MASTER ATIAM

THÈSE DE MASTER 2

Transcription automatique de batterie

Auteur:
Pierre-Amaury GRUMIAUX

Superviseur:
Geoffroy PEETERS

Institut de Recherche et Coordination Acoustique/Musique

Du 19 février au 31 juillet 2018

Contents

List of Figures

List of Tables

Chapter 1

Résumé

La transcription automatique de batterie est une sous-tâche de la transcription automatique de musique qui convertit un signal audio contenant de la batterie en évènements temporels. Les premières tentatives pour résoudre ce problème reposent sur des techniques de traitement du signal et de détection d'attaque. Les nouvelles techniques se concentrent sur l'utilisation de réseaux de neurones, récurrents, convolutifs ou convolutifs récurrents, pour analyser les caractéristiques acoustiques du signal ou les dépendances temporelles des éléments du signal entre eux. Nous proposons de reprendre les idées de deux publications récentes. Dans un premier temps nous évaluons différents réseaux de neurones, récurrents, convolutifs et convolutifs récurrents, sur 3 ensembles de données distincts : SMT-Drums (batterie seule), RBMA 13 (véritables morceaux) et ENST-drums (batterie seule puis batterie + accompagnement). Il est montré que les réseaux avec des composantes convolutives présentent des bonnes performances comparés aux réseaux seulement récurrents. Nous reprenons également l'idée d'ajouter des informations de temps et premiers temps, soit en entrée du réseau, soit avec un apprentissage en multi-tâche. Ces informations supplémentaires permettent d'améliorer les systèmes dans une certaine mesure, surtout lorsqu'elles sont renseignées en entrée. Le multi-tâche fonctionne mieux avec les réseaux récurrents, qui sont déjà capables de capter des dépendances temporelles entre les évènements du signal. Toutefois, l'apprentissage souffre de l'absence d'ensembles de données de gros volume. Pour lever ce problème, nous utilisons le paradigme élève-professeur pour entraîner nos réseaux sur un nouvel ensemble de données de grande taille, Billboard 800, composé de 800 extraits musicaux dans 4 genres (latino, hip-hop, pop, rock) issus des classements de Billboard. Nous montrons que cela permet d'améliorer la performance des systèmes dans une certaine mesure. Nos travaux ont permis d'explorer quelques pistes d'amélioration des systèmes de TAB, qui mériteraient des études plus poussées pour confirmer les résultats.

Chapter 2

Remerciements

Je remercie dans un premier temps mon tuteur, Geoffroy Peeters, pour m'avoir fait confiance et permis d'effectuer ce stage de recherche sur un sujet qui m'a passionné. Il a su me guider dans mes recherches pendant nos échanges pour que je ne m'égare pas en chemin. Je remercie également mes collègues du quotidien, de l'équipe Analyse-Synthèse, avec lesquels j'ai pu avoir des discussions constructives et très intéressantes ou des moments plus détendus : Hugo, Guillaume, Dogac, Hadrien, Alice, Gabriel, Rafael, Pierre, Jordan, et j'en oublie certainement. Enfin, je remercie mes collègues issus comme moi du master ATIAM, ou bien seulement de passage, avec qui j'ai pu avoir des débats toujours plus houleux lors des pauses déjeuner : Judy, Pierre, Pierre, Mathis, Victor, Thomas, Cyran, Bavo, Paul, Nicolas, Godelive.

Chapter 3

Introduction

Le problème de transcription automatique de batterie (TAB) fait partie intégrante du problème de transcription automatique de musique (TAM). Cette dernière est un des enjeux les plus compliqués dans la discipline d'extraction d'information musicale (MIR), et peut être simplement vue comme l'inverse de la création musicale : au lieu de créer la musique à partir d'une partition, un système TAM vise à créer la partition à partir d'un signal sonore. Concernant la TAM, les systèmes se concentrent sur la détection de la hauteur, du moment précis de l'attaque et de la durée de chaque note jouée par tous les instruments audibles d'un morceau. Plus spécifiquement, la TAB vise à transcrire seulement le moment précis de l'attaque car ce sont les instruments percussifs auxquels on s'intéresse. Ainsi, les deux systèmes s'avèrent relativement différents en terme d'approche : des algorithmes de TAB se concentrent davantage sur la détection d'évènements sonores impulsifs, alors que des systèmes de TAM pourront tirer profit des règles grammaticales de la théorie musicale, comme pour le problème de reconnaissance automatique de parole (automatic speech recognition, ASR).

Une batterie est un instrument à percussions constitué de plusieurs éléments, dont le nombre diffère en fonction des batteurs, mais dont nous donnons les principaux : la grosse caisse (bass drum, BD), la caisse claire (snare drum, SD), le charleston (hihat, HH), les cymbales et les toms. En général, dans la littérature, sont seulement considérés la grosse caisse, la caisse claire et le charleston, ce que nous faisons également dans nos travaux. Le jeu du batteur suit un certain rythme et plusieurs éléments de batterie peuvent être joués en même temps. Cette particularité rend la transcription exacte plus compliquée à obtenir puisque les BD, SD et HH possèdent des spectres musicaux qui se chevauchent. Une BD a un spectre qui s'étend habituellement dans l'intervalle 50Hz-5kHz, une SD peut s'entendre de 100Hz à 3kHz, et une HH s'étale sur un large spectre, d'environ 300Hz à 3kHz mais peut être entendue jusqu'à 15kHz.

Les applications possibles de la transcription automatique de batterie sont variées. D'un point de vue éducatif, un système de TAB pourrait assister un batteur qui s'entraîne en analysant sa performance en temps réel, en comparant la transcription réalisée avec le jeu attendu. Dans la production musicale, transcrire les parties de batterie pourrait être utile pour le remixage, par exemple en ayant la possibilité de remplacer les échantillons de batterie par d'autres, ou encore pour extraire des parties de batterie isolément. On pourrait également trouver des applications dans le MIR, en obtenant des informations sur le type de rythme. Ces informations peuvent être utiles pour structurer un ensemble de fichiers musicaux en différents styles, ou encore en les classant selon un critère de ce qui est dansable ou non.

Le but du stage dont traite ce rapport est l'implémentation et l'analyse de deux publications de l'état de l'art de la transcription automatique de batterie [Vog+17] [WL17]. Le premier propose l'utilisation de différents réseaux de neurones (récurrents, convolutifs et hybrides) ainsi que d'informations supplémentaires de temps et premiers temps, et le deuxième invite à utiliser le paradigme *élève-professeur* pour améliorer les performances de la

transcription automatique de batterie. Nous souhaitons *in fine* mélanger les deux approches.

La suite de ce rapport est structurée de la manière suivante. Le chapitre 4 fait un état de l'art des travaux proposés concernant la TAB. Le chapitre 5 présente les ensembles de données utilisés pour nos travaux et la manière dont nous allons les évaluer. Le chapitre 6 décrit le fonctionnement théorique des réseaux de neurones, puis le chapitre 7 explicite l'application de ces outils à la TAB et décrit le coeur de nos travaux. Enfin, le chapitre 8 présente le paradigme *élève-professeur* et les travaux effectués en lien avec la TAB.

Chapter 4

Etat de l'art de la transcription automatique de batterie

Dans ce chapitre, nous faisons une revue de la littérature liée à la transcription automatique de batterie. Dans le passé, les méthodes de TAB étaient regroupées en quatre catégories [GR08]: les méthodes de segmentation puis classification, les méthodes de séparation et de détection, les méthodes d'adaptation et les méthodes basées sur les modèles de Markov cachés (hidden Markov models, HMM). Plus récemment, [Wu+17] ont proposé de distinguer quatre types de méthodes de TAB plus ou moins semblables que la classification précédente : les méthodes de segmentation, les méthodes de classification, les modèles de language et les méthodes d'activation.

4.1 Méthodes de segmentation

Ce type d'approche se concentre sur les techniques de détection d'évènements, appliquées à la forme d'onde, parfois suivies de techniques de classification. Un évènement de batterie étant par nature percussif, il est direct de vouloir appliquer des méthodes de détection d'évènements comme la détection d'attaque par exemple.

Un des premiers systèmes de TAB a été proposé par [Sch85]. Il repose sur une estimation de l'enveloppe de la forme d'onde et détermine un évènement de batterie en fonction d'un seuil dans son attaque. Un tel système est capable d'identifier des coups de batterie dans des enregistements où elle est seule. [Zil+02] ont proposé une méthode de segmentation basée sur l'utilisation de patrons initiaux pour chaque élément de batterie, créés à partir d'impulsions filtrées. Une corrélation entre le signal analysé et ces patrons, suivie d'une détection de pics, permet d'identifier les événements de batterie attendus. In fine, les patrons de chaque élément de batterie sont mis à jour à partir de la moyenne des signaux correspondant à ces événements. Le procédé est répété jusqu'à une convergence des patrons. Ces méthodes d'analyse-synthèse a l'avantage d'avoir très peu d'a priori, mais est limité par la possible présence de voix ressemblant à des percussions aigües, ou encore par la présence de plusieurs éléments precussifs en même temps. [TKM05] ont proposé de mettre en valeur les éléments percussifs par des filtrages passe-bandes caractéristiques avant de procéder à une détection d'attaque. Ce système ne permet cependant pas une généralisation à d'autres types de percussions.

Ces méthodes dites de segmentation présentent plusieurs avantages. Leur utilisation directe de la forme d'onde permet une facilité d'interprétation des résultats. En terme de calculs, l'efficacité des algorithmes tels que la détection d'attaque ou la détection de pics permet d'utiliser ces systèmes en temps réel, pour des performances en temps réel par exemple. A l'inverse, un tel système ne sera pas robuste à un signal audio comprenant des instruments mélodiques. L'utilisation de la forme d'onde pour l'analyse ne permet également pas de capturer certaines caractéristiques du signal.

4.2 Méthodes de classification

Ce type de système comporte des blocs algorithmiques assez semblables aux méthodes de segmentation, mais met davantage l'accent sur la classification des divers éléments de batterie, quand au contraire les méthodes de segmentation privilégient les aspects d'efficacité et d'interpretabilité. Les méthodes de classification comprennent ainsi souvent un premier bloc de représentation du signal audio, suivi d'un bloc de localisation des potentiels événements de batterie, et enfin un bloc de classification des événements. Ce dernier bloc est d'ailleurs parfois utilisé seul pour le problème de classification d'échantillons de batterie, qu'on peut considérer comme sous-problème de la TAB.

En terme de représentation du signal audio, plusieurs formes ont été proposées dans la littérature. Des représentations temps-fréquence sont souvent avantageuses pour la TAB, majoritairement la transformée de Fourier à court terme [WL17] [SSH16] [SSH17], ou des variantes comme la transformation à Q constant [Ros+15] ou les spectrogrammes à échelle de mel [VDK16], [Vog+17]. Des méthodes ont utilisé d'autres types de caractéristiques comme la valeur efficace du signal [VS+] [TDDB05] [MF07], le barycentre temporel [VS+] [TDDB05] [MF07], le barycentre spectral [VS+] [TDDB05] [MDG13], ou encore les coefficients cepstraux en échelle de mel (mel frequency cepstral coefficient, MFCC) [VS+] [TDDB05] [MF07] [GR04]. Beaucoup utilisent plusieurs de ces caractéristiques à la fois pour augmenter le nombre d'informations utiles à la classification.

Concernant les classifieurs, des méthodes d'apprentissage supervisé ont souvent été utilisées. La méthode des plus proches voisins a beaucoup été mise en place par sa simplicité d'implémentation et d'interprétation [MDG13] [MF07]. Les machines à vecteurs de support ont été proposées avec différentes fonctions noyaux pour lever les problèmes de non-linéarité [GR08], [GR04], [VS+], [TDDB05].

Ces méthodes de classification présentent plusieurs avantages. Tout d'abord, la modularité entre l'étape d'extraction de caractéristiques du signal et le classifieur permet une implémentation aisée et une comparaison facile entre les différentes approches. Il est aussi pratique de rajouter des caractéristiques du signal dans la classification, en augmentant seulement la dimension du classifieur. Cependant, un tel système sera limité pour la transcription d'éléments de batterie qui peuvent être joués au même moment : il faudra par exemple ajouter des classes en combinant les éléments (i.e. BD+HH, BD+SD, HH+SD). Mais si l'on veut retranscrire d'autres éléments percussifs, le nombre de combinaisons va augmenter exponentiellement. Enfin, les méthodes de classification ont des limitations pour reconnaître des sons de batterie en présence d'autres instruments mélodiques non présents pendant l'apprentissage.

4.3 Modèles de langage

Contrairement aux méthodes de classification, les méthodes basées sur un modèle de langage prennent en compte le contexte autour d'un potentiel événement de batterie. Un tel modèle a été créé par une étape d'apprentissage qui va lui indiquer comment le signal évolue habituellement en fonction du temps. Il peut alors calculer la probabilité qu'une certaine séquence soit plausible dans un signal sonore. Un des premiers travaux utilisant un modèle de langage concerne l'annotation de *beatbox* par Nakano et al. [Nak+05]. Le système proposé utilise des MFCC en entrée, et ceux-ci sont décodés en une séquence d'onomatopée en utilisant des HMM et l'algorithme de Viterbi. Les HMM ont aussi été appliqué par Paulus [Pau09], mais avec un modèle sinusoïde-résidu en entrée pour supprimer les composants harmoniques du signal musical.

De tels systèmes utilisant des modèles de langages ne sont pas parfaitement adaptés au problème de TAB. Ces modèles ont été beaucoup utilisés pour la reconnaissance automatique de parole, mais celle-ci comportent de nombreuses différences par rapport à la batterie. La grammaire d'une langue, la longueur d'un phonème ou d'un mot n'ont pas d'homologue direct dans la TAB, et les éléments de batterie sont moins corrélés dans le temps que les phonèmes ou les mots entre eux. A l'inverse, les pauses et silences sont importants dans la musique, alors que dans la reconnaissance de parole ils le sont moins.

4.4 Méthodes d'activations

Les méthodes dites d'activation ont la spécificité de générer une fonction d'activation, c'est-à-dire une probabilité de présence ou non en fonction du temps, d'un élément de batterie pour un signal audio. Cette fonction d'activation est en général suivie d'un algorithme de détection de pic pour isoler les éléments signifiants du problème. Deux familles d'algorithmes se distinguent parmi ces méthodes : les factorisations de matrices et les réseaux de neurones profonds.

La première famille de méthodes applique une factorisation de matrices au spectrogramme (ou variantes) d'un signal audio de manière à le décomposer en des fonctions de bases (représentant dans notre cas les éléments à transcrire) et leur fonction d'activation correspondante. L'algorithme de base, la factorisation en matrice non-négative (non-negative matrix factorization, NMF) a été appliqué à la TAB en premier lieu [PV05]. Des concepts s'apparentant à la NMF ont également été proposées comme la déconvolution en matrices non-négatives [RPL15], la NMF semi-adaptative [DG14] ou la NMF partiellement fixée [WL15a] [WL15b]. Ces méthodes de factorisation ont la contrainte d'avoir une connaissance pré-établie sur les bases à obtenir (ici le contenu fréquentiel des éléments de batterie), sinon les performances chutent fortement.

La deuxième famille de méthodes propose d'utiliser des réseaux de neurones profonds pour la TAB. Ces réseaux de neurones ont l'avantage d'avoir la capacité à apprendre des correspondances non-linéaires entre des entrées et des sorties (ici les signaux sonores ou spectrogrammes, et les activations des éléments de batterie). Parmi les réseaux de neurones proposés pour répondre au problème de TAB, des réseaux de neurones récurrents (recurrent neural network, RNN) et RNN bidirectionnels ont été d'abord proposés [SSH16], ainsi que des RNN avec décalages du temps (annotations décalées avant l'apprentissage, puis redécalées après) [VDK16], ou des variantes comme les réseaux de neurones à mémoire court-terme et long-terme (long short-terme memory, LSTM) et les réseaux de neurones récurrents à portes (gated recurrent unit, GRU) [VDK17]. Les RNN ont la capacité de modéliser des séquences temporelles et ainsi capturer le rythme joué par une phrase de batterie. Plus récemment, les réseaux de neurones convolutifs (convolutional neural network, CNN) et des combinaisons de RNN et de CNN ont été utilisés pour le problème de TAB [SSH17] [Vog+17].Dans [WL17], Wu et al. ont proposé l'utilisation du paradigme *élève-professeur* pour la TAB.

C'est dans ce type de méthodes que nous nous plaçons au cours de ce stage. L'objectif du stage est dans un premier temps d'implémenter les réseaux proposés par Vogl et al. dans [Vog+17] qui constituent l'état de l'art, et de s'approcher de leurs résultats. Nous voulons ensuite utiliser l'approche de Wu et al. dans [WL17] en utilisant le paradigme *élève-professeur* avec des réseaux de neurones *élèves* et *professeurs* différents de ceux proposés dans [WL17] (nous utilisons justement ceux proposés par Vogl et al. que nous avons implémentés).

Chapter 5

Ensembles de données

Dans ce chapitre, nous présentons les différents ensembles de données utilisés pendant ce stage : SMT-Drums, ENST-drums, Red Bull Music Academy 2013 et Billboard 800.

5.1 SMT-Drums

L'ensemble de données SMT-Drums est un ensemble de 608 fichiers audio mono au format WAV (44.1 kHz, 16 bit), d'une durée totale d'environ 130 minutes [DG14]. 104 de ces 608 fichiers sont des boucles de batterie (seulement grosse caisse, caisse claire et charleston) seules, et sont également fournis 312 fichiers correspondant aux éléments de batterie isolés de chaque boucle (mais dont la transcription ne correspond pas avec celle du mélange des 3 éléments de batterie). Ces fichiers sont de plusieurs types :

- *RealDrum01* : véritables enregistrements de batterie acoustique ;
- *TechnoDrum01 et TechnoDrum02* : boucles créées avec des sons de batterie synthétisés ;
- *WaveDrum01 et WaveDrum02* : boucles créées avec des bibliothèques d'échantillons de batterie.

Les ensembles *WaveDrum02 et TechnoDrum02* comprennent 64 boucles de batterie donnés avec les pistes isolés de grosse caisse, caisse claire et charleston. Mélanger ces trois pistes redonnent parfaitement les boucles de batterie.

Pour nos travaux, nous utilisons les boucles mélangées de chaque ensemble, ainsi que les pistes séparées des ensembles *WaveDrum02*. Tous ces fichiers audio sont annotés en fournissant les occurrences en secondes de grosse caisse, caisse claire et charleston. Notons que les pistes séparées sont seulement utilisées pour l'entraînement de nos modèles, mais pas pour l'évaluation.

5.2 ENST-drums

ENST-Drums est une base de données contenant l'enregistrement de pistes audio jouées par trois batteurs professionnels, chacun sur son propre instrument [GR06]. La durée totale d'enregistrement par batteur est d'environ 75 minutes. Pour une sous-partie de cet ensemble (nommée *minus-one*), un fichier d'accompagnement musical est fourni pour chaque piste de batterie (les batteurs ayant joué par dessus ces accompagnements).

Dans nos travaux, seules les versions mixées (ayant été compressées et égalisées) des pistes de *minus-one* sont utilisées. D'une part nous utilisons seulement les pistes avec de la batterie seule (ensemble nommé *ENST solo*), et d'autre part ces pistes de batterie mixées par leur accompagnement respectif (ensemble nommé *ENST mix*). Cela donne un total

d'environ 1 heure pour chacun ensemble. Les annotations des occurrences de tous les éléments de batterie étant fournies, nous utilisons seulement celles relatives à la grosse caisse, la caisse claire et le charleston.

Des annotations de temps et premiers temps ont été également ajoutées pendant le stage. Nous avons utilisé le logiciel de l'IRCAM AudioSculpt en utilisant le module d'annotations de temps et premiers temps inclus, en ajustant minutieusement à la main lorsque l'annotation n'est pas parfaite. Ces annotations sont utilisées pour l'évaluation de modèles, comme cela sera expliqué dans le chapitre 8.

5.3 Red Bull Music Academy 2013 (RBMA 13)

Cet ensemble de données a été annoté par Richard Vogl et al. [Vog+17]. Il s'agit d'un ensemble de 30 fichiers audio issus de Red Bull Music Academy Various Assets 2013, une compilation libre créée par Red Bull Music Academy en 2013 comprenant des titres aussi bien électroniques qu'avec de la voix ou orientés jazz. Trois morceaux parmi ces trentes titres ne contiennent pas de pistes de batterie et sont donc ignorés. Les annotations créées manuellement comprennent la transcription en batterie ainsi que les occurrences de temps et premiers temps. La longueur totale de l'ensemble de données est 1h43.

L'avantage de ce nouvel ensemble de données est qu'il comprend de véritables morceaux de musique tels que l'on pourrait entendre à la radio ou sur des plateformes de streaming musical. Les sonorités de batterie sont diversifiées, les motifs de batterie sont arrangés pour des morceaux de longues durées et la plupart des pistes contiennent de la voix chantée, ce qui constitue une difficulté en plus pour des modèles entraînés sur de la musique sans chant.

5.4 Billboard 800

Cet ensemble de données a été constitué durant ce stage en se basant sur la description des travaux de Wu et al. [WL17]. Cet ensemble contient 800 morceaux audio de 30 secondes chacun. Ceux-ci ne sont pas annotés en vue de leur utilisation suivant le paradigme *élève-professeur*.

Les 800 morceaux audio constituant cet ensemble de données proviennent de listes de la plateforme *Billboard Charts* dans quatre genres différents : hip-hop, pop, rock et latino. Ces genres sont choisis pour leurs caractéristiques rythmiques très prononcées. Les 800 morceaux ont été téléchargés depuis YouTube [1], à l'aide de la bibliothèque Python pafy [2], puis ont été convertis au format mp3 (fréquence d'échantillonnage de 44.1 kHz) à l'aide de ffmpeg [3]. Pour diminuer la taille de la base de données et diminuer les temps de calculs, ainsi qu'éviter les introductions sans jeu musical, seules les 30 secondes à partir de la 30ème seconde sont conservées pour l'entraînement. Cet ensemble de données contient donc 400 minutes de flux audio.

Le tableau 5.1 résume les ensembles de données utilisés pour nos travaux.

5.5 Mesurer la performance des modèles

Pour évaluer la performance de nos modèles, nous comparons les prédictions des modèles avec les annotations des ensembles de données en utilisant la F-mesure. Les modèles nous

[1] https://www.youtube.com/, dernière visite: 11/07/2018
[2] https://pypi.org/project/pafy/, dernière visite: 11/07/2018
[3] https://www.ffmpeg.org/, dernière visite: 11/07/2018

SMT-Drums	*ENST-drums*		*RBMA 13*	*Billboard 800*
	solo	**mix**		
20 boucles (enregistrement) 14 boucles (synthèse) 70 boucles (échantillons) dont 60 avec les pistes séparées	63 pistes de batterie enregistrée 3 batteurs différents	63 pistes batterie + accompagnement 3 batteurs différents	27 titres musicaux	800 extraits de 30 secondes 4 genres : pop, rock, hip-hop, latino
130 minutes	**120 minutes**		**103 minutes**	**400 minutes**
annotations: BD, SD, HH	annotations: BD, SD, HH, temps, premiers temps		annotations: BD, SD, HH, temps, premiers temps	pas d'annotation

TABLE 5.1: Résumé du contenu des quatre ensembles de données utilisés.

	Vérité terrain	**Prédiction**
Vrai positif (TP)	Vrai	Vrai
Faux positif (FP)	Faux	Vrai
Vrai négatif (TN)	Faux	Faux
Faux négatif (FN)	Vrai	Faux

TABLE 5.2: Description de la classification entre vrai et faux positifs et négatifs.

prédisent une probabilité entre 0 et 1 de présence pour chaque trame de 10ms, par conséquent nous utilisons un algorithme de détection de pics pour extraire les trames de forte probabilité de présence de batterie. C'est un algorithme de détection de pics classique, qui détecte un pic au-dessus d'un seuil de 0.2 (établi de manière empirique), en prenant soin de ne pas extraire des pics trop proches dans le temps. Nous avons établi ce dernier paramètre à 5 trames minimum entre chaque pic, soit 50 ms : considérant un morceau à 180 battements par minute en 4/4 (ce qui est déjà plutôt rapide), les doubles croches sont jouées toutes les 80ms environ, et les HH sont rarement jouées à une cadence plus rapide que des doubles croches. Avec cet algorithme de détection de pics, nous obtenons donc in fine une liste de valeurs en seconde, correspondant aux instants d'occurrence des éléments de batterie. Cette liste est comparée à la liste des annotations fournies, et l'on considère que la prédiction est bonne si l'écart entre la prédiction et la vérité terrain est plus petit que 20ms (d'autres travaux utilisent 50 ms).

La F-mesure se calcule ensuite en mesurant le nombre de vrai et faux positifs ainsi que les vrai et faux négatifs dont le tableau 5.2 en fait la description. Ici, une prédiction est un vrai positif s'il existe une annotation à moins de 20ms d'écart, sinon c'est un faux positif. De même, s'il existe une annotation sans prédiction à moins de 20 ms d'écart, c'est un faux négatif, sinon c'est un vrai négatif.

Ensuite, nous mesurons la précision de la prédiction comme étant le nombre de prédictions du modèle qui s'avère être de justes sur le nombre total de prédictions : $precision = \frac{TP}{TP+FP}$. Le rappel se mesure comme le nombre de prédictions du modèle sur le nombre total d'annotations qu'il fallait trouver : $rappel = \frac{TP}{TP+FN}$. La F-mesure est finalement la moyenne géométrique de la précision et du rappel :

$$Fmesure = 2\frac{precision \times rappel}{precision + rappel} = \frac{2TP}{2TP + FP + FN}$$

En résumé, une F-mesure égale à 0 signifie qu'aucune des prédictions n'est bonne, alors qu'une F-mesure égale à 1 indique que toutes les prédictions sont justes et exhaustives.

Chapter 6

Les réseaux de neurones

Dans ce chapitre, nous présentons les principes théoriques des réseaux de neurones. Dans un premier temps nous montrons les limitations auxquelles ils répondent, puis nous détaillons leur fonctionnement. Dans une troisième et quatrième partie, nous présentons les réseaux de neurones convolutifs puis les réseaux de neurones récurrents.

6.1 L'insuffisance de la régression linéaire

Dans cette partie, nous mettons en lumière le besoin auquel répond les réseaux de neurones en présentant le problème de la régression linéaire.

6.1.1 La régression linéaire

Définition

Une régression linéaire cherche à établir une relation linéaire entre une famille de variables $x_1, ..., x_n$ et une variable y :

$$y = b + \theta_1 x_1 + ... + \theta_n x_n = \sum_{i=0}^{n} \theta_i x_i$$

où $\theta_1, ..., \theta_n$ sont les paramètres du modèle et b est le biais. Le biais b est souvent inclus dans les paramètres du modèle en tant que θ_0.

La figure 6.1 montre une vision en diagramme de la relation entre les données $x_1, ..., x_n$, les paramètres $\theta_1, ..., \theta_n$ et la valeur de sortie y. Le $+1$ correspond à l'ajout du biais $b = \theta_0$ dans la somme.

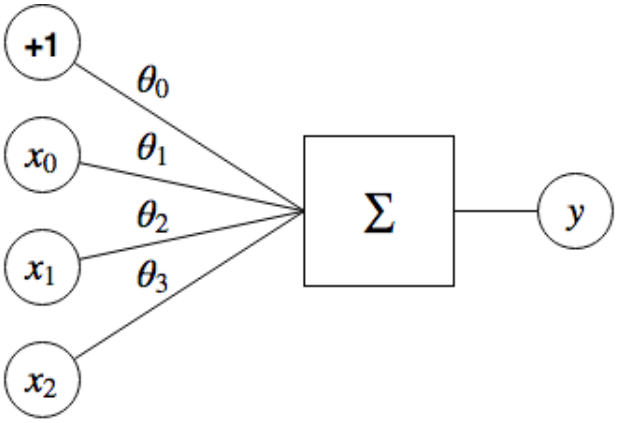


FIGURE 6.1: Représentation en diagramme de la relation linéaire entre y et $x_1, ..., x_n$.

Apprentissage

Le but de l'algorithme de régression linéaire est de trouver les valeurs des paramètres $\theta_0, ..., \theta_n$ qui correspondent le mieux aux données du problème, c'est-à-dire de sorte que $\hat{y} = \sum_{i=0}^{n} \theta_i x_i$ soit le plus proche possible de y pour tout jeu de données. La figure 6.2 illustre un cas de régression linéaire pour $n = 1$. Une fois le modèle entraîné, les valeurs de θ_0 et θ_1 sont ajustés de sorte que la ligne rouge soit le plus en adéquation avec les données.
L'algorithme d'apprentissage est le suivant:

- initialisation des paramètres $\theta_0, ..., \theta_n$;
- calcul des valeurs de $\hat{y}$ pour tout jeu de données ;
- calcul de la fonction de coût (*loss*) du jeu de paramètre $J(\theta)$. La formule de ce coût varie en fonction du problème, et représente l'écart entre le modèle en cours et la réalité ;
- calcul des gradients de $J(\theta)$ en fonction des paramètres $\theta_0, ..., \theta_n$;
- actualisation de la valeur des paramètres en utilisant la descente gradients ;
- itération des étapes précédentes jusqu'à minimiser le coût $J(\theta)$.

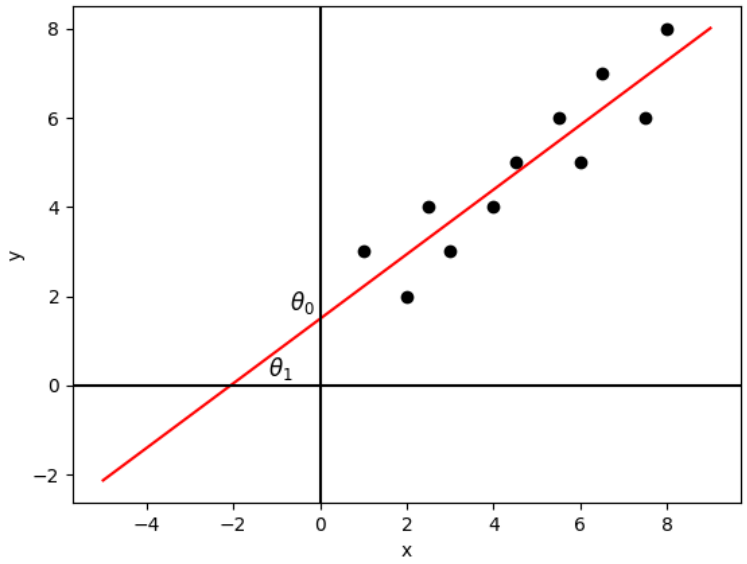


FIGURE 6.2: Visualisation d'une régression linéaire pour $n = 1$: en noir, les données d'apprentissage, en rouge la droite apprise par l'algorithme.

Grâce à cette algorithme, les paramètres s'adaptent au mieux pour correspondre aux données, mais on reste dans un cas linéaire.

On peut généraliser la régression linéaire pour prédire plusieurs variables $y_0, ..., y_p$ avec un jeu de données $x_0, ..., x_n$, comme le montre la figure 6.3.

6.1.2 Résolution limitée aux problèmes linéaires

La régression linéaire est un modèle qui fonctionne bien pour des données liées de façon à peu près linéaire, mais qui échoue lorsqu'on rentre dans le domaine du non-linéaire. Même si l'on ajoute des couches cachées à notre modèle, comme l'illustre la figure 6.4, cela revient

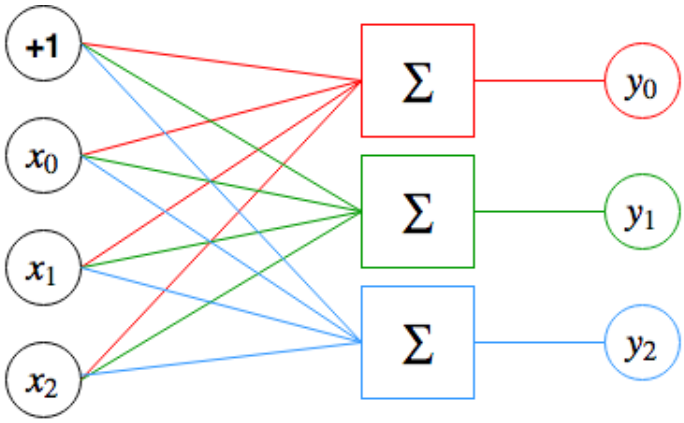


FIGURE 6.3: Représentation d'une régression linéaire prédisant plusieurs sorties.

finalement à un modèle linéaire sans couche cachée, simplement en factorisant les sommes et produits. Pour lever cette limitation, nous devons ajouter un élément non-linéaire à notre modèle, ce qu'on appelle des fonctions d'activation d'un neurone. Nous verrons alors qu'ajouter des couches cachées rend cette fois le modèle plus complexe.

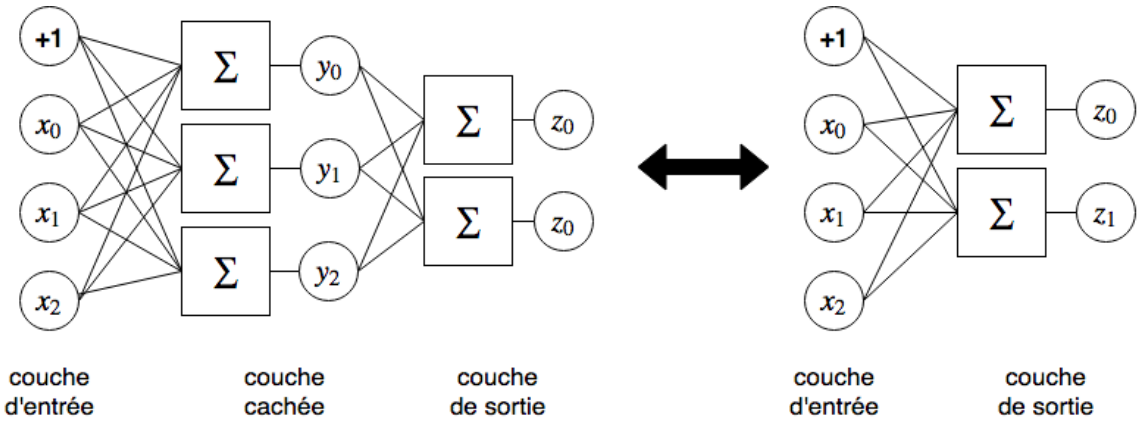


FIGURE 6.4: Un modèle à plusieurs couches est équivalent à un modèle avec une couche d'entrée et une couche de sortie, seuls les paramètres changent.

6.2 Vers le non-linéaire avec les réseaux de neurones

Dans cette partie, nous introduisons le principe des réseaux de neurone et comment ils apprennent à reproduire des fonctions non-linéaires.

6.2.1 Le perceptron multicouche

Qu'est-ce qu'un neurone artificiel ?

Un neurone artificiel est le bloc de base pour construire des réseaux de neurones. Il possède plusieurs entrées et une sortie. Pour calculer la sortie, un neurone effectue d'abord une combinaison linéaire de ses entrées, comme pour la régression linéaire :

$$y = \sum_{i=0}^{n} \theta_i x_i$$

A cela on ajoute une composante non-linéaire, appelée la fonction d'activation, que l'on notera σ. La figure 6.6 montre la courbe d'une fonction d'activation très utilisée, la sigmoïde, d'équation $\sigma(x) = \frac{1}{1+exp^{-x}}$. Nous présentons d'autres fonctions d'activation et leurs caractéristiques dans la partie 6.2.1.

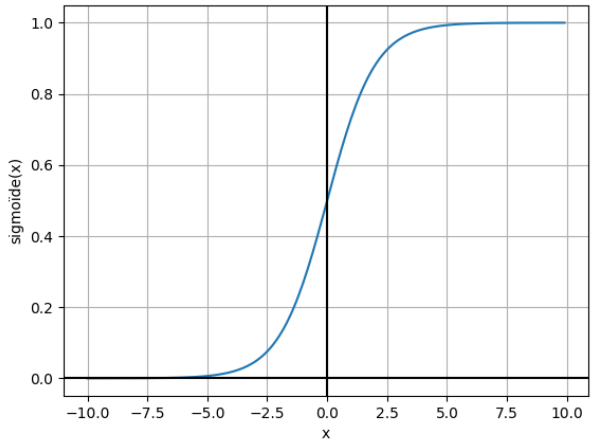


FIGURE 6.5: Représentation graphique de la fonction sigmoïde : $\sigma(x) = \frac{1}{1+exp^{-x}}$

Ainsi un neurone calcule une valeur de sortie en fonction des valeurs qu'il reçoit en entrée :

$$y = \sigma(\sum_{i=0}^{n} \theta_i x_i)$$

La figure 6.5 montre l'ajout de la fonction d'activation non-linéaire σ à l'architecture précédente.

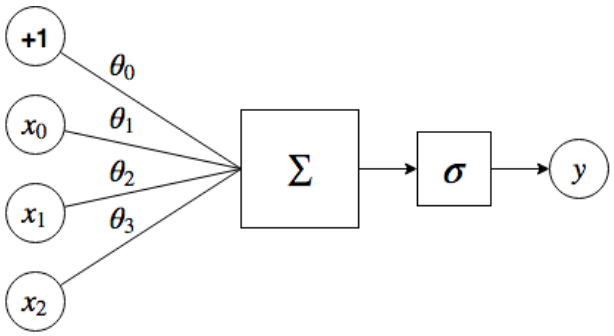


FIGURE 6.6: Représentation de l'architecture d'un neurone artificiel.

A l'instar de la régression linéaire, les paramètres $\theta_0, ..., \theta_n$ seront déterminés par un algorithme d'apprentissage. Cette fois-ci, l'utilisation de la fonction d'activation non-linéaire σ permet à un réseau de neurones de modéliser des fonctions non-linéaires, comme l'ont montré McCulloch et Pitts dans [MP43]. La dénomination d'*activation* provient de l'étude des vrais neurones cérébraux en biologie. Dans le cas de réseaux de neurones, cela se retrouve lorsque le calcul de sortie d'un certain neurone donne une valeur significative, on dit qu'il est *activé* : sa sortie aura un poids relativement important à l'entrée des neurones suivants.

Les couches cachées

Un réseau de neurones possède au moins une couche d'entrée, composée d'un certain nombre de neurones, et une couche de sortie, composée aussi d'un certain nombre de neurones. Dans ce cas-là, on connaît ce qui rentre dans la couche d'entrée, et ce qui sort de la sortie. A ces couches d'entrée et de sortie on peut rajouter ce que l'on appelle des couches cachées, comme l'illustre la figure 6.7. Ces couches sont appelées cachées car on ne connaît pas la nature de la sortie de ces couches. Cela constitue en quelque sorte une *boîte noire*, le réseau faisant ses propres représentations intermédiaires des données d'entrée. Seule la couche de sortie nous donne la représentation finale de ce que le réseau aura calculé.

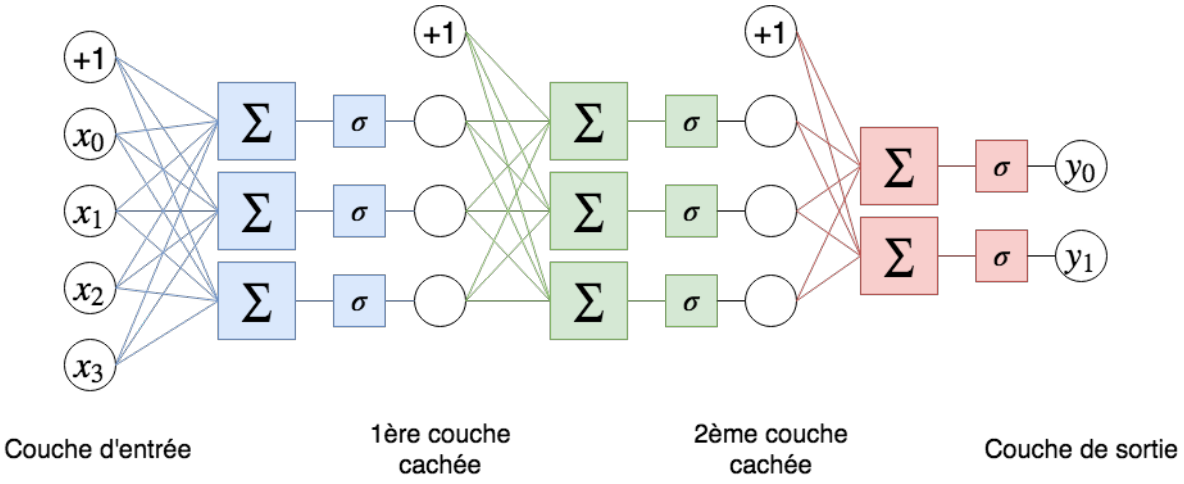


FIGURE 6.7: Ce réseau de neurones (perceptron multicouche) contient deux couches cachées.

Un tel réseau est appelé perceptron multicouche. Sa capacité à modéliser des fonctions non-linéaire est accrue grâce à l'utilisation de plusieurs couches cachées.

Dans la suite, nous omettrons les composants de somme et de fonction d'activation pour plus de clarté, comme sur la figure 6.8

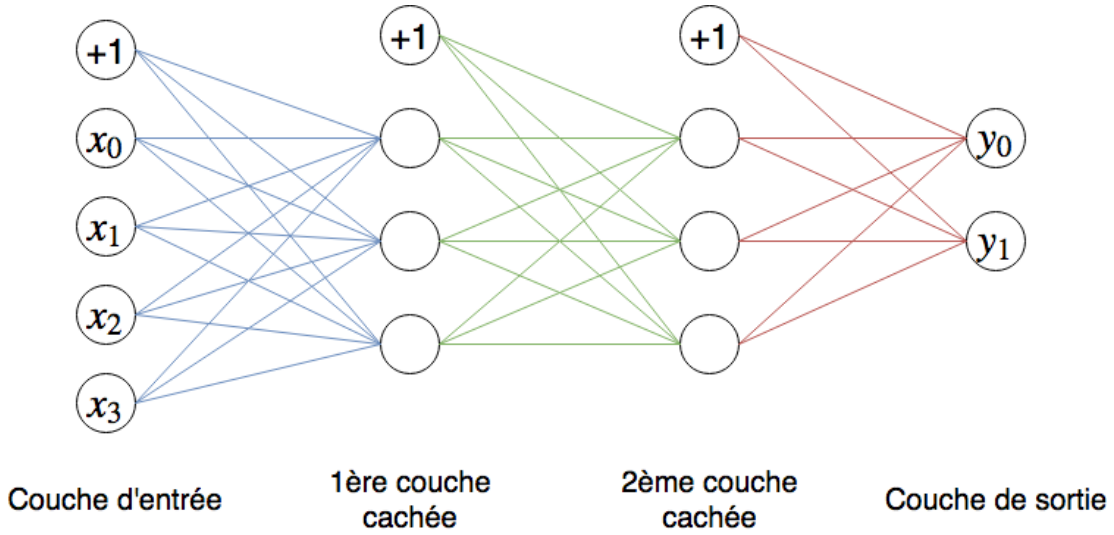


FIGURE 6.8: Représentation simplifiée d'un perceptron multicouche.

Les fonctions d'activation

Comme nous l'avons vu avec la fonction sigmoïde, les fonctions d'activation donnent la possibilité au réseau de neurones de modéliser des fonctions non-linéaires. Dans ce paragraphe, nous présentons quelques fonctions d'activation souvent utilisées dans les réseaux de neurone.

- *sigmoïde* – de fonction $\sigma(x) = \frac{1}{1+exp^{-x}}$. Cette fonction d'activation, dont la représentation graphique a été présentée sur la figure 6.6, est dérivable et comprise entre 0 et 1. C'est une fonction d'activation utile pour la couche de sortie d'un réseau où l'on veut modéliser une probabilité. Elle est utile pour des classifications binaires par exemple.
- *softmax* – la fonction softmax est quant à elle utilisée pour des classifications multi-classes, et donc appliquée en sortie de réseau. C'est une généralisation de la fonction sigmoïde. La fonction softmax représente une distribution de probabilité sur une variable discrète p, où p est le nombre de classes possibles d'un problème, et aussi généralement le nombre de neurones en sortie du réseau. Chaque neurone j prend alors comme valeur $\sigma(x)_j = \frac{e^{z_j}}{\sum_{j=1}^{p} e^{z_j}}$, de sorte que les valeurs de sortie de réseau somment à 1.
- *ReLU* – cette fonction vaut 0 pour des valeurs négatives et est l'identité pour des valeurs positives, elle est donc non-linéaire. Sa simplicité fait qu'elle est souvent utilisée dans les couches cachées d'un réseau. Elle n'est cependant jamais utilisée en sortie de réseau, n'ayant pas le *pouvoir* séparateur de la sigmoïde.

La propagation avant dans un réseau de neurones

La propagation avant consiste à injecter des données d'entrée dans la couche d'entrée et propager le calcul jusqu'à la couche de sortie qui produit les données de sortie. Mathématiquement, à chaque couche k, le calcul s'écrit :

$$sortie^k = \sigma(b^k + W^k sortie^{k-1})$$

où b^k et W^k sont respectivement le biais et la matrice de paramètres de la couche k.

Notons qu'avec cette manière de calculer, les réseaux de neurones sont déterministes : une même entrée correspond toujours à une même sortie. Cette propriété est utile pour les problèmes de classification mais devient une limite pour les problèmes de génération de contenus.

6.2.2 L'apprentissage pour un réseau de neurones

L'apprentissage d'un réseau de neurone se fait par un algorithme appelé la propagation arrière (*backpropagation* en anglais), qui se charge de modifier la valeur des paramètres du réseau de sorte, que pour chaque entrée, la sortie du réseau soit le plus fidèle aux données que l'on a déjà (la vérité terrain).

Fonctions de coûts

La performance d'un réseau, pour un jeu de paramètres donné, se calcule par ce qu'on appelle une fonction de coût, ou erreur. Cette fonction, que l'on note $\mathcal{L}$, mesure l'écart entre la prédiction du réseau $\hat{y}$ et la vérité terrain y. La forme de la fonction de coût dépend de

l'objectif du réseau de neurone. L'équation 6.1 montre une fonction de coût appelée entropie croisée binaire. Elle est utile pour calculer l'écart de prédiction pour une classification binaire. C'est celle que nous avons utilisée dans nos travaux.

$$\mathcal{L}(y, \hat{y}) = ylog(\hat{y}) + (1 - y)log(1 - \hat{y}) \tag{6.1}$$

D'autres fonctions de coût existent et sont utilisées en fonction du problème que le réseau cherche à modéliser.

La propagation arrière

La fonction de coût donne une mesure de la performance d'un réseau pour un problème donné. Le but est donc de réduire la valeur de cette fonction pour les valeurs de $\hat{y}$ données (y ne variant pas), et cela se fait par l'algorithme de descente de gradient.

Pour un échantillon $x = x_1, ..., x_n$ donné au réseau, on obtient des sorties $y = y_1, ..., y_p$, que l'on compare avec la vérité terrain $\hat{y}_1, ..., \hat{y}_p$ en mesurant le coût total:

$$\mathcal{L}(y, \hat{y}) = \sum_{i=1}^{p} \mathcal{L}(y_i, \hat{y}_i)$$

Nous avons donc une fonction de y (et a fortiori de x) qui représente l'erreur du réseau dans sa prédiction. L'idée de la descente de gradient est de considérer cette fonction comme convexe et d'utiliser les gradients afin de *descendre* vers le minimum (local) de cette fonction convexe. C'est en modifiant les paramètres de notre modèle que nous allons minimiser l'erreur. Ainsi, la descente de gradient peut se résumer ainsi :

- A l'itération t, nous avons un jeu de paramètres $w^t = (w_i^t)_i$ et une entrée x_n. Le réseau de paramètres w_i^t transforme l'entrée x_n en une sortie y_p.
- Nous calculons le différentiel de l'erreur en fonction du paramètre w_i^t : $\frac{\partial \mathcal{L}}{\partial w_i^t}$
- On met à jour le paramètre : $w_i^{t+1} = w_i^t - \eta \frac{\partial \mathcal{L}(y,\hat{y})}{\partial w_i^t}$ où η est le taux d'apprentissage (*learning rate*) : c'est la phase de **propagation arrière**.

Ces opérations sont réitérées jusqu'à convergence, ou dans la pratique jusqu'à un nombre déterminé de fois de sorte que l'erreur ne varie plus beaucoup. Le taux d'apprentissage η est un hyperparamètre important dans l'apprentissage des réseaux de neurone : il donne en quelque sorte la vitesse à laquelle le réseau va apprendre. Cependant lorsqu'on approche de la convergence de l'algorithme, un trop grand taux d'apprentissage peut empêcher l'algorithme de s'approcher du minimum local. Dans la pratique, on réduit parfois le taux d'apprentissage au cours des calculs.

Le piège sur surapprentissage

En apprentissage automatique, le surapprentissage est un phénomène qui survient lorsque le modèle est très performant sur les données d'apprentissage mais échoue sur de nouvelles données. On dit que le modèle ne généralise pas assez le problème. Le surapprentissage peut survenir lorsqu'on entraîne (c'est-à-dire lorsque que les paramètres sont ajustés longtemps et minutieusement) notre réseau trop longtemps sur un même ensemble de données. Le réseau modélise alors cet ensemble précis mais ne généralise pas à un autre ensemble de données censé ressembler au premier.

Un technique pour éviter un tel phénomène est la validation croisée. Celle-ci consiste à couper l'ensemble de données en deux : l'ensemble d'**apprentissage** et l'ensemble de **validation**. Le réseau est entraîné seulement sur l'ensemble d'apprentissage et l'ensemble de validation sert à vérifier la performance du modèle. Ainsi un bon apprentissage se fait lorsque l'erreur sur les deux ensembles diminue conjointement. Plus spécifiquement, on peut aussi diviser l'ensemble en 3 en ajoutant l'ensemble de **test**, pour chiffrer la performance finale du modèle. C'est ce que nous utilisons dans nos travaux.

6.3 Reconnaître des images avec les réseaux de neurones convolutifs (CNN)

Dans cette section, nous introduisons les concepts liés aux réseaux de neurones convolutifs (CNN) et les différentes couches qui peuvent apparaître dans ce genre de réseau de neurones.

6.3.1 Idée générale des réseaux de neurones convolutifs

Les réseaux de neurones convolutifs sont des réseaux spécialisés dans le traitement des images. Inspirés par le fonctionnement du cortex visuel, ils utilisent entre autre des neurones spécialisés qui analysent des parties d'une image en effectuant des convolutions, pendant que d'autres neurones se chargent de mettre en commun les informations. Les données d'entrée de tels réseaux sont des matrices représentant les pixels d'une image, qui peuvent alors être sous forme d'un empilement de 3 matrices (pour le rouge, le vert et le bleu).

6.3.2 Des filtres spécialisés : la couche de convolution

Les couches de convolutions au sein d'un réseau de neurones convolutifs sont à la base de la reconnaissance de motifs. Ils sont constitués d'un certain nombre de filtres d'une certaine taille. Etant donnée une matrice d'une certaine taille, un filtre de taille 3×3 (par exemple) effectue un calcul de convolution sur cette matrice : le filtre est multiplié terme à terme à chaque zone 3×3 de la matrice, donnant comme résultat une unique valeur pour chaque zone de la matrice, comme l'illustre la figure 6.9. En pratique, on utilise dans une même couche plusieurs filtres, ce qui donne en sortie une matrice de même profondeur que le nombre de filtres (voir figure 6.10). Deux hyperparamètres sont importants pour la convolution : le *stride* et le *zero-padding*. Le stride indique le pas de déplacement des filtres et donne la dimension de la sortie. Le zero-padding est l'opération de rajouter des 0 aux niveaux des bords de l'image pour permettre le filtrage de ces bordures. Les valeurs de ces filtres sont des paramètres du modèle, ils vont donc être appris pendant la phase d'entraînement. C'est pendant cet apprentissage qu'ils vont être façonnés de telle sorte à reconnaître certains motifs dans l'image, comme des lignes horizontales, verticales, diagonales, etc. La sortie d'une couche de convolution étant une nouvelle représentation de l'image (*feature maps*), on peut appliquer une nouvelle couche de convolution pour reconnaîtres des motifs un peu plus complexes, puis recommencer jusqu'à le niveau de complexité souhaité.

6.3.3 Compresser les informations : la couche de pooling

Un autre type de couche important dans les CNN est la couche de *pooling*. Celle-ci effectue une sorte de sous-échantillonnage de l'image pour réduire le nombre de paramètres et de calculs à effectuer dans la suite du réseau. Il s'agit d'une opération sur une certaine zone

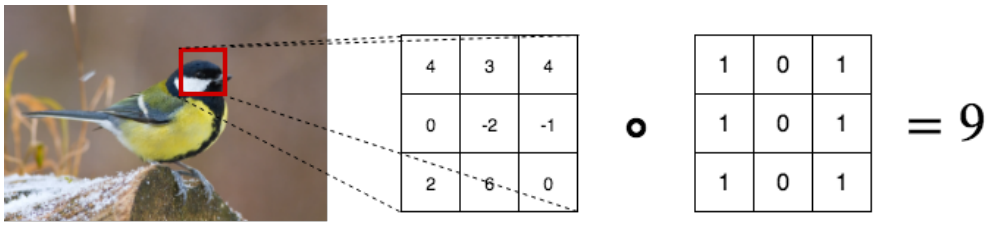


FIGURE 6.9: Convolution d'une image par un filtre 3×3.

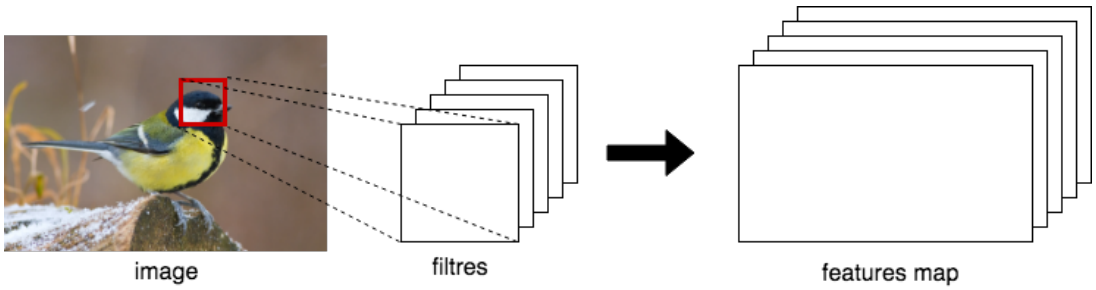


FIGURE 6.10: Exemple d'une couche de convolution avec plusieurs filtres.

de l'image qui donne en sortie une seule valeur : le maximum de la zone (on parle de *max-pooling*, la moyenne de la zone, etc. Par exemple, la figure 6.11 illustre un max-pooling de taille 2x2 avec un pas de 2.

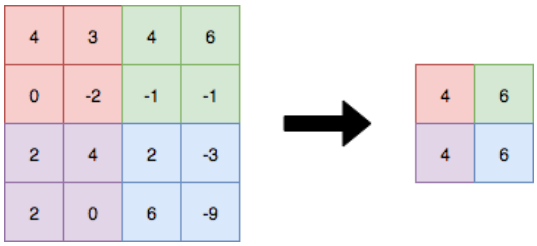


FIGURE 6.11: Opération de *max-pooling* 2×2 avec un pas de 2.

6.3.4 La couche d'activation

Comme dans les réseaux de neurones profonds basiques (DNN), les CNN ont des couches d'activation composées de fonctions non-linéaires telles que la ReLU, la tangente hyperbolique ou la sigmoïde. De même que pour les réseaux de neurones profonds, ces fonctions augmentent les propriétés non-linéaires du réseau. Dans un CNN, elles sont placées en général après les convolutions.

6.3.5 La couche entièrement connectée

Dans un CNN, après plusieurs couches de convolutions et de pooling, il est d'usage de placer des couches dites entièrements connectées (fully-connected, FC). Les neurones d'une telle couche prennent en entrée toutes les sorties de la couche précédente, d'où son nom. L'idée est de fournir au réseau un raisonnement haut niveau une fois toutes les convolutions et tous les pooling effectués. Les couches entièrements connectées sont en général aussi suivies de fonctions d'activation.

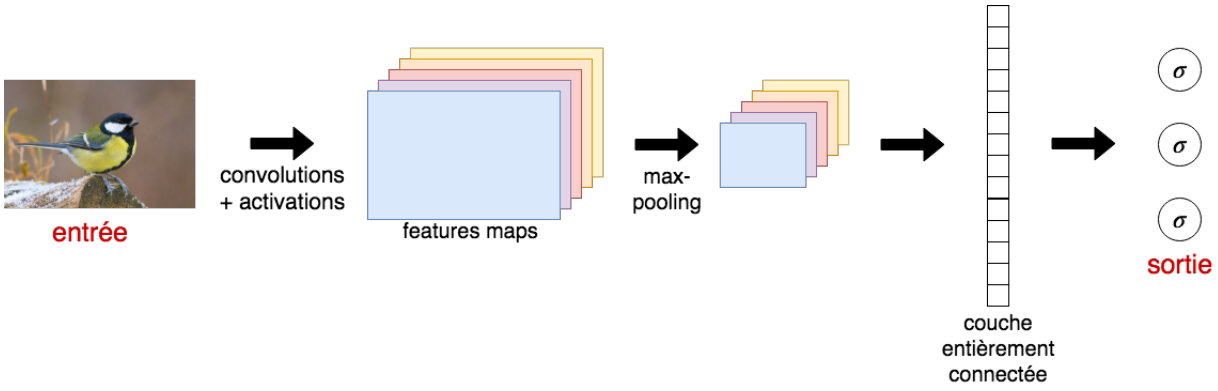


FIGURE 6.12: Exemple de CNN constitué d'une couche de convolution, d'une couche de max-pooling, d'une couche entièrement connectée et de 3 sigmoïdes en sortie.

La figure 6.12 est un exemple de CNN avec les couches décrites précédemment, avec comme couche de sortie 3 sigmoïdes (comme dans la suite). Il est bien sûr possible de créer des CNN avec beaucoup plus de couches.

6.4 Comprendre le temps avec les réseaux de neurones récurrents (RNN)

6.4.1 Réseaux de neurones récurrents

Les réseaux de neurones récurrents (RNN) sont spécialisés pour inclure des connexions récurrentes dans leurs calculs, en s'appliquant à des données séquentielles. L'idée des RNN est de considérer non pas une seule entrée et une seule sortie, mais une séquence d'entrées et une séquence de sorties qui sont indexées en fonction du temps. La sortie d'une couche va être réinsérée dans elle-même, de sorte qu'un neurone prenne en compte la donnée précédente dans le temps. Chaque couche réinsère sa propre sortie h_{t-1} sur les données au temps $t-1$ à l'entrée de cette même couche, mais pour les données suivantes au temps t (avec une matrice de pondération W_r): cela donne une nouvelle sortie h_t comme le montre l'équation 6.2.

$$h_t = \sigma(W.[x_t, h_{t-1}] + b) \tag{6.2}$$

où $[,]$ dénote la concaténation. W est la matrice de poids et b le biais.

La figure 6.13 montre ce mécanisme avec une représentation compacte (pour montrer qu'on a toujours les mêmes poids) et une représentation déroulée (pour montrer les liens temporels).

La figure 6.14 illustre le cas d'un RNN à plusieurs couches, où la représentation est plus difficile à se faire.

Réseaux de neurones récurrents bidirectionnels (BRNN)

Une caractéristique importante des réseaux de neurones récurrents est qu'ils peuvent être bidirectionnels, de sorte à ce qu'ils puissent reconnaître des motifs à la fois dans le sens du temps mais aussi dans le sens inverse. Cela peut être utile par exemple en musique, où le tempo d'un morceau peut se reconnaître dans un sens ou l'autre de la musique. Dans ce cas, l'état caché h_t n'est pas le même selon le sens du réseau. La figure 6.15 montre un réseau de neurone récurrent bidirectionnel à une seule couche.

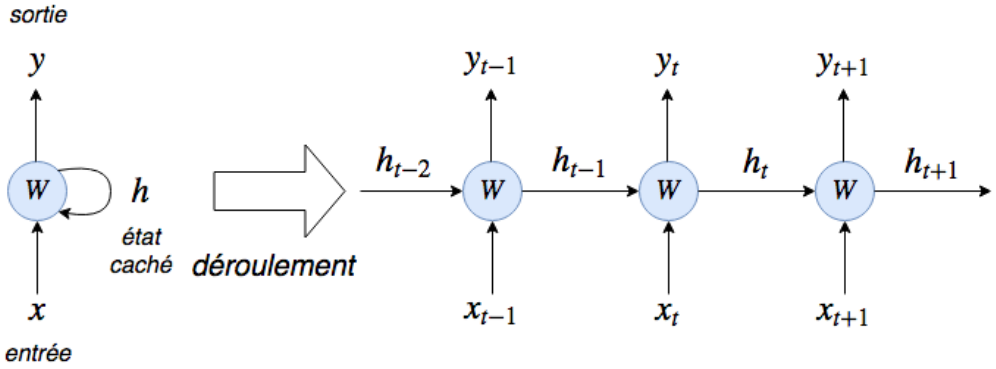


FIGURE 6.13: Illustration d'un réseau de neurones récurrents à une seule couche. A gauche une représentation compacte de l'unique couche, à droite une représentation déroulée dans le temps. Notons que pour tout t, $y_t = h_t$, mais nous distinguons les rôles ici.

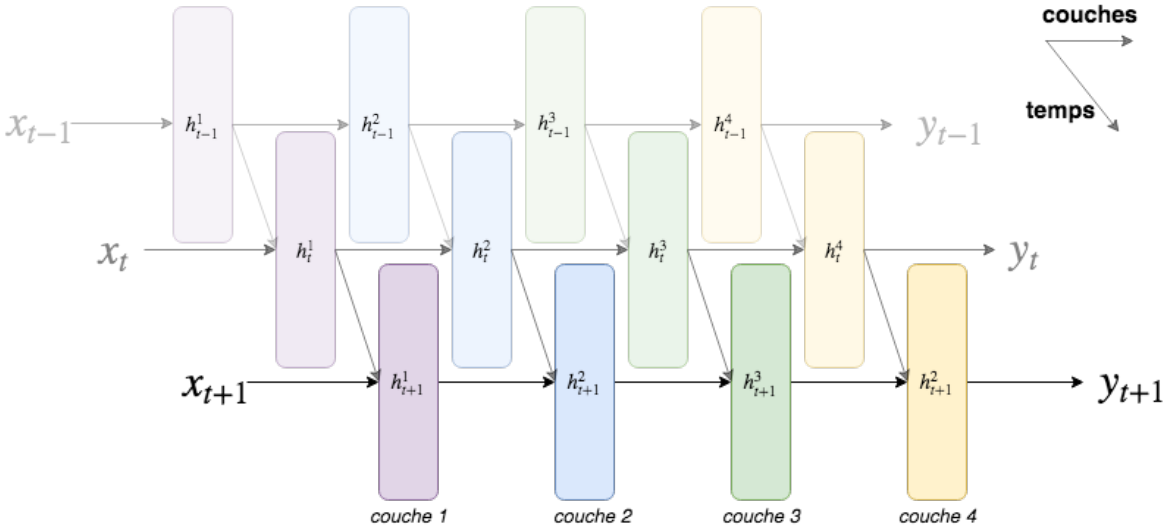


FIGURE 6.14: Schéma d'un réseau de neurones récurrent à plusieurs couches.

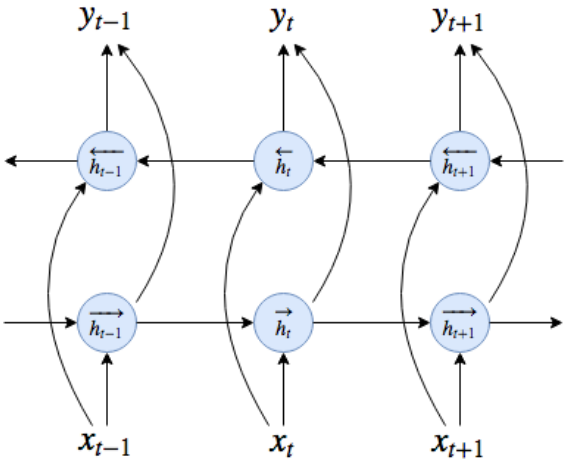


FIGURE 6.15: Représentation d'un réseau de neurones récurrent bidirectionnel.

Dans ce cas-là, les états cachées se calculent avec des matrices de poids différentes:

$$\overrightarrow{h_t} = \sigma(W_{\overrightarrow{h}}.[x_t, \overrightarrow{h_{t-1}}] + b_{\overrightarrow{h}})$$

$$\overleftarrow{h_t} = \sigma(W_{\overleftarrow{h}}.[x_t, \overleftarrow{h_{t-1}}] + b_{\overleftarrow{h}})$$

La sortie est ainsi déterminé par les contextes passés et futurs de la séquence d'entrée :

$$y_t = W_y.[\overrightarrow{h_t}, \overleftarrow{h_t}]$$

6.4.2 La propagation arrière à travers le temps (BPTT)

Pour entraîner un RNN, l'algorithme de propagation arrière est un peu modifié à cause de la présence de connexions récurrentes. Tout d'abord, le réseau étant entraîné sur des séquences, l'erreur totale est la somme des erreurs de chaque élément de la séquence:

$$E(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t)$$

Par conséquent, le gradient de l'erreur totale E dépendant aussi du temps en sommant les gradients des erreurs sur chaque pas du temps :

$$\frac{\partial E}{\partial w} = \sum_t \frac{\partial E_t}{\partial w}$$

où w est un paramètre du modèle. Si on reprend les notations de la figure 6.13, par exemple pour $t = 3$ la règle de la chaîne donne :

$$\frac{\partial E_3}{\partial w} = \frac{\partial E_3}{\partial y_3}\frac{\partial y_3}{\partial h_3}\frac{\partial h_3}{\partial w}$$

Mais comme s_3 dépend à la fois de w mais aussi de s_0, s_1 et s_2, l'erreur s'écrit alors :

$$\frac{\partial E_3}{\partial w} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial y_3}\frac{\partial y_3}{\partial h_3}\frac{\partial h_3}{\partial s_k}\frac{\partial h_3}{\partial w}$$

Ainsi la propagation arrière se fait sur les paramètres jusqu'au temps $t = 0$.

Le problème du gradient évanescent

La propagation arrière à travers le temps engendre un problème sur le gradient. Comme on peut le voir dans l'équation 6.4.2, les gradients sont multiplicatifs entre eux, et plus on avance dans le temps, plus on multiplie de termes puisque les pas de temps sont dépendants les uns des autres. Par conséquent, le gradient "loin" dans la séquence temporelle sera trop petit pour avoir une incidence sur l'apprentissage : les données temporelles "éloignées" ne servent plus pour entraîner le modèle. Le modèle présenté ci-après permet en partie de lever ce problème.

6.4.3 Les réseaux de neurones à porte (GRU)

Le problème du gradient évanescent pour les RNN basiques empêche donc l'apprentissage de trop longues dépendances temporelles. Les réseaux de neurones récurrents à porte (GRU) permet de lever cette limitation. Ceux-ci sont quelque peu similaires aux réseaux récurrents à mémoire court et long terme (*long short terme mémory*, LSTM) que nous ne présentons pas ici.

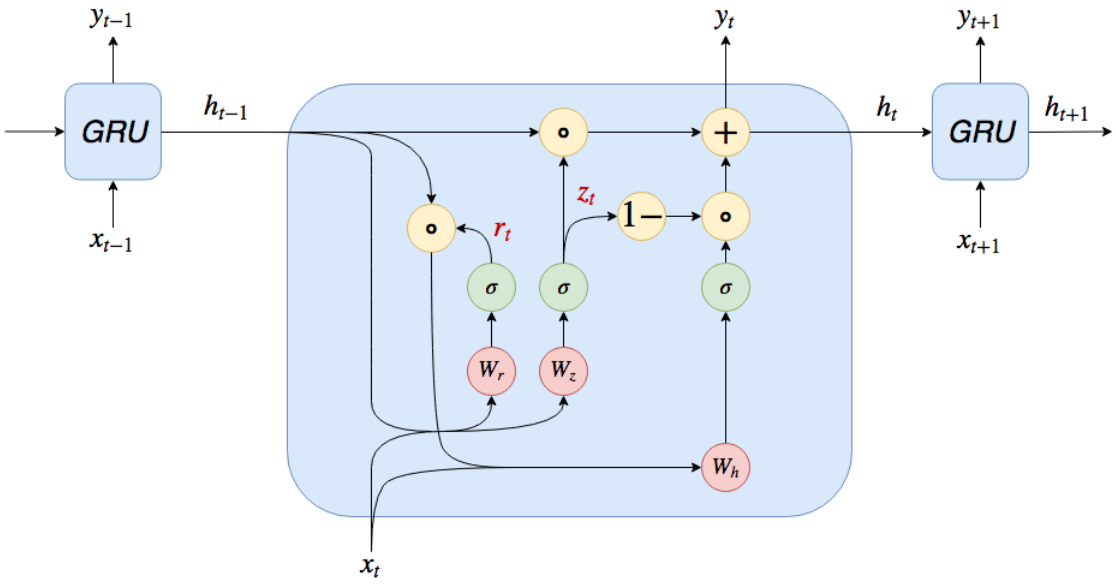


FIGURE 6.16: Schéma du mécanisme interne d'un réseau récurrent à porte.

La figure 6.16 présente le mécanisme interne au GRU. On observe que l'état caché h_{t-1} n'est pas utilisé tel quel dans le calcul de la sortie du réseau, car deux autres grandeurs sont calculés dans le mécanisme. Les calculs sont les suivants :

- porte de mise à jour : $z_t = \sigma(W_z.[x_t, h_{t-1}] + b_z)$
- porte de réinitialisation : $r_t = \sigma(W_r.[x_t, h_{t-1}] + b_r)$
- sortie : $h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \sigma(W_h.[x_t, (r_t \circ h_{t-1})] + b_h)$

où $\circ$ est le produit terme à terme.

La porte de réinitialisation détermine la manière de combiner l'entrée avec ce qui a été gardé en mémoire, et la porte de mise à jour définie quelle proportion de la mémoire précédente i faut garder pour le prochain pas de temps. Si la porte de réinitialisation est constitué de 1 partout et celle de mise à jour avec des 0 partout, on obtient un RNN basique. Cette façon de faire permet ainsi de garder des dépendances à plus long terme et contourner d'une certaine manière le problème du gradient évanescent.

Avec cette architecture, il est aussi possible de faire des réseaux récurrents bidirectionnels, de rajouter des couches, etc.

Chapter 7

Les réseaux de neurones pour la transcription automatique de batterie

Dans cette partie, nous introduisons les différents réseaux de neurones proposés par Vogl et al. dans [Vog+17], nous présentons les résultats obtenus et essayons d'en faire une analyse.

7.1 Types de réseaux de neurones

Dans [Vog+17], Vogl et al. ont proposé 3 types de réseaux pour répondre au problème de TAB : un réseau bidirectionnel constitué d'unités récurrentes à porte (GRU), un réseau convolutif et un réseau mélangeant les deux précédents. Une partie non négligeable du stage a occupé notre temps à implémenter ces réseaux, pas toujours parfaitement détaillés dans les publications, et à obtenir des résultats semblables à ceux dans [Vog+17].

7.1.1 Données d'apprentissage

Données d'entrée

Les signaux sonores qui constituent les ensembles de données sont sous forme de fichiers .wav ou .mp3, à la fréquence d'échantillonnage de 44.1 kHz. Nous calculons un spectrogramme en amplitude avec une fenêtre de taille de 2048 échantillons, soit 46 ms, pour un résultat d'un trame toutes les 10 ms. Les éléments fréquentiels sont transformés dans une échelle logarithmique à raison de 12 filtres triangulaires par octave, s'étalant de 20Hz à 20000 Hz (soit 84 éléments fréquentiels). Une fois ce spectrogramme obtenu, nous calculons la différence temporelle de premier ordre : il s'agit de la différence entre une trame et la précédente ; les valeurs négatives étant remplacées par 0. Cela permet de se concentrer sur les transitoires apparaissant dans le spectrogramme. Ce différentiel de spectrogramme est concaténé à ce dernier (concaténation selon l'axe fréquentiel). Finalement, le spectrogramme obtenu est constitué de 168 éléments en fréquence et les trames font 10 ms de longueur. La figure 7.1 donne un exemple de spectrogramme (sans le différentiel).

Données cibles d'apprentissage

Les couches de sortie de chaque réseau de neurone sont constituées de sigmoïdes, qui calculent une valeur entre 0 et 1. Cette valeur représente la probabilité de présence d'un élément de batterie pour une trame donnée (BD, SD ou HH) ou bien les temps et premiers temps. Nous transformons ces données présentes dans les ensembles de données en fonction *cible* pour les réseaux de neurone : pour chaque trame de 10 ms, nous avons des cibles dites *dures* pour BD, SD, HH, temps et premier temps. La cible vaut 0 s'il n'existe pas d'annotation et 1 s'il en existe une. La figure 7.2 montre les 3 fonctions cibles pour BD, SD et HH alignées avec le spectrogramme correspondant.

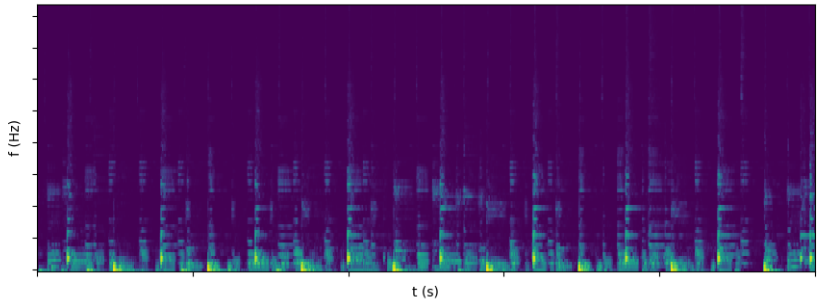


FIGURE 7.1: Spectrogramme d'un signal sonore ne contenant que de la batterie.

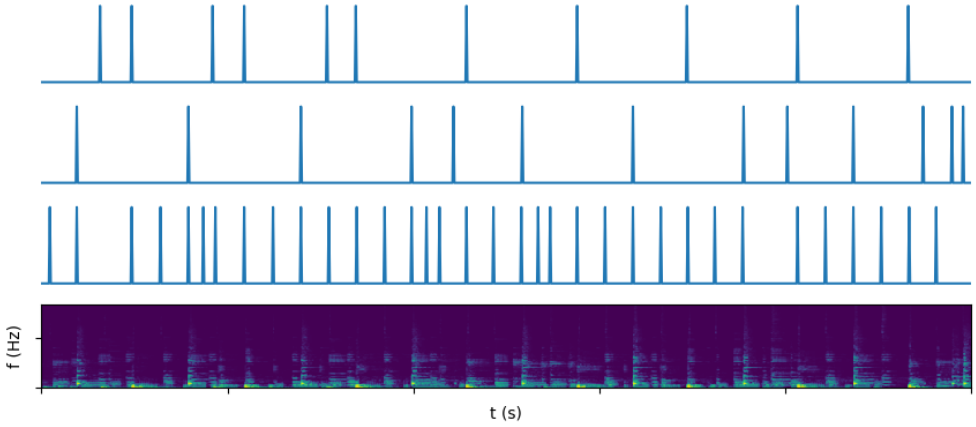


FIGURE 7.2: Visualisation des fonctions cibles donnant les annotations (0 ou 1) pour chaque trame du spectrogramme.

7.1.2 Réseaux de neurones bidirectionnels récurrents à porte

Le premier réseau proposé par Vogl et al. est un réseau de neurones bidirectionnel constitué de GRU (BGRU). Le choix des GRU est de donner au réseau une capacité de mémoire par rapport aux RNN classiques. L'auteur explique son choix des GRU par rapport aux LSTM par le fait qu'il est plus simple d'optimiser les hyperparamètres. L'idée générale d'utiliser des réseaux de neurones récurrents est d'espérer que le réseau puisse *comprendre* une certaine structure rythmique dans le signal. Par exemple, à 120 battements par minutes (bpm) sur un jeu en 4/4, il y a souvent (en electro) un coup de grosse caisse toutes les demi-secondes. On espère que les GRU aient cette capacité de capter de tels événements réguliers.

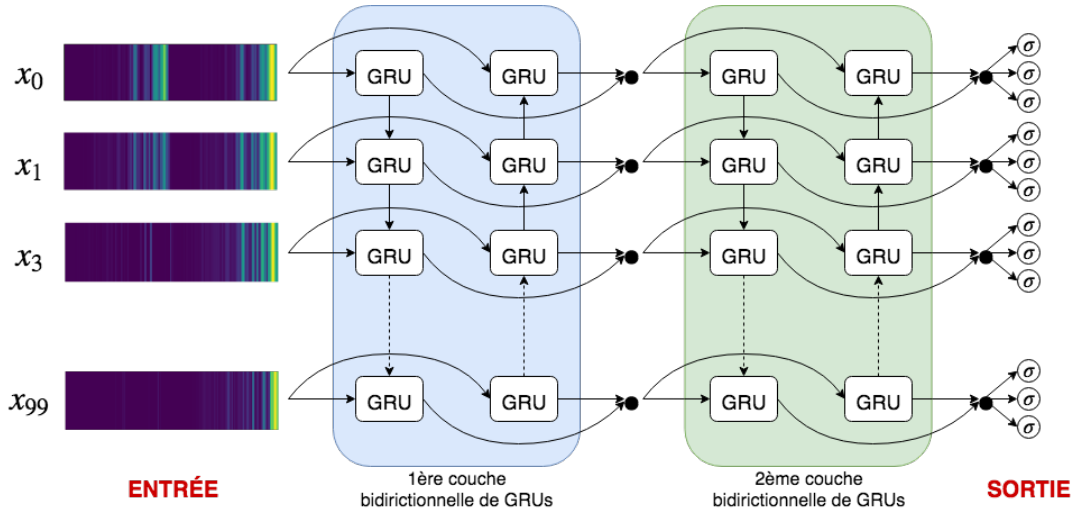
Deux variantes de ce BGRU est proposé par Vogl et al., un réseau à *petite* capacité, et un réseau à *grande* capacité :

- BGRUa (petite capacité): 2 couches de GRU avec une sortie de dimension 50, entraîné sur des séquences de 100 trames, soit 1 seconde dans notre cas (2 coups de grosse caisse en 4/4 à 120 bpm) ;
- BGRUb (grande capacité): 3 couches de GRU avec une sortie de dimension 30, entraîné sur des séquences de 400 trames, soit 4 secondes (8 coups de grosse caisse en 4/4 à 120 bpm).

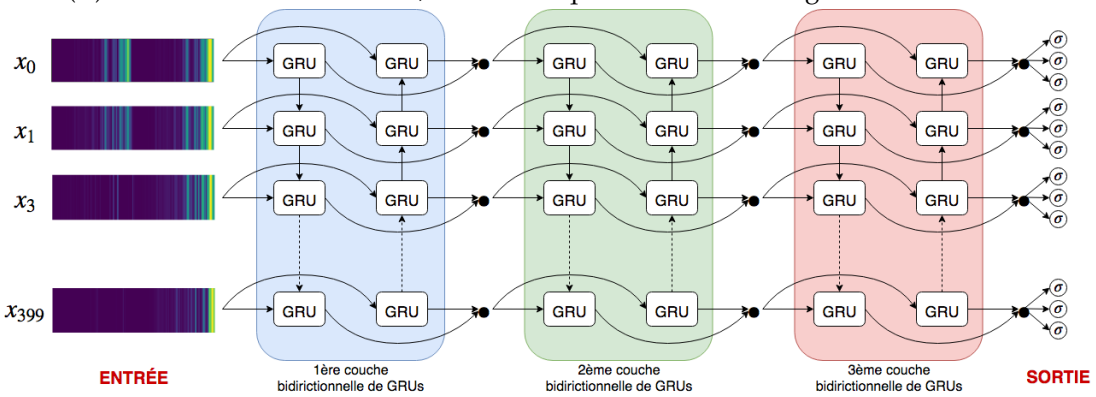
Pour chaque variante, la couche de sortie est composée de 3 sigmoïdes, une pour chaque élément de batterie (BD, SD, HH). Nous utilisons comme fonction de coût l'entropie croisée binaire, et un taux d'apprentissage de 0.007.

Les données d'entrée sont des séquences de 100 (BGRUa) ou 400 (BGRUb) trames issues du spectrogramme, soit des vecteurs de dimensions 168*1.

Les figures 7.3a et 7.3b illustrent ces modèles. Les hyperparamètres (nombre des couches, dimension de la sortie, taille des séquences pour l'entraînement) n'ont pas fait l'objet d'optimisation et de test par manque de temps. L'idée a d'abord été de reprendre les modèles proposés par Vogl et al. pour retrouver les résultats. Cependant nous pensons qu'il serait intéressant d'explorer les hyperparamètres dimension de sortie et nombre de couches car ils ne sont pas triviaux à interpréter. Concernant la taille des séquences pour l'entraînement, nous pouvons penser qu'une plus grande taille permet au réseau de mieux comprendre les dépendances rythmiques entre les éléments percussifs. Nous verrons avec les résultats comment cela se comporte.



(A) Schéma du réseau BGRUa, avec des séquences d'entrée longues de 100 trames.



(B) Schéma du réseau BGRUb, avec des séquences d'entrée longues de 400 trames.

FIGURE 7.3: Schémas des BGRUa et BGRUb.

7.1.3 Réseaux de neurones convolutifs

Le deuxième réseau proposé par Vogl et al. est un réseau de neurones convolutifs (CNN). L'idée pour ce réseau est de pouvoir modéliser les composantes acoustiques des éléments percussifs (BD, SD, HH) en analysant les spectrogrammes en tant qu'image. Par exemple, un

BD contient a priori plus de contenu dans les basses fréquences, alors qu'un HH s'étale sur les mediums et les hautes fréquences. Nous souhaitons que le réseau apprenne à reconnaître ces modèles acoustiques.

De même que pour les BGRU, 2 variantes sont proposées pour ces CNN, ce qui va changer la taille des images en entrée :

- CNNa (petite capacité): les images d'entrée sont de taille 168*9, c'est-à-dire que l'on donne au réseau le contenu de 9 trames, soit 90 ms. La fonction cible ne donne cependant que l'annotation de la trame du milieu. Les 8 autres trames (4 de part et d'autre) sert à donner un contexte acoustique au réseau pour l'apprentissage, de sorte à se concentrer davantage sur l'attaque des éléments percussifs.
- CNNb (grande capacité): les images d'entrée sont de taille 168*25, soit 250 ms de contexte.

Les réseaux CNNa et CNNb sont exactement les mêmes est (à part la taille des images en entrée) et sont constitués de 2 principaux blocs :

- bloc A : 2 couches de convolution de 32 filtres 3x3 avec *padding*, suivies d'une couche de *batch normalization*, d'une couche de *max-pooling* 3x3 et d'une couche de *dropout* (probabilité de 0.3) ;
- bloc B : mêmes éléments que le bloc A, sauf que les couches de convolution comportent 64 filtres 3X3.

Ces blocs sont mis bout à bout et sont suivis d'une couche de vectorisation (pour *applatir* les données) et de deux couches entièrement connectées constituées de 256 noeuds. La couche de sortie est, comme dans les BGRU, constituée de 3 sigmoïdes. La figure 7.4 illustre ce réseau.

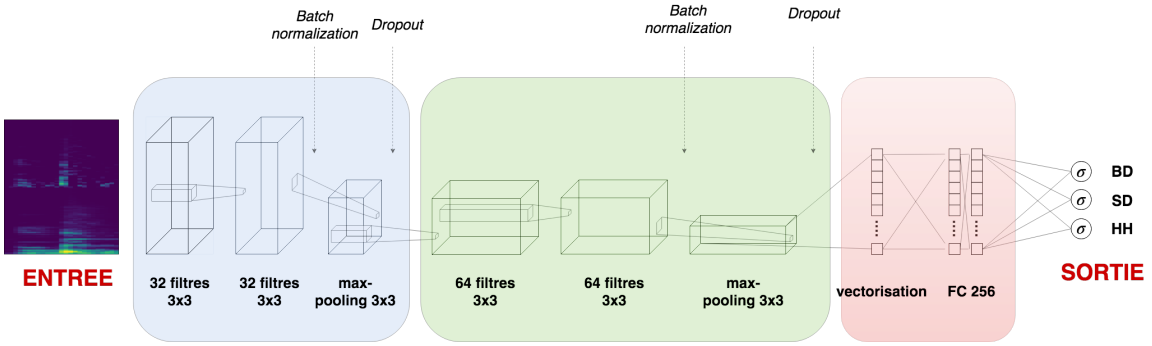


FIGURE 7.4: Schéma du CNNb utilisé pour la transcription automatique de batterie avec une dimension d'entrée de 250ms.

La fonction de coût est l'entropie croisée binaire et le taux d'apprentissage utilisé est de 0.001.

De même que pour les BGRU, les hyperparamètres ont été repris des travaux de Vogl et al. Le point important à analyser est selon nous l'utilisation d'un contexte de 90 ou 250 ms.

7.1.4 Réseaux de neurones convolutifs bidirectionnels récurrents à porte

Le troisième et dernier type de réseau proposé par Vogl et al. est une combinaison des deux réseaux précédents. On fait passer dans un premier temps les données d'entrée dans le CNN puis on fournit les données de sortie de ce dernier en entrée du BGRU. Encore une fois, deux variantes sont proposées :

- CBGRUa (petite capacité): 2 couches de GRU de sortie de dimension 30, avec un contexte de 9 trames pour les convolutions, et entraîné sur des séquences de 100 trames ;
- CBGRUb (grande capacité): 3 couches de GRU de sortie de dimension 60, avec un contexte de 13 trames pour les convolutions, et entraîné sur des séquences de 400 trames.

Les données d'entrée sont donc sous formes de séquences (de 100 ou 400) images (de dimension 168*9 ou 168*13). La figure 7.5 illustre pour l'exemple le CBGRUb. Nous utilisons comme fonction de coût l'entropie croisée binaire et un taux d'apprentissage de 0.0005.

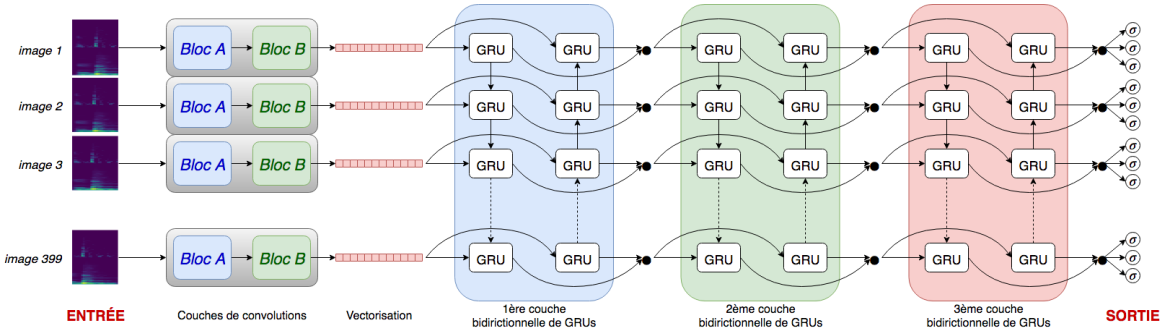


FIGURE 7.5: Schéma du CBGRUb utilisé pour la transcription automatique de batterie.

Encore une fois, les hyperparamètres choisis sont repris des travaux de Vogl et al. Ils sont cependant un peu troublants : pour le CBGRUa, il propose 2 couches de GRU avec une dimension de sortie de 30 alors que pour le BGRUa il avait proposé 2 couches de GRU de dimension de sortie 50 (mais une dimension de sortie de 30 pour le GRUb). De même, il propose cette fois-ci un contexte de 13 trames pour le CBGRU à grande capacité quand il avait proposé un contexte de 25 trames pour le CNNb. Quoiqu'il en soit, l'objectif du stage n'a pas été d'optimiser les hyperparamètres des modèles proposés (ce qui aurait pris du temps supplémentaire) mais bien de retrouver les résultats de Vogl et al. pour les combiner avec les idées de Wu et al.

7.1.5 Résultats

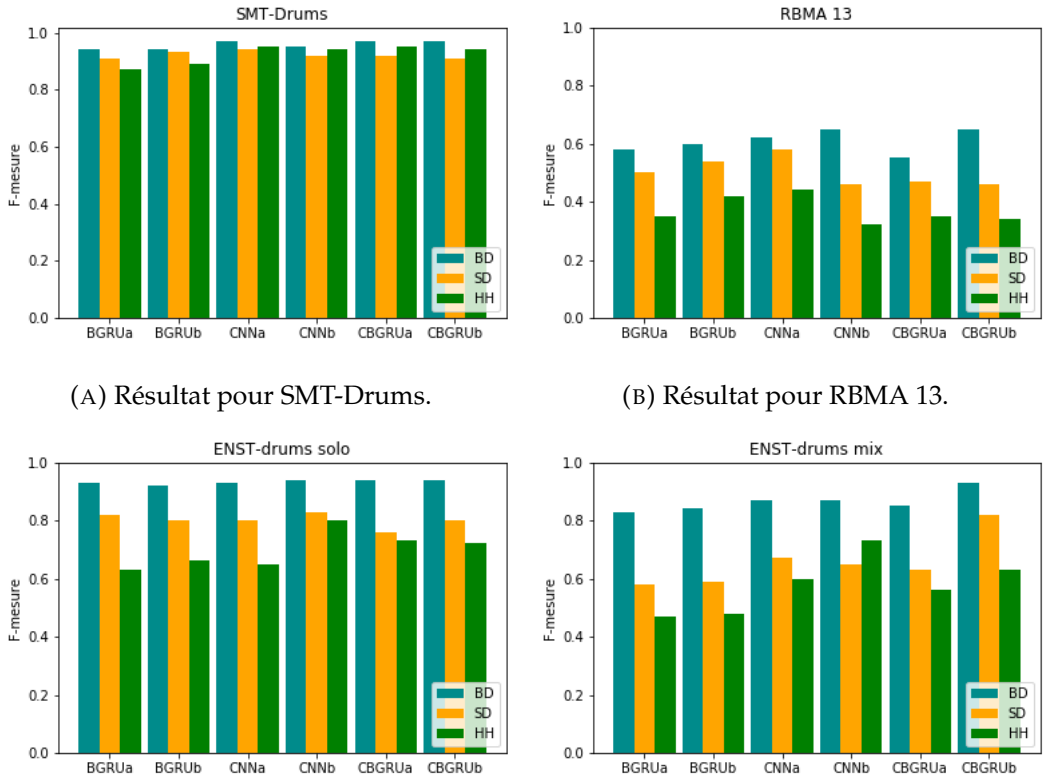
Les figures 7.6a, 7.6c, 7.6d et 7.7a montrent les F-mesures obtenues pour la transcription de batterie basique (*drum transcription*, DT) pour les 4 ensembles de données utilisés.

Dans un premier temps, on voit que BD est systématiquement mieux transcrit que SD et HH, et que SD est presque toujours mieux transcrit que HH (sauf pour les CNN et les CBGRU dans SMT-Drums et pour CNNb dans ENST-Drums). Ceci peut s'expliquer par l'étendue spectrale des éléments considérés : BD possède un spectre situé dans les basses fréquences et qui ne s'étale pas trop, tandis que HH possède un spectre très riche allant d'environ 300 à 3kHz voire plus haut. Des éléments sonores ayant un spectre étendu ressemblant tout ou partie à celui de HH, ayant pour effet d'activer certains neurones du réseau et augmenter la probabilité de présence d'un HH.

Ensuite on remarque que la performance est moindre pour RBMA 13 et ENST-Drums avec accompagnement, ce qui est attendu puisque la présence d'autres éléments sonores que la batterie seule rend plus difficile la transcription.

Concernant les différents modèles utilisés, voici ce que l'on peut analyser des résultats. On remarque une tendance à ce que les BGRU soient moins performants que les CNN et

CBGRU, notamment pour les HH (sauf pour RBMA 13). On peut penser que la transcription de batterie repose davantage sur le contenu spectral des éléments considérés que sur les récurrences temporelles. Par ailleurs, on ne remarque pas de différences flagrantes dans les performances des BGRUa et BGRUb, ce qui expliquerait que le réseau apprend aussi bien en analysant des séquences de 1 seconde ou de 4 secondes. Les CNN ont des performances différentes : le CNNa est meilleur que le CNNb pour l'ensemble SMT-Drums mais moins bon que le CNNb pour les ensembles RBMA 13 et ENST-Drums solo et mix. Les ensembles ENST-Drums et RBMA 13 sont constitués d'éléments de batterie plus longs dans le temps (moins synthétiques), donc le réseau demande une capacité de contexte temporel plus long pour apprendre à transcrire, d'où le fait que le CNNb (contexte temporel de 250 ms) soit meilleur que le CNNa (contexte de 90 ms) pour ces ensembles de données. Pour les CBGRU, il y a une légère tendance à ce que le CBRGUb soit meilleur que le CBGRUa, notamment pour les ensembles RBMA 13 et ENST-Drums, ce qui confirme la remarque précédente sur le contexte temporel (contexte de 130 ms pour CBGRUa et 250 ms pour CBGRUb).



(A) Résultat pour SMT-Drums.

(B) Résultat pour RBMA 13.

(C) Résultat pour ENST-Drums sans accompagnement.

(D) Résultat pour ENST-Drums avec accompagnements.

FIGURE 7.6: F-mesures pour BD, SD et HH pour chaque type de réseau utilisé, en fonction de l'ensemble de donnée considéré.

7.2 Transcription informée par les temps et premiers temps

Dans cette partie nous traitons de l'utilisation des informations de temps et premier temps (*beats* et *downbeats*) pour aider les réseaux pour la tâche de TAB. Ces idées sont reprises de Vogl et al.

7.2.1 Informer le réseau en entrée

Le jeu d'un batteur étant rythmique, il est d'une certaine manière corrélé avec le tempo et le battement d'un morceau. Par exemple, dans un morceau electro en 4/4, nous pouvons souvent entendre un coup de grosse caisse à chaque temps et un coup de caisse claire sur les temps 2 et 4. L'idée est alors de donner des informations supplémentaires en entrée du réseau pour l'aider, en espérant qu'il puisse ainsi faire le lien entre temps/premier temps et jeu de batterie. Pour cela, Vogl et al. ont proposé de concaténer les informations de temps/premier temps au spectrogramme, augmentant sa dimension fréquentielle de 168 à 170. Si la trame correspond à un temps et un premier temps, on concatène deux 1 à la suite du spectrogramme (pour la trame correspondante), et si c'est un temps mas pas un premier temps on concatène un 1 et un 0. Sinon on concatène deux 0. Ce type d'expérimentation a été nommé *beat features* (BF) par Vogl et al., nous reprenons cette appelation.

Cette idée de concaténer les informations de temps et premier temps nous a paru étrange dans un premier temps, avant obtention des résultats. En effet, quand il s'agit d'effectuer des convolutions sur les images des spectrogrammes pour identifier les formes acoustiques voulues, nous avons pensé que concaténer des 0 ou des 1 en bas de l'image n'aurait aucun sens, qu'ils seraient sûrement *perdus* au milieu des convolutions. Cela concernant les réseaux CNN et CBGRU, nous avons proposé de de fournir les informations de temps et premier temps **après** les convolutions, soit en les concatenant au vecteur de la couche vectorisation du CNN. Nous nommons ce procédé *beat features multi input* (BFMI).

Pour cette expérimentation nous utilisons l'ensemble de données RBMA 13 confectionné par Vogl et al. pour leur travaux [Vog+17].

7.2.2 Entraîner le réseau au multitâche

L'autre procédé proposé par Vogl et al. pour utiliser les informations de temps et premier temps dans le réseau est le multitâche (MT). Cette fois-ci, nous revenons à un spectrogramme sans concatenation (de dimension fréquentielle de 168) mais nous lui faisons apprendre à identifier les temps et premier temps en plus des trois éléments de batterie habituels BD, SD et HH. Nous modifions donc la couche de sortie en lui rajoutant 2 sigmoïdes correspondant au temps et premier temps. Nous fournissons donc pour l'apprentissage 5 fonctions cibles correspondant à BD, SD, HH, temps et premier temps. L'idée derrière l'utilisation du multitâche est que le problème de détection de batterie et celui de détection de temps et premier temps ont des caractéristiques communes, à savoir les propriétés temporelles du rythme. Nous espérons ainsi améliorer la TAB en apprenant les deux tâches en même temps.

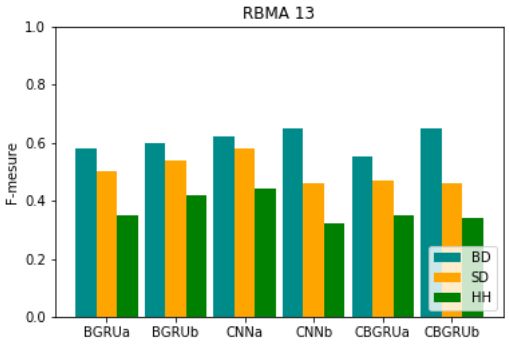
L'ensemble de données RBMA 13 est aussi utilisé pour cette tâche.

NB : L'ensemble de données ENST-drums a été annoté en temps et premier temps par nos soins lorsque nous avons expérimenté le paradigme professeur-élève, soit à une date ultérieure aux expérimentations dont traite cette partie. Nous n'avons malheureusement pas pris le temps d'entraîner de nouveaux modèles sur ENST-drums en utilisant les informations de temps et premier temps, mais il aurait été intéressant de voir si les résultats pour ENST-Drums auraient suivi les mêmes tendances que ceux pour RBMA 13.

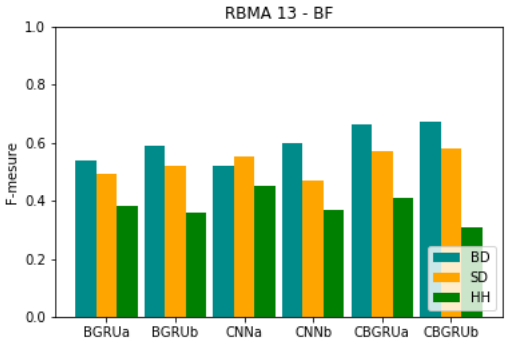
7.2.3 Résultats

Les figures 7.7a, 7.8a et 7.7c permettent de comparer les approches BF et MT avec la transcription basique DT. L'apport des temps et premiers temps en entrée (BF) dans les réseaux CNN et CBGRU améliore la performance, cependant pas pour les RNN. Ceci s'explique par le fait qu'un RNN possède déjà la capacité d'apprendre des dépendances temporelles, donc renseigner sur les temps et premiers temps n'aide apparemment pas davantage les

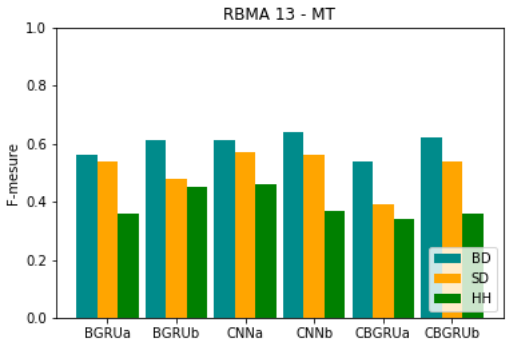
réseaux récurrents. En outre, pour le multitâche cela n'améliore pas la performance pour les réseaux CNN et CBGRU puisques de tels réseaux n'ont pas la capacité à apprendre des dépendances temporelles (ici les temps et premiers temps), alors que le multitâche améliore les performances du BGRUb (sauf pour SD). Ainsi un réseau récurrent à capacité suffisante (ici 4 secondes) peut conjointement apprendre à transcrire la batterie et extraire les temps et premiers temps, étant tous temporellement dépendants. On peut même remarquer que les résultats se détériorent pour les CNN lors de l'apport des temps et premiers temps. Les CBGRU ont cependant de meilleures performances pour BF mais pas pour MT. Nous attribuons cela au fait que les CBGRU ont des capacités acoustiques suffisantes pour transcrire les éléments de batterie mais pas pour l'apprentissage des temps et premiers temps.



(A) Résultat pour RBMA 13 en DT.



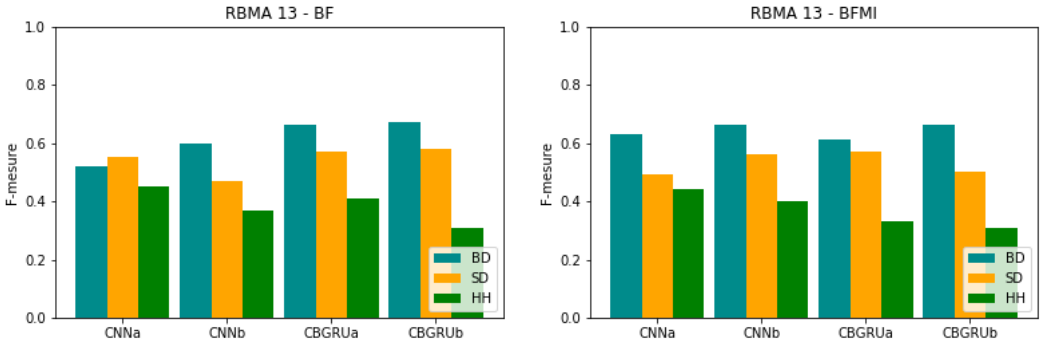
(B) Résultats pour RBMA 13 avec les temps et premiers temps en entrée du réseau.



(C) Résultat pour RBMA 13 en multi-tâche.

FIGURE 7.7: Comparaison des résultats de DT, BF et MT pour RBMA 13.

Les figures 7.8a et 7.8b permettent de comparer l'utilisation des temps et premiers temps avant ou après les convolutions pour les modèles CNN et CBGRU. On peut voir que, contrairement à ce que l'on pensait, l'apport des informations de temps et premiers temps directement en entrée du réseau (avec le spectrogramme) ne fonctionne pas moins bien que l'apport de ces informations après les couches de convolutions. On note cependant une amélioration pour le BD avec le CNNa, et pour le CNNb. On pense que l'ajout des informations de temps et premiers temps après les convolutions (donc après l'analyse acoustique du contenu spectral) permet de discriminer les éléments spectraux lorsqu'ils sont utilisés dans un contexte assez grand, ici 250 ms.



(A) Résultat pour RBMA 13 avec les temps et premiers temps en entrée du réseau.

(B) Résultats pour RBMA 13 avec les temps et premiers temps en entrée après les convolutions.

FIGURE 7.8: Comparaison des résultats de BF et BFMI pour RBMA 13.

Chapter 8

Le paradigme professeur-élève

Dans cette partie, nous présentons dans un premier temps le principe du paradigme *professeur-élève* appliqué aux réseaux de neurones. Ensuite, nous appliquons cette méthode aux différents réseaux de neurones analysés dans la partie précédente et examinons les résultats obtenus.

8.1 Explication du paradigme

Les ensembles de données régulièrement utilisés pour le problème de TAB ont des limitations : le nombre de données est relativement peu élevé, la majorité est constitué de signaux musicaux comportant de la batterie sans accompagnement, et le style de jeu percussif est parfois trop homogène au sein d'un même ensemble. Pour lever cette limitation, Wu et al. ont proposé d'appliquer le paradigme professeur-élève (*student-teacher*) à la TAB [WL17].

Ce paradigme a pour la première fois été proposé pour la compression de modèles. Le principe de ce paradigme est de transférer les connaissances d'un modèle dit *professeur* à un autre modèle *élève*, en utilisant les annotations. Il a ensuite été appliqué avec succès pour différentes tâches comme la reconnaissance vocale.

Le paradigme professeur-élève s'utilise de la manière suivante (illustré par la figure 8.1) :

- Entraîner un modèle *professeur* sur un ensemble de données A de taille moyenne parfaitement labellisé ;
- Utiliser ce modèle professeur pour annoter un ensemble de données B de grande taille non labellisé ;
- Entraîner un modèle *élève* sur l'ensemble de données B nouvellement labellisé par le modèle *professeur* ;
- Comparer les performances des modèles *professeur* et *élève* sur un ensemble de données C (pour éviter tout biais).

Différents travaux ont montré que le modèle élève obtient régulièrement de meilleures performances que le modèle professeur. On peut alors penser qu'il est préférable d'entraîner un modèle sur un grand ensemble de données aux annotations imparfaites que sur un petit ensemble de données aux annotations exactes.

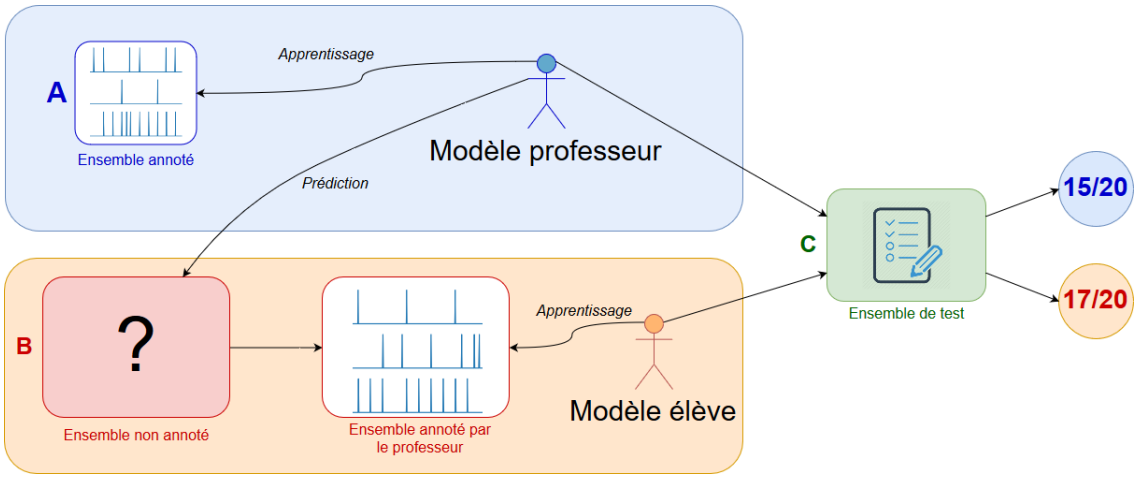


FIGURE 8.1: Schéma du paradigme professeur-élève. Le professeur, déjà entraîné sur le petit ensemble de données A, prédit les annotations du grand ensemble de données B qui sert à entraîner le modèle élève. On s'attend à ce que l'élève soit plus performant que le professeur.

8.2 Utilisation du paradigme pour la transcription automatique de batterie

Nous appliquons ce paradigme au problème de TAB. L'ensemble de données d'entraînement est l'ensemble SMT-Drums : il est de taille plutôt grande comparé à RBMA-13 et ENST-Drums, et les résultats sont plutôt bons pour l'apprentissage sur cet ensemble. Nous gardons également l'ensemble de données ENST-Drums (avec les mélanges batterie-accompagnement) pour l'évaluation de la performance des modèles. Présenté dans la partie 5.4, l'ensemble de données Billboard 800 est annoté par différents modèles professeur et est utilisé pour entraîner les modèles élèves.

8.2.1 Variation des modèles élèves et professeurs

Les 6 types de réseaux (BGRUa, BGRUb, CNNa, CNNb, CBGRUa, CBGRUb) sont utilisés en tant que modèle professeur et modèle élève. Nous avons dans un premier temps annoté 6 fois (une fois par type de réseau) l'ensemble de données Billboard 800 en BD, SD et HH. Nous souhaitons comparer les combinaisons entre différents modèles élèves et professeur et voir comment les connaissances sont transférées entre l'un et l'autre, en fonction du type de réseau. Cela donne un total de 36 modèles, en variant les types de modèles élève et professeur.

8.2.2 Utilisation des informations de temps et premiers temps

Nous avons également explorer l'utilisation des informations de temps et premiers temps avec le paradigme professeur-élève. Pour ce faire, nous avons eu besoin d'un système professeur assez performant pour prédire les temps et premiers temps. Nous avons choisi d'utiliser le bibliothèque python Madmom [1], qui implémente un des systèmes de l'état de l'art pour la détection de temps et premiers temps. Nous avons fait l'expérience seulement avec le réseau élève CBGRUb, qui présente les meilleurs résultats globaux selon nous. Nous reprenons les mêmes expériences que dans la section précédente, et comparons les résultats

[1] https://github.com/CPJKU/madmom, dernière visite 19/08/2018

du réseau CBGRUb sans information de temps et premiers temps (DT), avec ces informations en entrée (BF) et en multi-tâche (MT).

8.2.3 Résultats

Les figures 8.2a, 8.2b, 8.2c, 8.2d, 8.2e, 8.2f montrent les résultats obtenus en utilisant le paradigme élève-professeur.

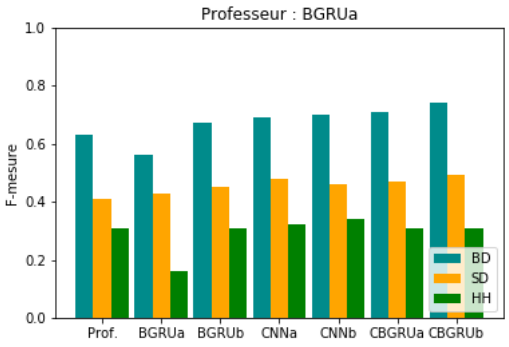
Dans un premier temps, on remarque que les réseaux élèves ont tendance, comme attendu, à obtenir environ les mêmes performances que le réseau professeur : par exemple pour le réseau professeur BGRUa, le professeur est meilleur pour transcrire le BD, moins bon pour le SD et encore moins bon pour le HH ; les élèves font de même. Pour le réseau professeur CNNa ou CNNb, le professeur est meilleur pour le BD, ensuite pour le HH et enfin pour le SD ; les élèves suivent ces tendances.

Par ailleurs, on remarque que lorsque le professeur est un BGRU, les réseaux élèves dotés de convolutions (CNNa, CNNb, CBGRUa et CBGRUb) obtiennent de meilleurs performances que le professeur en BD, avec également une légère amélioration en SD mais pas d'amélioration flagrante en HH. Cela se retrouve de manière moins prononcée dans la performance en BD lorsque le réseau professeur est un CNNa, un CBGRUa ou un CBGRUb. Par contre avec le professeur CNNb, on voit une nette amélioration de la performance en BD. De même, l'amélioration en performance pour SD ou HH ne sont pas évidentes avec les réseaux professeurs CNNb, CBGRUa et CBGRUb. Avec le réseau professeur CNNa, on peut noter une légère meilleure performance en HH.

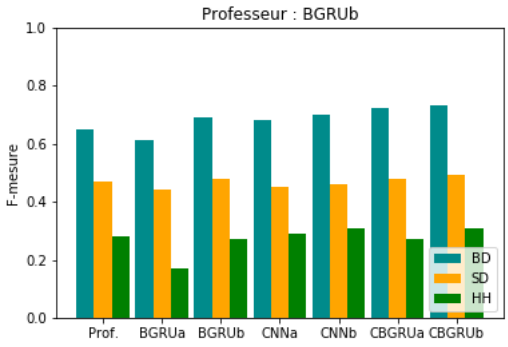
Si on regarde plus en détaille les résultats, on peut remarquer que globalement, lorsque le réseau professeur utilise des convolutions, les réseaux élèves BGRU ont du mal à répliquer les performances obtenues par les réseaux professeurs. Cela peut s'expliquer par le fait qu'un professeur qui a appris les modèles acoustiques de BD, SD et HH n'a aucune information basée sur les récurrences temporelles à fournir au réseau élève, qui par conséquent a du mal à s'adapter. A l'inverse, lorsque le réseau professeur est un BGRU, les réseaux CNN et CBGRU parviennent à obtenir de bonnes performances malgré le fait qu'ils n'ont pas de capacité à analyser des dépendances temporelles au sein des données. Il semble donc que le contenu spectral acoustique aide plus les réseaux que les dépendances temporelles pour la TAB.

Les figures 8.3a, 8.3b, 8.3c, 8.3d, 8.3e, 8.3f montrent les résultats pour l'expérience utilisant les informations de temps et premiers temps. Globalement, on note une amélioration de la transcription surtout pour BD, pour tous les réseaux professeurs. Cela peut s'expliquer par le fait que, étant donné les styles musicaux que contient Billboard 800 (latino, hip-hop, pop, rock), BD se retrouve essentiellement sur les temps, et dans la majorité des cas sur tous les temps (le fameux *four-on-the-floor* qui donne un coup de grosse caisse du chaque temps du morceau en 4/4). Nous ne notons pas d'amélioration nette de la transcription de HH, et celle de SD n'est améliorée sensiblement que pour le réseau professeur BGRUa, ce qui rend l'interprétation confuse. Cette expérience ne permet pas de conclusion évidente sur l'utilité du paradigme professeur avec les temps et premiers temps, mais cela est peut-être dû à la performance imparfaite de Madmom, méritant une investigation plus poussée.

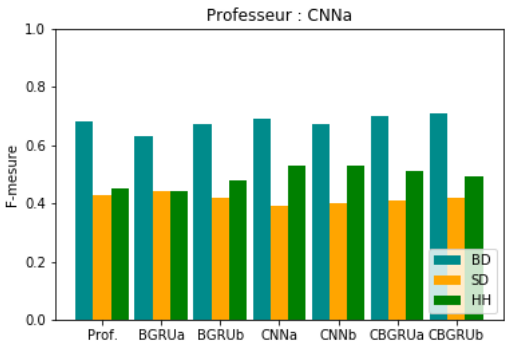
En conclusion, on peut dire que l'utilisation du paradigme professeur-élève permet l'amélioration de la transcription dans certain cas, selon le modèle utilisé pour le réseau professeur et celui de l'élève. L'utilisation de ce paradigme mériterait d'essayer davantage de combinaisons pour améliorer les performances. Par exemple, au lieu d'utiliser un seul réseau professeur pour annoter l'ensemble de données Billboard 800, on pourrait annoter le même ensemble plusieurs fois et prendre la moyenne des prédictions des professeurs comme annotation, à la



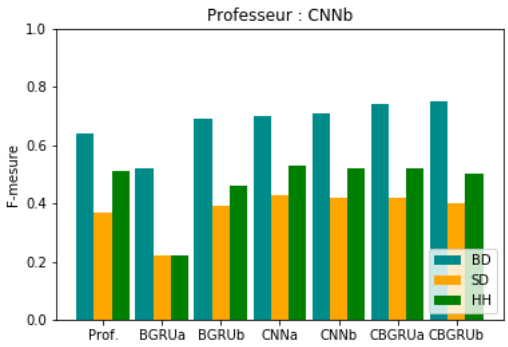
(A) Résultat des réseaux élèves entraîné sur les annotations du réseau professeur BGRUa.



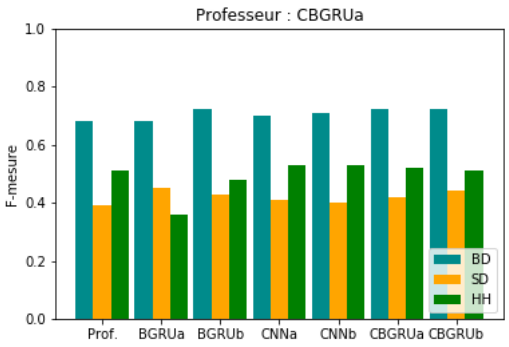
(B) Résultat des réseaux élèves entraîné sur les annotations du réseau professeur BGRUb.



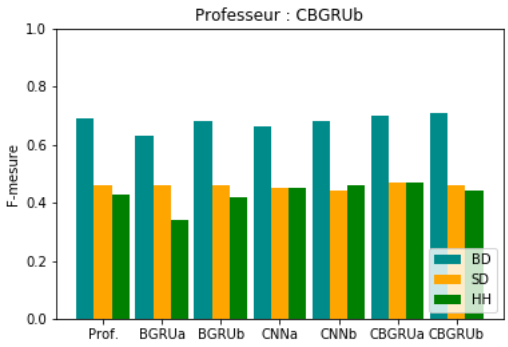
(C) Résultat des réseaux élèves entraîné sur les annotations du réseau professeur CNNa.



(D) Résultat des réseaux élèves entraîné sur les annotations du réseau professeur CNNb.



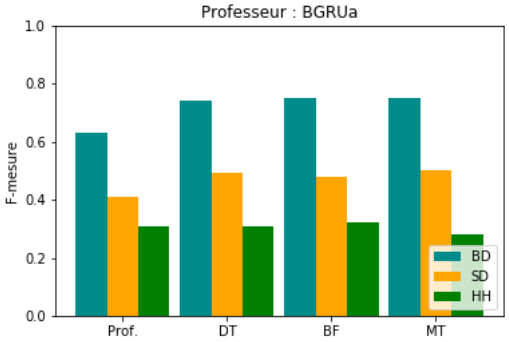
(E) Résultat des réseaux élèves entraîné sur les annotations du réseau professeur CBGRUa.



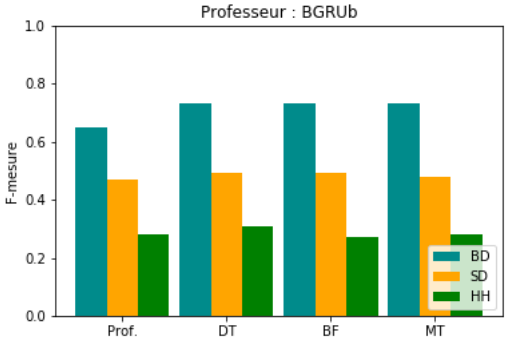
(F) Résultat des réseaux élèves entraîné sur les annotations du réseau professeur CBGRUb.

FIGURE 8.2: Comparaison des résultats de plusieurs réseaux élèves avec plusieurs réseaux professeurs. Les réseaux professeurs ont tous été entraînés sur SMT-Drums.

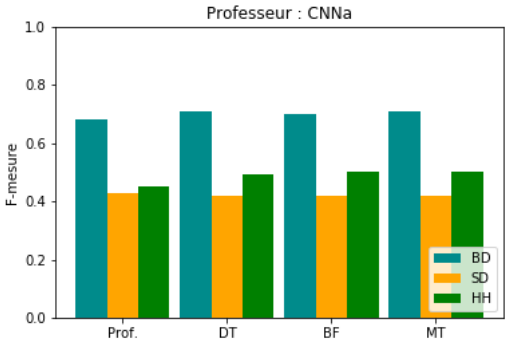
manière d'un vote démocratique. Par exemple, utiliser un professeur BGRU pour les caractéristiques temporelles et un professeur CNN pour les caractéristiques acoustiques permettrait peut-être de fournir au réseau élève différentes manières de comprendre les données.



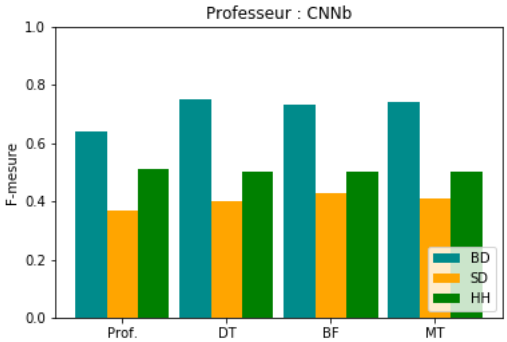
(A) Résultat du CBGRUb pour le réseau professeur BGRUa.



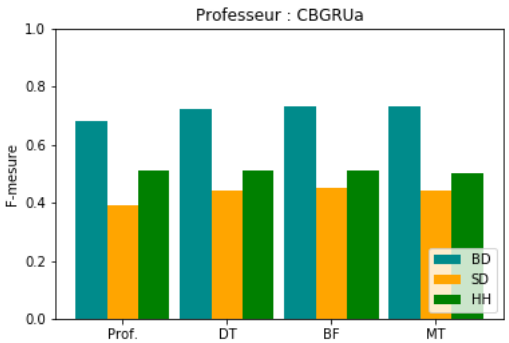
(B) Résultat du CBGRUb pour le réseau professeur BGRUb.



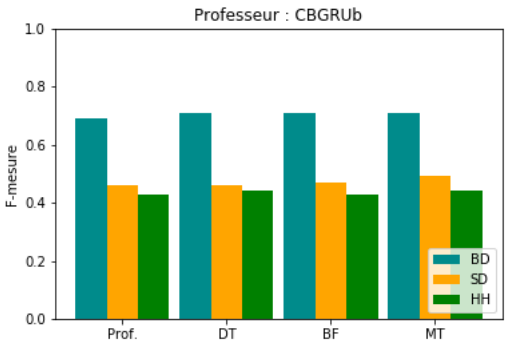
(C) Résultat du CBGRUb pour le réseau professeur CNNa.



(D) Résultat du CBGRUb pour le réseau professeur CNNb.



(E) Résultat du CBGRUb pour le réseau professeur CBGRUa.



(F) Résultat du CBGRUb pour le réseau professeur CBGRUb.

FIGURE 8.3: Comparaison des résultats du réseau BGRUb en utilisant les informations de temps et premiers temps, prédits sur Billboard 800 en utilisant Madmom.

Chapter 9

Conclusion

Ce rapport a présenté le travail que nous avons réalisé durant mon stage. Il a présenté plusieurs expériences réalisées pour répondre au problème de la transcription automatique de batterie. Dans un premier temps, nous avons exploré les capacités des réseaux récurrents, convolutifs et convolutifs récurrents à transcrire la batterie. Nous avons pu montrer que les réseaux comprenant des convolutions obtiennent de meilleures performances que les réseaux récurrents. Dans un second temps, nous avons expérimenté l'utilisation des informations de temps et premiers temps pour aider la transcription, en les renseignant en entrée du réseau tout d'abord, et en entraînant le réseau au multitâche ensuite. Les résultats ont prouvé que l'apport de ces informations en entrée permet une amélioration non-négligeable du système. Enfin, nous avons poursuivi en appliquant le paradigme élève-professeur au problème de TAB pour lever le problème d'ensembles de données de petites tailles. Nous avons pu constater une amélioration plutôt uniforme dans certains réseaux élèves, ce qui conforte l'intérêt de ce paradigme.

Plusieurs directions futures sont ouvertes pour poursuivre ces travaux :

- Analyser plus rigoureusement les valeurs des hyperparamètres utilisées pour les réseaux de neurones. Ceux-ci n'ont pas pu faire l'objet d'exploration par manque temps, mais une étude plus poussée serait intéressante ;
- Concernant le paradigme professeur-élève, il serait intéressant d'augmenter le nombre de réseaux professeurs pour analyser si cela augmente la performance des élèves.
- Evaluer l'utilisation de différentes représentations en entrée: les éléments de batterie étant percussifs, une représentation en échelle logarithmique de fréquence n'est peut-être pas toujours la mieux adaptée car dans ce problème la notion d'hauteur de notes n'existe pas, bien qu'elle peut guider les systèmes dans leur analyse. Il sera intéressant d'évaluer d'autres représentations en entrée des réseaux : transformation à Q-constant (CQT), cepstre ou ondelette.

Bibliography

[DG14] Christian Dittmar and Daniel Gärtner. "Real-time transcription and separation of drum recordings based on NMF decomposition". In: *Proceedings of the 17th International Conference on Digital Audio Effects (DAFx-14)*. 2014.

[GR04] Olivier Gillet and Gaël Richard. "Automatic transcription of drum loop". In: *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2004.

[GR06] Olivier Gillet and Gaël Richard. "ENST-Drums: an extensive audio-visual database for drum signals processing". In: *Proceedings of the 7th International Conference on Music Information Retrieval (ISMIR'06)*. 2006.

[GR08] Olivier Gillet and Gaël Richard. "Transcription and separation of drum signals from polyphonic music". In: *IEEE Transactions on Audio, Speech, and Language Processing* 16.3 (2008), pp. 529–540.

[MDG13] Marius Miron, Matthew E.P. Davies, and Fabien Gouyon. "An open-source drum transcription system for pure data and max MSP". In: *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2013, pp. 221–225.

[MF07] Arnaud Moreau and Arthur Flexer. "Drum transcription in polyphonic music using non-negative matrix factorisation". In: 2007.

[MP43] Warren S. McCulloch and Walter H Pitts. "A logical calculs of the ideas immanent in nervous activities". In: *Bulletin of Mathematical Biophysics* (1943).

[Nak+05] Tomoyasu Nakano et al. "Voice drummer: a music notation interface of drum sounds using voice percussion input". In: *Proceedings of the Annual ACM Symposium on User Interface Software and Technology (UIST)*. 2005, pp. 49–50.

[Pau09] Jouni Paulus. "Signal processing methods for drum transcription and music structure analysis". PhD thesis. Tampere University of Technology, 2009.

[PV05] Jouni Paulus and Tuomas Virtanen. "Drum transcription with non-negative spectrogram factorisation". In: *Proceedings of the European Signal Processing Conference (EUSIPCO)*. 2005.

[Ros+15] Mathias Rossignol et al. "Alternate level clustering for drum transcription". In: *Proceedings of the European Signal Processing Conference (EUSIPCO)*. 2015.

[RPL15] Axel Röbel, Jordi Pons, and Marco Liuni. "On automatic drum transcription using non-negative matrix deconvolution and itakura-saito divergence". In: *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2015.

[Sch85] W. Andrew Schloss. "On the automatic transcription of percussive music - from acoustic signal to high-level analysis." PhD thesis. Stanford University, 1985.

[SSH16] Carl Southall, Ryan Stables, and Jason Hockman. "Automatic drum transcription using bi-directional recurrent neural networks". In: *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*. 2016.

[SSH17] Carl Southall, Ryan Stables, and Jason Hockman. "Automatic drum transcription for polyphonic recordings using soft attention mechanisms and convolutional neural networks". In: *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*. 2017.

[TDDB05] Koen Tanghe, Sven Degroeve, and Bernard De Baets. "An algorithm for detecting and labeling drum events in polyphonic music". In: *Proceedings of first MIREX*. 2005.

[TKM05] George Tzanetakis, Ajay Kapur, and Richard I. McWalter. "Subband-based drum transcription for audio signals". In: *Proceedings of Workshop on multimedia signal processing*. 2005.

[VDK16] Richard Vogl, Matthias Dorfer, and Peter Kness. "Recurrent neural networks for drum transcription". In: *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*. 2016.

[VDK17] Richard Vogl, Matthias Dorfer, and Peter Knees. "Drum transcription from polyphonic music with recurrent neural networks". In: *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2017.

[Vog+17] Richard Vogl et al. "Drum transcription via joint beat and drum modeling using convolutional recurrent neural networks". In: *Proceedings of the 18th International Conference on Music Information Retrieval (ISMIR'17)*. 2017.

[VS+] Dirk Van Steelant et al. "Classification of percussive sounds using support vector machines". In:

[WL15a] Chih-Wei Wu and Alexander Lerch. "Drum transcription using partially fixed non-negative matrix factorization". In: *Proceedings of the European Signal Processing Conference (EUSIPCO)*. 2015.

[WL15b] Chih-Wei Wu and Alexander Lerch. "Drum transcription using partially fixed non-negative matrix factorization with template adaptation". In: *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*. 2015.

[WL17] Chih-Wei Wu and Alexander Lerch. "Automatic drum transcription using the student-teacher learning paradigm with unlabeled music data". In: *Proceedings of the 18th International Conference on Music Information Retrieval (ISMIR'17)*. 2017.

[Wu+17] Chih-Wei Wu et al. "A review of automatic drum transcription". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* (2017).

[Zil+02] Aymeric Zils et al. "Automatic extraction of drum tracks from polyphonic music signals". In: *Proceedings of the 2nd International Conference on Web Delivering of Music*. 2002.