mathbf{P}) = w_H \sum_{n=1}^{N-1}\sum_{k=0}^{K-1}(|H_{n,k}|^\gamma - |H_{n-1,k}|^\gamma)^2 + w_P \sum_{n=0}^{N-1}\sum_{k=1}^{K-1}(|P_{n,k}|^\gamma - |P_{n,k-1}|^\gamma)^2 \tag{2.3.1}$$

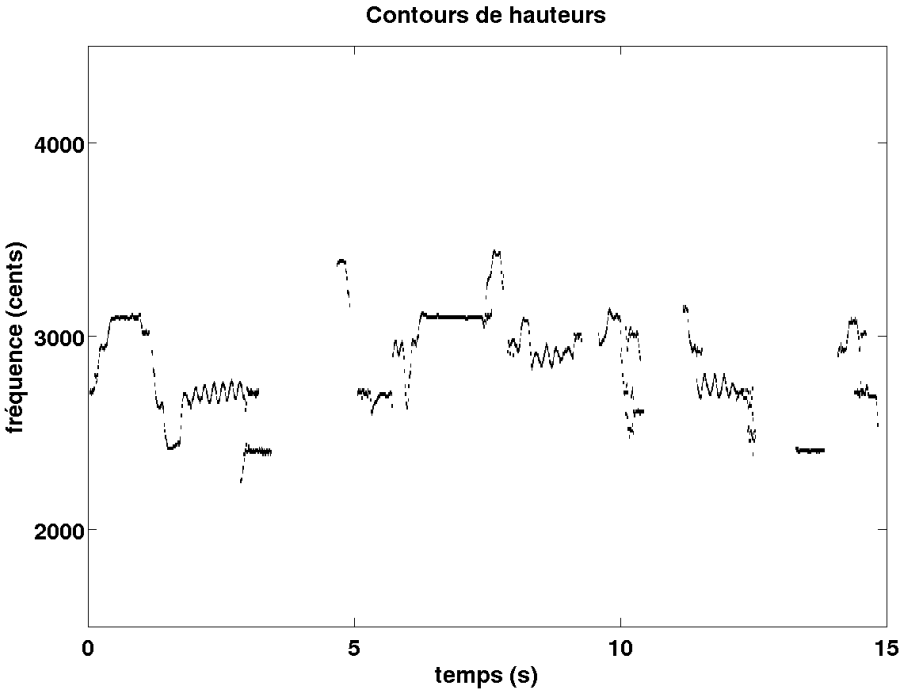
$$\text{s.c. } \mathbf{H}+\mathbf{P} = \mathbf{W} \tag{2.3.2}$$

où w_H et w_P sont des paramètres de pondération fixés empiriquement (ici à 1) et contrôlant le degré de lissage temporel et fréquentiel, γ est un coefficient qui permet de prendre en compte la loi de Stevens concernant la perception de l'intensité sonore : la sensation varie comme la puissance 0.6 de l'excitation, par soucis de simplification des calculs, les auteurs fixent $\gamma = 0.5$. On observe que le premier terme de l'équation (2.3.1) contraint $\mathbf{H}$ à être lisse temporellement, le deuxième terme contraint $\mathbf{P}$ à être lisse fréquentiellement. Le problème de minimisation peut-être résolu grâce à un algorithme itératif, les règles de mise à jour des deux matrices à estimer sont données dans [14].

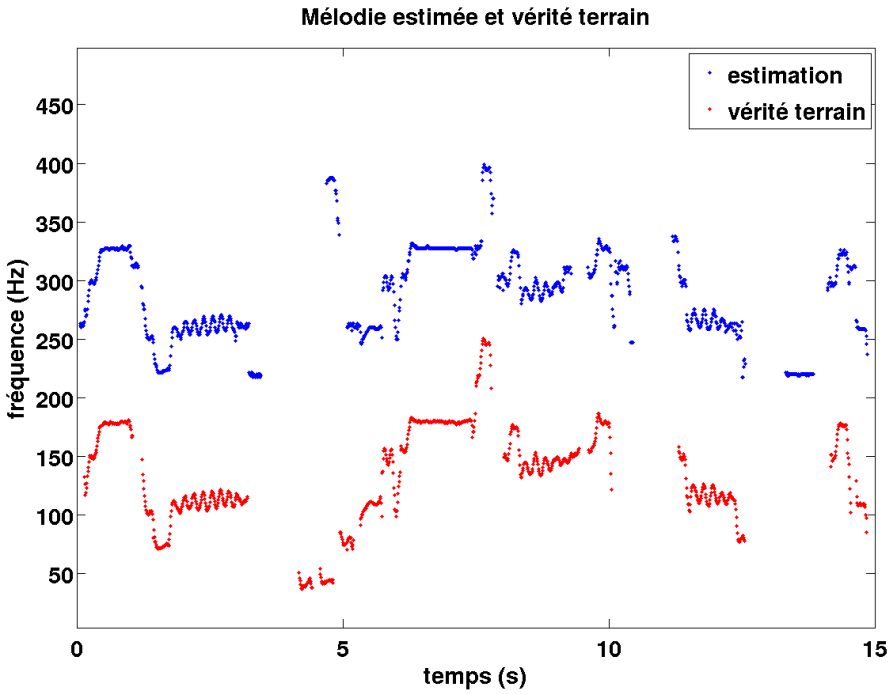
2.3.2.2 Mise en avant de la source mélodique par double HPSS

Bien que cette méthode d'extraction de source mélodique puisse se généraliser à tout type d'instrument porteur de mélodie, nous allons la présenter ici dans le cadre de la voix chantée, car c'est en effet pour ce type de signal que les résultats sont les meilleurs et de plus, rappelons que notre travail s'inscrit ici dans le cadre d'une application de séparation automatique de voix chantée.

Une caractéristique forte de la voix chantée est qu'elle présente un grand nombre de fluctuations en fréquence et en amplitude. Ceci est notamment dû au fait que l'appareil phonatoire soit un instrument de musique plus libre mécaniquement que les autres instruments. A la différence d'instruments tels que le piano par exemple, dont



(a) Contours de hauteurs après la succession de filtrages



(b) Mélodie finale estimée (en haut) et vérité terrain (en bas) décalée pour plus de clarté

la fréquence fondamentale et les harmoniques ne varient que peu en fréquence au sein d'une même note, on peut considérer que la voix est un instrument moins "sinusoïdal", dont les notes sont moins stables temporellement. Il n'en reste pas moins que la voix chantée est généralement bien plus "sinusoïdale" que les percussions. C'est ainsi que l'on peut considérer la voix chantée comme un intermédiaire entre les instruments harmoniques, stables temporellement et ceux percussifs. Dans le cadre de l'HPSS définie précédemment, la voix se trouve donc entre les composantes harmoniques et percussives que l'on cherche à extraire. Cette caractéristique de variabilité temporelle est ce qui permet de distinguer la voix chantée de l'accompagnement, mais cela peut aussi s'étendre à des mélodies jouées par d'autres instruments pour lesquelles on peut de la même façon, mais sûrement dans une moindre mesure, observer le même phénomène.

Ce raisonnement nous amène à considérer trois catégories de sons :

- $\mathscr{H}$: Les sons stables dans le temps, associés à l'harmonie.

- $\mathscr{V}$: Les sons fluctuants dans le temps, quasi-stables, associés à la mélodie (particulièrement dans le cas de la voix).

- $\mathscr{P}$: Les sons percussifs, non-stables dans le temps.

Nous nous attachons désormais à extraire du signal de départ la source associée à la catégorie $\mathscr{V}$. Le principe est ici d'effectuer successivement deux HPSS à partir de deux spectrogrammes dont les résolutions fréquentielles et temporelles sont différentes, paramètres contrôlés par la taille de la fenêtre dans le cas de la TFCT.

Considérons dans un premier temps une fenêtre d'analyse très grande, dans ce cas le spectrogramme a une très bonne résolution fréquentielle et les fluctuations des sons de la catégorie $\mathscr{V}$ se retrouveront dans de nombreux *bins* fréquentiels au sein d'une même trame. Ces sons seront par contre discontinus dans le temps, la fréquence fondamentale ayant sûrement changé entre deux trames successives. Les sons de la catégorie $\mathscr{V}$ seront donc plus lisses fréquentiellement que temporellement. Si l'on applique une HPSS sur un tel spectrogramme dont la longueur de fenêtre est longue, le signal de départ $w(t)$ sera séparé en un signal harmonique $h_1(t) \approx \mathscr{H}$ et un signal percussif $p_1(t) \approx \mathscr{P} + \mathscr{V}$.

Il nous faut maintenant réussir à isoler les sons de la catégorie $\mathscr{V}$ depuis le signal $p_1(t)$. Considérons pour cela une fenêtre d'analyse très courte pour le calcul de la TFCT, dans ce cas le spectrogramme associé à $p_1(t)$ a une très mauvaise résolution fréquentielle et les fluctuations des sons $\mathscr{V}$ ne se retrouveront que dans très peu de *bins* fréquentiels, ils occuperont par contre un nombre important de trames car la fréquence fondamentale ne changera pas sur une très courte fenêtre de signal. Le spectrogramme paraîtra donc très lisse temporellement. C'est pour cette raison que si l'on applique une HPSS sur ce spectrogramme, le signal de départ $p_1(t)$ sera séparé en $h_2(t) \approx \mathscr{V}$ et $p_2(t) \approx \mathscr{P}$.

Finalement, le processus de séparation de la source mélodique par double HPSS se résume ainsi :

$$w(t) \xrightarrow[\text{grande fenêtre d'analyse}]{HPSS} \{h_1(t), p_1(t)\} \tag{2.3.3}$$

$$h_1(t) \approx \mathscr{H} \text{ et } p_1(t) \approx \mathscr{P} + \mathscr{V} \tag{2.3.4}$$

$$p_1(t) \xrightarrow[\text{courte fenêtre d'analyse}]{HPSS} \{h_2(t), p_2(t)\} \tag{2.3.5}$$

$$h_2(t) \approx \mathscr{V} \text{ et } p_2(t) \approx \mathscr{P} \tag{2.3.6}$$

Le spectrogramme du signal d'entrée $w(t)$ ainsi que ceux des trois signaux $h_1(t)$, $h_2(t)$ et $p_2(t)$ issus de la double HPSS sont représentés figure 2.6. On observe clairement les trois catégories de sons $\mathscr{H}$, $\mathscr{V}$ et $\mathscr{P}$ auxquels ils sont respectivement associés. Le signal $h_2(t)$ correspond à la source mélodique.

2.3.2.3 Suivi de la mélodie par programmation dynamique

Afin d'effectuer le suivi de la mélodie dans le signal issu de la double HPSS, un schéma probabiliste est mis en place, basé sur deux hypothèses : la hauteur associée à une note jouée à un instant donné est proche fréquentiellement de celle à l'instant précédent et les sons formant la mélodie présentent une structure harmonique.

Soit $S_n = \{s_t\}_{1 \leq t \leq n}$ la série temporelle associée aux trames de la CQT du signal, où $s_t = \{s_{t,x}\}_{1 \leq x \leq m}$ correspond aux valeurs de la CQT à l'instant t et pour chaque *bin* $x \in [1, m]$. Soit $X_n = \{x_t\}_{1 \leq t \leq n}$ la série temporelle des hauteurs instantanées x_t que l'on souhaite estimer.

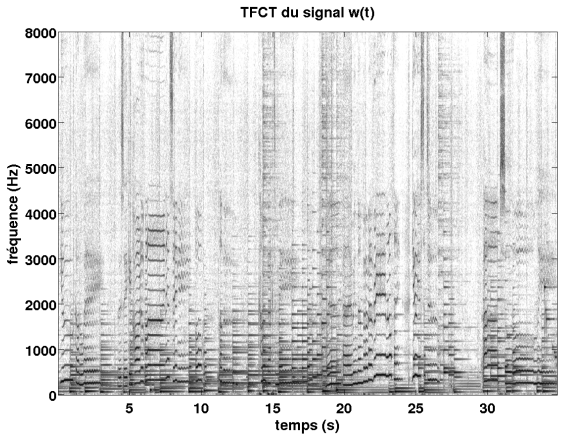
Etant donné S_n, on estime la mélodie la plus vraisemblable $\hat{X}_n$ qui maximise la probabilité *a posteriori* $p(X_n|S_n)$, c'est-à-dire,

$$\hat{X}_n = \underset{X_n}{\operatorname{argmax}}\ p(X_n|S_n) = \underset{X_n}{\operatorname{argmax}}\ p(X_n, S_n) \tag{2.3.7}$$

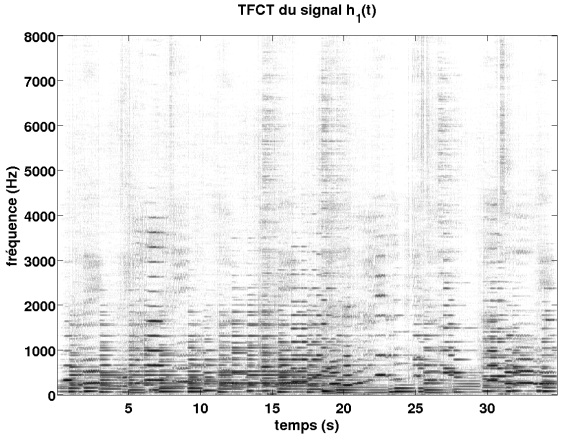
Si on suppose que la hauteur à l'instant t x_t dépend uniquement de celle à l'instant $t-1$ x_{t-1}, et que la trame s_t issue de la CQT dépend uniquement de la hauteur instantanée x_t, on peut écrire l'équation récurrente :

$$\ln p(X_t, S_t) = \ln p(X_{t-1}, S_{t-1}) + \ln p(s_t|x_t) + \ln p(x_t|x_{t-1}) \tag{2.3.8}$$

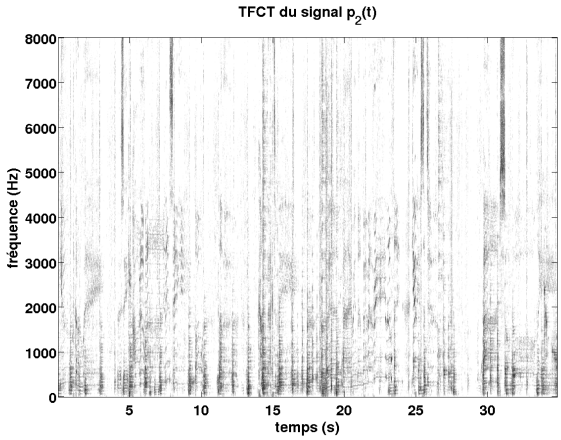
Le problème d'optimisation est ainsi divisé en optimisations locales, sur chaque trame, la résolution peut donc se faire par programmation dynamique afin de trouver la solution optimale globale.



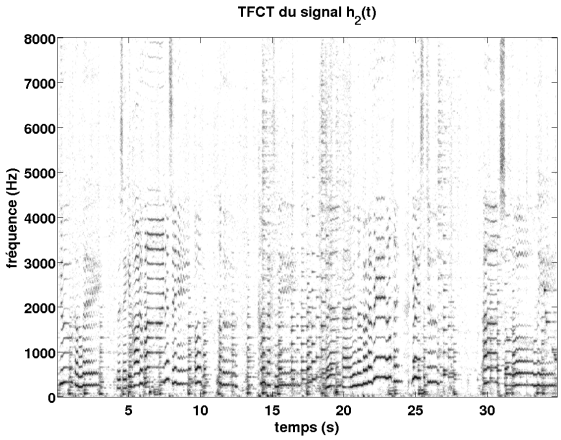
(a) TFCT du signal $w(t)$



(b) TFCT du signal $h_1(t)$



(c) TFCT du signal $p_2(t)$



(d) TFCT du signal $h_2(t)$

FIGURE 2.6 – Mise en avant de la source mélodique par double HPSS

Modèle de vraisemblance des observations $p(s_t|x_t)$ et modèle de transition $p(x_t|x_{t-1})$:

Par inter-corrélation entre chaque trame de la CQT et un modèle de son harmonique $q(k)$, on obtient $\hat{s}_t(x_t)$ qui présentera de fortes valeurs si x_t correspond à la vraie fréquence fondamentale dans la trame, on peut rapprocher cette opération d'une somme-spectrale :

$$\hat{s}_t(x_t) = \sum_{k=0}^{k_{Nyq}-x_t} s_t(k+x_t)q(k) \tag{2.3.9}$$

où x_t et k correspondent aux bins de la CQT, k_{Nyq} est associé à la fréquence de Nyquist et $q(k)$ est un modèle de son harmonique :

$$q(k) = \sum_{h=0}^{H-1} a_h g(k - hk_{min}) \tag{2.3.10}$$

g est une fonction correspondant à un partiel, elle est approximée par une gaussienne, H est le nombre d'harmoniques du modèle, $a_h = 1/h$ est la pondération de chaque harmonique et k_{min} est associé à la fréquence minimale considérée pour le calcul de la CQT. On considère finalement que $p(s_t|x_t) \propto \hat{s}_t(x_t)$.

En faisant l'hypothèse que la hauteur la plus probable à l'instant t, x_t, soit la plus proche de la précédente x_{t-1}, on définit un modèle de transition $p(x_t|x_{t-1})$ suivant une densité normale de moyenne x_{t-1} et de variance σ définie empiriquement :

$$p(x_t|x_{t-1}) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_t - x_{t-1})^2}{2\sigma^2}} \tag{2.3.11}$$

La séquence optimale de hauteurs $\hat{X}_n$ correspondant à l'estimation de la mélodie est alors décodée par l'algorithme de Viterbi en utilisant les modèles de probabilité définis précédemment.

2.3.2.4 Détection de la présence de mélodie

Il n'est pas mentionné dans [14] la méthode qu'ont utilisée les auteurs afin de détecter quand la mélodie est présente, or les résultats du concours MIREX indiquent que lors de la soumission de leur algorithme, une méthode de détection de la mélodie était bien présente. Cette méthode est brièvement introduite dans le résumé accompagnant la soumission à MIREX [26]. Les auteurs y mentionnent un résultat intéressant concernant les statistiques permettant de caractériser le bruit présent dans le signal $h_2(t)$ après la deuxième HPSS. La méthode utilise deux spectrogrammes d'amplitude $H^{(2)}_{n,k}$ et $P^{(2)}_{n,k}$ associés respectivement aux signaux $h_2(t)$ et $p_2(t)$ issus de la double HPSS (voir formule (2.3.5)). Il est possible de considérer que le spectrogramme $H^{(2)}_{n,k}$ soit essentiellement du bruit pour les trames où la mélodie n'est pas présente. Chaque point temps/fréquence pour ces trames peut alors être modélisé comme la réalisation d'une variable aléatoire suivant une loi normale de paramètres sa moyenne et sa variance. Si cette modélisation fonctionne en théorie, il est rare que les segments temporels sans mélodie dans $H^{(2)}_{n,k}$ correspondent effectivement à du bruit car des interférences entre les sources séparées restent présentes. L'hypothèse faite est alors que les statistiques du bruit de $H^{(2)}_{n,k}$ peuvent être approximées par celles de $P^{(2)}_{n,k}$.

Considérons les spectrogrammes $H^{(2)}_{n,k}$ et $P^{(2)}_{n,k}$ comme un ensemble d'observations de vecteurs à plusieurs variables. Les observations correspondent à l'axe temporel (indicées par n) et les variables à l'axe fréquentiel (indicées par k). Il faut considérer les variables comme indépendantes et définissant un espace multidimensionnel dans lequel chaque observation est représentée. A partir de $P^{(2)}_{n,k}$ on définit un point qui représente les moyennes des variables sur toutes les observations, c'est le vecteur $\boldsymbol{\mu} = (\mu_1, \mu_2, ..., \mu_K)$, on calcule également la matrice de covariance des observations $\boldsymbol{\Sigma}$. Pour chaque trame $H^{(2)}_{n,k}$, la distance de Mahalanobis du vecteur $\mathbf{h_n^{(2)}} = (H^{(2)}_{n,1}, H^{(2)}_{n,2}, ..., H^{(2)}_{n,K})$ à un ensemble de vecteurs de valeurs moyennes $\boldsymbol{\mu} = (\mu_1, \mu_2, ..., \mu_K)$ et possédant une matrice de covariance $\boldsymbol{\Sigma}$ est alors calculée :

$$D_{\text{Mahalanobis}}(\mathbf{h_n^{(2)}}) = \sqrt{(\mathbf{h_n^{(2)}} - \boldsymbol{\mu})\boldsymbol{\Sigma}^{-1}(\mathbf{h_n^{(2)}} - \boldsymbol{\mu})^T} \tag{2.3.12}$$

où la notation $\mathbf{v}^T$ correspond au vecteur transposé du vecteur $\mathbf{v}$.

Finalement, pour chaque observation $H^{(2)}_{n,k}$, si la distance à ce point central est assez faible, alors c'est que pour la trame associée la mélodie n'est pas présente. Cette distance est superposée au spectrogramme $H^{(2)}_{n,k}$ sur la figure 2.7.

Il est nécessaire de fixer un seuil sur la distance de Mahalanobis ainsi calculée, pour décider l'absence de mélodie quand la distance est assez faible. Ce seuil est calculé de façon dynamique par la méthode d'Otsu [27]. Cette méthode provient du domaine de la vision par ordinateur et permet à l'origine d'effectuer un seuillage automatique d'une image à partir de son histogramme pour obtenir une image binaire ne contenant que deux classes de pixel (premier plan et arrière plan). Le seuil est calculé afin de minimiser la variance intra-classe, ce qui revient en fait à maximiser la variance inter-classe. Dans notre cas, le seuil est estimé à partir de l'histogramme de la distance de Mahalanobis calculée pour chaque trame de $H^{(2)}_{n,k}$. Comme on peut le voir sur la figure 2.7, la distance est très bruitée, ce qui explique les performances relativement faibles de ce système pour la détection de voix (voir section 2.3.3.2).

2.3.3 Evaluation des méthodes

Pour évaluer les deux méthodes de référence et celle actuellement utilisée dans ADX TRAX, il nous a fallu mettre en place un banc d'essai. L'extraction de mélodie dans ADX TRAX est faite sur le signal de voix chantée préalablement séparé, cette approche est donc basée séparation de sources.

L'étape d'évaluation est importante car elle permet de situer la méthode d'ADX TRAX par rapport aux différents algorithmes que nous avons implémentés, et surtout, nous verrons qu'elle met en évidence l'importance de la détection de voix pour obtenir de bonnes performances globales.

2.3.3.1 Mesures de performances

Il nous faut tout d'abord définir un ensemble de mesures permettant d'évaluer les algorithmes.

Mesures de performances de la détection de voix :

- Vrais Positifs (VP) : Nombre de trames effectivement en présence de voix et détectées comme telle.

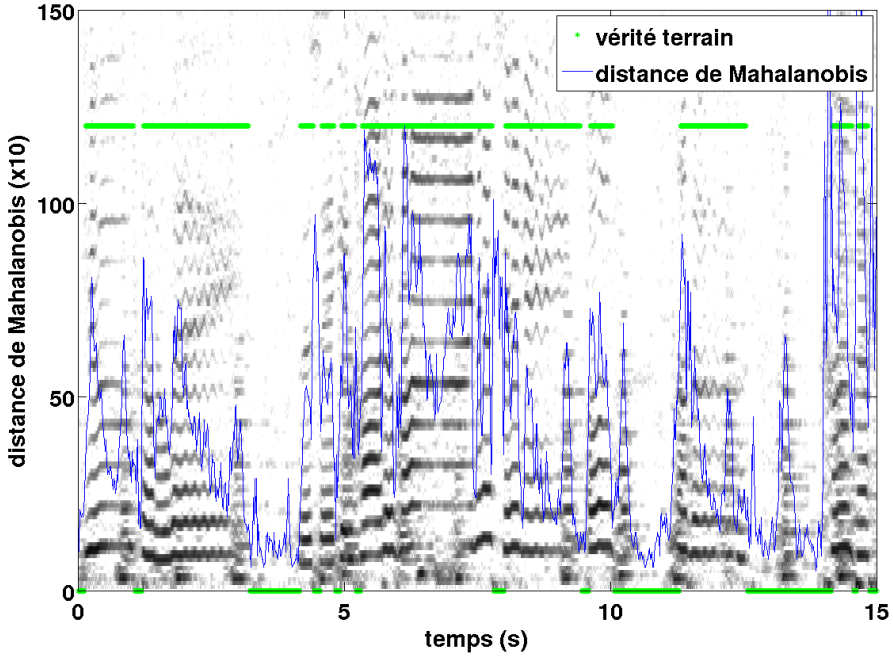


FIGURE 2.7 – Distance de Mahalanobis

- Faux Positifs (FP) : Nombre de trames détectées comme en présence de voix mais à tort.
- Vrais Négatifs (VN) : Nombre de trames effectivement sans présence de voix et détectées comme telle.
- Faux Négatifs (FN) : Nombre de trames détectées comme sans présence de voix mais à tort.

- Rappel = VP/(VP+FN) : C'est la proportion de trames avec voix dans la vérité terrain qui sont effectivement estimées comme en présence de voix par l'algorithme.
- Taux de fausse-alarme = FP/(VN+FP) : C'est la proportion de trames sans voix dans la vérité terrain qui sont estimées à tort comme en présence de voix par l'algorithme.

Ces mesures sont celles utilisées dans le contexte de l'estimation de mélodie. Lorsqu'on travaille uniquement sur la détection de voix, il est d'usage d'en utiliser d'autres que nous présenterons section 3.3. Dans [9], il est indiqué que ces mesures sont moyennées sur l'ensemble de la base de test. Nous notons cependant une incohérence avec les instructions du concours MIREX [1], spécifiant que les mesures de détection de voix sont calculées non pas en moyennant sur l'ensemble des fichiers de la base mais en considérant toutes les trames conjointement, ceci afin d'éviter d'obtenir des résultats biaisés si par exemple certains fichiers sont très courts et en permanence avec de la voix. Après vérification des résultats de la méthode de Salamon et de celle de Tachibana présentées pour l'évaluation MIREX 2011 [2], les mesures sont effectivement moyennées sur l'ensemble de la base de données. Nous procédons donc de cette façon pour présenter des résultats de détection de voix dans le cadre de l'estimation de mélodie.

Mesures de performances de l'estimation de la ligne mélodique :

- *Raw Pitch Accuracy* : C'est la proportion de trames avec voix dans la vérité terrain pour lesquelles le pitch est correctement estimé par l'algorithme (plus ou moins un quart de ton).
- *Raw Chroma Accuracy* : C'est le même critère que précédemment mais ramené à l'octave, ce qui permet de ne pas tenir compte des erreurs d'octave.

[1] http://www.music-ir.org/mirex/wiki/2014:Audio_Melody_Extraction
[2] http://nema.lis.illinois.edu/nema_out/mirex2011/results/ame/adc04/index.html

Mesure de performance globale :

- *Overall Accuracy* : C'est la proportion de trames correctement estimées par l'algorithme, c'est-à-dire estimées sans voix quand effectivement celle-ci n'est pas présente, et pour celles où elle est présente, l'algorithme doit les avoir détectées avec voix et le pitch doit être correctement estimé (plus ou moins un quart de ton).

Pour plus de détails sur ces mesures on pourra se référer à [9].

2.3.3.2 Résultats de l'estimation de mélodie

Pour évaluer l'extraction de mélodie, nous avons constitué la base de test [TestMélodie] (cf. annexe A) comprenant uniquement des morceaux où la mélodie principale est vocale. Les résultats moyennés sur l'ensemble de cette base sont présentés dans le tableau 2.1 pour les deux méthodes de référence et pour celle utilisée dans ADX TRAX. Pour cette dernière, nous indiquons les résultats avec et sans l'étape de détection de voix (DV) comme celle-ci est effectuée en début d'algorithme et contraint totalement l'estimation de mélodie : la mélodie n'est pas estimée quand la voix n'est pas détectée, voir section 3.4 pour plus de détails.

TABLE 2.1 – Résultats moyens de détection de mélodie sur la base [TestMélodie] - SG : Salamon et Gómez - TOOS : Tachibana et al. - ADX avec/sans DV : méthode utilisée dans ADX TRAX avec/sans Détection de Voix

	rappel (%)	**taux de fausse alarme (%)**	
SG	82.87	44.49	
TOOS	58.54	40.82	
ADX avec DV	53.87	34.25	
ADX sans DV	-	-	
	***raw pitch accuracy* (%)**	***raw chroma accuracy* (%)**	***overall accuracy* (%)**
SG	66.70	73.90	57.04
TOOS	74.98	82.05	49.37
ADX avec DV	41.09	46.42	49.64
ADX sans DV	73.61	77.56	-

La méthode de Salamon et Gómez est celle qui d'après le concours MIREX présente actuellement les meilleurs résultats, c'est également le cas pour l'évaluation que nous avons effectuée. En évaluant notre implémentation avec la base de données ADC2004 uniquement (20 fichiers), qui fait partie des bases d'évaluation de MIREX, nous obtenons une mesure d'*overall accuracy* de 66.3% alors que MIREX rapporte un score de 73.5%. Cette différence peut s'expliquer notamment par le fait que nous n'avons pas utilisé un incrément aussi faible que celui spécifié par les auteurs pour le calcul de la TFCT. Ceux-ci mentionnent un pas de 2.9ms, ce qui représente une précision temporelle telle que la méthode devient très coûteuse en mémoire. Nous avons donc choisi un incrément de 25% de la longueur de la fenêtre d'analyse, soit 11.6ms. Pour la méthode de Tachibana, nous obtenons des résultats équivalents à ceux reportés dans MIREX pour la base ADC2004 (sans tenir compte de la détection de voix qui n'était pas décrite dans [14]).

Lors de nos tests, nous avons fait varier les paramètres des seuillages de la méthode de Salamon et Gómez, utilisés pour filtrer les pics de la fonction de saillance avant l'étape de création des contours (τ_+ et τ_σ) et pour la détection de présence de mélodie (τ_v). Sans effectuer une mesure précise de l'impact de ces paramètres sur les performances avec la base de test [TestMélodie], nous nous sommes tout de même rendus compte que les résultats variaient fortement avec ces derniers, c'est pour cela qu'ils doivent être optimisés. Ces optimisations ne sont pas évidentes, compte tenu de la sensibilité des paramètres et du fait qu'ils doivent être ajustés conjointement. En effet, modifier l'étape de création des contours impactera celle de détection de présence de mélodie. De plus, optimiser les paramètres implique l'utilisation de bases de données, il existe donc un risque de sur-apprentissage et le contrôle sur ce risque n'est pas simple. Dans l'article de Salamon et Gómez, les seuils mis en jeu sont optimisés à partir de trois bases utilisées pour le concours MIREX, ce qui peut expliquer le fait que le résultat d'*overall accuracy* diminue fortement (plus de 9%) entre la base ADC2004 et la base que nous avons constituée [TestMélodie].

Salamon et Gómez mentionnent plusieurs pistes d'amélioration de leur méthode, notamment pour la création des contours. Il faudrait par exemple prendre en compte une information de timbre pour la création des contours de hauteurs. En ce qui concerne le filtrage des contours, plusieurs descripteurs sont utilisés, une amélioration envisageable serait d'introduire de nouveaux descripteurs afin de mettre en place un système de classification plus élaboré.

On remarque à partir du tableau 2.1 que pour l'ensemble des méthodes, il est difficile d'avoir un bon taux de rappel en même temps qu'un faible taux de fausse-alarme. Salamon et Gómez effectuent dans leur article une estimation du plafond des performances de leur méthode, et la plus grande perspective d'amélioration se situe justement au niveau du taux de fausse-alarme. Concernant les deux méthodes basées séparation de sources, on observe que leurs performances pour l'estimation de mélodie seule (*raw pitch accuracy*, *raw chroma accuracy*), sans tenir compte de la détection de voix, sont très bonnes et largement au dessus de la méthode de Salamon et Gómez. Cependant, les performances en détection de voix restent assez faibles.

Il existe une différence principale pour la détection de voix entre la méthode de Salamon, et les deux basées séparation de sources, Tachibana et ADX TRAX. Pour la première, cette détection est totalement liée au reste de l'algorithme, elle se situe juste après la création de contours et avant l'étape de réduction des erreurs d'octave et d'élimination des contours singuliers. Elle est donc totalement dépendante des traitements effectués précédemment et influe sur les performances des blocs de traitements suivants. A l'inverse, pour la méthode de Tachibana ou d'ADX TRAX, la détection de voix constitue une étape du processus à part. Dans le premier, cas elle se fait en fin d'algorithme par seuillage de la distance de Mahalanobis entre les spectrogrammes des signaux issus de la double HPSS. Dans le second cas elle s'effectue en tout début d'algorithme par un système de classification à partir de descripteurs spectraux. Il s'agit donc d'un bloc de traitement sur lequel des améliorations sont envisageables indépendamment de l'estimation de mélodie, et qui au vu des résultats permettraient d'améliorer significativement les performances globales, la mesure d'*overall accuracy* étant fortement impactée lorsque la détection de voix n'est pas correcte. On remarque d'ailleurs que celle-ci est très faible pour l'ensemble des méthodes. En comparaison avec les résultats de MIREX, ceci est dû au fait que la base de test sur laquelle nous travaillons est plus hétérogène et conséquente que celles utilisées pour ce concours.

Rappelons qu'un critère important que nous souhaitons prendre en compte est le temps de calcul, car notre travail se place dans le cadre d'une application de séparation automatique de la voix chantée, la méthode doit donc être assez rapide car elle sert de pré-traitement à la séparation de sources. Voici ci-dessous les temps de calculs approximatifs pour les trois méthodes que nous avons évaluées, ceci pour 1s de signal sur un Intel Core i7-960 cadencé à 3.2GHz avec 8 coeurs :

- Salamon et Gómez : 9s
- Tachibana et al. : 0.5s
- ADX TRAX avec détection d'activité vocale : 0.4s

La méthode de Salamon et Gómez est très lourde en temps de calculs, en effet, nous avons vu qu'elle mettait en oeuvre un grand nombre d'étapes différentes, qui s'avèrent coûteuses en temps et également en mémoire (estimation des fréquences instantanées, calcul de la fonction de saillance, création de contours de hauteurs, multiples seuillages...).

2.4 Conclusion

Dans cette partie nous avons présenté les différentes approches existantes pour résoudre le problème d'extraction automatique de mélodie. Nous avons plus particulièrement porté notre attention sur deux méthodes, celle de Salamon et Gómez et celle de Tachibana et al. La première est caractéristique des approches basées saillance, où de nombreux paramètres sont mis en jeu pour définir l'ensemble des règles permettant d'extraire progressivement la mélodie (transformation spectrale, seuillages, ensemble d'heuristiques...). Ces méthodes nécessitent une étape d'optimisation des représentations et des paramètres impliqués, celle-ci étant critique afin d'obtenir de bons résultats. C'est d'ailleurs l'objectif de l'article [24] dont les résultats ont été utilisés pour la méthode de Salamon et Gómez et qui a permis à l'algorithme d'améliorer ses performances significativement entre sa soumission au concours MIREX en 2010 et celle en 2011.

Le second algorithme que nous avons implémenté, basé sur la méthode de Tachibana, est quant à lui représentatif des méthodes basées séparation de sources, approche sur laquelle se base également l'algorithme utilisé dans ADX TRAX. Dans ces méthodes, la mélodie est extraite à partir d'un signal pour lequel la source mélodique est isolée.

Le banc d'essai mis en place pour évaluer ces 3 stratégies d'extraction automatique de mélodie nous a permis de mettre en évidence l'importance de la détection de voix. En effet, des perspectives d'amélioration claires se dégagent des méthodes basées séparation de sources en ce qui concerne la détection de voix, étape critique limitant les résultats globaux de ces méthodes (*overall accuracy*) alors même que la mesure de *raw pitch accuracy* est bonne. C'est pour cette raison que nous avons choisi de nous concentrer pour la suite sur la détection de la voix chantée. Dans le prochain chapitre nous présentons un état de l'art de cette tâche.

Chapitre 3

Etat de l'art de la détection de voix chantée

La majorité des méthodes de l'état de l'art pour la détection de voix chantée se basent sur des techniques d'apprentissage automatique. Il s'agit d'extraire un ensemble de descripteurs par une analyse à court-terme du signal, ceux-ci sont ensuite fournis en entrée d'un système de classification. Les paramètres du classifieur doivent préalablement être appris sur une base de données. Parmi ces méthodes, on peut considérer un sous-ensemble constitué de celles effectuant la classification par des seuillages. Ces dernière mettent généralement en place un système d'extraction de descripteurs élaboré, représentant de l'information très haut niveau, et effectuent ensuite une classification simple où les seuls paramètres à déterminer sont les seuils.

Une telle approche a été utilisée dans [28] et se base sur le fait que la voix présente naturellement un vibrato (modulation de la fréquence fondamentale) et un trémolo (modulation de l'amplitude) de façon simultanée, alors que les instruments ne peuvent jouer que sur un de ces deux effets à la fois. Après avoir extrait les partiels sinusoïdaux du signal, deux descripteurs caractérisant la présence de vibrato et de trémolo sont estimés. Le premier en calculant l'amplitude maximale entre 4 et 8Hz de la transformée de Fourier de l'évolution de la fréquence du partiel dans le temps. Le second est estimé de la même façon mais en considérant l'évolution de l'amplitude du partiel dans le temps. L'idée est qu'une forte composante entre 4 et 8Hz dans la transformée de Fourier indiquera la présence de vibrato ou de trémolo. La détection des partiels vocaux se fait en seuillant les amplitudes de vibrato et de trémolo estimées pour chaque partiel et en introduisant un critère d'harmonicité.

3.1 Descripteurs et classifieurs usuels

Les descripteurs les plus utilisés sont sans doutes ceux issus du domaine du traitement de la parole. Les Coefficients de Prédiction Linéaire (*Linear Predictive Coefficients*) (LPCs) sont calculés à partir d'une modélisation auto-régressive du signal. Les Coefficients de Prédiction Linéaire Perceptuels (*Perceptual Linear Predictive Coefficients*) (PLPs) correspondent à une version perceptuelle des LPCs, le signal étant tout d'abord traité dans le domaine fréquentiel sur une échelle de Bark (pré-accentuation selon la sensibilité fréquentielle de l'oreille et compression de l'amplitude) avant l'extraction des coefficients LPCs.

Les *Log Frequency Power Coefficients* (LFPCs) sont des coefficients issus d'un banc de filtres répartis de façon logarithmique en fréquence. Les *Harmonic Attenuated* LFPC (HA-LFPCs) sont identiques mais un filtrage pour atténuer les composantes harmoniques est effectué avant l'extraction des coefficients. Ces derniers descripteurs sont utilisés dans [29], l'atténuation harmonique est justifiée par le fait que la structure en fréquence du signal de voix chantée est beaucoup moins harmonique que pour les signaux instrumentaux, il s'agit donc de détecter ces irrégularités dans le spectre (surplus d'énergie) en atténuant les composantes harmoniques. L'ensemble des filtres à appliquer pour l'atténuation est déduit de la tonalité du morceau, définissant ainsi un nombre restreint de notes.

Les MFCCs sont une représentation cepstrale utilisant des bancs de filtres répartis sur une échelle Mel et une transformée en cosinus discrète, ils sont les plus utilisés dans les systèmes de détection de la voix chantée. Dans [30], les MFCCs semblent fournir les meilleurs résultats parmi un ensemble de comparaisons de descripteurs. Il est commun d'ajouter les dérivées temporelles première et seconde de tous ces descripteurs.

D'autres descripteurs de la forme du spectre peuvent également être utilisés pour la détection de voix, on peut par exemple estimer le centroïde spectral ou le *Roll-Off* spectral [31]. Le premier correspond au centre de gravité du spectre et le second est la fréquence au dessous de laquelle se situe un certain pourcentage de l'énergie du signal. Certains descripteurs utilisés sont estimés dans le domaine temporel, comme le niveau RMS caractérisant l'énergie du signal ou bien le nombre de fois où le signal passe par zéro (*Zero Crossing Rate* (ZCR)). Le ZCR sera élevé lorsque le signal est essentiellement du bruit, à l'inverse il sera faible pour des signaux harmoniques. Certaine méthodes utilisent également une estimation de la ligne mélodique [32, 33].

On trouvera dans [34] une descriptions de nombreux descripteurs audio pouvant être utilisés pour des tâches de classification.

Plusieurs classifieurs ont été proposés pour détecter la présence de voix à partir d'un ensemble de descripteurs, notamment les SVM, GMM, HMM et les réseaux de neurones.

Les systèmes de classification basés sur une analyse à court-terme du signal ont tendance à produire une sortie bruitée, c'est-à-dire avec des sauts fréquents entre les deux classes possibles (présence/absence de voix). Il est commun d'effectuer un post-traitement pour lisser temporellement la décision prise à chaque instant d'analyse par le classifieur. On peut dans un premier temps éliminer les segments courts sans présence de voix [28, 32]. Il est également possible d'effectuer un filtrage median sur une fenêtre glissante ou bien une technique plus élaborée consiste à utiliser un HMM [32].

3.2 Méthodes de l'état de l'art

Dans [35], les auteurs utilisent un ensemble de 13 MFCCs, 39 PLPs et 12 LFPCs, ils comparent un réseau de neurones et un SVM pour opérer la classification, ils obtiennent les meilleurs résultats avec ce dernier.

Dans [32], un grand nombre de descripteurs est extrait du signal par deux analyses à différentes échelles temporelles. Cela permet de créer un vecteur d'entrée de 116 composantes, dont la dimension est ensuite réduite en ne gardant que les descripteurs les plus discriminants. Après une détection du silence basée sur l'énergie dans chaque trame de signal, le vecteur de descripteurs est fourni en entrée d'un SVM pour effectuer la classification. Un HMM est ensuite utilisé pour lisser temporellement le résultat de classification. Les résultats de cette méthode sur la base de données Jamendo sont présentés tableau 3.1.

Dans [6], les auteurs utilisent la technique de séparation de sources HPSS comme pré-traitement avant d'extraire des MFCCs et d'utiliser un HMM comme système de classification. Il est montré que le pré-traitement par HPSS permet d'améliorer significativement les résultats. Dans cet article, la détection de voix est utilisée dans le cadre d'une technique de séparation de la voix chantée, afin d'annuler les segments non-vocaux dans le signal séparé.

Dans [33], trois descripteurs sont basés sur l'extraction au préalable de la mélodie principale : pour chaque trame de signal, les auteurs mesurent les fluctuations de la ligne mélodique, ils calculent les amplitudes des partiels de la voix principale, et finalement ils extraient les MFCCs d'un signal monophonique correspondant à la voix principale re-synthétisée (à partir de la ligne mélodique extraite et des partiels associés). Les derniers descripteurs sont les MFCCs du signal original. Le classifieur utilisé est un SVM-HMM.

Dans [36], Lehner montre qu'en utilisant uniquement des MFCCs et un système de classification par forêt aléatoire, il est possible d'obtenir des résultats comparables à l'état de l'art. Pour cela il optimise sur une base d'apprentissage plusieurs paramètres :

- Taille du banc de filtres pour l'extraction des MFCCs : 30.
- Nombre de coefficients MFCCs retenus : 30 inclus le premier ainsi que leurs dérivées premières, soit 60 coefficients au total.
- Taille de la fenêtre d'analyse : 800ms avec un incrément de 200ms.

Dans [37], Lehner utilise toujours un classifieur par forêt aléatoire mais améliore nettement ses précédents résultats sur la base de données Jamendo (voir tableau 3.1). Les descripteurs utilisés permettent de réduire considérablement le taux de fausse-alarme (donc d'améliorer la précision), grâce au fait qu'ils soient conçus pour mieux discriminer la voix chantée des autres instruments harmoniques. Cette méthode permet d'obtenir à notre connaissance les meilleurs résultats sur Jamendo, elle nous servira donc de point de comparaison par la suite. Trois descripteurs sont extraits à partir d'une analyse à court-terme du signal dans 17 bandes fréquentielles :

- Caractérisation des fluctuations des partiels inférieures au demi-ton à l'aide d'un *Fluctuogram*.
- Mesure de la dispersion de l'énergie en fréquence (*Spectral Contraction*) afin de savoir si l'énergie est concentrée au milieu du spectre ou diffuse.
- Caractérisation du spectre par une mesure de *Spectral Flatness*.

Les descripteurs précédents ne permettent pas de bien discriminer les instruments harmoniques de la voix chantée. Les cinq premiers MFCCs (sans le premier) sont alors calculés car ils représentent les variations lentes du spectre qui sont liées aux changement de forme du conduit vocal. Il est montré qu'ils sont particulièrement représentatifs de la voix chantée. Le vecteur des descripteurs est alors de dimension 116 et comprend :

- 17 coefficients pour les variances du *fluctuogram* sur 40 trames dans chaque bande de fréquences.
- 17 coefficients pour les moyennes de la mesure de *spectral flatness* sur 40 trames dans chaque bande de fréquences.

- 17 coefficients pour les variances de la *spectral contraction* sur 40 trames dans chaque bande de fréquences.
- 30 MFCCs ainsi que leurs dérivées premières, soit 60 coefficients.
- 5 coefficients pour les variances sur 11 trames des 5 premiers MFCCs (sauf le premier).

3.3 Mesures de performances

Les mesures de performances pour la détection de voix comprennent le rappel qui a été défini section 2.3.3.1, s'ajoutent également :

- Le taux de bonne classification = (TP+TN)/#{trames} : C'est la proportions de trames correctement classées.
- La précision = VP/(VP+FP) : C'est la proportions de trames estimées comme en présence de voix par l'algorithme à juste titre, c'est-à-dire qui sont effectivement avec voix d'après la vérité terrain.
- La F-mesure = 2(rappel.précision)/(rappel+précision) : C'est la moyenne harmonique du rappel et de la précision.

Il est important de noter que les conventions liées à ces mesures sont différentes quand il s'agit de les calculer dans le contexte d'une tâche d'estimation de mélodie ou lorsqu'elles sont présentées dans le cadre d'une tâche de détection de voix uniquement. Dans ce dernier cas, les mesures communes sont le taux de bonne classification, le rappel, la précision et la F-mesure. Il est d'usage dans ce contexte de ne pas moyenner sur les fichiers constituant la base mais de considérer l'ensemble des trames pour évaluer les performances. Nous faisons donc de même pour présenter des résultats de détection de voix qui ne sont pas placés dans le contexte de l'estimation de mélodie.

3.4 Evaluation de la méthode d'ADX TRAX

La détection de voix est effectuée dans ADX TRAX en tout début d'algorithme, car elle permet de contraindre par la suite la séparation. Cette détection utilise un système de classification à partir de descripteurs spectraux afin de décider pour chaque trame de signal la présence ou l'absence de voix. Les trois étapes mises en jeu successivement dans ADX TRAX sont : détection de voix, séparation de la voix chantée, estimation de la mélodie. L'étape de détection de voix est donc primordiale car elle contraint tout le processus, c'est pour cette raison que nous nous sommes attachés à l'évaluer seule.

Le classifieur jusqu'à alors utilisé avait été entraîné et évalué uniquement avec les bases Jamendo, composées de 61 fichiers pour l'entraînement, et de deux fois 16 fichiers pour la base de validation et la base de test. On peut observer tableau 3.1 les résultats du système de classification mis en place à partir des bases Jamendo, pour différentes méthodes de l'état de l'art présentées précédemment [32, 36, 37] et pour celle utilisée dans ADX TRAX. On note que les résultats de cette dernière sont assez faibles comparés aux méthodes de l'état de l'art et qu'une amélioration du système de détection de voix d'ADX TRAX serait très bénéfique au processus complet de séparation et d'estimation de mélodie comme ceux-ci sont contraints par la décision présence/absence de voix.

Tableau 3.2, les résultats sont présentés pour la base de test élargie [TestDVC] (cf. annexe A pour l'ensemble des bases de données). On remarque, en comparaison avec le tableau 3.1 que toutes les mesures, excepté la précision, diminuent lorsque la méthode d'ADX Trax est évaluée sur cette nouvelle base plus grande mais toujours entraînée avec Jamendo. Nous supposons alors que cela vient d'un problème de généralisation et qu'en définissant une nouvelle base d'apprentissage [ApprentissageDVC] plus conséquente nous obtiendrons de meilleurs performances. En entraînant le classifieur avec la base [ApprentissageDVC], on obtient les résultats indiqués dans la deuxième colonne du tableau 3.2. On remarque une légère amélioration des performances, particulièrement pour la mesure de rappel, se répercutant également sur la F-mesure. Les nouvelles données d'apprentissage pour la détection de voix ont été intégrées au logiciel ADX TRAX comme les performances ont été améliorées en augmentant la base d'apprentissage.

TABLE 3.1 – Résultats détection de voix avec les bases de données Jamendo

	Ramona [32]	Lehner2013 [36]	Lehner2014 [37]	ADX TRAX
Taux de bonne classification (%)	82.2	84.8	88.2	76.93
Rappel (%)	n/a	90.4	86.2	96.89
Précision (%)	n/a	79.5	88	67.55
F-mesure	84.3	84.6	87.1	79.6

TABLE 3.2 – Résultats de la détection de voix d'ADX Trax avec les nouvelles bases de données (cf. annexe A)

base d'entraînement	Jamendo	[ApprentissageDVC]
base de test	[TestDVC]	[TestDVC]
Taux de bonne classification (%)	73.31	73.72
Rappel (%)	92.86	95.20
Précision (%)	67.78	67.56
F-mesure	78.36	79.03

3.5 Conclusion

La détection de voix est primordiale dans l'algorithme d'ADX TRAX pour séparer la voix chantée et également pour extraire la mélodie car c'est le premier bloc de traitement. Nous avons vu dans le chapitre 2 que cette étape est essentielle pour obtenir de bonnes performances d'estimation de mélodie, et qu'à la fois la méthode de Tachibana et d'ADX TRAX présentent de bons résultats d'estimation de la ligne mélodique (*raw pitch accuracy*) indépendamment de la détection de voix. Ces considérations nous amènent à nous concentrer sur l'élaboration d'un système robuste de détection de voix. La direction vers laquelle nous nous tournons est dans la continuité de la méthode de Tachibana. Nous souhaitons en effet utiliser les signaux obtenus par double HPSS, cette étape constituant un pré-traitement intéressant pour extraire de l'information automatiquement dans la musique. La décision absence/présence de mélodie actuellement utilisée par la méthode de Tachibana peut-être vue comme un système de classification très simple, à un descripteur, qui est la valeur pour chaque trame de signal de la distance de Mahalanobis entre les deux spectrogrammes associés aux signaux issus de la double HPSS. L'idée serait donc toujours d'exploiter l'information disponible à partir de ce pré-traitement, mais cette fois avec un classifieur plus élaboré. Notons de plus que cette stratégie s'intègrerait facilement avec le processus de séparation automatique de la voix chantée mis en place dans ADX TRAX. Finalement, en prenant cette direction nous faisons également le choix d'aller vers une méthode efficace en temps de calcul, puisque comme nous l'avons vu, la méthode de Tachibana est rapide (0.5s de calcul pour 1s de signal).

Nous souhaitons donc mettre en oeuvre un système robuste de classification automatique pour la détection de la voix chantée. Pour cela, nous choisissons d'explorer une approche d'apprentissage profond. L'objectif de la suite de ce rapport sera de décrire le fonctionnement des réseaux de neurones et l'architecture que nous avons choisie. Finalement nous présenterons et évaluerons la nouvelle méthode mise en place pour la détection de voix. Nous verrons qu'elle améliore significativement l'état de l'art sur la base de données Jamendo.

Chapitre 4

Réseaux de neurones

Les réseaux de neurones artificiels sont constitués d'un assemblage d'opérateurs élémentaires inter-connectés. A l'origine, ils correspondent à un modèle mathématique du traitement de l'information effectué par le cerveau. Chaque opérateur élémentaire peut être considéré comme une représentation du fonctionnement d'un neurone biologique, ces unités de calcul sont appelées "neurones formels" par McCulloch et Pitts en 1943. Un neurone biologique possède des entrées (les dendrites), une sortie (l'axone) et une partie centrale appelée corps cellulaire (le soma). Les neurones sont inter-connectés grâce aux synapses, celles-ci désignent les zones de contact entre deux neurones ou entre un neurone et une autre cellule (récepteur sensoriel, cellule musculaire,...). Quand la somme des potentiels provenant des entrées d'un neurone biologique atteint un certain seuil, un potentiel d'action (impulsion électrique) est envoyé le long de l'axone vers la sortie. C'est sur ce schéma qu'est défini le fonctionnement d'un neurone formel.

La première réalisation majeure de système de classification à base de réseaux de neurones correspond au perceptron de Rosenblatt (1958) [38], c'est le premier modèle auquel est associé un processus d'apprentissage des paramètres. Il correspond à un système de classification linéaire. Dans sa version la plus simple il ne contient qu'un unique neurone formel.

Dans ce chapitre nous présentons dans un premier temps le fonctionnement d'un réseau de neurones par couches, appelé perceptron multi-couches. Nous verrons comment un tel modèle est utilisé pour définir un système de classification et quelle méthode d'entraînement peut-être utilisée. Nous détaillerons ensuite le fonctionnement des réseaux de neurones récurrents, notamment les architectures utilisant des blocs longue mémoire à court-terme. Celles-ci permettent de résoudre des tâches de classification de séquences en prenant en compte un contexte temporel pour classer un vecteur d'entrée. Finalement nous discuterons des motivations et difficultés liées aux architectures profondes, c'est-à-dire comprenant plusieurs couches cachées.

4.1 Architecture d'un perceptron multi-couches

Un réseau de neurones associe une entrée (vecteur d'état) à une sortie (étiquette scalaire ou vecteur étiquette). Pour cela le réseau effectue un certain nombre d'opérations internes, à partir d'un élément de base capable d'effectuer des opérations simples, c'est le neurone formel. Le réseau comprend un assemblage de neurones formels. Cette assemblage constitue l'architecture (ou la structure) du réseau qu'il est donc nécessaire de spécifier selon la tâche considérée. Une fois l'architecture fixée, la phase d'apprentissage permet d'optimiser les paramètres du réseau afin que la sortie soit la plus proche de celle désirée. Aucune hypothèse n'est à faire pour définir un réseau de neurones, c'est uniquement à partir de la base d'entraînement étiquetée que chaque neurone est optimisé. Une fois que le réseau est appris, la même architecture est utilisée pour classer un nouvel échantillon.

Cette section présente le neurone formel et un type d'architecture spécifique correspondant aux réseaux de neurones par couches, appelés également perceptrons, adaptés à des tâches de classification.

4.1.1 Le neurone formel

Un neurone formel j, illustré figure 4.1, possède plusieurs entrées et une unique sortie, il est décrit par :

- Ses I entrées x_i.
- Son activité a_j correspondant à une somme pondérée de ses entrées :

$$a_j = \sum_{i=1}^{I} w_{ij} x_i \tag{4.1.1}$$

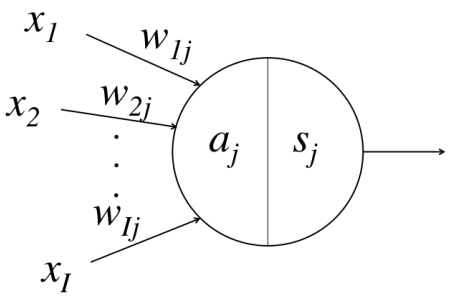
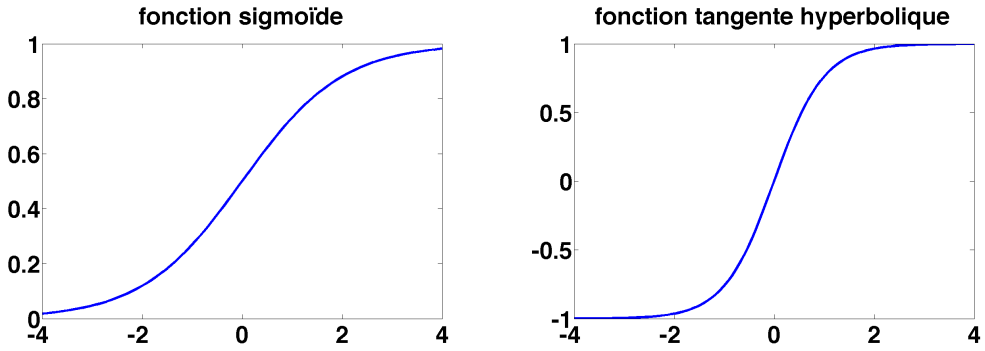


FIGURE 4.1 – Neurone formel



(a) Sigmoïde (b) Tangente hyperbolique

FIGURE 4.2 – Fonctions de transition usuelles

- Sa sortie s_j qui est le résultat de l'application d'une fonction de transition f_j à l'activité a_j :

$$s_j = f_j(a_j) \tag{4.1.2}$$

Un neurone formel est donc essentiellement caractérisé par ses poids w_{ij} qui correspondent aux paramètres à estimer lors de la phase d'apprentissage.

Si à l'origine, la fonction de transition utilisée par Rosenblatt était une fonction seuil, les fonctions de transition les plus communes sont aujourd'hui la tangente hyperbolique (figure 4.2b) :

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{4.1.3}$$

et la fonction sigmoïde (figure 4.2a) :

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{4.1.4}$$

Ces deux fonctions étant reliées linéairement par $\tanh(x) = 2\sigma(2x) - 1$, un réseau construit avec l'une des deux fonctions de transition sera totalement équivalent à un autre construit avec la seconde fonction de transition.

Le rôle de ces fonctions est d'introduire des éléments non-linéaires. Un modèle non-linéaire, qu'il soit destiné à une tâche de classification ou à une tâche de régression sera beaucoup plus puissant qu'un modèle linéaire, dans le premier cas il sera capable de résoudre des problèmes de classification non-linéairement séparables, dans le second il pourra modéliser des fonctions non-linéaires. Un autre avantage de ces fonctions de transition est qu'elles sont dérivables, ce qui permettra d'entraîner le réseau par un algorithme de descente de gradient.

Leurs dérivées sont respectivement pour la tangente hyperbolique et pour la sigmoïde :

$$\tanh'(x) = 1 - \tanh(x)^2 \tag{4.1.5}$$

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)) \tag{4.1.6}$$

4.1.2 Perceptron de Rosenblatt et introduction du biais

Le réseau le plus simple correspond au perceptron de Rosenblatt constitué d'un unique neurone formel. Il permet de résoudre des problèmes de classification binaire linéairement séparables en apprenant l'hyperplan H d'équation $\sum_{i=1}^{I} w_i x_i + b = 0$ tel que pour tout $\mathbf{x}$ de la classe 1, $\sum_{i=1}^{I} w_i x_i + b > 0$, et pour tout $\mathbf{x}$ de la classe 2, $\sum_{i=1}^{I} w_i x_i + b < 0$.

Le vecteur étendu $\mathbf{w_e} = (w_1, w_2, ..., w_I, b)$ représente le vecteur orthogonal à la direction de H. Il caractérise donc l'hyperplan séparateur. Les éléments de ce vecteur sont les paramètres à apprendre.

On remarque la nécessité d'augmenter la dimension du vecteur d'entrée $\mathbf{x}$ en rajoutant un coefficient constant à 1 afin de permettre au réseau d'apprendre le biais b de la même façon que les poids w_i. Ce biais peut être vu comme permettant de décaler horizontalement la fonction de transition. Dans la suite de ce rapport, pour ne pas avoir à différencier l'apprentissage des poids de celui du biais, nous n'écrirons pas ce dernier. Cependant, il ne faudra pas oublier que lors de la conception du réseau il est nécessaire d'ajouter une entrée unité pour chaque couche, donc pour pour chaque neurone.

4.1.3 Perceptron multi-couches

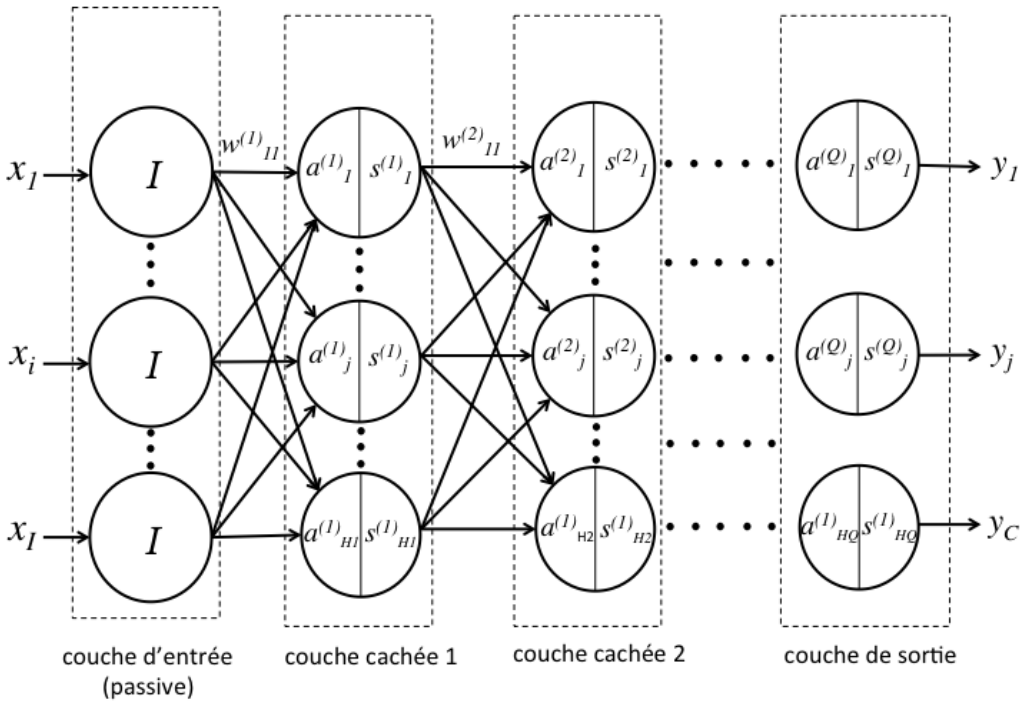


FIGURE 4.3 – Perceptron multi-couches

Toute architecture peut-être considérée pour connecter les neurones formels entre eux afin de construire un réseau. Cependant, nous allons nous attacher à décrire le cas où les neurones sont regroupés en couches : les entrées d'un neurone d'une couche l sont les sorties des neurones de la couche $l-1$, comme illustré figure 4.3. On parle alors de réseau ou perceptron multi-couches. Parmi cet ensemble de couches, certaines réalisent l'interface avec "le monde extérieur", la couche d'entrée du réseau est reliée à l'espace des données, c'est une couche passive, c'est-à-dire que ses neurones se contentent de recopier leur entrée en sortie. La couche de sortie quant à elle est reliée par exemple à l'espace des classes. Les autres neurones organisés par couches n'étant pas visibles, on parle de couches cachées. Chaque couche l possède H_{l-1} entrées $e_j^{(l)} = s_j^{(l-1)}$, H_l unités d'activités $a_j^{(l)}$ et H_l sorties $s_j^{(l)}$. $H_0 = I$ est la dimension de l'espace d'état, $e_j^{(1)} = s_j^{(0)} = x_j$ est l'élément j du vecteur d'état en entrée. Pour un réseau à Q couches, $H_Q = C$ est le nombre de sorties et $s_j^{(Q)} = y_j$ est l'élément j du vecteur de sortie.

Considérons la couche l d'un perceptron multi-couches comprenant H_l neurones. Pour tout neurone j de cette couche, $j \in \{1, ..., H_l\}$, on peut calculer son activité $a_j^{(l)}$:

$$a_j^{(l)} = \sum_{i=1}^{H_{l-1}} w_{ij}^{(l)} e_i^{(l)} = \sum_{i=1}^{H_{l-1}} w_{ij}^{(l)} s_i^{(l-1)} \tag{4.1.7}$$

Rappelons que pour le calcul de l'activité du neurone j de la couche l, un biais est bien pris en compte de façon implicite, il est confondu avec un poids associé à la connexion entre le neurone j et une entrée de valeur unité.

Connaissant la fonction de transition de transition $f_{l,j}$ du neurone j, on calcule sa sortie $s_j^{(l)}$:

$$s_j^{(l)} = f_{l,j}(a_j^{(l)}) = f_{l,j}(\sum_{i=1}^{H_{l-1}} w_{ij}^{(l)} s_i^{(l-1)}) \tag{4.1.8}$$

On voit à partir de l'expression précédente que le calcul des sorties de la couche l nécessite de connaître celles de la couche $l-1$, on parle de propagation.

4.1.4 Cas particulier de la couche de sortie

Le vecteur de sortie du réseau est donné par les sorties des neurones de la dernière couche. Le nombre de neurones dans cette dernière couche et le choix de la fonction de transition qui leur est associée dépendent de la tâche considérée.

Pour une tâche de classification binaire, c'est-à-dire à deux classes, la configuration standard pour la couche de sortie est un unique neurone d'activité a et de sortie y, avec une fonction de transition sigmoïde (équation (4.1.4)). Puisque cette fonction est à valeur dans $[0,1]$, on peut directement interpréter la sortie de l'unique neurone sur la dernière couche comme la probabilité que le vecteur d'entrée appartienne à la première classe (on en déduit également celle d'appartenance à la seconde classe) :

$$\begin{aligned} p(c=1|\mathbf{x}) &= y = \sigma(a) \\ p(c=2|\mathbf{x}) &= 1-y \end{aligned} \tag{4.1.9}$$

Pour une classification à $C > 2$ classes, le réseau aura C neurones sur sa couche de sortie. Les sorties y_c, $c \in \{1,...,C\}$ correspondent aux activités a_c auxquelles est appliquée la fonction de transition *softmax*, ainsi elles appartiennent à l'intervalle $[0,1]$ et peuvent être interprétées comme une estimation d'appartenance du vecteur $\mathbf{x}$ à la classe c :

$$p(c|\mathbf{x}) = y_c = \frac{\mathrm{e}^{a_c}}{\sum_{k=1}^{C} \mathrm{e}^{a_k}} \tag{4.1.10}$$

Ces spécifications d'architecture de la couche de sortie correspondent exactement au schéma de classification probabiliste que nous allons définir section 4.2.2.2. On utilisera alors pour entraîner le réseau l'erreur d'entropie croisée : équation (4.2.13) pour le cas à deux classes ou équation (4.2.9) dans le cas général.

4.2 Réseaux de neurones et système de classification

Le cadre de l'utilisation des réseaux de neurones en classification est celui de l'apprentissage supervisé, pour lequel on dispose d'une base de données étiquetées servant à apprendre les paramètres du modèle. A chaque individu de la base est associée une classe parmi un ensemble de C classes. On dispose de R individus, chaque individu p, $p \in \{1,...,R\}$, est décrit par un vecteur d'état $\mathbf{x_p}$ appartenant à l'espace d'état de dimension I. Un vecteur d'état $\mathbf{x_p}$ a donc I composantes $x_{i,p}$, $i \in \{1,...,I\}$. A chaque individu représenté par $\mathbf{x_p}$ est associée une étiquette d_p correspondant au numéro de la classe donc une valeur entière entre 1 et C. Afin d'alléger les notations nous omettons par la suite l'indice p.

4.2.1 Taux de bonne classification

Entraîner un système de classification correspond à apprendre les paramètres qui le caractérise de telle sorte qu'idéalement, pour un individu étiqueté, la grandeur $f(\mathbf{x})$ qui lui est associée par le système soit l'étiquette fournie par la base d'apprentissage. De ce fait, on peut définir une première mesure de performance du classifieur f qui correspond à analyser ses sorties pour la totalité de la base d'apprentissage et à compter les individus mal classés. Ceci correspond à une définition d'erreur de classification $E^{class}(f,S)$ pour un classifieur f et un ensemble S de paires entrée-étiquette $(\mathbf{x},d)$:

$$E^{class}(f,S) = \frac{1}{R} \sum_{(\mathbf{x},d)\in S} \begin{cases} 0 & \text{si } f(\mathbf{x}) = d \\ 1 & \text{sinon} \end{cases} \tag{4.2.1}$$

On peut également définir le taux de bonne classification $A^{class} = 1 - E^{class}$.

Un outil de classification associe une étiquette à une entrée quelconque. On attend en général d'un tel système qu'il se généralise à des entrées n'appartenant pas à la base d'apprentissage. Afin de mesurer cette capacité de généralisation, on définit généralement une base de test S' composée également de paires entrée-étiquette mais n'ayant pas été utilisées pour l'apprentissage. On pourra alors mesurer les performances du classifieur à l'aide de cette base en calculant $E^{class}(f,S')$. La représentativité de la base d'apprentissage est un élément essentiel pour obtenir un classifieur présentant de bonnes capacités de généralisation.

4.2.2 Fonction de coût

Le taux de bonne classification est insuffisant pour caractériser les performances d'un classifieur. Soit par exemple le cas d'un système de classification à $C = 3$ classes. A chaque individu $\mathbf{x}$ n'est pas associée directement la classe d'appartenance mais un vecteur de sortie réel $\mathbf{y} \in [0,1]^C$ tel que l'individu soit classé dans la classe j si $\max_{i\in\{1,...,C\}} y_i = y_j$. On a donc la sortie du classifieur $f(\mathbf{x}) = \arg \max_{i\in\{1,...,C\}} y_i$.

On définit le vecteur cible $\mathbf{t} \in \{0,1\}^C$ associé à l'individu $\mathbf{x}$ et permettant d'encoder sa classe d'appartenance tel que :

$$t_i = \begin{cases} 1 & \text{si } d = i \\ 0 & \text{sinon} \end{cases} \tag{4.2.2}$$

avec $i \in \{1,...,C\}$.

TABLE 4.1 – Exemple de classification 1

Numéro d'individu	Vecteur de sortie $\mathbf{y}$	Vecteur étiquette $\mathbf{t}$	Classification correcte ?	Taux de bonne classification
p = 1	(0.1,0.2,0.7)	(1,0,0)	non	33%
p = 2	(0.3,0.4,0.3)	(0,1,0)	oui	
p = 3	(0.3,0.3,0.4)	(0,0,1)	oui	

TABLE 4.2 – Exemple de classification 2

Numéro d'individu	Vecteur de sortie $\mathbf{y}$	Vecteur étiquette $\mathbf{t}$	Classification correcte ?	Taux de bonne classification
p = 1	(0.3,0.4,0.3)	(1,0,0)	non	33%
p = 2	(0.1,0.7,0.2)	(0,1,0)	oui	
p = 3	(0.1,0.2,0.7)	(0,0,1)	oui	

Les tableaux 4.1 et 4.2 illustrent (pour $R = 3$ individus) le fait qu'il soit possible d'avoir le même taux de bonne classification pour deux systèmes de décision aux performances différentes. En observant les vecteurs des sorties $\mathbf{y}$, on voit que l'outil de classification associé au tableau 4.2 présente de meilleurs performances. En effet, la décision de classification se fait en cherchant le maximum du vecteur de sortie. On souhaite donc que lorsque le classifieur effectue une décision correcte, cette valeur soit la plus grande possible, et lorsqu'il se trompe, cette valeur doit être la plus faible possible. Le taux de bonne classification ne nous permet pas de mesurer cela.

Il est nécessaire de définir une fonction de coût afin de prendre en compte ce critère de performance. Cette fonction de coût est essentielle, car elle sera prise en compte dans l'algorithme d'apprentissage des paramètres du modèle. On cherchera en effet à estimer les paramètres qui la minimise. Du fait de la minimisation, il est nécessaire que la fonction de coût présente de bonnes propriétés de dérivabilité.

4.2.2.1 Erreur quadratique

Plusieurs fonction de coût peuvent être utilisées mais un premier choix, peut-être le plus naturel, correspond à une mesure d'erreur quadratique. En effet, on peut considérer le réseau de neurones comme un outil de régression, c'est-à-dire un système d'approximation d'une fonction de ses entrées. Dans le cas de l'apprentissage d'une courbe dans le plan, il existe un polynôme de degré R passant exactement par les R points décrits par la base d'apprentissage, mais un tel polynôme, très oscillatoire, ne permettra aucune généralisation à un nouvel individu. On cherche donc plutôt à définir une courbe plus simple passant à peu près par les R points initiaux.

L'erreur quadratique prend en compte cette notion de distance et surtout, elle permet de mettre en place un calcul de type minimisation au sens des moindres carrés.

On peut par exemple utiliser la Somme des Carrés des Résidus (*Sum of Squared Error*) (SSE) :

$$E_{SSE} = \sum_{S} \sum_{i=1}^{C} (y_i - t_i)^2 \tag{4.2.3}$$

Comme on mesure des carrés, on majore l'importance des grands écarts entre observations et prévisions. Afin de ne pas prendre en compte la taille de la base dans le calcul des performances, il est également possible de calculer l'Erreur Quadratique Moyenne (*Mean Squared Error*) (MSE) $E_{MSE} = \frac{1}{R} E_{SSE}$, le problème d'optimisation sera en revanche exactement identique. Finalement, on peut également calculer la Racine Carrée de l'Erreur Quadratique Moyenne (*Root Mean Squared Error*) (RMSE) $E_{RMSE} = \sqrt{E_{MSE}}$.

Pour les deux exemples précédents, on a dans le cas du tableau 4.1 $E_{SSE} = 2.42$ et pour le tableau 4.2 $E_{SSE} = 1.02$, cette mesure d'erreur reflète donc bien le fait que le deuxième exemple de classification soit plus performant.

4.2.2.2 Entropie croisée

Système de classification probabiliste :

L'outil de classification peut être vu comme un moyen d'estimer la probabilité conditionnelle des C classes $p(c|\mathbf{x})$, $c \in \{1, ..., C\}$, étant donné un vecteur d'entrée inconnu $\mathbf{x}$. La classe qui maximise cette probabilité est alors associée à la sortie du classifieur :

$$f(\mathbf{x}) = \arg \max_{c \in \{1,...,C\}} p(c|\mathbf{x}) \tag{4.2.4}$$

Ce schéma probabiliste que nous utiliserons par la suite dans la mise en oeuvre des réseaux de neurones permet de définir une stratégie d'entraînement du classifieur basée sur le maximum de vraisemblance.

Entraînement :

Si le système de classification est caractérisé par un vecteur de paramètres $\boldsymbol{\theta}$, on peut estimer les paramètres optimaux $\boldsymbol{\theta}_{ML}$ au sens du maximum de vraisemblance à partir de l'ensemble des paires entrée-étiquette $(\mathbf{x}, d)$ de la base d'apprentissage S, considérées indépendantes et identiquement distribuées (iid) :

$$\boldsymbol{\theta}_{ML} = \arg \max_{\boldsymbol{\theta}} p(S|\boldsymbol{\theta}) = \arg \max_{\boldsymbol{\theta}} \prod_{(\mathbf{x},d) \in S} p(d|\mathbf{x}, \boldsymbol{\theta}) \tag{4.2.5}$$

Afin d'estimer les paramètres $\boldsymbol{\theta}_{ML}$, on cherche à maximiser la vraisemblance $p(S|\boldsymbol{\theta})$, ce qui est équivalent à maximiser la log-vraisemblance $\ln p(S|\boldsymbol{\theta})$ ou encore à minimiser la fonction de coût suivante :

$$\mathcal{L}(\boldsymbol{\theta}) = -\ln \prod_{(\mathbf{x},d) \in S} p(d|\mathbf{x}, \boldsymbol{\theta}) = -\sum_{(\mathbf{x},d) \in S} \ln p(d|\mathbf{x}, \boldsymbol{\theta}) \tag{4.2.6}$$

Entropie croisée :

A partir d'un vecteur d'entrée $\mathbf{x}$ de la base d'apprentissage S, le classifieur défini par son vecteur de paramètres $\boldsymbol{\theta}$ calcule un vecteur de sortie $\mathbf{y}$ tel que $p(d = j|\mathbf{x}, \boldsymbol{\theta}) = y_j$ avec $j \in \{1, ..., C\}$.

En utilisant un schéma de codage de la classe d'appartenance de $\mathbf{x}$ défini par le vecteur $\mathbf{t}$ tel que

$$t_j = \begin{cases} 1 & \text{si } d = j \\ 0 & \text{sinon} \end{cases} \tag{4.2.7}$$

on obtient :

$$p(d|\mathbf{x}, \boldsymbol{\theta}) = p(\mathbf{t}|\mathbf{x}, \boldsymbol{\theta}) = \prod_{j=1}^{C} (y_j)^{t_j} \tag{4.2.8}$$

En injectant ce résultat dans l'équation (4.2.6), on obtient :

$$\mathcal{L}(\boldsymbol{\theta}) = -\sum_{(\mathbf{x},d) \in S} \sum_{j=1}^{C} t_j \ln y_j \tag{4.2.9}$$

La fonction de coût définie ci-dessus est appelée erreur d'entropie croisée. Si nous appliquons cette erreur pour l'exemple présenté plus haut, on trouve pour le tableau 4.1 $\mathcal{L} = 1.38$ et pour le tableau 4.2 $\mathcal{L} = 0.64$.

L'Erreur d'entropie croisée se simplifie dans le cas d'une classification binaire, c'est-à-dire à deux classes. En effet, le classifieur n'a besoin que d'une seule sortie y telle qu'on considère :

$$p(d = 1|\mathbf{x}, \boldsymbol{\theta}) = y \tag{4.2.10}$$

et

$$p(d = 2|\mathbf{x}, \boldsymbol{\theta}) = 1 - y \tag{4.2.11}$$

Dans ce cas, en utilisant le schéma de codage $t = 1$ si la classe correcte est $d = 1$ et $t = 0$ si la classe correcte est $d = 2$, les deux expressions (4.2.10) et (4.2.11) peuvent être combinées pour écrire :

$$p(d|\mathbf{x}, \boldsymbol{\theta}) = p(t|\mathbf{x}, \boldsymbol{\theta}) = y^t(1 - y)^{1-t} \tag{4.2.12}$$

On obtient alors grâce à l'équation (4.2.6) l'erreur d'entropie croisée dans le cas d'une classification binaire :

$$\mathcal{L}(\boldsymbol{\theta}) = -\sum_{(\mathbf{x},d)\in S} t \ln y + (1 - t) \ln(1 - y) \tag{4.2.13}$$

Nous avons vu que le taux de classification n'était pas suffisant pour caractériser les performances d'un système de classification et qu'il était nécessaire de définir une fonction de coût plus représentative. Au travers d'un exemple, nous avons vu qu'à la fois une erreur quadratique et l'erreur d'entropie croisée permettait de prendre en compte le fait que le système de classification représenté par le tableau 4.2 soit plus robuste. Cette notion de fonction de coût est de plus indispensable à tout système de classification car elle permet d'entraîner les paramètres du classifieur.

4.3 Algorithme d'apprentissage

Nous présentons dans cette section une technique d'apprentissage des poids d'un réseau de neurones, ceci dans le cadre d'un schéma de classification probabiliste tel que défini précédemment.

4.3.1 Rétropropagation de l'erreur

Du fait des fonctions de transition que nous avons définies précédemment, les perceptrons multi-couches sont des opérateurs différentiables. Ils peuvent donc être entraînés pour minimiser une fonction de coût à partir d'un algorithme de descente de gradient. L'idée est de parcourir l'espace des poids afin de trouver le point associé à un minimum de l'erreur. Pour cela, il est nécessaire de calculer la dérivée de la fonction de coût par rapport à chacun des poids du réseau et de mettre à jour la valeur de ces poids selon le sens inverse du gradient.

Les erreurs d'entropie croisée telles que nous les avons définies précédemment contiennent une somme sur l'ensemble d'apprentissage S, c'est-à-dire qu'on évalue l'erreur sur la totalité de la base. De ce fait, le gradient de la fonction de coût est la somme de tous les gradients partiels calculés pour chacun des exemples de la base d'apprentissage : $\nabla\mathcal{L}(\boldsymbol{\theta}) = \sum_{(\mathbf{x},d)\in S} \nabla\mathcal{L}_{(\mathbf{x},d)}(\boldsymbol{\theta})$. Afin d'alléger les notations, nous considérerons à partir de maintenant la fonction de coût par rapport à une unique paire entrée-étiquette et nous la noterons $\mathcal{L}$. Le gradient total pourra être calculé en sommant sur toutes les paires de la base d'apprentissage S. Cette simplification nous permet également de prendre en compte une modalité d'apprentissage échantillon par échantillon, que nous détaillerons dans la section 4.3.2 liée à l'algorithme de descente de gradient.

Nous allons présenter dans cette section le calcul de la dérivée partielle de l'erreur par rapport à chaque poids du réseau. Nous verrons qu'il s'agit en fait de rétropropager la dérivée de l'erreur de la couche de sortie vers la couche d'entrée, en utilisant à chaque fois le théorème de dérivation des fonctions composées.

A partir des équations (4.2.9) et (4.2.13), on a l'expression de l'erreur d'entropie croisée pour une unique paire entrée-étiquette dans le cas d'une classification binaire où $y = \sigma(a)$ correspond à la sortie de l'unique neurone de sortie :

$$\mathcal{L} = -t \ln y + (1 - t) \ln(1 - y) \tag{4.3.1}$$

et pour une classification à C classes donc pour un réseau à C sorties y_c :

$$\mathcal{L} = -\sum_{k=1}^{C} t_k \ln y_k \tag{4.3.2}$$

Dérivée partielle de la fonction de coût par rapport à un poids $w_{ij}^{(l)}$ de la couche l :

On a l'expression de la dérivée de la fonction de coût par rapport au poids $w_{ij}^{(l)}$:

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^{(l)}} = \frac{\partial \mathcal{L}}{\partial s_j^{(l)}} \frac{\partial s_j^{(l)}}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial w_{ij}^{(l)}} \tag{4.3.3}$$

En dérivant l'équation (4.1.7) et en posant l'expression des "deltas" telle que

$$\delta_j^{(l)} = \frac{\partial \mathcal{L}}{\partial a_j^{(l)}} = \frac{\partial \mathcal{L}}{\partial s_j^{(l)}} \frac{\partial s_j^{(l)}}{\partial a_j^{(l)}} \tag{4.3.4}$$

on trouve

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^{(l)}} = \delta_j^{(l)} s_i^{(l-1)} \tag{4.3.5}$$

Cette dernière expression est générale et peut s'appliquer à n'importe quel neurone j du réseau pour calculer la dérivée partielle de la fonction de coût. Cependant, on voit que les termes $\delta_j^{(l)}$ dépendent de la fonction de coût à travers $\mathcal{L}$ et de la fonction de transition du neurone j de la couche l à travers la sortie $s_j^{(l)}$. Il nous faut donc distinguer trois cas pour les calculer :

- δ_q pour la couche de sortie. Il nous faudra considérer deux sous-cas correspondant à la classification à deux classes ou plus.
- $\delta_j^{(Q-1)}$ pour un neurone j de la couche cachée $Q-1$ précédent la couche de sortie.
- $\delta_h^{(l)}$ pour un neurone h de la couche cachée l. Ce cas correspond à une généralisation du précédent.

Dérivée partielle de la fonction de coût au niveau d'un neurone q de la couche de sortie :

Considérons dans un premier temps le calcul de la dérivée de la fonction de coût par rapport aux poids de la couche de sortie, il nous faut calculer δ_q qui dépend de la fonction de coût et de la fonction de transition du neurone q. On distingue donc le problème à deux classes de celui à $C > 2$ classes.

Cas binaire :

On dispose d'un unique neurone sur la dernière couche, d'activité $a_q = a$ et de sortie $y_q = y$, la fonction d'activation est une sigmoïde. Ainsi, en réécrivant l'équation (4.3.4),

$$\delta_q = \delta = \frac{\partial \mathcal{L}}{\partial y} \frac{\partial y}{\partial a} \tag{4.3.6}$$

En dérivant l'équation (4.3.1) par rapport à la sortie du réseau, on trouve

$$\frac{\partial \mathcal{L}}{\partial y} = \frac{y - t}{y(1-y)} \tag{4.3.7}$$

Et en utilisant la dérivée de la fonction sigmoïde définie équation (4.1.6) on obtient finalement

$$\delta = y - t \tag{4.3.8}$$

Cas à $C > 2$ classes :

Pour un problème à $C > 2$ classes on dispose de C neurones sur la dernière couche, d'activités a_q et de sorties y_q, $q \in \{1, ..., C\}$, la fonction de transition est la fonction *softmax* définie équation (4.1.10). Comme chaque sortie y_q dépend de toutes les activations de la couche de sortie a_k, $k \in \{1, ..., C\}$, on a :

$$\delta_q = \sum_{k=1}^{C} \frac{\partial \mathcal{L}}{\partial y_k} \frac{\partial y_k}{\partial a_q} \tag{4.3.9}$$

En dérivant l'équation 4.3.2 par rapport à la sortie k, on trouve

$$\frac{\partial \mathcal{L}}{\partial y_k} = -\frac{t_k}{y_k} \tag{4.3.10}$$

A partir de la dérivée de la fonction *softmax*[1] on obtient

$$\frac{\partial y_k}{\partial a_q} = y_k(\delta_{qk} - y_q) \tag{4.3.11}$$

où δ_{qk} est la fonction de dirac égale à 1 si $q = k$.

En injectant les équations 4.3.10 et 4.3.11 dans 4.3.9 et en utilisant le fait que $\sum_{k=1}^{C} t_k = 1$ on a l'expression de δ_q qui est très similaire au cas binaire (équation (4.3.8))

$$\delta_q = y_q - t_q \tag{4.3.12}$$

Dérivée de la fonction de coût au niveau d'un neurone j de la couche cachée $Q-1$:

On cherche maintenant à calculer le terme $\delta_j^{(Q-1)}$ pour un neurone j de la couche cachée $Q-1$ précédent celle de sortie, on réécrit pour ce cas l'équation (4.3.4)

$$\delta_j^{(Q-1)} = \frac{\partial \mathcal{L}}{\partial s_j^{(Q-1)}} \frac{\partial s_j^{(Q-1)}}{\partial a_j^{(Q-1)}} \tag{4.3.13}$$

Sachant que la fonction de coût $\mathcal{L}$ dépend uniquement du neurone j de la couche $Q-1$ à travers son influence sur les neurones q de la couche de sortie, on a

$$\frac{\partial \mathcal{L}}{\partial s_j^{(Q-1)}} = \sum_{q=1}^{C} \frac{\partial \mathcal{L}}{\partial a_q} \frac{\partial a_q}{\partial s_j^{(Q-1)}} = \sum_{q=1}^{C} \delta_q \frac{\partial a_q}{\partial s_j^{(Q-1)}} \tag{4.3.14}$$

Or d'après l'équation (4.1.7),

$$\frac{\partial a_q}{\partial s_j^{(Q-1)}} = w_{jq} \tag{4.3.15}$$

donc

$$\frac{\partial \mathcal{L}}{\partial s_j^{(Q-1)}} = \sum_{q=1}^{C} \delta_q w_{jq} \tag{4.3.16}$$

En considérant que la fonction de transition du neurone j de la couche $Q-1$ est $f_{Q-1,j}$, on a

$$\frac{\partial s_j^{(Q-1)}}{\partial a_j^{(Q-1)}} = f'_{Q-1,j}(a_j^{(Q-1)}) \tag{4.3.17}$$

En injectant les équations (4.3.16) et (4.3.17) dans (4.3.13) on obtient finalement

$$\delta_j^{(Q-1)} = f'_{Q-1,j}(a_j^{(Q-1)}) \sum_{q=1}^{C} \delta_q w_{jq} \tag{4.3.18}$$

Dérivée de la fonction de coût au niveau d'un neurone h de la couche cachée l :

Les termes $\delta_h^{(l)}$ pour chaque neurone h d'une couche cachée l précédant la dernière peuvent être calculés de la même façon, en utilisant les termes $\delta_h^{(l+1)}$ calculés pour la couche supérieure $l+1$ comprenant H_{l+1} neurones :

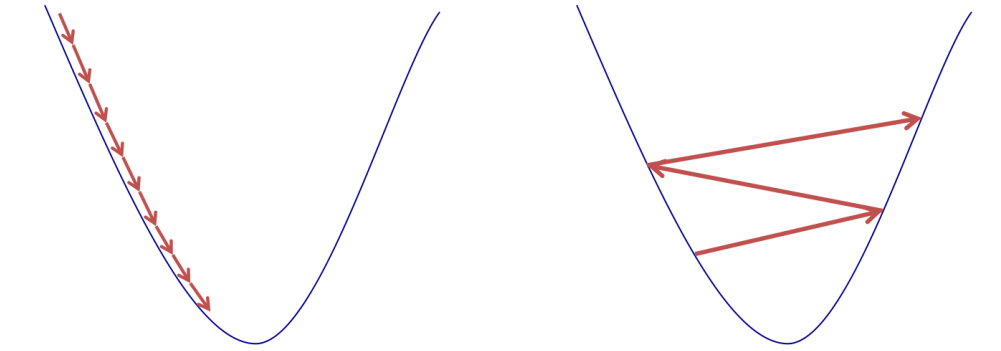
$$\delta_h^{(l)} = f'_{l,h}(a_h^{(l)}) \sum_{j=1}^{H_{l+1}} \delta_j^{(l+1)} w_{hj} \tag{4.3.19}$$

Cette dernière expression généralise à n'importe quelle couche cachée l'équation (4.3.18) associée à la dernière couche cachée, car $H_Q = C$.

On parle de rétropropagation pour calculer la dérivée de la fonction de coût par rapport à chacun des poids du réseau car on commence par calculer les deltas au niveau de la couche de sortie, et on va progressivement vers la première couche.

Finalement, la dérivée de la fonction de coût par rapport à chacun des poids s'obtient grâce à l'expression (4.3.5) et aux deltas calculés pour chaque neurone du réseau.

[1] Considérer les cas $k = c$ et $k \neq c$ pour dériver à partir de l'équation 4.1.10



(a) Trop petit pas de gradient : apprentissage lent (b) Trop grand pas de gradient : divergence

FIGURE 4.4 – Pas du gradient

4.3.2 Descente de gradient

L'objectif de la phase d'apprentissage est de modifier les poids pour arriver à un minimum de la fonction de coût, qui correspond à une mesure d'erreur. Le principe de l'algorithme de descente est de partir d'un point initial dans l'espace des paramètres $\boldsymbol{\theta}$, de trouver une direction dans le sens de la minimisation de l'erreur et de se déplacer d'un pas dans cette direction. La procédure est renouvelée jusqu'à ce qu'un critère d'arrêt soit satisfait (une certaine valeur de la fonction de coût, un nombre d'itérations, stagnation...). Un algorithme de descente de gradient considère comme direction de descente l'opposé du gradient de la fonction de coût. Le calcul des dérivées partielles par rapport à chacun des poids du réseau permet de connaître ce gradient. Le paramètre η est le "pas", il quantifie la taille du saut fait dans la direction opposée au gradient. C'est un paramètre important de l'algorithme car s'il est trop grand, on risque de manquer un bon minimum, voire de faire diverger l'erreur (figure 4.4b), s'il est trop petit, l'apprentissage risque d'être lent (figure 4.4a). Le pas peut être fixé, adaptatif ou planifié pour varier au fur et à mesure de l'apprentissage. Dans tous le cas, le choix du pas de gradient est essentiellement empirique et dépend du problème de minimisation.

Deux modalités opératoires peuvent être envisagées pour l'algorithme de descente de gradient. La première correspond à la descente de gradient stochastique, c'est une approche dite "en ligne" où les poids sont modifiés échantillon par échantillon dans la base d'apprentissage S. Le gradient total est donc approché par celui calculé pour un individu. Nous avions mentionné cette modalité au début de la section précédente lorsque nous avions omis la somme sur S dans l'expression de l'erreur d'entropie croisée. La descente de gradient stochastique est détaillée dans l'algorithme 4.1.

Algorithme 4.1 : Descente de gradient stochastique

```
initialiser les poids
répéter
    pour chaque (x, d) ∈ S faire
        calculer les sorties du réseau
        pour l ⟵ Q à 1 faire
            pour chaque neurone j de la couche l faire
                calculer les termes δ_j^(l)
                pour chaque poids w_ij^(l) faire
                    Δw_ij^(l) ⟵ −η ∂ℒ/∂w_ij^(l) = −η δ_j^(l) s_i^(l−1)
                    w_ij^(l) ⟵ w_ij^(l) + Δw_ij^(l)
jusqu'à condition d'arrêt vérifiée
```

La seconde modalité correspond à la descente de gradient standard détaillée dans l'algorithme 4.2, elle permet une mise à jour globale des poids selon le gradient calculé pour l'ensemble de la base d'apprentissage. Ce gradient total est la somme des gradients individuels pour chaque individu de la base.

Algorithme 4.2 : Descente de gradient standard

initialiser les poids
répéter
  $\Delta w_{ij}^{(l)} \longleftarrow 0$
  pour chaque $(\mathbf{x}, d) \in S$ **faire**
    calculer les sorties du réseau
    pour $l \longleftarrow Q$ **à** 1 **faire**
      pour chaque *neurone* j *de la couche* l **faire**
        calculer les termes $\delta_j^{(l)}$
        pour chaque *poids* $w_{ij}^{(l)}$ **faire**
          $\Delta w_{ij}^{(l)} \longleftarrow \Delta w_{ij}^{(l)} - \eta \frac{\partial \mathcal{L}}{\partial w_{ij}^{(l)}} = \Delta w_{ij}^{(l)} - \eta \delta_j^{(l)} s_i^{(l-1)}$
  pour chaque *poids* $w_{ij}^{(l)}$ **faire**
    $w_{ij}^{(l)} \longleftarrow w_{ij}^{(l)} + \Delta w_{ij}^{(l)}$
jusqu'à *condition d'arrêt vérifiée*

Pour l'algorithme en ligne (algorithme 4.1), la modification du poids $w_{ij}^{(l)}$ prend en compte la dérivée partielle de la fonction de coût par rapport au poids $w_{ij}^{(l)}$ pour un unique individu $(\mathbf{x}, d)$. Alors que dans le cas de l'algorithme standard (algorithme 4.2), les dérivées partielles par rapport à un poids sont cumulées sur l'ensemble de la base d'apprentissage avant de modifier ce poids. On appelle "époque" le fait de visiter l'ensemble des échantillons de la base d'apprentissage. Une troisième modalité appelée descente de gradient stochastique par mini-lots est un intermédiaire entre la descente stochastique et standard. Il s'agit de considérer un certain nombre d'individus (plus de un et moins que la totalité de la base) pour le calcul du gradient et la mise à jour des poids.

4.3.3 Modalités opératoires

4.3.3.1 Descente de gradient standard ou en stochastique

La méthode de descente de gradient stochastique est généralement appliquée dans le cadre de l'apprentissage des réseaux de neurones.

Elle est en générale plus rapide car le nombre de mises à jour est beaucoup plus grand que pour l'algorithme standard, particulièrement lorsqu'on dispose d'une grande base d'apprentissage. En effet, si la base présente un certain nombre d'échantillons redondants, seulement quelques individus permettront de progresser en moyenne autant dans la minimisation de la fonction de coût que pour une mise à jour à partir du gradient calculé sur toute la base.

Un algorithme de descente de gradient ordinaire converge vers un minimum local de la fonction de coût, l'aspect stochastique permet de limiter le risque de rester bloquer dans un mauvais minimum local car ces minimas seront différents pour chaque individu de la base.

Afin d'augmenter encore l'aspect stochastique de cet algorithme et de favoriser l'alternance des classes, il est commun de présenter les échantillons de la base d'apprentissage de façon aléatoire pour la mise à jour des poids.

4.3.3.2 Critère d'arrêt

Il est nécessaire de fixer un critère d'arrêt à la phase d'apprentissage, cela peut-être :

- Un nombre maximal d'époques, permettant ainsi de limiter la durée de la phase d'apprentissage.
- Une valeur minimale de la fonction de coût à atteindre.
- Un taux de bonne classification à atteindre. Mais comme nous l'avions montré au travers d'un exemple section 4.2, ce critère ne reflète pas bien les performances du classifieur en phase d'apprentissage, il est donc préférable d'utiliser la fonction de coût.

Cependant, ces deux dernières démarches ne sont pas correctes, car avec un nombre de poids suffisant on pourra toujours trouver une architecture de réseau permettant d'arriver à un taux de bonne classification de presque 100% ou à une erreur quasi nulle sur la base d'entraînement (rien ne dit qu'on pourra par contre arriver

à entraîner cette architecture). Le critère d'arrêt de la phase d'apprentissage ne peut donc se déduire d'une mesure sur cette base.

4.3.3.3 Base de validation et base de test

Il est indispensable de disposer au minimum d'une autre base étiquetée sur laquelle on évaluera les résultats de l'apprentissage après chaque époque, et ce sera à partir de celle-ci qu'on définira le critère d'arrêt. On appelle ce deuxième ensemble de données "base de validation".

Le réseau est en quelque sorte également appris sur cette base, car même si les poids ne sont pas mis à jour en considérant les individus de la base de validation, on répète leur optimisation tant que cela se fait au bénéfice d'une mesure sur celle-ci. Le risque que le réseau s'ajuste trop aux données (on appelle ce phénomène sur-apprentissage) existe donc également pour une base de validation proche de celle d'apprentissage.

Un troisième ensemble de données, appelé "base de test", permettra d'évaluer à la fin de la phase d'apprentissage les performances du réseau. Cette base est très utile lorsque l'on recherche l'architecture optimale d'un réseau. On apprend chaque architecture grâce à la base d'apprentissage, permettant de trouver les poids qui minimise la fonction de coût, l'apprentissage est arrêté par un critère fixé sur la base de validation, et les performances du modèle de classification ainsi défini sont finalement évaluées sur la base de test. L'architecture optimale du réseau sera celle qui présente les meilleurs performances sur la base de test.

4.3.3.4 Sur-apprentissage

Le risque de sur-apprentissage existe toujours. Il est nécessaire lors de l'entraînement du réseau de neurones de vérifier que le modèle est capable de se généraliser à de nouvelles données non utilisées pour calculer les poids du réseau, c'est le rôle de la base de test. Plusieurs techniques permettent d'éviter de sur-apprendre :

- **Arrêt prématuré** (*early stopping*) : Lorsque le modèle n'est pas trop ajusté à la base d'entraînement, l'erreur (évaluée à partir de la fonction de coût) sur celle-ci et sur la base de validation décroient de la même façon. Lorsque le phénomène de sur-apprentissage apparaît, l'erreur sur la base de validation commence à croître alors qu'elle continue de descendre sur la base d'entraînement (figure 4.5). La technique de l'arrêt prématuré consiste donc à stopper la phase d'apprentissage avant la convergence, quand cet effet d'augmentation de l'erreur sur la base de validation apparaît. Cette technique simple à mettre en oeuvre est très populaire dans le cadre de l'entraînement des réseaux de neurones, car elle s'avère efficace. Cependant, il est parfois difficile de déterminer le moment exacte auquel arrêter l'apprentissage car les performances sur la base de validation ne se dégradent pas forcément clairement, l'erreur pouvant être assez oscillante, notamment pour une descente de gradient stochastique.

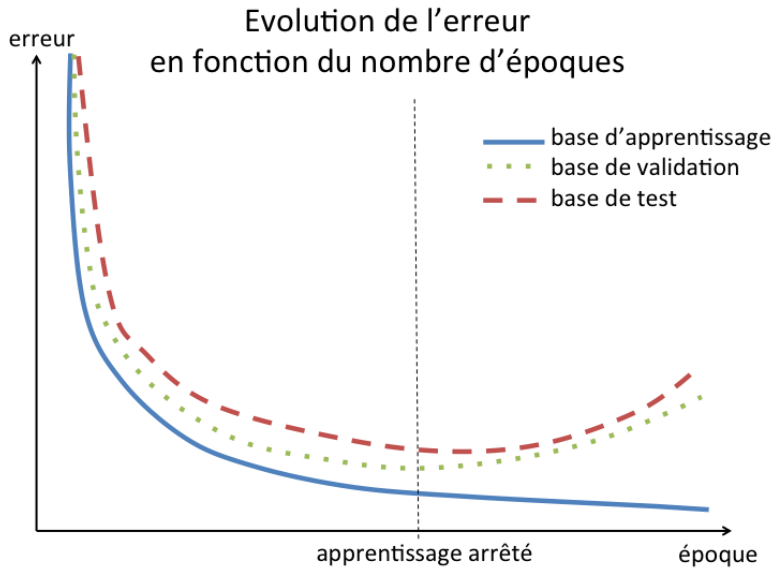


FIGURE 4.5 – Sur-apprentissage

- **Dégradation des poids** (*weight decay*) : Lorsque les poids du réseau ont une grande amplitude, les fonctions de transition auront tendance à agir dans leur zone de saturation et de ce fait à introduire des variations brusques dans la fonction modélisée par le réseau. Afin de limiter le sur-apprentissage, il est

préférable d'avoir une fonction lisse. Pour cela, la technique de régularistaion par dégradation des poids ajoute un terme à la fonction de coût pour limiter l'amplitude des poids du réseau. Ce terme correspond classiquement au carré de la norme L_2 du vecteur de paramètres $\boldsymbol{\theta}$ associé aux P poids :

$$\mathcal{L}_{\mathcal{WD}}(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}) + \lambda \sum_{i=1}^{P} \theta_i^2 \tag{4.3.20}$$

λ est un hyper-paramètre à fixer empiriquement.

On peut également minimiser la somme des valeurs absolues des poids, ce qui correspond à une régularisation L_1.

- **Limiter le nombre de neurones** : Ce n'est pas vraiment une technique pour éviter le sur-apprentissage mais plutôt une règle à respecter. Nous l'avons déjà mentionné précédemment, si le nombre de poids est proche du nombre d'échantillons de la base d'apprentissage alors le modèle aura tendance à trop s'ajuster aux données. Il faut donc essayer d'avoir les meilleurs performances sur la base de test tout en gardant un nombre de paramètres faible, ceci de manière empirique.

4.3.3.5 Problème des minima locaux

Une caractéristique des réseaux de neurones est la difficulté de l'entraînement du fait de la complexité de la fonction de coût, qui s'accroît avec le nombre de couches cachées. Celle-ci étant fortement irrégulière, elle présente de nombreux minima locaux (figure 4.6), ainsi que des plateaux et vallées. Une descente de gradient standard convergera vers le minimum local le plus proche de l'initialisation, alors que l'algorithme stochastique, du fait que le gradient soit bruité (une approximation du gradient total), pourra plus facilement s'échapper des minima locaux.

Cependant, rien ne pourra garantir que le minimum local dans lequel l'algorithme aura convergé (s'il n'est pas stoppé avant convergence) sera associé à de bonnes performances. Mais cela n'est pas un réel problème pour les réseaux de neurones, car on s'attache plus à obtenir de bonnes performances de généralisation qu'à trouver le minimum global de la fonction de coût calculée à partir de la base d'apprentissage. On se contentera donc d'un "bon minimum" ou même uniquement d'une "bonne" valeur d'erreur, qu'elle corresponde à un minimum ou non.

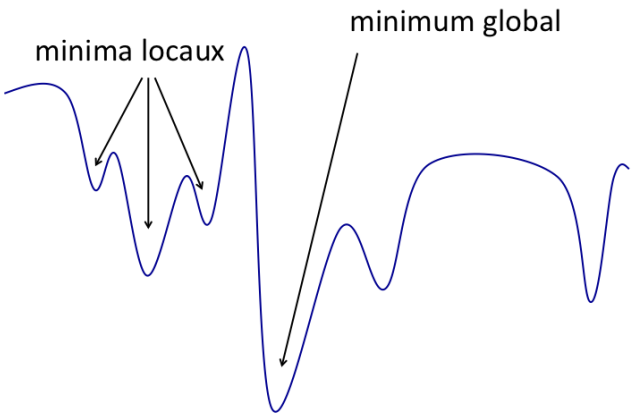


FIGURE 4.6 – Minima locaux et minimum global

4.3.3.6 Ajout d'un terme d'inertie

Lors de la mise à jour des poids du réseau par l'algorithme de descente de gradient stochastique, il est possible d'ajouter un terme d'inertie : le *momentum*. Il s'agit de calculer une moyenne mobile des mises à jours pour chaque individu, c'est donc un moyen de lisser la trajectoire de minimisation (figure 4.7) qui, dans le cas d'un algorithme en ligne, est très mouvementée du fait que le changement des poids ne se fasse pas dans la direction du gradient total. C'est pour cette raison que ce terme d'inertie permettra d'accélérer la phase d'apprentissage. La moyenne mobile est utilisée pour modifier les poids à l'instant τ (pour un individu) :

$$\Delta w_{ij}^{(l)}(\tau) \longleftarrow m\Delta w_{ij}^{(l)}(\tau - 1) - \eta \frac{\partial \mathcal{L}}{\partial w_{ij}^{(l)}(\tau - 1)} \tag{4.3.21}$$

avec $0 \leq m \leq 1$ le *momentum* contrôlant la prise en compte des modifications précédentes pour la mise à jour courante. Si $m = 0$, on est dans le cas de la descente de gradient stochastique standard, plus m se rapproche de un, plus on tient compte des gradients précédents pour mettre à jour les poids. Lorsqu'on utilise un terme d'inertie, on diminuera le pas du gradient pour que le terme de modification des poids ne soit pas trop grand.

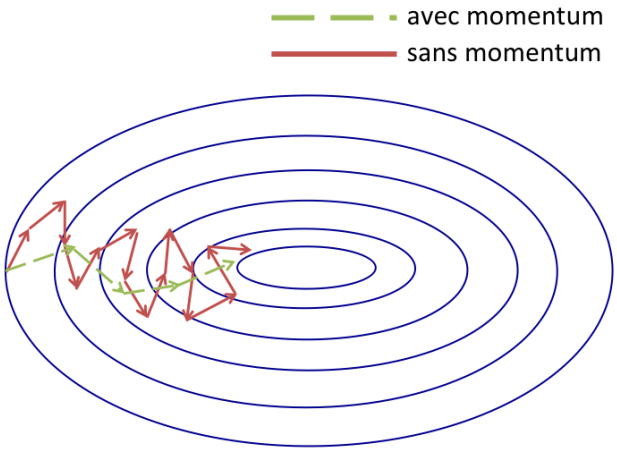


FIGURE 4.7 – Descente de gradient stochastique avec et sans momentum

4.3.3.7 Normalisation des entrées

Les entrées (et les poids comme nous allons le voir après) doivent permettre aux fonctions de transition (prenant en entrée l'activité des neurones) de travailler dans une région non trop linéaire (activité des neurones proche de 0) car sinon on perd l'avantage de la non-linéarité du modèle. Les fonctions de transition ne doivent pas non plus saturer (activité des neurones trop grande), sinon le gradient sera trop faible et la modification des poids non significative lors de la phase d'apprentissage. Un bon choix de mise en forme des données consiste à faire en sorte que sur l'ensemble de la base d'apprentissage, chaque composante du vecteur d'entrée soit de moyenne proche de 0 et de variance proche de 1. La moyenne et l'écart-type calculés pour normaliser les données sur la base d'apprentissage sont utilisés pour faire de même sur la base de validation et la base de test.

4.3.3.8 Initialisation des poids

Toujours afin d'éviter que les fonctions de transition travaillent dans leurs régions extrêmes, il semble raisonnable de souhaiter que les activités des neurones, c'est-à-dire la sommes pondérée des entrées par les poids, soient également centrées-réduites.

Supposons que les poids sont initialisés dans l'intervalle $[-U, U]$. Considérons un neurone j avec I entrées x_i et d'activité $a_j = \sum_{i=1}^{I} w_{ij} x_i$. En considérant les entrées x_i comme indépendantes, centrées et réduites, on peut écrire

$$\mathrm{Var}(a_j) = \sum_{i=1}^{I} \mathrm{Var}(w_{ij} x_i) = \sum_{i=1}^{I} w_{ij}^2 \mathrm{Var}(x_i) \approx \sum_{i=1}^{I} w_{ij}^2 \leq IU^2 \tag{4.3.22}$$

Une intuition pour initialiser les poids serait donc de faire en sorte qu'ils appartiennent à un intervalle proche de $[-\frac{1}{\sqrt{I}}, \frac{1}{\sqrt{I}}]$. Ils peuvent donc être initialisés uniformément dans cet intervalle, mais on peut également envisager de les initialiser selon une loi normale de moyenne nulle et d'écart-type $\frac{1}{\sqrt{I}}$.

4.4 Réseaux de neurones récurrents

Les systèmes de classification automatique dans le domaine du MIR utilisent généralement des vecteurs d'entrée associés à des descripteurs extraits grâce à une analyse à court-terme du signal. Cette analyse se fait en fenêtrant le signal sur une certaine durée autour de différents points d'analyse. La longueur de la fenêtre définie le contexte temporel pris en compte pour l'extraction du descripteur, le taux de recouvrement entre deux fenêtres successives fixe la précision temporelle de l'analyse. L'aspect temporel dans la musique est primordial, chaque vecteur de descripteur ne peut être considéré comme indépendant des précédents ou des suivants, il en est de même pour l'étiquette qui lui est associée dans le cadre d'un problème de classification. Pour un signal de musique, l'ensemble des descripteurs extraits, et les étiquettes associées, forment donc des séquences. Plutôt que de classer un vecteur

d'entrée vers un vecteur de sortie, il paraît plus pertinent de classer une séquence de vecteurs d'entrée vers une séquence de sortie. C'est donc un problème de classification de séquences.

Prenons l'exemple de l'extraction automatique de la mélodie associée à un signal de voix chantée. Les entrées, calculées à partir du signal de voix chantée, sont produites par le mouvement continu de l'appareil phonatoire, et les étiquettes de sortie, correspondent aux notes chantées, généralement organisées selon des règles d'écriture musicale. Dans la musique tonale par exemple, la mélodie pourra se contraindre à un jeu d'écriture basé sur des successions de tensions/résolutions (résolution de la sensible vers la tonique...), la mélodie pourra également suivre la progression harmonique définie par l'accompagnement. On voit qu'à la fois en entrée et en sortie se trouvent des séquences pour lesquelles chaque élément ne peut être considéré comme indépendant de ceux qui l'entourent.

Plusieurs méthodes permettent de prendre en compte l'aspect temporel pour un système de classification, on peut par exemple utiliser un HMM après avoir estimer pour chaque vecteur d'entrée la probabilité d'appartenance à chaque classe. On peut également mettre en entrée du classifieur non seulement le vecteur associé à la trame n, mais également ceux des trames $n-1$ et $n+1$.

Nous allons voir qu'il est possible de définir des réseaux de neurones permettant de prendre en compte un contexte temporel dans la tâche de classification, en ajoutant des connexions récurrentes aux couches cachées. La particularité de ces Réseaux de Neurones Réurrents (*Recurrent Neural Networks*) (RNNs) est que la taille du contexte temporel est apprise automatiquement grâce à l'optimisation des poids associés aux connexions récurrentes. De plus, si le RNN possède plusieurs couches cachées, il semble que chacune d'entre elle puisse apprendre un contexte temporel de taille différente, ce qui a été montré expérimentalement pour une tâche de prédiction de texte dans [39]. Dans cette section nous présenterons également l'extension des RNN avec des blocs longue mémoire à court-terme.

4.4.1 Réseaux de neurones récurrents mono-directionnels

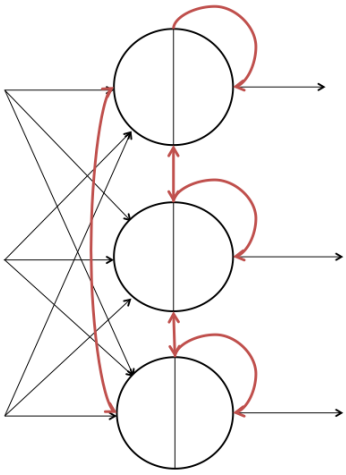


FIGURE 4.8 – Couche récurrente

Un RNN correspond à un classifieur automatique de séquences. Chaque individu de la base de données étiquetée n'est plus un couple vecteur d'entrée-étiquette mais est désormais un couple séquence de vecteurs d'entrée-séquence d'étiquettes.

Il existe de nombreuses architectures de RNNs (réseaux de Elman, de Jordan, de Hopfield, ...), nous considèrerons ici un réseau récurrent à couches vu comme une extension du perceptron multi-couches auquel sont ajoutées des connexions récurrentes dans les couches cachées. Ces connexions relient les neurones d'une couche cachée entre eux, de telle sorte que chaque neurone soit lié à n'importe quel autre de cette couche, y compris lui-même. Une telle topologie est représentée figure 4.8. L'activité d'un neurone j d'une couche cachée l à l'instant n est donc non seulement la somme pondérée des sorties de la couche précédente $l-1$ mais aussi des sorties de tous les neurones de la couche l à l'instant $n-1$. On peut calculer l'activité et la sortie de ce neurone :

$$a_j^{(l)}(n) = \sum_{i=1}^{H_{l-1}} w_{ij}^{(l)} s_i^{(l-1)}(n) + \sum_{h=1}^{H_l} w_{hj}^{(l)} s_h^{(l)}(n-1) \tag{4.4.1}$$

$$s_j^{(l)}(n) = f_{l,j}(a_j^{(l)}(n)) \tag{4.4.2}$$

Rappelons que pour le calcul de l'activité du neurone j de la couche l, un biais est bien pris en compte de façon implicite, il est confondu avec un poids associé à la connexion entre le neurone j et une entrée de valeur

unité. Les connexions récurrentes créent un effet de mémoire des entrées précédentes, celles-ci persistent dans les états internes du réseau et permettent d'influencer la sortie du réseau à l'instant courant. On voit à partir de l'équation 4.4.1 qu'il est nécessaire de définir la sortie de chaque neurone $s_h^{(l)}(n=0)$ avant l'arrivée d'un vecteur d'entrée à l'instant $n=1$, un choix possible simple est de considérer ces états initiaux nuls. Dans une telle architecture de réseau, la couche de sortie ne présente aucune connexion récurrente, ses sorties sont donc calculées de la même façon que pour le perceptron multi-couches. Les fonctions de transition sont également les mêmes que dans le cas non-récurrent, ce qui permet d'utiliser la même fonction de coût que précédemment, à savoir l'erreur d'entropie croisée dans le cas d'un schéma de classification probabiliste. Cependant, la fonction de coût que nous avions défini précédemment équations (4.3.1) et (4.3.2) correspond à un instant n de la séquence, il faut donc ajouter une somme sur la longueur de la séquence, c'est-à-dire :

$$\mathcal{L} = \sum_{n=1}^{T} \mathcal{L}(n) \tag{4.4.3}$$

où $\mathcal{L}(n)$ correspond aux expressions (4.3.1) et (4.3.2).

L'entraînement d'un réseau de neurones récurrent se fait également par descente de gradient en calculant les dérivées partielles de la fonction de coût par rapport à chaque poids. Un algorithme permettant de calculer ces termes est la rétropropagation dans le temps (*Backpropagation Through Time*) (BPTT), introduite dans [40]. C'est l'algorithme pour entraîner les réseaux récurrents le plus populaire, de par sa simplicité et sa ressemblance avec la rétropropagation standard pour les perceptrons multi-couches. En effet il s'agit également d'utiliser la technique de dérivation des fonctions composées et tout se passe de la même façon que précédemment sauf qu'il est nécessaire de considérer en entrées de chaque neurone les sorties de la couche précédente et celles de la couche courante à l'instant précédent. Pour appliquer la rétropropagation dans le temps, il faut prendre en compte le fait que la fonction de coût dépende des sorties de la couche cachée l non seulement à travers son influence sur la couche de sortie, mais également à travers son influence sur cette couche cachée l à l'instant suivant, on a alors les deltas pour chaque neurone h d'une couche cachée l :

$$\delta_h^{(l)}(n) = f'_{l,h}(a_h^{(l)}(n)) \left(\sum_{j=1}^{H_{l+1}} \delta_j^{(l+1)}(n) w_{hj} + \sum_{h'=1}^{H_l} \delta_{h'}^{(l)}(n+1) w_{hh'} \right) \tag{4.4.4}$$

Comme la couche de sortie n'est pas récurrente, les deltas qui lui sont associés se calculent de la même façon que pour le perceptron multi-couches.

On remarque dans l'expression 4.4.4 une certaine similarité entre la rétropropagation liée aux couches suivantes (terme en $l+1$) et celle liée aux instants suivants (terme en $n+1$). La séquence complète de deltas est alors calculée par double récursion, dans la profondeur du réseau et dans le temps, c'est-à-dire en appliquant l'équation (4.4.4) de la couche de sortie vers la couche d'entrée et de l'instant $n=T$ à l'instant $n=1$, sachant que $\delta_h^{(l)}(n=T+1) = 0 \ \forall h, \forall l$ comme l'erreur est considérée nulle pour les instants supérieurs à la longueur de la séquence, $n > T$.

Rappelons qu'un individu de la base d'apprentissage est désormais un couple séquence d'entrée-séquence d'étiquettes, la mise à jour des poids se fait par descente de gradient stochastique après avoir calculé les deltas à partir de l'ensemble de la séquence. A chaque instant, les poids sont donc identiques pour un individu. On somme finalement sur l'ensemble de la séquence pour obtenir les dérivées partielles de la fonction de coût par rapport à chaque poids w_{ij} :

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^{(l)}} = \sum_{n=1}^{T} \frac{\partial \mathcal{L}(n)}{\partial w_{ij}^{(l)}} = \sum_{n=1}^{T} \frac{\partial \mathcal{L}}{\partial a_j^{(l)}(n)} \frac{\partial a_j^{(l)}(n)}{\partial w_{ij}^{(l)}} = \sum_{n=1}^{T} \delta_h^{(l)}(n) s_i^{(l-1)}(n) \tag{4.4.5}$$

Il est possible de visualiser le réseau récurrent déroulé dans le temps, comme représenté figure 4.9, où chaque carré correspond à une couche constituée de plusieurs neurones. Cette représentation permet de comprendre pourquoi on considère qu'un RNN forme une architecture profonde. En effet, il faut prendre en compte deux directions de propagation au sein du réseau, celle dans la profondeur des couches (comme pour le perceptron classique), et celle temporelle. Les connexions récurrentes sont associées à la direction temporelle et forment un réseau de profondeur égale à la taille de la séquence, c'est-à-dire éventuellement très profond.

Les RNNs mettent donc en place un effet de mémoire interne, permettant de tenir compte d'un contexte temporel passé. Ce contexte, principalement défini par sa longueur, est appris automatiquement au travers des poids associés aux connexions récurrentes. Cependant, il n'est généralement pas plus grand qu'une dizaine d'instants. Cette limitation vient du fait que l'influence d'une entrée sur les couches cachées et sur la couche de sortie décroit exponentiellement au fur et à mesure que cet entrée se propage dans le temps. Il en est de même pour la rétropropagation du gradient dans le temps mise en oeuvre dans l'algorithhme BPTT pour apprendre les poids, ce problème est connu sous le nom de *vanishing gradient problem* [41] et rend l'apprentissage des RNNs

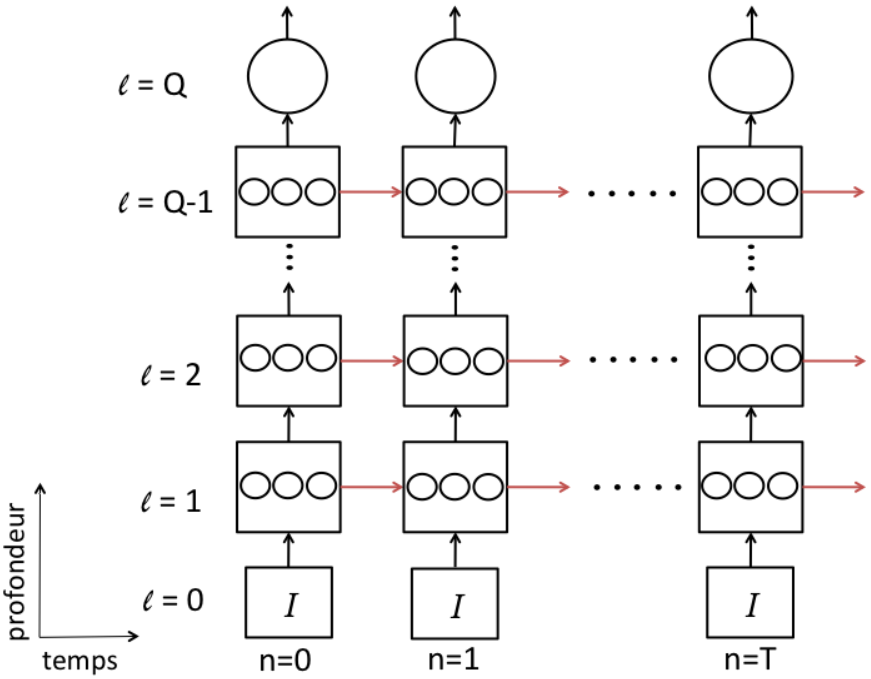


FIGURE 4.9 – RNN déroulé dans le temps

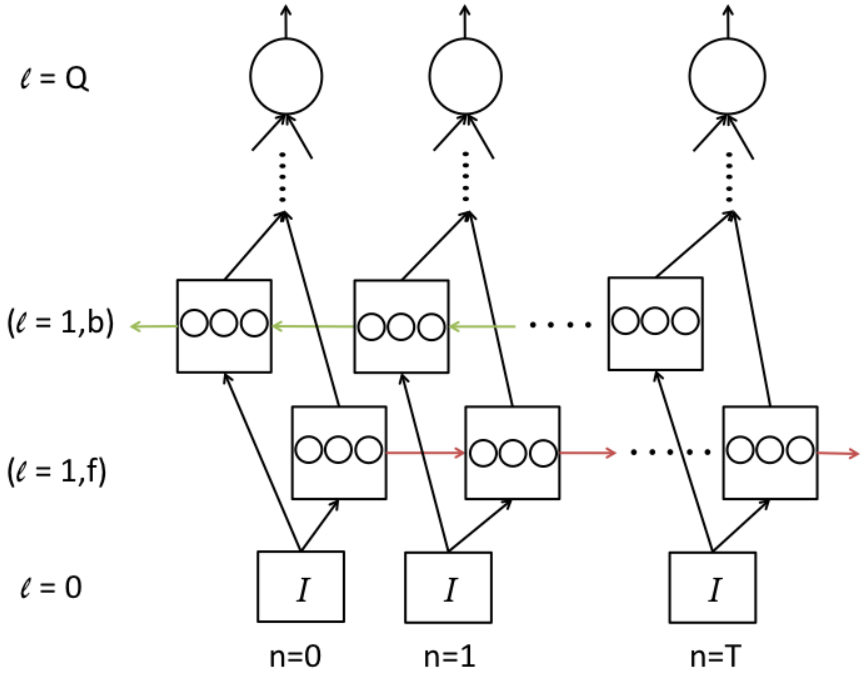


FIGURE 4.10 – BRNN déroulé dans le temps

difficile lorsqu'un contexte temporel grand est nécessaire. Nous verrons dans la section 4.4.3 une architecture permettant de solutionner ce problème.

4.4.2 Réseaux de neurones récurrents bi-directionnels

Pour certaines tâches de classification de séquence, où il est possible de relâcher la contrainte de causalité, il peut être utile de prendre en compte également un contexte temporel futur. Ceci implique de connaître entièrement la séquence de vecteurs d'entrée avant d'opérer la classification. Prenons l'exemple de systèmes d'extraction automatique d'information dans la musique, dont les tâches de détection de voix ou d'extraction de mélodie font partie : sans contrainte de traitement en temps réel il est possible de calculer les vecteurs d'entrée (associés à des descripteurs, généralement extraits par une analyse à court-terme) sur tout le morceau de musique avant d'opérer la classification.

Les Réseaux de Neurones Réurrents Bi-directionnels (*Bidirectional Recurrent Neural Networks*) (BRNNs) [42] permettent de prendre en compte un contexte temporel passé et futur. L'idée de base, comme illustrée figure 4.10, est de dédoubler chaque couche cachée récurrente en une couche "avant" (l, f) et une couche "arrière" (l, b). On présente la séquence d'entrée en avant $(n = 1, ..., T)$ et en arrière $(n = T, ..., 1)$ pour les deux premières couches avant et arrière. Les sorties de celles-ci sont ensuite concaténées dans un seul vecteur fourni en entrée des couches avant et arrière supérieures et la propagation se fait en itérant ce processus jusqu'à la couche de sortie.

L'algorithme de propagation des entrées du réseau à Q couches vers les sorties est le suivant :

Algorithme 4.3 : Propagation avant dans un réseau de neurones récurrent bi-directionnel

pour $l = 1$ **à** $Q - 1$ **faire**
  pour $n = 1$ **à** T **faire**
    calculer et enregistrer les sorties $a_j^{(l,f)}(n)$ de la couche cachée avant
  pour $n = T$ **à** 1 **faire**
    calculer et enregistrer les sorties $a_j^{(l,b)}(n)$ de la couche cachée arrière
  mettre en forme les entrées pour la couche $l + 1$ en concaténant les deux séquences de sorties issues des couches avant (l, f) et arrière (l, b)
pour $n = 1$ **à** T **faire**
  calculer les sorties $a_q^{(Q)}(n)$ de la couche de sortie

Le calcul des activités des neurones de la couche cachée avant (l, f) se fait comme pour un RNN mono-directionnel, en présentant les vecteurs d'entrée de l'instant $n = 1$ à $n = T$. Pour la couche cachée arrière (l, b), il suffit de les présenter de $n = T$ à $n = 1$.

$$a_j^{(l,f)}(n) = \sum_{i=1}^{H_{l-1,f}} w_{ij}^{(l,f)} s_i^{(l-1,f)}(n) + \sum_{i=1}^{H_{l-1,b}} w_{ij}^{(l,b)} s_i^{(l-1,b)}(n) + \sum_{h=1}^{H_{l,f}} w_{hj}^{(l,f)} s_h^{(l,f)}(n-1) \tag{4.4.6}$$

$$a_j^{(l,b)}(n) = \sum_{i=1}^{H_{l-1,f}} w_{ij}^{(l,f)} s_i^{(l-1,f)}(n) + \sum_{i=1}^{H_{l-1,b}} w_{ij}^{(l,b)} s_i^{(l-1,b)}(n) + \sum_{h=1}^{H_{l,b}} w_{hj}^{(l,b)} s_h^{(l,b)}(n+1) \tag{4.4.7}$$

avec $H_{l,f}$ et $H_{l,b}$, les tailles de couche avant et arrière, respectivement, pour la couche cachée l.

Le calcul des sorties du réseau est identique au cas du perceptron multi-couches, excepté le fait que le vecteur d'entrée de la couche de sortie est constitué de la concaténation des sorties des couches cachées avant et arrière précédentes.

La rétropropagation se déroule comme pour les RNNs mono-directionnels, c'est-à-dire que l'entraînement se fait par l'algorithme BPTT (cf. section 4.4). Il est cependant nécessaire de calculer en premier lieu les deltas $\delta_q(n)$, $n = T, ..., 1$, pour tous les neurones q de la couche de sortie et de les rétropropager aux couches inférieures avant et arrière, en sens inverse :

Algorithme 4.4 : Rétropropagation dans un réseau de neurones récurrent bi-directionnel

pour $n = T$ **à** 1 **faire**
  calculer par BPTT et enregistrer les deltas pour chaque neurone q de la couche de sortie
pour $l = Q - 1$ **à** 1 **faire**
  pour $n = T$ **à** 1 **faire**
    calculer par BPTT et enregistrer les deltas pour la couche cachée avant (l, f), en utilisant les deltas de la couche supérieure $l + 1$
  pour $n = 1$ **à** T **faire**
    calculer par BPTT et enregistrer les deltas pour la couche cachée arrière (l, b), en utilisant les deltas de la couche supérieure $l + 1$

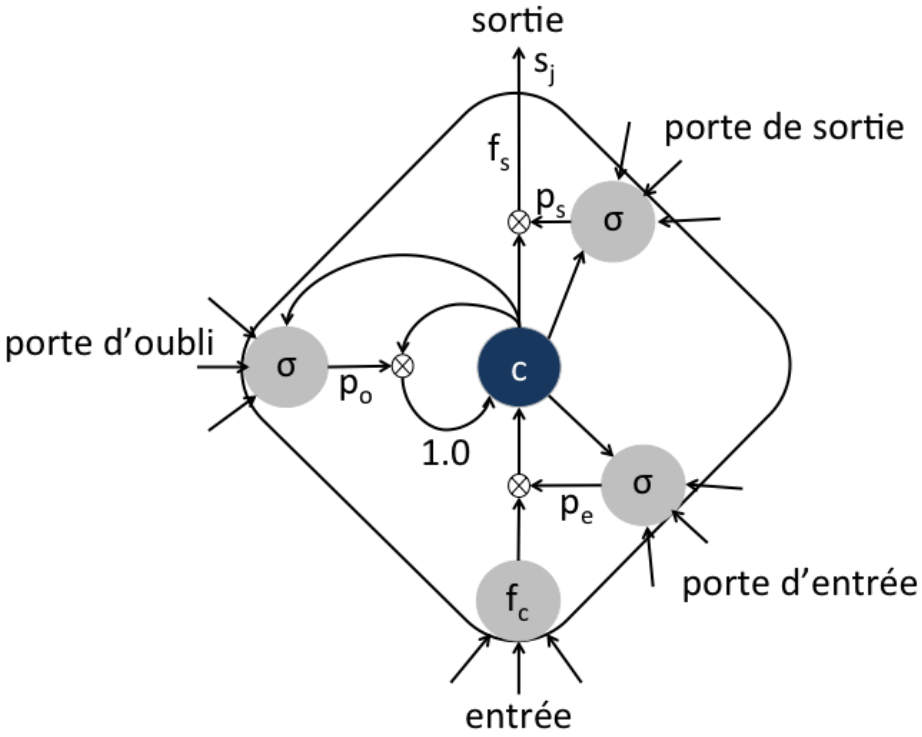


FIGURE 4.11 – Bloc LSTM

4.4.3 Longue Mémoire à Court-Terme

Afin de permettre aux RNNs de prendre en compte un contexte temporel plus long, permettant alors de résoudre le problème de diffusion du gradient (*vanishing gradient*), il a été proposé initialement dans [43] d'utiliser une architecture basée sur des blocs Longue Mémoire à Court-Terme (*Long Short-Term Memory)* (LSTM). Cette architecture que nous allons présenter ici inclue les modifications ajoutées aux blocs LSTM dans [44], leur permettant d'oublier le passé et mettant en place des connexions rétroactives de la cellule mémoire vers les portes contrôlant le fonctionnement du bloc LSTM.

4.4.3.1 Fonctionnement d'un bloc LSTM

Comme représenté figure 4.11, chaque bloc LSTM contient une entrée, une sortie, une (ou plusieurs) cellule mémoire connectée à elle même et trois unités multiplicatives : la porte d'entrée, la porte de sortie et la porte d'oubli qui correspondent respectivement aux opération écrire, lire et remise à zero pour la cellule mémoire. A chaque porte correspond un neurone à valeur réelle dans $[0, 1]$ (utilisation d'une fonction sigmoïde), si sa sortie est proche de 1, on dit que la porte est ouverte, si elle est proche de 0, la porte est fermée. En entrée du bloc LSTM agit un neurone dont la sortie pourra être stockée dans la cellule mémoire, la valeur contenue par cette dernière pourra ensuite être présentée directement en sortie du bloc LSTM. L'activation d'une porte est contrôlée par ses poids, les entrées du bloc, et la valeur stockée par la cellule mémoire.

Les portes multiplicatives permettent aux cellules mémoire des blocs LSTM d'enregistrer et d'accéder à un long contexte temporel. Ces portes pouvant prendre toute valeur entre 0 et 1, elles ne correspondent pas exactement à des interrupteurs ouvert/fermé, mais plutôt à des gains, cependant il est plus clair d'expliquer leur fonctionnement en considérant ces deux états, mais il est nécessaire de garder à l'esprit que le fonctionnement réel n'est pas "en tout ou rien" :

- Lorsque la porte d'entrée s'ouvre, elle permet à la cellule mémoire de stocker l'entrée du bloc LSTM. Tant que la porte d'entrée reste fermée, la valeur stockée dans la cellule mémoire ne pourra être écrasée.
- La valeur stockée par la cellule mémoire est le résultat de l'entrée du bloc LSTM multipliée par la porte d'entrée, s'ajoute à cela la valeur de la cellule mémoire à l'instant précédent multipliée par la porte d'oubli. La porte d'oubli, si elle est fermée, permettra donc d'éliminer le contexte passé précédemment stocké.
- Lorsque la porte de sortie s'ouvre, la valeur stockée par la cellule mémoire devient disponible en sortie du bloc LSTM. Si cette porte est fermée, la sortie du bloc est nulle.

Equations de propagation :

Le fonctionnement que nous venons de décrire peut se mettre en équation afin de calculer la propagation des entrées du bloc LSTM vers les sortie, pour l'ensemble de la séquence $n = 1, ..., T$. Comme précédemment, w_{ij} représente le poids associé à la connexion entre les neurones i et j. Les indices e, o et s réfèrent respectivement aux portes d'entrée, d'oubli et de sortie. Nous cherchons à calculer la sortie $s_j(n)$ d'un unique bloc mémoire à l'instant n, possédant I entrées $x_i(n)$ et faisant partie d'une couche cachée comprenant H blocs LSTM de sorties $s_h(n)$. La valeur stockée par la cellule mémoire à l'instant n est $c(n)$. σ est la fonction de transition des trois portes, c'est-à-dire une sigmoïde et f_c, f_s sont les fonctions de transitions respectivement en entrée et en sortie de la cellule mémoire. Pour calculer la sortie du bloc $s_j(n)$, il est nécessaire de procéder aux calculs suivants dans l'ordre :

Porte d'entrée

$$p_e(n) = \sigma\left(\sum_{i=1}^{I} w_{ie}x_i(n) + \sum_{h=1}^{H} w_{he}s_h(n-1) + w_{ce}c(n-1)\right) \tag{4.4.8}$$

Porte d'oubli

$$p_o(n) = \sigma\left(\sum_{i=1}^{I} w_{io}x_i(n) + \sum_{h=1}^{H} w_{ho}s_h(n-1) + w_{co}c(n-1)\right) \tag{4.4.9}$$

Cellule mémoire

$$c(n) = p_o(n)c(n-1) + p_e(n)f_c\left(\sum_{i=1}^{I} w_{ic}x_i(n) + \sum_{h=1}^{H} w_{hc}s_h(n-1)\right) \tag{4.4.10}$$

Porte de sortie

$$p_s(n) = \sigma\left(\sum_{i=1}^{I} w_{is}x_i(n) + \sum_{h=1}^{H} w_{hs}s_h(n-1) + w_{cs}c(n-1)\right) \tag{4.4.11}$$

Sortie du blocs LSTM

$$s_j(n) = p_s(n)f_s(c(n)) \tag{4.4.12}$$

Rappelons que pour calculer la sortie d'un des quatre neurones du bloc LSTM, un biais est bien pris en compte mais il est implicitement confondu avec un poids associé à la connexion entre ce neurone et une entrée de valeur unité.

Certaines architectures de blocs LSTM contiennent plusieurs cellules mémoires, dans ce cas il faudra introduire une somme sur le nombre de ces cellules dans les équations précédentes (voir [45]). Dans ce rapport nous n'utiliserons que des blocs comprenant une unique cellule mémoire.

4.4.3.2 Entraînement d'un réseau LSTM

Les réseaux LSTM sont également des opérateurs différentiables pouvant être entraînés par descente de gradient. A l'origine [43], les réseaux LSTM étaient entraînés en calculant le gradient tronqué dans le temps de la fonction de coût. Cependant, il est possible de calculer le gradient total et d'entraîner le réseau en utilisant exactement comme précédemment l'algorithme BPTT, ce qui permet d'interpréter directement les réseaux LSTM comme une extension des RNNs, où les neurones sont simplement remplacés par des blocs LSTM. Le calcul de la dérivée de la fonction de coût, par rapport à chaque poids du réseau présent dans les équations précédentes, est présenté dans [45] (page 39) et permet d'entraîner le réseau par l'algorithme BPTT.

4.4.4 Conclusion

Les réseaux LSTM ont pu dépasser le problème de diffusion du gradient (*vanishing gradient*) des RNNs. Ils ont permis de résoudre plusieurs tâches complexes artificielles, nécessitant la prise en compte d'un long contexte temporel, et qui n'avaient pu être résolus par l'utilisation de RNNs simples [43, 44].

D'un point de vue architectural, il suffit de remplacer les neurones des réseaux récurrents par des blocs LSTM afin d'obtenir un réseau pouvant prendre en compte un long contexte temporel passé. Il est donc est également possible d'utiliser les blocs LSTM pour construire une architecure bi-directionnelle. Un réseau Bi-directionnel à Longue Mémoire à Court-Terme (*Bidirectionnal Long Short-Term Memory*) (BLSTM) [46] permet alors de prendre en compte un long contexte temporel passé et futur.

Plusieurs travaux sur des tâches de classification de séquences à partir de données réelles ont montré récemment l'efficacité des réseaux LSTM et BLSTM, en dépassant à chaque fois l'état de l'art. On peut par exemple citer [47] pour la détection automatique de parole avec un réseau LSTM, [48] en reconnaissance de parole avec un réseau BLSTM ou encore [49] pour une tâche de détection automatique de vocalisation (rires, mots de remplissages comme "ah", "eh",...) dans les signaux de paroles avec un réseau BLSTM.

Il est également possible d'utiliser ces réseaux pour mettre en place des systèmes autres que des classifieurs de séquences. Dans [50], un réseau LSTM est entraîné pour composer automatiquement de la musique, avec une structure temporelle locale cohérente, mais aussi globale, puisque le réseau a appris à reproduire un blues de 12 mesures. Dans [51], un réseau BLSTM permet de débruiter des descripteurs MFCCs dans le cadre d'une application de reconnaissance de parole.

4.5 Vers des architectures profondes

Comme nous l'avons vu, les réseaux récurrents sont profonds par nature, car une fois déroulés dans le temps, on peut considérer deux directions de propagation : de la couche d'entrée vers la couche de sortie et dans le temps. Selon la direction temporelle, la profondeur est égale à la taille de la séquence, elle est donc potentiellement très grande et c'est de cela que venait les difficultés d'entraînement des RNNs, le gradient se rétropropageant mal dans le temps.

Les méthodes de classification de séquences à base de blocs LSTM que nous avons citées précédemment, présentant de bons résultats pour des tâches diverses, possèdent toutes des architectures comprenant plusieurs couches cachées, jusqu'à trois. Cette caractéristique s'inscrit dans une tendance plus générale appartenant au domaine de l'apprentissage profond (*Deep Learning*). La profondeur d'un réseau non-récurrent correspond au nombre de couches du réseau, c'est-à-dire le nombre de couches cachées plus la couche de sortie, un réseau est typiquement considéré comme profond s'il a au moins deux couches cachées. Un des atouts des réseaux de neurones est que la profondeur est un hyper-paramètre du système de classification, ce qui n'est par exemple pas le cas des SVM où la profondeur est figée à 2 : projection des entrées par transformation non-linéaire dans l'espace de redescription, puis calcul de la sortie par combinaison linéaire du vecteur d'entrée projeté avec un vecteur de paramètres. Cet atout est aussi une difficulté, car aucune règle théorique ne définit la profondeur optimale pour une tâche de classification donnée, l'approche est donc essentiellement empirique.

4.5.1 Motivations des architectures profondes

On peut montrer qu'un réseau avec deux couches cachées peut résoudre tout problème de classification non-linéaire [52], cependant cela nécessitera potentiellement un très grand nombre de neurones. Ceci n'est qu'un théorème d'existence, aucune méthode n'existe pour trouver l'architecture et les paramètres de ce réseau. Si tout problème de classification peut en théorie être résolu par un réseau de profondeur 3, certains résultats théoriques permettent de prouver l'intérêt des architectures plus profondes [53, 54]. En effet, on peut montrer que pour certaines familles de fonctions pouvant être représentées en $\mathcal{O}(n)$ neurones, pour n entrées lorsque la profondeur est K, le nombre de neurones sera exponentiellement plus grand ($\mathcal{O}(2^n)$) pour représenter la même fonction avec une architecture de profondeur $K-1$. Une augmentation de la profondeur du réseau peut-être vue comme une forme de factorisation de la fonction à approcher, il est donc nécessaire que cette fonction possède une certaine structure pour pouvoir être représentée de façon compacte avec un réseau profond, rien ne garantit a priori que la fonction associée à un problème donné présentera cette propriété.

Une des motivations premières à l'utilisation d'architectures profondes pour des tâches d'intelligence artificielle vient du fonctionnement du cortex visuel. Le traitement des informations visuelles dans le cerveau semble en effet fonctionner par couche afin d'extraire des informations à différents niveaux d'abstraction (orientation, contours à bas niveau et formes, mouvements à haut niveau).

D'un point de vue cognitif également, les humains organisent leurs idées de façon hiérarchique. L'apprentissage se fait de cette façon, en commençant par des concepts simples qui, combinés ensemble, permettent d'atteindre un niveau d'abstraction plus élevé. Lorsqu'un opérateur humain cherche à construire une solution à un problème, il aura également tendance à procéder de façon hiérarchique. Ceci est notamment mis en évidence pour le domaine du MIR dans [55], où les systèmes d'extraction de descripteurs construits "à la main", dans le cadre du traitement des signaux musicaux, tels que les MFCCs ou les chroma, sont constitués d'une cascade d'opérations simples : fenêtrage à court-terme, transformation affine (banc de filtres, CQT), non-linéarité (module, log), réduction de dimension (réduction à l'octave) et transformation affine (transformation en cosinus discrète). Nous avons également vu cela pour les méthodes d'estimation de mélodie, où il s'agit d'appliquer successivement différentes opérations, de mise en forme du signal dans une représentation adéquate (fonction de saillance), d'extraction de descripteurs (contours de mélodie), de filtrages et seuillages...

4.5.2 Difficulté de l'entraînement des réseaux profonds

Apprendre un réseau profond n'est pas chose facile, et c'est à cette difficulté qu'est due l'absence des architectures profondes dans les méthodes neuronales avant 2006 (excepté pour les réseaux convolutionnels). Il n'est pas du tout garanti qu'en augmentant la profondeur du réseau l'erreur diminue. Bien souvent, l'optimisation de la fonction de coût converge vers un minimum local tel que les performances d'apprentissage et de généralisation sont moins bonnes que pour un réseau peu profond. une hypothèse est formulée dans [54] pour expliquer cette difficulté : Ajouter des couches cachées augmente les irrégularités de la fonction de coût et limite l'efficacité de la phase d'entraînement en bloquant l'apprentissage dans des minima locaux associés à de faibles performances. De la même façon que pour les RNNs où l'apprentissage est difficile à cause de l'extinction du gradient lors de la rétropropagation dans le temps, il semblerait que le gradient lorsqu'il est rétropropagé de la couche de sortie vers la couche d'entrée dans un réseau profond souffre du même problème.

4.5.3 Pré-entraînement des réseaux profonds

Afin de pallier la difficulté d'apprentissage des réseaux profond, plusieurs techniques de pré-entraînement, supervisé ou non-supervisé, peuvent être mises en place pour trouver une "bonne" initialisation des poids avant d'effectuer l'apprentissage complet du réseau profond, cette fois-ci de façon supervisée. Bien qu'il n'y ait pas d'explication théorique sur ce que serait une bonne initialisation des paramètres pour l'apprentissage, ou quel critère utiliser pour l'apprentissage non-supervisé correspondant à l'étape de pré-entraînement, deux techniques ont fait leurs preuves : les Machines de Boltzmann Restreintes (MBR) et les auto-encodeurs. Ces méthodes ont prouvé expérimentalement leur efficacité [56, 54].

L'ensemble du réseau est pré-entraîné de façon vorace [57], c'est-à-dire que la première couche est pré-entraînée de façon non-supervisée, ses sorties sont ensuite calculées pour l'ensemble de la base d'apprentissages et utilisées comme entrées pour le pré-entraînement de la couche suivante, et ainsi de suite jusqu'à la couche de sortie. Finalement, l'ensemble du réseau est entraîné (phase de raffinement, *fine tuning*) de façon supervisée à partir de l'initialisation des poids issue du pré-entraînement.

Le pré-entraînement est donc local à chaque couche. Par exemple, chaque couche du réseau profond peut être entraînée pour fonctionner comme un auto-encodeur. Un auto-encodeur permet de réduire la dimension du vecteur d'entrée en calculant une transformation de celui-ci telle que l'entrée puisse ensuite être reconstruite grâce à une couche de décodage. Si on relâche la contrainte de réduction de dimension, l'auto-encodeur pourra éventuellement apprendre l'identité, il alors est nécessaire d'utiliser un auto-encodeur débruitant [58], permettant de reconstruire l'entrée originale à partir d'une version corrompue de celle-ci (bruitée). Chaque couche du réseau profond est alors pré-entraînée pour trouver la transformation "encodant" le vecteur d'entrée telle qu'il soit ensuite possible de le reconstruire avec une erreur minimale au sens des moindres carrés.

Deux hypothèses sont formulées et argumentées principalement par l'exemple pour expliquer l'efficacité du pré-entraînement dans le cadre de l'apprentissage de réseaux de neurones profonds :

- Problème d'optimisation : Les poids issus du pré-entraînement non-supervisé sont dans une région de l'espace des paramètres permettant d'atteindre un meilleur minimum lors de la phase de raffinement [57].
- Problème de régularisation : Le pré-entraînement non-supervisé permet d'ajouter une pénalité infinie sur les poids qui sont en dehors d'une certaine région de l'espace des paramètres [56]. Le point d'initialisation ainsi trouvé par le pré-entraînement permettrait d'imposer implicitement des contraintes sur les poids du réseau.

Il est également possible de mettre en place un pré-entraînement supervisé et vorace du réseau profond. Pour cela, chaque couche est pré-entraînée individuellement de façon supervisée, en étant mise en place dans réseau à une couche cachée où les entrées sont les sorties de la couche précédente du réseau profond, ayant été pré-entraînée de la même façon, ou bien les entrées du réseau dans le cas du pré-entraînement de la première couche cachée. Les étiquettes utilisées sont celles liées à la tâche considérée. Les couches cachées pré-entraînées sont ensuite utilisées comme initialisation du réseau profond avant d'effectuer la phase de raffinement (apprentissage du réseau complet de façon supervisée) [57].

Les techniques de pré-entraînement que nous venons de présenter n'ont jamais été utilisées, à notre connaissance, pour des RNNs. Néanmoins, dans [39], une autre approche de pré-entraînement supervisé est proposée pour un RNN. Les auteurs proposent d'entraîner le réseau en ajoutant chaque couche de façon incrémentale. La première couche cachée est entraînée seule, de façon supervisée par descente de gradient, puis la deuxième couche est ajoutée entre la première couche et la couche de sortie, le réseau est alors à nouveau entraîné avec les poids de la première couche qui correspondent à ceux appris à l'étape précédente, les poids de la deuxième et de la dernière couche sont quant-à-eux initialisés aléatoirement. Ce processus est itéré pour toute la profondeur du réseau. Cette technique est justifiée par l'intérêt pour chaque couche d'être connectée pendant un certain temps d'apprentissage à la couche de sortie. En effet, le problème de l'entraînement des réseaux profonds comme

nous l'avons expliqué, pourrait venir du fait que l'information sur l'erreur au niveau de la sortie se propage mal jusqu'aux couches les plus basses [57], c'est-à-dire que cette information n'est plus présente pour apprendre correctement les poids des couches proches de l'entrée car le gradient est trop diffus après rétropropagation. D'où l'idée de "casser" la profondeur lors du pré-entraînement, en considérant chaque couche une à une ou bien en connectant chaque couche cachée à la couche de sortie pendant une partie de l'apprentissage. Pour notre méthode de détection de voix avec un réseau BLSTM, que nous allons présenter chapitre 5, nous avons utiliser la méthode d'entraînement incrémentale utilisée dans [39]. Celle-ci nous permettant d'arriver à de meilleurs résultats que par un entraînement brut du réseau entier. Nous avons également essayé de mettre en oeuvre la méthode de pré-entraînement non-supervisée par auto-encodeur, mais le gain apporté par cette technique était moins évident.

Chapitre 5

Détection de la voix chantée par réseau BLSTM

Les réseaux de neurones récurrents sont particulièrement adaptés pour des tâches d'extraction automatique d'information dans la musique, car comme nous l'avons expliqué en introduction de la section 4.4, prendre en compte l'aspect temporel est essentiel, chaque vecteur de descripteur ne pouvant être considéré comme indépendant des précédents ou des suivants dans une même séquence de descripteurs issue d'une analyse à court-terme d'un signal de musique. De plus, les blocs LSTM ont déjà montré leur pertinence pour des applications musicales, dans [50], un réseau LSTM apprend à composer automatiquement de la musique en respectant des structures non seulement locales mais également à long-terme, en apprenant la structure d'un blues de 12 mesures. Ajoutons à cela le fait que des réseaux LSTM ont été utilisés avec succès pour des tâches de reconnaissance vocale [48] ou de détection de la parole [47], en dépassant significativement l'état de l'art. Il semble alors naturel d'utiliser un tel outil de classification, prenant en compte directement l'aspect temporel sans nécessiter de post-traitement, pour le système de détection automatique de la voix chantée que nous souhaitons mettre en place.

L'objectif de ce chapitre est de présenter cette nouvelle méthode de détection de la voix chantée. Nous décrirons les descripteurs utilisés, la méthode que nous avons employée pour définir l'architecture du réseau BLSTM et pour l'entraîner, nous tenterons également d'explorer ce qui se passe dans le réseau au niveau des couches cachées. Finalement, nous présenterons les résultats de cette méthode sur différentes bases de données, notamment Jamendo afin de pouvoir comparer nos résultats à différentes méthodes de l'état de l'art. Nous verrons que l'approche tournée vers des réseaux de neurones BLSTM est très encourageante, car le système que nous avons mis en place dépasse les performances des derniers algorithmes utilisés pour la détection de la voix chantée.

5.1 Descripteurs

Récemment, il a été montré en reconnaissance de parole avec un réseau de neurones profond que des descripteurs issus d'un banc de filtres distribué sur une échelle Mel permettaient d'obtenir de meilleurs résultats que pour des MFCCs [59, 60], ces derniers étant pourtant devenus progressivement les descripteurs timbraux de référence, initialement pour la parole mais par extension également pour la musique. L'approche classique en reconnaissance de parole est d'utiliser un HMM où les distributions de probabilité d'émission des vecteurs de descripteurs acoustiques associées à chaque état sont modélisées à l'aide de GMMs. Les paramètres des GMMs sont généralement estimés grâce à l'algorithme Espérance-Maximisation (*Expectation-Maximisation*) (EM) et à un ensemble de données. Pour le traitement de la parole, il est souvent nécessaire de simplifier le modèle afin de limiter le nombre de paramètres à apprendre, ainsi les matrices de covariances des densités gaussiennes multidimensionnelles à apprendre sont en pratique systématiquement diagonales. Ceci implique une hypothèse sous-jacente qui est que chaque élément d'un vecteur de descripteurs acoustiques est supposé décorrélé des autres. Cette hypothèse justifie le fait que les système de reconnaissance de parole actuels n'utilisent pas les coefficients issus d'un banc de filtres comme vecteurs de descripteurs, car ils sont fortement corrélés. En revanche, dans le cas d'un réseau de neurones profond, aucune hypothèse n'est à faire, ce qui permet d'utiliser un vecteur d'entrée dont les composantes sont corrélées. Il faut tout de même nuancer cette explication, en notant qu'il est parfois conseillé de décorréler les entrées du réseau à l'aide par exemple d'une analyse en composantes principales. Concevoir des descripteurs pour une tâche spécifique ne garantit donc pas la pertinence de ces derniers pour une autre application. Un des objectifs de l'apprentissage profond, comme illustré dans [55] pour des tâches d'extraction automatique d'information dans la musique, est d'essayer d'apprendre les descripteurs optimaux pour la tâche considérée, et non de partir de descripteurs à un haut-niveau d'abstraction dont la conception est essentiellement manuelle et potentiellement sous-optimale.

C'est pour ces raisons que nous avons choisi d'utiliser comme descripteurs pour notre système de détection de la voix chantée des coefficients issus d'un banc de filtres distribués sur une échelle Mel. Ce que nous espérions

à partir de ce choix est que la profondeur du réseau permette de passer d'une représentation bas-niveau du signal en entrée du réseau, à des représentations simplifiées au fur et à mesure de la propagation dans les couches-cachées, éliminant l'information non pertinente, permettant ainsi d'obtenir de bons résultats de classification peu importe les spécificités du signal d'entrée : style, instrumentation, tessiture de la voix, etc.

Avant d'extraire les descripteurs, le signal est converti en mono et sous-échantilloné à 16kHz [1], deux pré-traitements sont ensuite effectués. Le premier est une normalisation du niveau du signal, le second est une double séparation HPSS (voir sections 5.1.1 et 5.1.2). A l'issu du deuxième pré-traitement, nous obtenons deux signaux $h_2(t)$ et $p_2(t)$ (voir formule (2.3.5)). Les spectrogrammes associés sont calculés à l'aide de la TFCT avec une fenêtre de Hann de longueur 32ms et un taux de recouvrement de 50%. Un banc de 40 filtres répartis sur une échelle Mel permet ensuite d'extraire 40 coefficients pour chacun des deux signaux, le logarithme de ces coefficients constitue finalement le vecteur de 80 descripteurs par trame.

Le logarithme est utilisé car les données issues de la TFCT d'un signal audio ont une très grand dynamique. Cette opération de réduction de dynamique permet alors d'obtenir une distribution de chaque coefficient sur l'ensemble de la base d'apprentissage plus proche d'une gaussienne. C'est une opération nécessaire car nous effectuons ensuite une normalisation de telle sorte que chaque dimension des données soit centrée-réduite (moyenne nulle et variance unité) sur l'ensemble de la base d'apprentissage. Cette mise en forme des données permet de prévenir la saturation des neurones du réseau (voir section 4.3.3.7). Les moyennes et variances calculées sur la base d'apprentissage sont utilisées pour normaliser les données extraites depuis la base de validation et la base de test.

5.1.1 Normalisation RMS

Pour que la détection de présence de voix ne dépende pas du volume du signal d'entrée, nous utilisons une normalisation des signaux basée sur le niveau RMS. La procédure que nous détaillons ici est indiquée dans [61]. On procède par une analyse à court-terme pour déterminer le niveau RMS du signal :

$$L_{RMS}^{ST}(k) = \sqrt{\frac{1}{M}\sum_{n=0}^{M-1}(x(n+kM)w(n))^2} \tag{5.1.1}$$

avec $w(n) = \mathbb{1}_{n=0,\dots,M-1}$ une fenêtre rectangulaire de taille M correspondant à 50ms.

Ces valeurs sont ensuite triées par ordre croissant. Les instants de niveaux les plus élevés participent le plus à la sensation du volume perçu, c'est pourquoi la valeur RMS globale L_{RMS} est choisie comme celle telle que 95% de l'ensemble des niveaux calculés sur le signal soient inférieurs. Le niveau de référence utilisé est $(L_{RMS}^{ref})_{dB} = -20$dB. Le signal normalisé $x_N(n)$ associé à $x(n)$ est finalement calculé par :

$$x_N(n) = \frac{L_{RMS}^{ref}}{L_{RMS}}x(n) \tag{5.1.2}$$

5.1.2 Pre-traitement par double HPSS

Avant d'extraire les descripteurs, nous effectuons un pré-traitement basé sur la technique de mise en avant de la voix chantée par double HPSS [62] (voir section 2.3.2 pour plus de détails). Cette technique de a déjà montré son intérêt dans [6] pour la détection de la voix chantée, cependant dans cette méthode, seul le signal associé à la mélodie vocale est utilisé. Nous allons voir qu'il est intéressant de prendre en compte un autre signal obtenu par double HPSS.

Grâce à cette double séparation de sources, nous disposons de trois signaux $h_1(t)$, $h_2(t)$ et $p_2(t)$ (voir section 2.3.2.2) dont la somme est égale au signal original. $h_2(t)$ est le signal pour lequel la mélodie vocale est mise en avant, il s'agit donc d'extraire les descripteurs sur celui-ci. Cependant, il serait peut-être intéressant d'inclure également l'information contenue dans les autres signaux. Dans la méthode d'extraction de mélodie de Tachibana [26] (utilisant la double HPSS), la détection de voix se fait en calculant une distance entre le spectrogramme associé à $h_2(t)$ et celui associé à $p_2(t)$ (cf. section 2.3.2.4), il semblerait donc que ce dernier signal qui correspond aux composantes percussives contienne de l'information complémentaire pour détecter la présence de voix chantée. Il en est peut-être de même pour $h_1(t)$.

Pour justifier le choix des descripteurs, plusieurs combinaisons (40 coefficients à chaque fois par signal) sont considérées : (1) le signal brut sans pré-traitement par HPSS, (2) $h_2(t)$, (3) $h_2(t)$, $p_2(t)$ et (4) $h_2(t)$, $p_2(t)$, $h_1(t)$. Afin d'identifier la meilleur combinaison de descripteurs, nous avons entraîné un réseau BLSTM à une couche cachée sur la base de données Jamendo, et à chaque fois pour quatre tailles de couche cachée : 10, 20, 30 ou 40 blocs LSTM (répartis équitablement entre couche avant et couche arrière).

[1] Les résultats sont peu modifiés par rapport à 44.1kHz

Nous présentons les erreurs de classification obtenues pour chaque combinaison sur la base de test Jamendo, nous considérons à chaque fois la taille de couche cachée permettant d'obtenir la plus faible erreur de classification :

(1) Descripteurs extraits depuis le signal brut sans pré-traitement : 18.55% avec 20 blocs LSTM et un vecteur d'entrée de dimension 40.

(2) Descripteurs extraits depuis $h_2(t)$: 14.11% avec 20 blocs LSTM et un vecteur d'entrée de dimension 40.

(3) Descripteurs extraits depuis $h_2(t)$ et $p_2(t)$: 13.55% avec 30 blocs LSTM et un vecteur d'entrée de dimension 80.

(4) Descripteurs extraits depuis $h_2(t)$, $p_2(t)$ et $h_1(t)$: 15.79% avec 10 blocs LSTM et un vecteur d'entrée de dimension 120.

Ces résultats de classification permettent de justifier le choix des descripteurs et l'intérêt du pré-traitement par double HPSS. Comme supposé, le fait d'utiliser des descripteurs extraits depuis $h_2(t)$ et $p_2(t)$ augmente le taux de bonne classification. On remarque par contre que l'information contenue dans $h_1(t)$, liée aux composantes harmoniques du signal de départ, ne semble pas pertinente pour la détection de voix chantée, elle limite même les performances bien que les descripteurs issus de $h_2(t)$ et $p_2(t)$ soient également contenus dans les vecteurs d'entrée. Une intuition pour expliquer cette observation est que dans ce signal se trouvent les autres instruments harmoniques ayant des caractéristiques communes avec la voix, il n'est donc sûrement pas souhaitable de l'inclure dans les descripteurs.

Afin de nous assurer de la pertinence d'un réseau BLSTM, nous avons également essayé de mettre en oeuvre un système de détection de la voix chantée à partir d'un perceptron à une couche cachée et des mêmes descripteurs. En comparant les performances pour différentes tailles de couches cachées - 16, 32, 64, 128, 256, 512, 1024 - nous obtenons la plus petite erreur de classification, 20.88%, pour une couche cachée de 32 neurones. Ce résultat confirme l'intérêt d'utiliser un réseau BLSTM plutôt qu'un simple perceptron.

5.2 Modalités d'apprentissage

Nous utilisons un réseau BLSTM pour une tâche de classification binaire. La fonction de coût utilisée est l'erreur d'entropie croisée car on se place dans un schéma de classification probabiliste (voir section 4.2.2.2). L'entraînement d'un réseau BLSTM se fait par descente de gradient avec *momentum* en utilisant l'algorithme de rétropropagation dans le temps BPTT. La mise à jour des poids se fait après chaque séquence de la base d'apprentissage, ces dernières étant sélectionnées aléatoirement au sein de chaque époque pour être présentées en entrée du réseau.

Nous employons une technique de contrôle du sur-apprentissage par arrêt prématuré (cf. figure 5.1) : l'apprentissage débute avec un pas $\eta = 10^{-5}$, si pendant 20 époques il n'y a pas d'amélioration de l'erreur d'entropie croisée, le pas est divisé par 10 ($\eta = 10^{-6}$) et l'apprentissage reprend depuis les paramètres associés à la dernière amélioration. Si pendant 10 époques il n'y a pas eu d'amélioration, le pas est de nouveau divisé par 10 ($\eta = 10^{-7}$), l'apprentissage continu depuis la dernière amélioration. La phase d'entraînement est finalement arrêtée s'il n'y a pas d'amélioration pendant 10 époques avec ce nouveau pas. Le *momentum* est typiquement pris proche de 1 afin de garder une intertie assez grande pour éviter de tomber dans un mimimum local et pour atténuer la trajectoire très oscillante de la minimisation, on choisit $m = 0.9$. Les poids sont initialisés aléatoirement selon une loi normale centrée et d'écart-type 0.1 d'après le raisonnement décrit section 4.3.3.8 et appliqué pour un vecteur d'entrée de dimension 80.

Tous nos essais pour mettre en oeuvre un réseau basé sur des blocs LSTM ont été faits grâce à la libraire CURRENNT [2] développée par Felix Weninger, permettant de tirer avantage d'une implémentation GPU de la phase d'entraînement.

5.3 Apprentissage de l'architecture

La taille de la couche de sortie est définie par le problème de classification. Nous considérons deux classes C_1 = présence et C_2 = absence de voix chantée, dans ce cas, un unique neurone de sortie est nécessaire, avec une fonction de transition sigmoïde. Cette configuration permet d'interpréter la sortie comme la probabilité d'appartenance du vecteur d'entrée à la classe C_1.

Aucun élément théorique ne permet de connaître pour une tâche donnée l'architecture optimale du réseau de neurones permettant de la résoudre. Dans les articles liés aux méthodes d'apprentissage profond, il est rare que l'architecture soit justifiée, cela étant sûrement dû au fait que la construction d'un réseau est essentiellement

[2] http://sourceforge.net/projects/currennt/

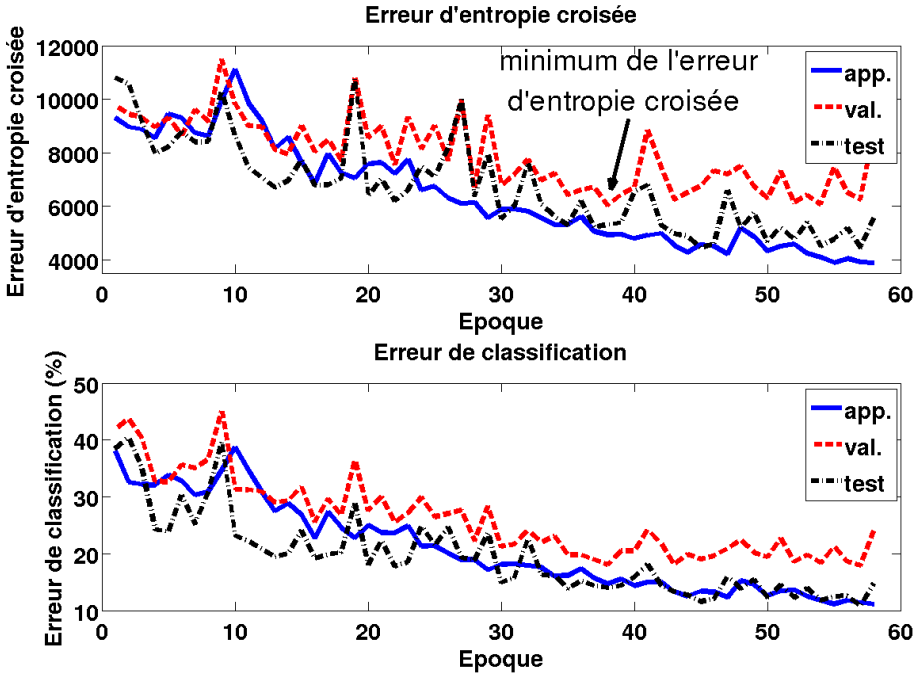


FIGURE 5.1 – Illustration de l'arrêt prématuré : l'apprentissage est stoppé prématurément à l'époque 58 car il n'y a pas d'amélioration de l'erreur d'entropie croisée sur la base de validation depuis l'époque 38

empirique. La méthode que nous avons mise en place pour définir la structure du réseau BLSTM est certes empirique, mais peut être automatisée. Il s'agit de trouver la profondeur optimale du réseau L^* et le nombre de blocs LSTM b_l^*, $l \in 1, ..., L^*$ optimal pour chaque couche de telle sorte que l'erreur de classification $E^{class}(l, b_l)$ sur la base de test soit minimale.

Du fait que les performances de chaque architecture soient évaluées sur la base de test, celle-ci sert en quelque sorte également à l'apprentissage. En toute rigueur, pour les systèmes de classification dont l'architecture doit être apprise, il serait nécessaire de disposer d'une quatrième base pour évaluer le modèle après apprentissage de la structure. Pour pouvoir comparer nos résultats avec l'état de l'art, nous n'avons cependant utilisé que les bases d'apprentissage, de validation et de test Jamendo. Nous présenterons tout de même section 5.4 les résultats de classification sur notre base de test élargie [TestDVC].

On utilise la procédure d'entraînement incrémental suggérée dans [39] : la profondeur du réseau est augmentée en ajoutant progressivement des couches. Cette technique permet à chaque couche cachée d'être reliée pendant une certaine durée d'apprentissage à la couche de sortie, ce qui a pour effet de limiter les difficultés d'entraînement d'un réseau profond (voir section 4.5.3). Pour chaque couche cachée l, on évalue différentes largeurs b_l, la largeur étant le nombre de blocs LSTM qu'elle contient. Lorsqu'une couche cachée est ajoutée, les poids des couches inférieures appris aux étapes précédentes de la construction du réseau sont gardés, les poids de la couche cachée courante et de la couche de sortie sont initialisés aléatoirement. L'algorithme 5.1 décrit le processus utilisé pour construire le réseau.

Pour mettre en place l'architecture du réseau BLSTM, il faut définir l'espace $\mathbb{B}$ de recherche du nombre de blocs LSTM par couche. Dans notre cas, nous avons choisi arbitrairement $\mathbb{B} = \{10, 20, 30, 40, 50\}$, ces valeurs sont assez faibles pour contrôler le nombre de paramètres du modèle et éviter le sur-apprentissage. La figure 5.2 présente les résultats de classification pour toutes les structures de réseau envisagées. L'erreur de classification sur la base de test Jamendo est représentée en fonction de la largeur b_l de la couche cachée l. Chacune des courbes correspond à un réseau de profondeur différente, par exemple, la courbe en trait plein est associée à un réseau BLSTM de 80 entrées et un neurone de sortie, avec une unique couche cachée pour laquelle on fait varier le nombre $b_1 \in \mathbb{B}$ de blocs LSTM, d'où la notation en légende "$80 - b_1 - 1$". La courbe en pointillés juste en dessous est associée à un réseau BLSTM de 80 entrées, avec une première couche cachée de taille fixée 30, une seconde couche cachée pour laquelle on fait varier le nombre $b_2 \in \mathbb{B}$ de blocs LSTM, et un seul neurone en sortie, d'où la notation en légende "$80 - 30 - b_2 - 1$"... Rappelons que pour un réseau bi-directionnel, si on indique b_l blocs LSTM dans la couche cachée l, cela signifie qu'il y a $b_l/2$ blocs dans la couche cachée avant et de même pour la couche cachée arrière.

Algorithme 5.1 : Apprentissage de l'architecture du réseau

$l \longleftarrow 0$
$E^{class}(l, b_l^*) \longleftarrow \infty$
répéter
  ajouter une couche cachée : $l \longleftarrow l + 1$
  $E_{min}^{class} \longleftarrow \infty$
  pour chaque $b_l \in \mathbb{B}$ **faire**
    entraîner le réseau à l couches cachées, avec b_k^* blocs LSTM par couche $k = 1, ..., l-1$ et b_l blocs LSTM pour la couche l
    si $E^{class}(l, b_l) < E_{min}^{class}$ **alors**
      $b_l^* \longleftarrow b_l$
      $E_{min}^{class} \longleftarrow E^{class}(l, b_l)$
jusqu'à $E^{class}(l, b_l^*) > E^{class}(l-1, b_{l-1}^*)$
$L^* \longleftarrow l - 1$

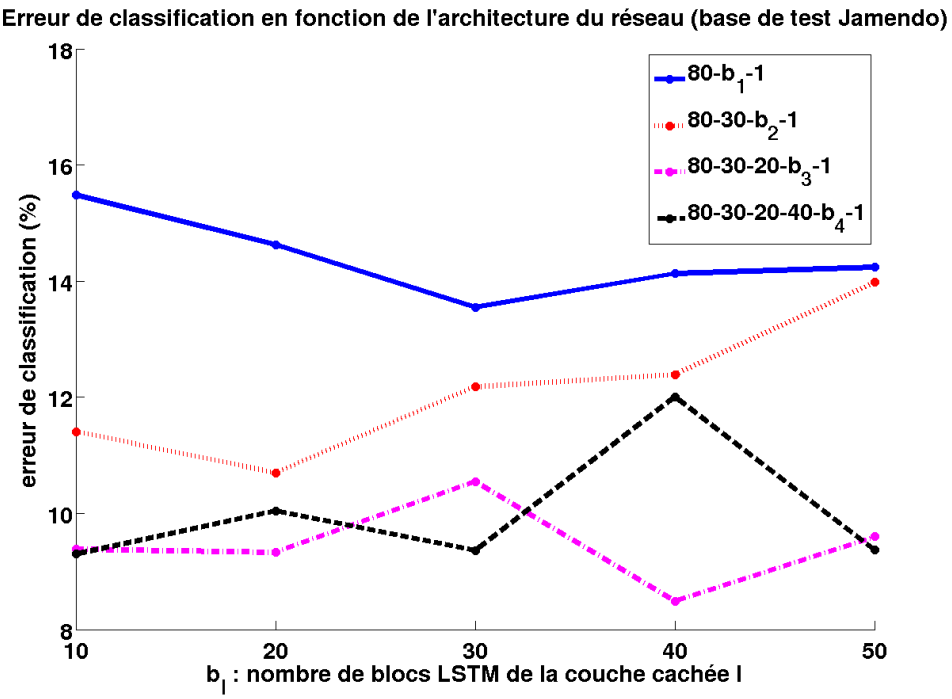


FIGURE 5.2 – Erreur de classification sur la base de test Jamendo en fonction de l'architecture du réseau

A chaque ajout de couche cachée, le minimum d'erreur de classification atteint sur la base d'apprentissage est plus faible. Ceci est normal car on ajoute plus de degrés de liberté donc le modèle peut mieux s'ajuster aux données. Cependant, ce n'est pas le problème d'optimisation qui nous intéresse mais celui de généralisation : ce sont les résultats de classification sur la base de test qui importent. La figure 5.2 nous montre qu'il n'est possible d'améliorer la capacité de généralisation que jusqu'à trois couches cachées, pour notre système de détection de voix chantée. En ajoutant une quatrième couche, l'erreur est plus grande sur la base de test. Les meilleurs résultats de classification sont donc donnés pour un réseau à 3 couches cachées de tailles par ordre croissant de couche (de l'entrée vers la sortie) : 30, 20 et 40. Un tel réseau comprend 21671 poids.

Rappelons que les performances évaluées suite à la phase d'apprentissage dépendent de l'initialisation des poids. En effet, l'initialisation contraint la région à partir de laquelle l'entraînement se fait et comme la fonction de coût est très irrégulière, les performances ne seront sûrement pas les mêmes avec des initialisations différentes. En toute rigueur, il faudrait donc répéter l'algorithme d'apprentissage de l'architecture du réseau afin de s'assurer d'obtenir celle qui est optimale indépendamment de l'initialisation. Cependant, cela requiert un certain temps d'apprentissage et nous n'avons pu répéter la procédure pour nos expériences.

5.4 Résultats de la détection de voix chantée par réseau BLSTM

Afin d'évaluer les performances du système, les mesures définies section 3.3 sont calculées sur la base de test Jamendo et présentées tableau 5.1. Les résultats de la méthode de Lehner [37], qui à notre connaissance présentait jusqu'à maintenant les meilleurs performances sur la base de données Jamendo, sont également indiqués. Les mesures sont calculées en considérant la classification de l'ensemble des trames de la base de test et non en moyennant les résultats évalués par fichier. Nous voyons que le réseau BLSTM mis en oeuvre ici dépasse sur tous les critères celui de Lehner, il faut de plus remarquer que les descripteurs que nous utilisons (coefficients issus d'un banc de filtres sur une échelle Mel) sont bas niveau et très peu mis en forme en comparaison avec la méthode de Lehner (voir section 3). Une perspective pour de futurs travaux serait d'évaluer l'importance des descripteurs pour les performances du réseau BLSTM. On pourrait envisager d'utiliser des MFCCs par exemple, éventuellement en complément des descripteurs actuels.

TABLE 5.1 – Résultats de détection de voix chantée sur la base de test Jamendo. LEHNER : extraits de [37]

	LEHNER	réseau BLSTM 80-30-20-40-1
Taux de bonne classification (%)	88.2	91.24
Rappel (%)	86.2	92.26
Précision (%)	88.0	89.23
F-mesure	87.1	90.72

Comme nous l'avions vu avec l'évaluation de la méthode de détection de voix d'ADX TRAX (cf. tableau 3.1), les performances chutaient lorsque le système de classification était appris sur la base Jamendo mais testé sur [TestDVC]. Afin d'évaluer la capacité de généralisation de notre système après entraînement sur la base Jamendo, nous mesurons ses performances sur [TestDVC], les résultats sont présentés tableau 5.2 et comparés à ceux de la méthode actuellement implémentée dans ADX TRAX. Par manque de temps, nous n'avons malheureusement pu implémenter la méthode de Lehner pour effectuer une comparaison sur [TestDVC].

Pour notre système de détection de la voix chantée, on observe une baisse du taux de bonne classification de 5.4% par rapport à l'évaluation sur Jamendo (voir tableau 5.1), ce qui n'est pas négligeable et peut s'expliquer par le fait que la base de test est assez proche de la base d'apprentissage pour Jamendo. Cependant, la classification reste correcte à 86.10%, ce qui correspond tout de même à de bonnes performances et constitue un gain de plus de 12.5% par rapport à la méthode d'ADX TRAX. On note un taux de rappel plus faible pour le réseau BLSTM mais cela est bien compensé par les autres mesures.

TABLE 5.2 – Résultats de détection de voix, entraînement sur Jamendo et test sur [TestDVC]

	ADX TRAX	réseau BLSTM 80-30-20-40-1
Taux de bonne classification (%)	73.31	86.10
Rappel (%)	92.86	88.96
Précision (%)	67.78	84.97
F-mesure	78.36	86.92

5.5 Ré-entraînement

Pour un obtenir un système capable de bien se généraliser, une première solution est de diversifier la base d'apprentissage. Nous avions mis en oeuvre une grande base [ApprentissageDVC] dans ce but pour la méthode employée dans ADX TRAX et cela avait permis d'améliorer les résultats. Afin de correspondre aux modalités opératoires d'apprentissage du réseau BLSTM, il est nécessaire de disposer d'un base de test mais aussi d'une base de validation. Nous séparons donc la base [TestDVC] en deux bases [TestDVC-BLSTM] et [ValDVC-BLSTM] (voir annexe A pour plus de détails).

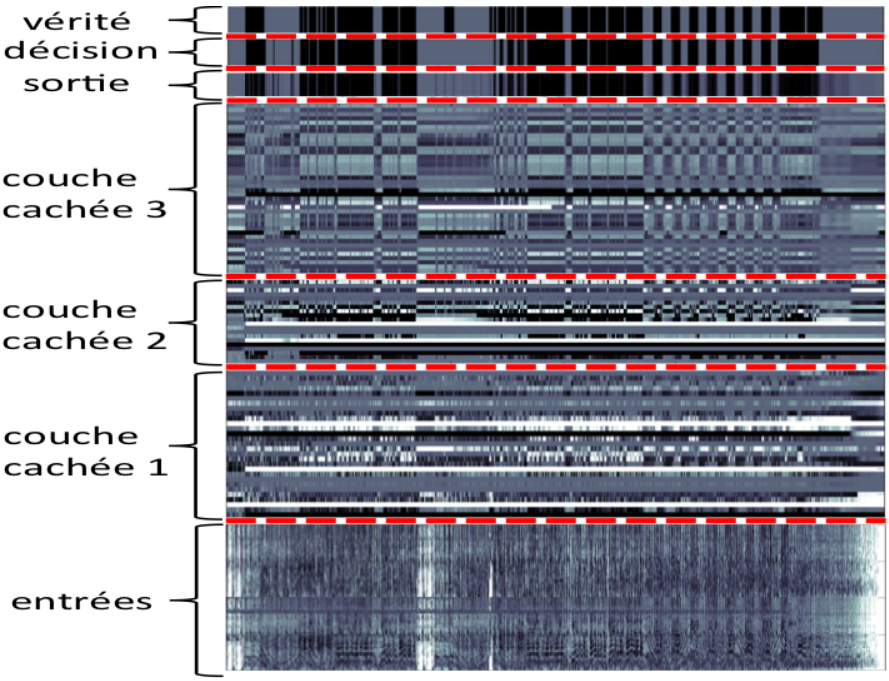
Nous avons deux possibilités pour entraîner le réseau avec ces nouvelles bases. Nous pouvons partir d'un réseau initialisé aléatoirement et procéder de façon incrémentale en apprenant les couches petit à petit comme précédemment ("entraînement brut"), ou nous pouvons utiliser le réseau déjà entraîné sur Jamendo et effectuer une phase de "ré-entraînement" global afin d'ajuster les poids avec les nouvelles données d'apprentissage. Les résultats après apprentissage sont donnés pour ces deux approches tableau 5.3.

Les performances du réseau après ré-entraînement sont légèrement meilleurs que celles obtenues avec l'entraînement complet depuis une initialisation aléatoire, sauf pour le taux de précision qui baisse de plus de 2%. Ré-entraîné un réseau BLSTM est une technique intéressante car cela permet d'ajuster les poids précédemment estimés si de nouvelles données d'apprentissage sont ajoutées. Cette phase de ré-entraînement est beaucoup plus rapide que d'entraîner le réseau en ajoutant les couches une à une depuis une initialisation aléatoire (30 époques contre 87), ceci pouvant être interprété comme venant du fait que les poids en début de ré-entraînement

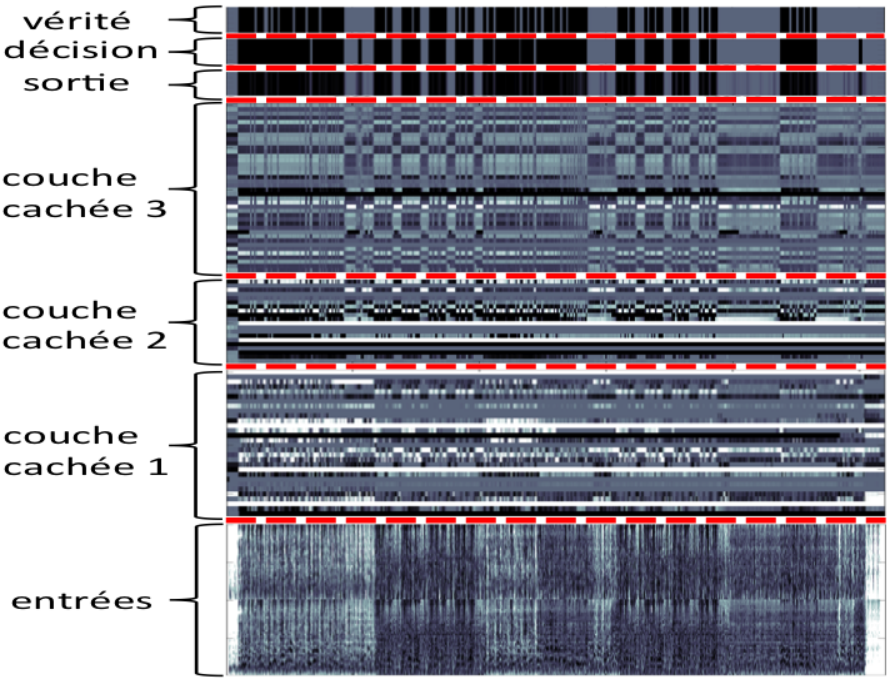
sont dans une région de l'espace des paramètres déjà associée à de bonnes performances : après une époque de ré-entraînement, le taux de bonne classification est déjà de 85.58%.

TABLE 5.3 – Résultats de détection de voix, entraînement sur [ApprentissageDVC], validation sur [ValDVC-BLSTM] et test sur [TestDVC]

	Entraînement brut	Ré-entraînement
Taux de bonne classification (%)	86.24	87.00
Rappel (%)	84.26	89.49
Précision (%)	86.40	84.14
F-mesure	85.32	86.73



(a) Fichier "03 - School" de la base de test Jamendo



(b) Fichier "05 - Crepuscule" de la base de test Jamendo

FIGURE 5.3 – Représentations internes du réseau - échelle de couleur entre -1 (blanc) et 1 (noir)

5.6 Analyse du fonctionnement du réseau

5.6.1 Fonctionnement interne

Afin d'essayer de mettre en évidence ce qu'il se passe dans le réseau lors de la classification d'une séquence, on représente figure 5.3 la séquence de vecteurs d'entrée ainsi que les sorties de chaque couche pour deux fichiers de la base de test Jamendo. Ces morceaux ont été choisis pour être assez différents en style et instrumentation. Les vecteurs d'entrée sont extraits à partir des signaux issus de la double HPSS (voir formule 2.3.5), la moitié inférieure correspond au signal $h_2(t)$ et celle supérieur à $p_2(t)$. On observe également la décision du système de classification qui correspond au seuillage de la sortie du réseau ainsi que la vérité terrain. Au fur et à mesure de la profondeur du réseau, les sorties des couches sont de plus en plus stables et une structure temporelle claire se dégage, avec l'apparition de segments associés à la présence ou à l'absence de voix. On passe ainsi d'une représentation bas-niveau et très variable, issue d'un banc de filtres sur une échelle Mel, à une représentation simple en sortie de la troisième couche cachée, où les neurones rendent compte de la présence ou de l'absence de voix. Cette structure en sortie de la dernière couche cachée est de plus assez générale : comme le montre la comparaison des figures 5.3a et 5.3b, les sorties des deux premières couches cachées varient selon le fichier en entrée, mais la dernière couche présentera toujours cet apparence proche d'un damier, où les neurones semblent en quelque sorte osciller entre deux états selon la présence ou l'absence de voix.

On remarque cependant que certains neurones ont tendance à saturer en permanence, et qu'au sein de chaque couche, des neurones présentent des sorties très corrélées. Nous pouvons déduire de ces observations que le nombre de neurones composant l'architecture pourrait être réduit. Un neurone qui agit de la même façon qu'un autre étant sûrement peu utile.

5.6.2 Mise en évidence de la prise en compte d'un contexte temporel

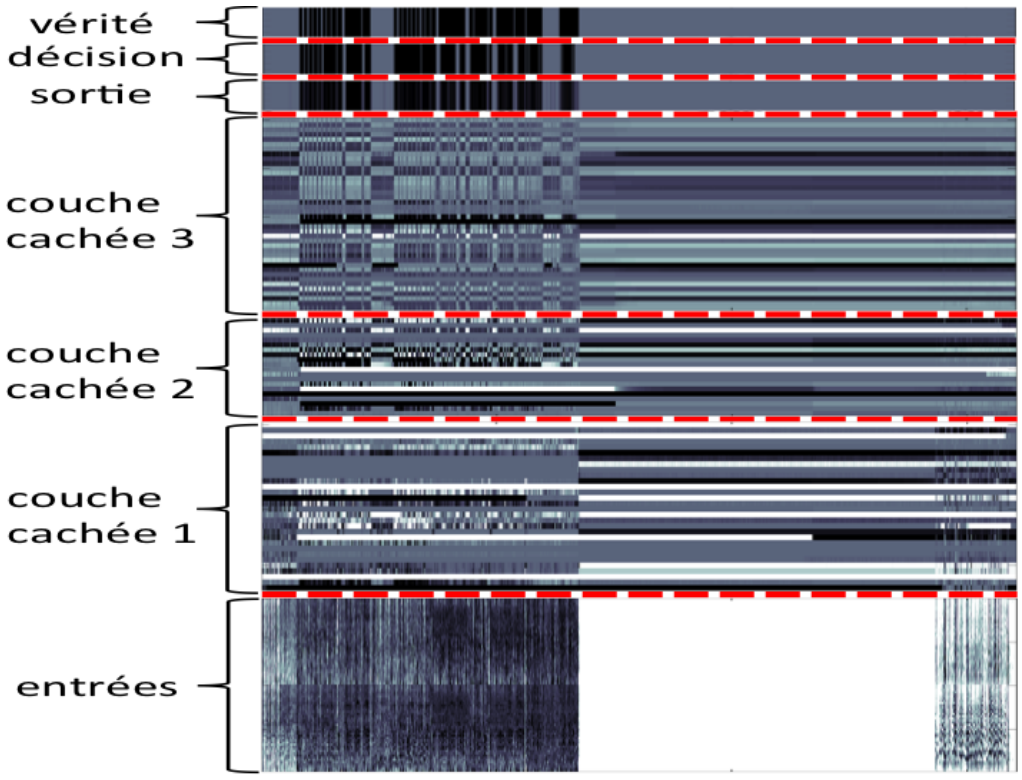


FIGURE 5.4 – Mise en évidence de la prise en compte d'un contexte temporel à partir d'un extrait de "03 - Say me goodbye"

Le morceau "03 - Say me goodbye" de la base de test Jamendo contient un longue partie de silence. En observant le fonctionnement interne du réseau pour ce silence figure 5.4, on se rend compte que les sorties de certains neurones continuent de varier bien que les vecteurs d'entrée restent fixes. Cette observation permet de mettre en évidence le fait qu'un contexte temporel soit pris en compte par le système de classification. En particulier, cet effet peut être compris grâce au fonctionnement des blocs LSTM, où les cellules mémoires sont capables de mémoriser un état pendant une certaine durée.

5.6.3 Estimation de la probabilité de présence de voix

La sortie du réseau est tracée figure 5.5 pour un extrait du morceau "03 - School". Bien que l'estimation de la probabilité de présence de voix soit légèrement bruitée, la décision effectuée en la seuillant (par rapport à 0.5) reste stable à l'intérieur des zones en présence de voix. La décision prise par le système de classification est ainsi très proche de la vérité terrain et aucun post-traitement ne semble nécessaire pour améliorer les résultats de classification. Les seules imperfections se situent aux extrémités des zones d'activité vocale. L'absence de nécessité d'un lissage temporel vient du fait que la méthode de classification utilisée ici prend en compte de façon intrinsèque un contexte temporel. On voit donc l'intérêt d'utiliser un réseau BLSTM pour des tâches de classification de séquences.

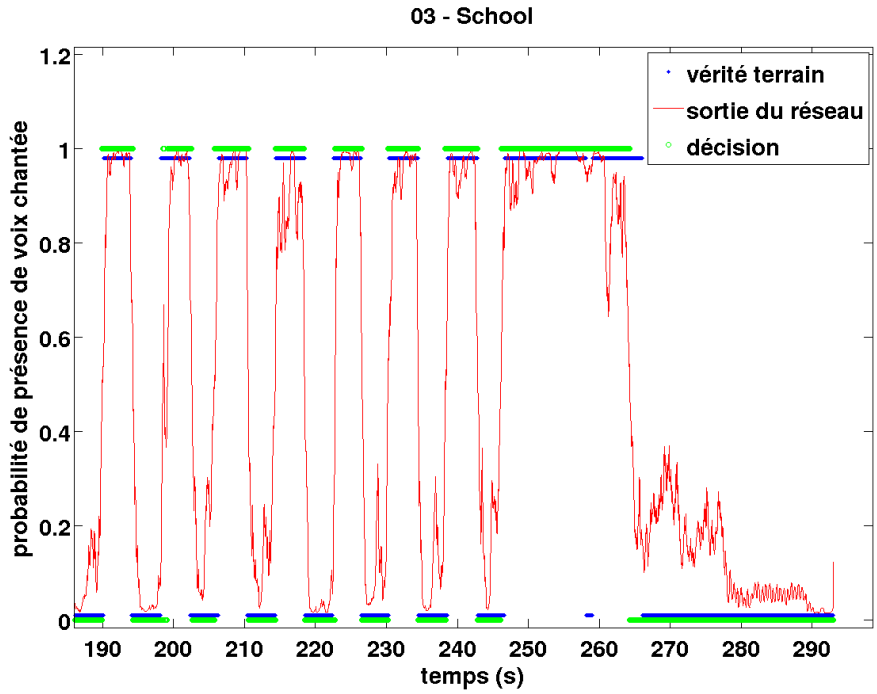


FIGURE 5.5 – Sortie du réseau BLSTM pour un extrait de "03 - School" issu de la base de test Jamendo

5.7 Intégration à un système d'estimation de mélodie

Pour mettre en évidence le gain apporté par une détection de voix robuste au niveau des performances globales d'un système d'estimation de mélodie, nous avons intégré notre méthode à l'estimation de mélodie de Tachibana. Il s'agit donc de remplacer la détection d'activité vocale utilisant un seuillage de la distance de Mahalanobis (voir section 2.3.2.4) par la méthode que nous avons décrite précédemment, employant un réseau BLSTM entraîné sur la base [ApprentissageDVC]. Les résultats sont présentés tableau 5.4. L'augmentation de l'*overall accuracy* est de plus de 15%, ce qui correspond à une nette amélioration. On voit que le taux de fausse-alarme a augmenté, mais cela est largement compensé par le gain au niveau du rappel. Rappelons que ces mesures sont moyennées sur l'ensemble des fichiers de la base.

TABLE 5.4 – Résultats moyens de détection de mélodie sur la base [TestMélodie] - TOOS1 : Tachibana et al. (ancienne méthode de détection de voix) - TOOS2 : Tachibana et al. (nouvelle méthode de détection de voix)

	rappel (%)	**taux de fausse alarme (%)**	
TOOS1	58.54	40.82	
TOOS2	92.30	54.14	
	***raw pitch accuracy* (%)**	***raw chroma accuracy* (%)**	***overall accuracy* (%)**
TOOS1	74.98	82.05	49.37
TOOS2	74.98	82.05	64.45

5.8 Conclusion

Dans cette partie nous avons présenté la nouvelle méthode de détection de la voix chantée que nous avons mise en place. Celle ci utilise des descripteurs bas niveau en entrée d'un réseau de neurones BLSTM. Définir l'architecture du réseau est une partie essentielle dans une méthode d'apprentissage profond. Cependant, c'est souvent une étape qui est peu détaillée. Nous avons ici présenté une approche d'apprentissage de l'architecture du réseau, certes empirique, mais automatique et générale. Notons cependant que cette technique incrémentale fait l'hypothèse que l'ajout d'une couche cachée ne modifie pas la largeur optimale estimée pour les couches en dessous. Cette hypothèse, assez forte, est cependant nécessaire pour rendre l'apprentissage de l'architecture du réseau possible. L'évaluation de notre méthode de détection de la voix chantée sur une base de référence nous a permis de montrer une nette amélioration de l'état de l'art. Nous avons également montré que notre système permettrait d'améliorer significativement la détection de voix dans ADX TRAX. Nous avons aussi effectué une première analyse du fonctionnement interne du réseau, en observant les sorties de chaque neurone. Ceci nous a permis de mettre en évidence qu'au fur et à mesure de la propagation dans la profondeur du réseau, la représentation des données se simplifie et s'adapte à notre tâche de détection de voix. Nous avons également remarqué la prise en compte du contexte temporel et le fait que ce système de classification ne nécessite pas de post-traitement. Finalement, en intégrant cette méthode de détection de la voix chantée au système d'estimation de mélodie de Tachibana, les performances globales ont été améliorées de plus de 15%, d'où l'importance de cette étape de traitement pour estimer de façon robuste la mélodie.

Chapitre 6

Conclusion et perspectives

La problématique de ce stage était d'évaluer quel type d'information permettrait d'améliorer l'automatisation du processus de séparation de la voix chantée. Dans un premier temps, nous nous sommes concentrés sur l'estimation de la mélodie vocale. Grâce à la mise en place d'un banc d'essai, nous avons pu comparer les performances de deux algorithmes de l'état de l'art que nous avons implémentés, ainsi que celles de la méthode d'estimation de mélodie d'ADX TRAX. Cette évaluation nous a permis de mettre en évidence l'importance de la détection de la voix chantée pour obtenir de bonnes performances globales d'estimation de mélodie. A fortiori, les méthodes de séparation de la voix chantée utilisant la mélodie comme *a priori* nécessitent d'avoir un système de détection de la voix chantée robuste. Et on peut de plus argumenter que de façon générale, l'information de présence de voix est utile à n'importe quel algorithme de séparation de la voix chantée, que ce soit *a priori* ou *a posteriori*. Nous avons ainsi pu conclure que l'information de présence de voix était la plus importante pour automatiser la séparation de la voix chantée. Dans le cadre du logiciel ADX TRAX, celle-ci se fait au début de la chaîne de traitement et contraint l'estimation de mélodie et la séparation de la voix, c'est donc un bloc de traitement essentiel.

Après avoir identifié cette axe de recherche, nous nous sommes attachés à mettre en place un système robuste de détection de la voix chantée. Pour ce faire, nous avons chosi de nous orienter vers une approche d'apprentissage profond. La nouvelle méthode que nous avons mise en place s'appuie sur deux caractéristiques principales : l'extraction de descripteurs bas niveau à partir des signaux issus d'une double séparation harmonique/percussif et un système de classification par réseau de neurones récurrent utilisant des blocs longue mémoire à court-terme. Cette approche présente des résultats meilleurs que l'état de l'art sur une base de données de référence. De plus, lorsque nous l'avons intégrée à la méthode d'estimation de mélodie de Tachibana et al., la mesure d'*overall accuracy* a augmenté de 15% sur notre base de test. En combinant l'estimation de mélodie de Tachibana et al. et notre détection de voix, nous avons obtenu un système d'estimation de mélodie dont les performances sont les plus élevées sur notre base de test.

Deux perspectives principales se dégagent pour de futurs travaux concernant la détection de la voix chantée :

- Mettre en évidence l'importance du choix des descripteurs pour le système de classification par réseau BLSTM. Bien que les performances soient bonnes avec des descripteurs issus d'un banc de filtres sur une échelle Mel, il serait intéressant de savoir si elles peuvent encore être améliorées avec des descripteurs plus élaborés, notamment en ce qui concerne la mesure de précision pour laquelle l'amélioration est la moins importante. La méthode proposée par Lehner dans [37] a pour objectif de réduire le taux de fausse-alarme (lié à la mesure de précision) en concevant des descripteurs qui permettent de discriminer la voix chantée des autres instruments harmoniques. Ces descripteurs constituent donc une piste à envisager pour améliorer notre méthode de détection de la voix chantée.

- Il serait également intéressant d'explorer les approches d'apprentissage profond pour l'extraction des descripteurs. Il s'agirait d'entraîner un réseau profond pour extraire des descripteurs, ceux-ci étant ensuite mis en entrée d'un second réseau indépendant du premier, typiquement un réseau récurrent pour opérer une classification de séquences. Un réseau pour extraire des descripteurs peut-être entraîné de façon supervisée en associant à chaque vecteur d'entrée un vecteur cible qui l'encode. Cette approche est utilisée dans [63] pour une tâche de reconnaissance de parole. Il est montré qu'utiliser les sorties des couches cachées comme entrées du système de classification permet d'améliorer les résultats. Cette observation est particulièrement intéressante car elle permet d'interpréter les sorties des couches cachées comme des descripteurs complémentaires. La piste des réseaux de neurones convolutionnels pour l'extraction de descripteurs est également à envisager, ces derniers exhibent des propriétés d'invariance des sorties pour la direction sur laquelle les poids du réseau sont partagés. En reconnaissance de parole, cette caractéristique est utile par exemple dans la direction fréquentielle, car les formants d'un même phonème peuvent être localisés légèrement différemment en fréquence selon le locuteur ou également pour un même locuteur [64].

Les résultats obtenus grâce au réseau BLSTM pour la détection de la voix chantée permettent d'espérer obtenir de bonnes performances si on étend cette approche à l'estimation de mélodie. Il est important de mentionner que les résultats provenant de l'évaluation MIREX en estimation de mélodie ne s'améliorent quasiment plus depuis 2010 (voir [9], figure 7). Il semble donc pertinent d'explorer la piste des méthodes d'apprentissage profond pour l'estimation de mélodie, celle-ci n'ayant jamais été encore considérée à notre connaissance. Du fait de la généralité d'un réseau BLSTM pour définir un système de classification de séquences, on peut également considérer utiliser une telle méthode pour d'autres tâches permettant de guider le processus de séparation de la voix chantée : détection de consonnes, détection des instruments présents à chaque instant...

Remerciements

Je tiens à remercier tout particulièrement mon directeur de stage, Romain Hennequin, pour la confiance qu'il m'a accordée et pour son aide durant ces presque 6 mois de stage. Je garderai assurément à l'esprit pour la suite de mon parcours sa façon de travailler. Merci également à Pierre Leveau, qui avec Romain a su m'orienter vers des approches d'apprentissage profond. Ce choix de thématique de recherche ayant grandement participé à ma motivation. Je tiens à remercier également Telma Oliva et Guillaume Vincke pour leur accueil chez Audionamix. Merci à mes co-stagiaires, Alexandre Vaneph et aussi Sébastien Eggenspieler qui m'a laissé utiliser ses ressources GPU. Merci également à Melody, qui rien que par son prénom était dans le thème de ce stage.

Bibliographie

[1] Alexey Ozerov and Cédric Févotte. Multichannel nonnegative matrix factorization in convolutive mixtures for audio source separation. *Audio, Speech, and Language Processing, IEEE Transactions on*, 18(3) :550–563, 2010.

[2] Jean-Louis Durrieu, Alexey Ozerov, Cédric Févotte, Gaël Richard, and Bertrand David. Main instrument separation from stereophonic audio signals using a source/filter model. In *European Signal Processing Conference (EUSIPCO), Glasgow, Scotland*, 2009.

[3] Romain Hennequin, Bertrand David, and Roland Badeau. Score informed audio source separation using a parametric model of non-negative spectrogram. In *ICASSP*, pages 45–48, 2011.

[4] Paris Smaragdis and Gautham J Mysore. Separation by "humming" : User-guided sound extraction from monophonic mixtures. In *Applications of Signal Processing to Audio and Acoustics, 2009. WASPAA'09. IEEE Workshop on*, pages 69–72. IEEE, 2009.

[5] Roland Badeau. Gaussian modeling of mixtures of non-stationary signals in the time-frequency domain (hr-nmf). In *Applications of Signal Processing to Audio and Acoustics (WASPAA), 2011 IEEE Workshop on*, pages 253–256. IEEE, 2011. Présentation disponible sur http://techtalks.tv/talks/gaussian-modeling-of-mixtures-of-non-stationary-signals-in-the-time-frequency-domain-hr-nmf/54682/.

[6] Chao-Ling Hsu, DeLiang Wang, Jyh-Shing Roger Jang, and Ke Hu. A tandem algorithm for singing pitch extraction and voice separation from music accompaniment. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(5) :1482–1491, 2012.

[7] Li Deng and Dong Yu. Deep learning : Methods and applications. Technical Report MSR-TR-2014-21, January 2014.

[8] J Stephen Downie. The music information retrieval evaluation exchange (2005-2007) : A window into music information retrieval research. *Acoustical Science and Technology*, 29(4) :247–255, 2008.

[9] J. Salamon, E. Gomez, D. Ellis, and G. Richard. Melody extraction from polyphonic music signals : Approaches, applications and challenges. *IEEE Signal Processing magazine*, 31(2) :118–134, March 2014.

[10] J. Salamon and E. Gómez. Melody extraction from polyphonic music signals using pitch contour characteristics. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(6) :1759–1770, Aug. 2012.

[11] Derek W Robinson and R So Dadson. A re-determination of the equal-loudness relations for pure tones. *British Journal of Applied Physics*, 7(5) :166, 1956.

[12] Tzu-Chun Yeh, Ming-Ju Wu, Jyh-Shing Roger Jang, Wei-Lun Chang, and I-Bin Liao. A hybrid approach to singing pitch extraction based on trend estimation and hidden markov models. In *ICASSP*, pages 457–460. IEEE, 2012.

[13] Chao ling Hsu and Jyh shing Roger Jang. Singing pitch extraction at mirex 2010. In *In 6 th Music Information Retrieval Evaluation eXchange (MIREX), extended abstract.*

[14] Hideyuki Tachibana, Takuma Ono, Nobutaka Ono, and Shigeki Sagayama. Melody line estimation in homophonic music audio signals based on temporal-variability of melodic source. In *ICASSP*, pages 425–428, 2010.

[15] Matija Marolt. On finding melodic lines in audio recordings. In *7th Int. Conf. on Digital Audio Effects (DAFx-04)*, Naples, Italy, Oct. 2004.

[16] Seokhwan Jo, Sihyun Joo, and Chang D Yoo. Melody pitch estimation based on range estimation and candidate extraction using harmonic structure model. In *INTERSPEECH*, pages 2902–2905, 2010.

[17] Karin Dressler. Audio melody extraction for mirex 2009. *5th Music Inform. Retrieval Evaluation eXchange (MIREX)*, 79 :100–115, 2009.

[18] Pablo Cancela. Tracking melody in polyphonic audio. mirex 2008. In *In MIREX Audio Melody Extraction Contest Abstracts*. Citeseer, 2008.

[19] Masataka Goto. A real-time music-scene-description system : predominant-f0 estimation for detecting melody and bass lines in real-world audio signals. *Speech Communication*, 43(4) :311–329, 2004.

[20] J-L Durrieu, Gaël Richard, Bertrand David, and Cédric Févotte. Source/filter model for unsupervised main melody extraction from polyphonic audio signals. *Audio, Speech, and Language Processing, IEEE Transactions on*, 18(3) :564–575, 2010.

[21] Graham E Poliner and Daniel PW Ellis. A classification approach to melody transcription. In *ISMIR 2005 : 6th International Conference on Music Information Retrieval : Proceedings : Variation 2 : Queen Mary, University of London & Goldsmiths College, University of London, 11-15 September, 2005*, pages 161–166. Queen Mary, University of London, 2005.

[22] Christopher Sutton. Transcription of vocal melodies in popular music. *Report for the degree of MSc in Digital Music Processing at the University of London*, 2006.

[23] Robert C Maher and James W Beauchamp. Fundamental frequency estimation of musical signals using a two-way mismatch procedure. *The Journal of the Acoustical Society of America*, 95(4) :2254–2263, 1994.

[24] J. Salamon, E. Gómez, and J. Bonada. Sinusoid extraction and salience function design for predominant melody estimation. In *14th Int. Conf. on Digital Audio Effects (DAFx-11)*, pages 73–80, Paris, France, Sep. 2011.

[25] Equal loudness filter. http://replaygain.hydrogenaud.io/proposal/equal_loudness.html. [Online].

[26] Hideyuki Tachibana, Takuma Ono, Nobutaka Ono, and Shigeki Sagayama. Extended abstract for audio melody extraction in mirex 2010 (to be modified).

[27] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *Automatica*, 11(285-296) :23–27, 1975.

[28] Lise Regnier and Geoffroy Peeters. Singing voice detection in music tracks using direct voice vibrato detection. In *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*, pages 1685–1688. IEEE, 2009.

[29] Tin Lay Nwe, Arun Shenoy, and Ye Wang. Singing voice detection in popular music. In *Proceedings of the 12th annual ACM international conference on Multimedia*, pages 324–327. ACM, 2004.

[30] Martın Rocamora and Perfecto Herrera. Comparing audio descriptors for singing voice detection in music audio files. In *Brazilian Symposium on Computer Music, 11th. San Pablo, Brazil*, volume 26, page 27, 2007.

[31] George Tzanetakis. Song-specific bootstrapping of singing voice structure. In *Multimedia and Expo, 2004. ICME'04. 2004 IEEE International Conference on*, volume 3, pages 2027–2030. IEEE, 2004.

[32] Mathieu Ramona, Gaël Richard, and Bertrand David. Vocal detection in music with support vector machines. In *Proc. ICASSP '08*, pages 1885–1888, March 31 - April 4 2008.

[33] Matthias Mauch, Hiromasa Fujihara, Kazuyoshi Yoshii, and Masataka Goto. Timbre and melody features for the recognition of vocal activity and instrumental solos in polyphonic music. In *ISMIR*, pages 233–238, 2011.

[34] Geoffroy Peeters. A large set of audio features for sound description (similarity and classification) in the CUIDADO project. Technical report, Icram, 2004.

[35] Shankar Vembu and Stephan Baumann. Separation of vocals from polyphonic audio recordings. In *IN : PROC. ISMIR2005. (2005) 337–344*, pages 337–344, 2005.

[36] Bernhard Lehner, Reinhard Sonnleitner, and Gerhard Widmer. Towards light-weight, real-time-capable singing voice detection. In *ISMIR*, pages 53–58, 2013.

[37] Bernhard Lehner, Gerhard Widmer, and Reinhard Sonnleitner. On the reduction of false positives in singing voice detection. In *Proc. ICASSP '14*, May 2014.

[38] Frank Rosenblatt. The perceptron : a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6) :386, 1958.

[39] Michiel Hermans and Benjamin Schrauwen. Training and analysing deep recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 190–198, 2013.

[40] Paul J Werbos. Backpropagation through time : what it does and how to do it. *Proceedings of the IEEE*, 78(10) :1550–1560, 1990.

[41] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02) :107–116, 1998.

[42] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, 45(11) :2673–2681, 1997.

[43] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8) :1735–1780, 1997.

[44] Félix Gers. *Long Short-Term Memory in Recurrent Neural Networks*. PhD thesis, Ecole Polytechnique Fédérale de Lausanne, 2001.

[45] Alex Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*. PhD thesis, Technische Universität München, July 2008.

[46] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5) :602–610, 2005.

[47] Florian Eyben, Felix Weninger, Stefano Squartini, and Bjorn Schuller. Real-life voice activity detection with lstm recurrent neural networks and an application to hollywood movies. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 483–487. IEEE, 2013.

[48] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6645–6649. IEEE, 2013.

[49] Raymond Brueckner and Bjorn Schulter. Social signal classification using deep blstm recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 4823–4827. IEEE, 2014.

[50] Douglas Eck and Juergen Schmidhuber. A first look at music composition using lstm recurrent neural networks. *Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale*, 2002.

[51] Felix Weninger, Jürgen Geiger, Martin Wöllmer, Björn Schuller, and Gerhard Rigoll. The munich feature enhancement approach to the 2nd chime challenge using blstm recurrent neural networks. In *Proceedings of the 2nd CHiME workshop on machine listening in multisource environments*, pages 86–90, 2013.

[52] Richard P Lippmann. An introduction to computing with neural nets. *ASSP Magazine, IEEE*, 4(2) :4–22, 1987.

[53] Johan Hastad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 6–20. ACM, 1986.

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

[55] Eric J Humphrey, Juan P Bello, and Yann LeCun. Feature learning and deep architectures : new directions for music informatics. *Journal of Intelligent Information Systems*, 41(3) :461–481, 2013.

[56] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning ? *The Journal of Machine Learning Research*, 11 :625–660, 2010.

[57] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19 :153, 2007.

[58] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[59] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition : The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6) :82–97, 2012.

[60] Abdel-rahman Mohamed, George E Dahl, and Geoffrey Hinton. Acoustic modeling using deep belief networks. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(1) :14–22, 2012.

[61] Rms normalization. http://wiki.hydrogenaud.io/index.php?title=ReplayGain_1.0_specification. [Online].

[62] Hideyuki Tachibana, Nobutaka Ono, and Shigeki Sagayama. Singing voice enhancement in monaural music signals based on two-stage harmonic/percussive sound separation on multiple resolution spectrograms. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 22(1) :228–237, 2014.

[63] Li Deng and Jianshu Chen. Sequence classification using the high-level features extracted from deep neural networks. In *Proc. ICASSP*, May 2014.

[64] Ossama Abdel-Hamid, Li Deng, and Dong Yu. Exploring convolutional neural network structures and optimization techniques for speech recognition. In *INTERSPEECH*, pages 3366–3370, 2013.

Annexe A

Bases de données

Nous présentons ici les bases de données constituées pour l'entraînement et le test des systèmes de détection de voix chantée ainsi que la base de test constituée pour les méthodes d'estimation de mélodie. Les fichiers sélectionnés sont tous associés à une mélodie principale portée par la voix chantée.

Ces bases correspondent à un assemblage de différents sous-ensembles de bases utilisées dans le domaine du MIR, pour les deux tâches considérées, elles intègrent également une base dont nous disposons en interne.

ADC2004 et MIREX05 [1] sont des bases utilisées pour le concours MIREX, elles sont annotées en mélodie. La première correspond exactement à une base de MIREX, la seconde est un sous-ensemble. Les styles représentés sont Pop, Jazz, Opéra, R'n'B et Rock. ADC2004 comprend 12 extraits d'environ 20s de musique avec voix chantée et MIREX05 en comprend 9 d'environ 30s.

RWC Popular Music et RWC Royalty Music [2] sont des bases annotées en mélodie et également en activité vocale pour RWC Popular Music. La première est assez homogène et représentative du style Pop/Rock, la seconde contient des chansons pour enfants. Elles disposent respectivement de 100 et 20 morceaux de quelques minutes.

MIR-1k pour MIREX [3] est une base annotée en mélodie. Elle est composée d'extraits de karaoke amateur chinois d'une durée entre 4 et 13 secondes.

Jamendo est une base de donnée ayant été constituée pour les travaux sur la détection de voix chantée menés dans [32] [4]. Elle est plutôt représentative de la musique pop/rock. Elle est constituée d'une base d'apprentissage de 61 morceaux, d'une base de validation et d'une base de test comprenant chacune 16 morceaux. Chacun des fichiers fait quelques minutes.

Base de test [TestMélodie] pour l'estimation de mélodie vocale :

- 4 extraits d'environ 30 secondes des bases de données ADC2004 et MIREX05. Pop, Jazz, Opéra, R'n'B, Rock.
- 100 extraits de quelques secondes issus de la base de données MIR-1k. Karaoke amateur chinois.
- 2 morceaux de quelques minutes issus d'une base de donnée interne. Pop, Rap.
- 10 morceaux de quelques minutes issus de la base de donnée RWC Popular Music. Pop, Rock.
- 1 morceau de quelques minutes issus de la base de donnée RWC Royalty Music. Chansons pour enfants.

Base d'apprentissage [ApprentissageDVC] pour la détection de voix chantée :

- 12 extraits d'environ 30 secondes issus des bases de données ADC2004 et MIREX05. Pop, Jazz, Opéra, R'n'B, Rock.
- 8 morceaux de quelques minutes issus d'une base de donnée interne. Pop, Rock, R'n'B, Rap, Jazz...
- 80 morceaux de quelques minutes issus de la base de donnée RWC Popular Music. Pop, Rock.

[1] disponibles depuis http://labrosa.ee.columbia.edu/projects/melody/
[2] disponibles depuis https://staff.aist.go.jp/m.goto/RWC-MDB/AIST-Annotation/
[3] disponible depuis https://sites.google.com/site/unvoicedsoundseparation/mir-1k
[4] disponible depuis http://www.mathieuramona.com/wp/data/jamendo/

- 12 morceaux de quelques minutes issus de la base de donnée RWC Royalty Music. Chansons pour enfants.
- 61 morceaux de quelques minutes issus de la base de donnée d'entraînement Jamendo. Pop, Rock.

Base de test [TestDVC] pour la détection de voix chantée :

- 9 extraits d'environ 30 secondes issus des bases de données ADC2004 et MIREX05. Pop, Jazz, Opéra, R'n'B, Rock.
- 3 morceaux de quelques minutes issus d'une base de donnée interne. Pop, Rock, Rap.
- 20 morceaux de quelques minutes issus de la base de donnée RWC Popular Music. Pop, Rock.
- 3 morceaux de quelques minutes issus de la base de donnée RWC Royalty Music. Chansons pour enfants.
- 16 morceaux de quelques minutes issus de la base de test Jamendo. Pop, Rock.

Base de validation [ValDVC-BLSTM] pour la détection de voix chantée avec réseau BLSTM :

- 5 extraits d'environ 30 secondes issus des bases de données ADC2004 et MIREX05. Pop, Jazz, Opéra.
- 1 morceau de quelques minutes issu d'une base de donnée interne. Rock.
- 10 morceaux de quelques minutes issus de la base de donnée RWC Popular Music. Pop, Rock.
- 2 morceaux de quelques minutes issus de la base de donnée RWC Royalty Music. Chansons pour enfants.
- 16 morceaux de quelques minutes issus de la base de validation Jamendo. Pop, Rock.

Base de test [TestDVC-BLSTM] pour la détection de voix chantée avec réseau BLSTM :

- 4 extraits d'environ 30 secondes issus des bases de données ADC2004 et MIREX05. Pop, Jazz, Opéra.
- 2 morceaux de quelques minutes issus d'une base de donnée interne. Pop, Rap.
- 10 morceaux de quelques minutes issus de la base de donnée RWC Popular Music. Pop, Rock.
- 1 morceau de quelques minutes issu de la base de donnée RWC Royalty Music. Chanson pour enfants.
- 16 morceaux de quelques minutes issus de la base de test Jamendo. Pop, Rock.

Evidemment, bien qu'assemblées depuis les mêmes ensembles, aucun fichier n'est présent à la fois dans une base ayant servie à l'apprentissage et dans une base de test.